# TABLE OF CONTENTS

1 Getting started 3
   1.1 Using Python 3
   1.2 Installing MongoDB 4
   1.3 Installing CytoPy 4

2 Creating Projects and populating with data 5
   2.1 Connecting to the database 5
   2.2 Projects & Subjects 5
   2.3 Adding Cytometry data 6

3 Autonomous gating 11
   3.1 GatingStrategy & Gate’s 12
   3.2 Types of Gates 18
   3.3 Editing a GatingStrategy & FileGroup 24
   3.4 Using control data 25

4 Assessment of inter-sample variation 27
   4.1 Loading data for analysis 27
   4.2 Choosing a reference sample 28
   4.3 Visualising univariant differences 28
   4.4 Visualising multivariant differences 29
   4.5 Building a similarity matrix 30

5 Single cell phenotype classification by supervised learning 33
   5.1 Introducing the CellClassifier 34
   5.2 Training 36
   5.3 Validating 37
   5.4 Predicting populations and troubleshooting with backgating 39
   5.5 Keras 41
   5.6 Saving classifiers 42

6 Single-cell phenotype classification by high-dimensional clustering 43
   6.1 The Clustering class 43
   6.2 Clustering functions 44
   6.3 Clustering a single sample 45
   6.4 Introducing exploratory data analysis with Explorer 45
   6.5 MetaClustering with PhenoGraph 48

7 Feature extraction, selection, and summarisation 53
   7.1 Summarising the proportion of cell populations/clusters 53
   7.2 Dimensionality reduction 54
In recent years there has been an explosion in Cytometry data analysis tools in the open source scientific community. This expansion is looking to soon replace traditional methods such as manual gating with sophisticated automated algorithms.

Although exciting, this can be daunting to those from a traditional immunology background and those that are new to programming. Additionally, current tools have a loose structure in the steps taken in analysis, resulting in large custom scripts, poor reproducibility, and insufficient data management.

CytoPy was created to address these issues. It was created with the general philosophy that given some cytometry data and a clinical/experimental endpoint, we wish to find what properties separate groups (e.g. what cell populations are important for identifying a disease? What phenotypes are changing in response to a stimulus? etc). The pipeline itself is centered around a MongoDB database, is built in the Python programming language, and designed with a ‘low code’ API, greatly simplifying cytometry analysis.

CytoPy was authored by Ross Burton and the Eberl Lab at Cardiff University Infection and Immunity Research Institute. CytoPy is maintained on GitHub (https://github.com/burtonrj/CytoPy) and all the latest developments can be found here. This project is a working progress and we are eager to expand and improve it’s capabilities. If you would like to contribute to CytoPy please make a pull request or email us at burtonrj@cardiff.ac.uk. For news and latest developments, follow us on Twitter @EberlLab and @burtondatasci

Our accompanying manuscript details the application of CytoPy to a novel immunophenotyping project focused on patients receiving peritoneal dialysis who were admitted on day 1 of acute peritonitis before commencing antibiotic treatment. The original peritonitis dataset can be accessed here.
Welcome to CytoPy, a data-centric analytical framework for cytometry data. The source code for CytoPy is stored and maintained at https://github.com/burtonrj/CytoPy. You can also read our pre-print manuscript. It is our hope that CytoPy opens the door to a bioinformatics approach to Cytometry analysis by using the beginner friendly programming language, Python. CytoPy was developed in Python 3.7. If you’re new to programming that is fine, as we have added some information below for installing Python 3.7.

CytoPy makes the assumption that your hypothesis is as follows:

“We have collected data on humans/mice/cell-lines in X experimental/clinical conditions and we want to test for cell phenotypes that differentiate between these conditions”

We recognise that there has been an extraordinary effort to develop bioinformatics tools for addressing questions like the one above using Cytometry data. Some of these tools even feature within CytoPy itself (see https://www.biorxiv.org/content/10.1101/2020.04.08.031898v2 for details). So why CytoPy and not one of the 30+ tools in the literature? CytoPy is an agnostic framework that will allow you to apply autonomous gates, supervised classification, and high-dimensional clustering algorithms, and it achieves all this whilst providing a low-code interface and a central data repository in the form of a MongoDB database. We want to make the amazing tools in the literature more accessible to immunologists whilst improving the analysts experience.

Bioinformatics is a jungle of possible methods and an analysis tends to amount hundreds of scripts, lots of csv files, and lots of headaches. In CytoPy all experimental/clinical metadata is housed within a central database, which can be hosted locally or online. Linked to this metadata are the results of your gating, classification and clustering, and this is all stored in one central repository. Analysis is iterative and this fact has steered the design of CytoPy; an object-orientated interface built atop the MongoDB ORM makes interacting with this database a breeze.

Let’s get started by explaining how to setup CytoPy on your local system...

1.1 Using Python

For those of you who are brand-new to programming or not familiar with Python, this section will provide some helpful tips on how to get started.

Note: CytoPy assumes you are familiar with Python version 3, have some experience with object-orientated programming, and are happy with the concepts of Numpy arrays, Pandas DataFrames, and general data science concepts like machine learning and clustering

If the above note is daunting, please don’t threat! There are lots of resources linked below and I believe with 6/8 weeks hard work anyone can grasp enough to start using CytoPy. So if you’re brand new to Python, please start at one of the following resources:

- https://www.learnpython.org/ (Basics)
• https://www.freecodecamp.org/news/want-to-learn-python-heres-our-free-4-hour-interactive-course/ (Basics)
• https://www.youtube.com/watch?v=rfscVS0vrbw (Basics)
• https://jakevdp.github.io/PythonDataScienceHandbook/ (Data science)

To install Python 3 locally we recommend either:

• Anaconda: a popular data science platform that bundles Python, R and a very intuitive environment manager together. This also comes with a handy graphical user interface for managing all your software. For newbies we recommend this. Consult the Anaconda website for a tutorial on how to download and install.

• Alternatively, you can download and install from Python.org. For this approach we recommend installing virtualenv (see here for information about venv) to manage your Python environments.

We recommend making yourself familiar with programming environments before getting started. CytoPy has many dependencies and the best way to prevent any problems down the line is to keep CytoPy contained within it’s own programming environment.

1.2 Installing MongoDB

CytoPy assumes you have a MongoDB server up and running, either locally or in the cloud. If you wish to install MongoDB locally, then download the community edition here. CytoPy makes the assumption that MongoDB is hosted locally, but if it is hosted remotely this can be specified when connecting to the database by providing the host address, port, and authentication data.

Although to use CytoPy you don’t need to know much about MongoDB, we recommend that the user learns a bit about this powerful document-based database and how to perform simple troubleshooting and queries. To learn more we recommend checking out the resources over at FreeCodeCamp (https://www.youtube.com/watch?v=E-1xI85Zog8).

We also suggest using some sort of MongoDB GUI for troubleshooting and helping with understanding how data is stored. Robo3T is a free tool that can be used for this purpose.

1.3 Installing CytoPy

So you have Python 3 installed, you have MongoDB installed, and now you’re ready to get started with CytoPy. First you want to create a new programming environment and activate that environment. Once inside that programming environment, either run the following command:

```
pip3 install git+https://github.com/burtonrj/CytoPy.git
```

Or, alternatively, download the source code and run the setup file as so:

```
python3 setup.py install
```

For a detailed overview of CytoPy we direct you to our manuscript. The remaining tutorials on this site display the functionality of CytoPy by replicating the analysis described within our manuscript.
CREATING PROJECTS AND POPULATING WITH DATA

To start any analysis we will need to populate our database. Here we will detail how to connect to a database, create projects, populate those projects with subjects, add experiments, and add *.fcs files.

2.1 Connecting to the database

If the database is hosted locally, then connecting is simple. We use the global_init function at the beginning of our analysis (whether this is in a script or in a notebook). All we have to do is pass the name of our database into this function, for example this will connect to the database called “CytoPy”:

```python
from CytoPy.data.setup import global_init
global_init('CytoPy')
```

If the database does not yet exist, it will automatically be generated.

If you’re connecting to a remote database, you can pass other arguments such as host name and port as keyword arguments. These are passed to a call to mongoengine.connection.register_connection. Please see the mongoengine documentaiton for details.

Currently CytoPy does not support connection to multiple databases at once. All connections default to the alias “core”.

2.2 Projects & Subjects

Inside CytoPy all data is contained within MongoDB documents. In the hierarchy of control, the Project document is top and controls access to all Experiments and contained data. We can start by defining a Project like so:

```python
from CytoPy.data.project import Project
# Create a project for the PD data
pd_project = Project(project_id='Peritonitis', owner='ross')
pd_project.save()
```

We then retrieve this project using the get function and the relevant query:

```python
pd_project = Project.objects(project_id='Peritonitis').get()
```

We use the Project document to access all the data within our analysis. A Project will have connections to one or more Subject’s. A Subject can be associated to more than one Project.

A Subject contains all the metadata related to one individual patient/mouse/cell-line (some biological ‘subject’). The Subject document is dynamic, that is, we can add any variables we like. In addition to this, there are special embedded documents that can be added to a Subject. Multiple embedded documents can be added for each:
• **Bug** - each **Bug** document details a single microbiological isolate related to the associated subject e.g. if a patient had bacterial pneumonia and *Pseudomonas aeruginosa* had been isolated, you can add a **Bug** document detailing this event (see CytoPy.data.subject.Bug)

• **Drug** - each **Drug** document details the name of a drug administrated, the initiation data, and the end date (see CytoPy.data.subject.Drug)

• **Biology** - each **Biology** document details some biological measurement (see CytoPy.data.subject.Biology)

The purpose of these embedded documents is that some metadata is complicated; a subject can have multiple bacterial isolates, many drugs administrated, or many other biological measurements taken. To be able to suitably associate this metadata to our single cell data in exploratory analysis, we must account for this.

A **Subject** can be created and associated to a **Project** like so:

```python
from CytoPy.data.subject import Subject, Bug, Biology, Drug

bugs = [Bug(org_name='staph', gram_status='positive'), Bug(org_name='e.coli', gram_status='negative')]
bio = [Biology(test='white_cell_count', result=8000, units='cubic millimeter')]
patient = Subject(subject_id='patient_001', peritonitis=True, infection_data=bugs, patient_biology=bio, age=50, gender=1)
patient.save()
pd_project.subjects.append(patient)
pd_project.save()
```

Or using the **Project** API:

```python
pd_project.add_subject(subject_id='patient_001', peritonitis=True, infection_data=bugs, patient_biology=bio, age=50, gender=1)
```

Most tasks can be simplified by using the methods CytoPy classes.

A couple of things to highlight from what we just covered:

- Notice that we added a field called ‘peritonitis’. This is pretty specific to this project. This doesn’t matter because **Subject** is a dynamic document. This means you can add whatever fields you want ad hoc.

- Always remember to save your documents! Changes are not committed to the database unless you call the `save` method.

### 2.3 Adding Cytometry data

To add Cytometry data we have to add an **Experiment** to our project. An **Experiment** houses one or more **FileGroup** documents. Each **FileGroup** represents a biological specimen that can consist of one or more *.fcs* files (primary staining and any controls e.g. Full Minus One or Isotype controls).

An **Experiment** should be generated for each unique staining panel.
2.3.1 Cytometry Panels

Cytometry data can be related according to some staining profile/panel used to describe the cellular contents of a biological sample. In CytoPy, an Experiment must be created for each set of staining conditions. The staining conditions are described by a Panel document.

The Panel document is special because it standardises the input of cytometry data at the point of entry. What do I mean by this? Well imagine a user has collected 10 samples, all stained for the same markers, but each time they collected their data they labelled the metadata in their *.fcs file differently. E.g. in the first sample CXCR3 was labelled “CXCR3”, but in the next it was “CXCR 3”, then “CXCR-3” and then “cxcr-3” and so on. You can see how this could be an issue when it comes to analysing our data.

To overcome this problem, we introduce the Panel document. It contains a description of the expected marker and channel names, the marker and channel names that should be set as standard, and then a regular expression used to match markers and channels to their appropriate standard. If you haven’t heard of regular expression before, you can find info here:

- https://docs.python.org/3/howto/regex.html
- https://www.youtube.com/watch?v=ZiQFUJhPqMM

To create a Panel we first prepare an Excel template. You can get a blank template here: https://github.com/burtonrj/CytoPy/tree/master/CytoPy/assets

The template Excel spreadsheet contains two sheets:

- Mappings - these are the channel/marker mappings and names of the channels and markers that should be used as standard. The channels and markers will appear with these names throughout your analysis as every file will be standardised accordingly.

- nomenclature - this sheet details, for each channel and each marker standard, a regular expression search term, a list of possible “permutations” and whether the search should be case sensitive. This sheet is used to search each *.fcs file and match the marker/channel to the appropriate standard name

Below is an example of the mappings sheet for one staining panel in the Peritonitis study detailed in our manuscript:
<table>
<thead>
<tr>
<th></th>
<th>channel</th>
<th>marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FSC-A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FSC-W</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FSC-H</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SSC-A</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SSC-H</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SSC-W</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Alexa Fluor 488-A</td>
<td>CD14</td>
</tr>
<tr>
<td>10</td>
<td>PerCP-A</td>
<td>CD16</td>
</tr>
<tr>
<td>11</td>
<td>Alexa Fluor 647-A</td>
<td>Siglec8</td>
</tr>
<tr>
<td>12</td>
<td>Alexa Fluor 700-A</td>
<td>CD45</td>
</tr>
<tr>
<td>13</td>
<td>APC-Cy7-A</td>
<td>CD3</td>
</tr>
<tr>
<td>14</td>
<td>Alexa Fluor 405-A</td>
<td>CD1c</td>
</tr>
<tr>
<td>15</td>
<td>AmCyan-A</td>
<td>LD</td>
</tr>
<tr>
<td>16</td>
<td>BV605-A</td>
<td>CD15</td>
</tr>
<tr>
<td>17</td>
<td>BV711-A</td>
<td>HLA-DR</td>
</tr>
<tr>
<td>18</td>
<td>PE-A</td>
<td>CD116</td>
</tr>
<tr>
<td>19</td>
<td>PE-Cy7-A</td>
<td>CD19</td>
</tr>
</tbody>
</table>

Cell at row 25, column 3 is blank.
An example of the nomenclature sheet:

This study spanned over 2 years and so there was a vast array of naming conventions used within the cytometry files provided. Using regular expression and a list of known permutations, we can match the channels and markers to the correct standard.

Templates can be made manually by editing the default template. A useful resource for testing regular expressions is https://regexr.com/

Some convenience functions for exploring the range of channel mappings and creating templates are:

- CytoPy.data.read_write.explore_channel_mappings - given the path to a directory containing one or more *.fcs files, returns a list of dictionaries for all unique channel/marker names
- CytoPy.data.read_write.fcs_mappings - given the path to a single *.fcs file, return the channel/marker names

Once we have our template ready, we can create out Panel document:

```python
from CytoPy.data.experiment import Panel
n_panel = Panel(panel_name='PD_N_Panel')
n_panel.create_from_excel('path/to/template.xlsx')
n_panel.save()
```

Or, we can provide the template when generating our Experiment and associate it to the Experiment at the same time. This is the advised method for creating panels and experiments.

### 2.3.2 Creating Experiments

As explained above, for each staining condition we are going to create an Experiment. It is the Experiment that will be used later on to access single cell data. To create a new Experiment we use the add_experiment method of Project:

```python
from CytoPy.data.project import Project
pd_project = Project.objects(project_id='Peritonitis').get()
new_experiment = pd_project.add_experiment('PD_N_Panel',
                                        panel_definition='path/to/template.xlsx',
                                        data_directory='path/to/store/single/
```

2.3. Adding Cytometry data
By providing the path to our panel template, the **Panel** will be created and automatically associated to our new **Experiment**. Alternatively you can provide `panel_name` instead to associate an existing **Panel** to the **Experiment**.

Notice that we also provide the parameter `data_directory`. Single cell data is large and stored in matrices of float values. CytoPy stores the single cell data to disk in HDF files which are mapped to the **Experiment** in the database. We can change the location of this single cell data store at any time using the `update_data_directory` method of **Experiment**. This makes it easy to migrate data between drives. This means we can migrate the data by just updating the mappings in the database.

To load an **Experiment** from our **Project** we use the `load_experiment` method:

```python
experiment = pd_project.load_experiment('PD_N_Panel')
```

We load an **Experiment** and interact with that object to create and access single cell data.

### 2.3.3 Adding *.fcs files

*.fcs files are stored within something called a **FileGroup**. Within a **FileGroup** there should be one primary file and then any other *.fcs files attributed to controls. We refer to this group of related files as one biological sample. So for example if I had three different mice (a, b, and c) and I stained for CD4 and CD8 but had an isotype control for CD4 and CD8 in each, then each mouse would have one **FileGroup** and within each **FileGroup** would be 3 files: the primary staining and two isotype controls.

We add the files for a biological sample like so:

```python
experiment.add_new_sample(sample_id='patient_001_N',
                          primary_path='path/to/primary_staining.fcs',
                          controls_path={'CD27': 'path/to/CD27_FMO.fcs'},
                          subject_id='patient_001',
                          compensate=True)
```

In the method call above we provided the following:

- **sample_id**: this is the ID we will use to retrieve data from this sample
- **primary_path**: this is the file path to the primary fcs file
- **controls_path** (optional): a dictionary with the key corresponding to the ‘control_id’ and the ‘path’ where the control file is located as the value
- **subject_id** (optional): the **Subject** this sample should be referenced too. This will allow metadata to be pulled and associated at a later date
- **compensate**: whether to apply compensation at the point of entry. CytoPy will search the fcs file for an embedded compensation matrix. If this is not present, the user should provide the path to a csv file of the compensation matrix with the parameter `comp_matrix`. 

---

**Chapter 2. Creating Projects and populating with data**
CHAPTER
THREE

AUTONOMOUS GATING

We were heavily inspired by the great work of OpenCyto, an autonomous gating framework developed for R Bioconductor. We recognised early on however, that autonomous gating is limited in its capabilities. First of all, what do we mean by an “autonomous gate”.

An autonomous gate is one that replicates the actions of a manual gate by applying some sort of algorithm to the same data in one/two dimensional space. Autonomous gates as a means of automated cytometry analysis have the following issues:

• Being a direct emulation of manual gating, they suffer from the same bias as a manually derived gating strategy
• Unsupervised algorithms are often heavily influenced by hyperparameters, the optimal values of which will deviate from one biological sample to the next
• Algorithms applied in this way don’t take into consideration the “global topology” of the immunological landscape that could be captured by considering all available variables

This is why CytoPy focuses instead on using supervised machine learning and high-dimensional clustering that has access to all available variables when modelling the cytometry data and generalise better.

Despite this, we decided to include automated gating as a function of CytoPy. The reason for this is that we found, no matter the quality of data, some amount of ‘gating’ is required. Before we can classify cells autonomously we must remove debris, dead cells, and other artifacts. The efficiency and standardisation of this process can be greatly improved through the use of autonomous gates, which we found had good performance for this task.

Pre-processing normally follows these steps:

1. Design a GatingStrategy using some chosen example from your Experiment, adding Gate’s for the populations you want
2. Apply that GatingStrategy to each biological sample within an Experiment
3. The GatingStrategy generates Population’s as a result of this gating process and they are saved to the underlying database in each biological samples FCSGroup
4. The GatingStrategy results in an ‘identical’ root population being generated for each biological sample e.g. T cells or CD45+ leukocytes. This root population is the point from which supervised classification and high-dimensional clustering take place.

A Population is generated whenever a gate is applied or when classified by some supervised classification algorithm. High-dimensional clustering algorithms can be applied to a Population and the resulting clusters saved within that Population.
3.1 GatingStrategy & Gate’s

The GatingStrategy class is used to create and apply gates, generate Population**s and visualise data. The **GatingStrategy class is very powerful and we recommend checking out the API reference for details (see CytoPy.data.gating_strategy.GatingStrategy).

Often what we want to do is create a ‘gating strategy’: a sequence of gates applied to each biological sample, rendering the root population from which we will start more thorough analysis (e.g. CD3+ lymphocytes or CD45+ single live cells). The GatingStrategy allows us to add Gate’s in sequence using some example data from our Experiment. This can then be saved to the database and applied to subsequent data without manual intervention.

We initiate a GatingStrategy object and load a FileGroup from an Experiment like so:

```python
from CytoPy.data.gating_strategy import GatingStrategy
# Load the project
project = Project.objects(project_id='Peritonitis').get()
# Load the experiment
exp = project.load_experiment('PD_T_PDMCs')
# Create a new gating strategy
template = GatingStrategy(name="T cells", verbose=True)
# Load a FileGroup (biological sample)
template.load_data(experiment=exp, sample_id='307-01')
# Save the template
template.save()
```

We can save a GatingStrategy to the database using the save method. When calling this, the associated FileGroup is also saved with any new Populations generated being commited to the database.

3.1.1 Plotting

When plotting single cell data in one or two dimensional space, behind the scenes the CreatePlot class is used. This generates a Matplotlib Axes object upon which data, gates, and Populations are plotted. For all plotting methods of the GatingStrategy, the plots can be customised by specifying arguments for CreatePlot using the create_plot_kwargs parameter, a dictionary of keyword arguments. The user should consult CytoPy.flow.plotting for details, but notable mentions are:

- transform_x & transform_y: how the axis should be transformed (see CytoPy.flow.transforms) for available methods; defaults to ‘logicle’ transform
- xlabel & ylabel: how to label the x and y axis; defaults to the name of the fluorochrome/isotype/marker chosen for each axis
- ax: an existing Matplotlib axes object can be provided for additional control
- bins: either an integer value or a string value signifying the method that should be used to estimate bin size for 2D histograms (defaults to “scott” method)
- bw: either a float value or string value signifying the method that should be used to estimate the optimal bandwidth for 1D kde plot (defaults to “scott” method)

CreatePlot has numerous methods that GatingStrategy will call to plot data. The inputs to these methods can be modified or overwritten by supplying additional keyword arguments (kwargs) to the GatingStrategy method.

For every FileGroup there will always be one population present by default. This is called the ‘root’ population. Not to be confused with what we refer to before. This ‘root’ is a population that contains all the events in an fcs file. We can plot the ‘root’ population using the plot_population method:
3.1.2 Creating and applying a Gate

Creating a new gate is performed by the following steps:

1. Define the gate and create a new Gate object
2. Preview the Gate using the preview_gate method of GatingStrategy
3. Label the child populations created by the Gate object
4. Apply the Gate in the GatingStrategy using the apply_gate method

There are three types of Gate objects, all of which inherit their functionality from Gate. But you will commonly interact instead with these three subtypes. They are: ThresholdGate, PolygonGate and EllipseGate. There classes are described in detail in the section below titled Types of Gates.
These Gate objects are independent of the GatingStrategy and instead act on a Pandas DataFrame of single cell data, returning Population objects. Population’s represent a group of single cell events that are contained within the geometric definitions of a Gate. This geometry is described in the PopulationGeometry.

We want gates to be autonomous and data-driven but traditionally we inject our knowledge and understanding of the immune system to ‘annotate’ populations of interest. In CytoPy, we label the child populations generated when creating our Gate using some example data. When we label the child populations, we are telling CytoPy “this is a population I am interested in and I want to identify similar populations in other data you’re exposed too”. The geometry of the population will be described and saved within the Gate object, so when exposed to new data in the future, the same unsupervised algorithm will be applied, generating slightly different results (since it is new data), but the new populations will be labelled by matching them to the original children of the Gate.

How does this matching work? CytoPy first calculates the total area overlapping between the geometries. This is then multiplied by the minkowski distance between the vector mean of the original child population and the newly discovered population. The new population will inherit the label of the child deemed most ‘similar’ to it using this score.

So let’s go over an example. Say we want to gate live CD3+ cells:

.. image:: images/gating/cd3_ld.png

We would start by defining our Gate. We’re going to use a PolygonGate and the MiniBatchKMeans algorithm from Scikit-Learn (methods we can access will be explained in detail in Types of Gates):

```python
# define the gate

gate = PolygonGate(gate_name="cd3_live_gate",
                   parent="root",
                   x="CD3",
                   y="LD",
                   transformations={"x": "logicle",
                                    "y": "logicle"},
                   method="MiniBatchKMeans",
                   method_kwargs={"n_clusters": 4,
                                   "batch_size": 1000,
                                   "n_init": 10,
                                   "random_state": 42})

template.preview_gate(gate)
```
There are some common arguments we provide to a Gate, regardless of the type of Gate we are using:

- **gate_name**: the name we want the gate to have in the gating strategy. This will be saved to the database and will be how we refer to the gate in future tasks
- **parent**: the name of the population this gate is being applied to
- **x**: name of the variable on the x-axis
- **y** (optional): if gating two-dimensions, the name of the variable on the y-axis (note: only ThresholdGate supports one dimensional gating)
- **transformations**: a dictionary describing how each axis should be transformed prior to applying the gate

Note: we can also modify the appearance of the plot generated by passing dictionaries to `create_plot_kwargs` and `plot_gate_kwargs` arguments of `preview_gate`.

When we have previewed the gate, we then label the child populations:

```python
gate.label_children({'A': 'Live CD3+'})
```

We’re only interested in that population in the bottom right, so we just label “A”. This signifies to CytoPy that the other populations identified are not of interest and will be ignored.

Once they have been labelled, we use the `apply_gate` method and use the argument `add_to_strategy` to associate this Gate to the GatingStrategy:

```python
template.apply_gate(gate, add_to_strategy=True)
```
Some simple statistics regarding the newly generated population will also be printed to screen:

```
----- cd3_live_gate -----
Parent (root) n: 585650
...child Live CD3+ n: 303811; 51.87586442414411% of parent
```

The ‘Live CD3+’ **Population** is now associated to the **FileGroup** currently connected to the **GatingStrategy**, ‘template’. The **Population** will not be saved to the database however until we call the **save** method of **GatingStrategy**. We can view the populations currently stored in the linked **FileGroup** using the **list_populations** method or we can print a population tree using the **print_population_tree** method:

```
template.print_population_tree()
```

```
root
   — Live CD3+
```

You can access the **FileGroup** directly by accessing the **filegroup** attribute of the **GatingStrategy**.
### 3.1.3 Actions

There are some processes in the identification of cell populations that falls outside the scope of polygon or threshold gates that subset ‘clusters’ of cells. The **GatingStrategy** offers the **Action** class (which is embedded within the **GatingStrategy** class) and this allows a user to define merging or subtracting tasks. This **Action** is applied to a target population, *left*, in relation to some other population, *right*. The result is the generation of a new **Population**, either by subtracting the elements of *right* from *left* or by merging *left* with *right*.

As an example, say we had two populations derived from our ‘Live CD3+’ population, we will call them ‘Population A’ and ‘Population B’. We can create a new population by merging these two, creating a ‘Population AB’, using an **Action**:

```python
from CytoPy.data.gating_strategy import Action
merged_ab = Action(action_name="merge_ab",
                   method="merge",
                   left="A",
                   right="B",
                   new_population_name="AB")
template.apply_action(action=merged_ab, add_to_strategy=True)
```

Now when we apply this **GatingStrategy** to new data, gates will be applied in order of their parent population and once the populations “A” and “B” have been generated, this merge operation will be performed.

### 3.1.4 Applying a GatingStrategy to new data

Once we’re happy with the gates and actions we have added to a **GatingStrategy** we commit it to the database using the **save** method. We can then load an existing **GatingStrategy** using the mongoengine API:

```python
# Save to database
template.save()
# Load a gating strategy
template = GatingStrategy.objects(name="T cells").get()
```

We can load a new sample into the **GatingStrategy** like before:

```python
template.load_data(experiment=exp, sample_id='297-02')
```

And either apply gates/actions individually:

```python
template.apply_gate("cd3_live_gate")
template.apply_action("merge_ab")
```

Or, use the **apply_all** method to apply all gates and actions at once:

```python
template.apply_all()
```
3.2 Types of Gates

We described before that there are three different types of Gate available for autonomous gating. We will now go through these and explain their use in detail.

3.2.1 ThresholdGate

The ThresholdGate is subsetting data in one or two-dimensions. The class offers accessing to gating either through manual one/two dimensional thresholds (a straight line that separates data), quantile gates that draw the threshold at the specified quantile of the data it interacts with, or density driven gates that use a novel algorithm that sets the threshold values based on the properties of the estimated probability density function (PDF) of the data encountered.

To specify which method to use, you can change the value of method when initializing the object. The value should be ‘manual’, ‘quantile’ or ‘density’.

The density driven algorithm estimates the PDF for each dimension using convolution based kernel density estimation, provided by the KDEpy library <https://kdepy.readthedocs.io/en/latest/>. A peak find algorithm <https://github.com/demotu/detecta> is applied and the following performed:

1. If a single peak is found, the threshold is drawn at the inflection point between the local maxima and the minima to the right of the peak. This can be reversed to draw the threshold at the left of the peak by passing \{“inflection_point_kwargs”: \{“incline”: True\}\} to method_kwargs.

2. If two peaks are found, then the threshold is drawn at the local minima between the two peaks

3. If more than two peaks are found, then the PDF is ‘smoothed’ using a Savitzky-Golay filter. The algorithm is rerun until 2 or less peaks are found, with the window of the filter increased at each iteration

Lets see an example of the ThresholdGate applied to identifying the CD3+ population we saw before:

```python
gate = ThresholdGate(gate_name="cd3_live_gate",
                     parent="root",
                     x="CD3",
                     y="LD",
                     transformations={"x": "logicle",
                                      "y": "logicle"},
                     method="density",
                     method_kwargs={"min_peak_threshold": 0.05,
                                     "peak_boundary": 0.1})
template.preview_gate(gate)
```
Additional keyword arguments used to manipulate the behaviour of the underlying algorithm are passed as a dictionary to `method_kwargs`. This is the case for all the Gate classes. We would then label the resulting populations just as before using the `label_children` method and then apply the gate.

The ThresholdGate provides the same functionality for gating in a single dimension. For example, we could gate just along the CD3+ axis:

```python
gate = ThresholdGate(gate_name="cd3_gate",
    parent="root",
    x="CD3",
    transformations={"x": "logicle"},
    method="density")

template.preview_gate(gate)
```
CytoPy, Release 1.0.0

If we want the gate applied to a single axis but plotted on two dimensions, we can modify the plot by passing the
following to the preview_gate method:
template.preview_gate(gate, plot_gate_kwargs={"y": "LD",
"transform_y": "logicle"})

20

Chapter 3. Autonomous gating


3.2.2 PolygonGate

The **PolygonGate** does as its name describes, it generates polygon shapes, or ‘gates’, that encircle events to create **Populations**. A **PolygonGate** can create one or more polygon shapes at once (stored as a **PolygonGeom**, which inherits from **PopulationGeom**). These shapes are formed by calculating the convex envelope of the clustering assignments of a clustering algorithm. Alternatively, the user can also create static gates by setting `method` to ‘manual’ and providing polygon coordinates as ‘x_values’ and ‘y_values’ in `method_kwargs`.

As for the algorithms available, the **PolygonGate** provides access to any clustering algorithm from the popular Scikit-Learn `cluster` module. Additionally, the HDBSCAN <https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html> library is also accessible. CytoPy’s design is such that it uses the common Scikit-Learn signature for these algorithms. If you know of or have an unsupervised clustering algorithm that you would like included, that follows the Scikit-Learn template, please make a pull request at our Github repository <https://github.com/burtonrj/CytoPy> or contact us at burtonrj@cardiff.ac.uk.

Using an algorithm is simple; we specify the class name in `method`, like we did before when gating live CD3+ cells, where we used mini-batch K means. The parameters that you would pass to initiate the scikit-learn object are given as a dictionary in `method_kwargs`.
3.2.3 EllipseGate

The EllipseGate inherits all functionality of the PolygonGate and is similar in many ways. The difference is that elliptical gates are generated instead of polygons, and this is achieved using probabilistic models from the Scikit-Learn *mixture* module <https://scikit-learn.org/stable/modules/mixture.html#mixture>.

The shapes themselves are generated by drawing a confidence interval around the models components, creating elliptical ‘gates’. The confidence intervals themselves are estimated from the covariance matrices of each component and because of this, only ‘full’ covariance type is supported.

An example of gating live CD3+ cells is shown below when using Gaussian mixture models:

```python
gate = EllipseGate(gate_name="cd3_gate", parent="root", x="CD3", y="LD", transformations={"x": "logicle", "y": "logicle"}, method="GaussianMixture", method_kwargs={"n_components": 3})
template.preview_gate(gate)
```

Some more examples of popular clustering algorithms that can be accessed for autonomous gates are shown in the figure below.
3.2. Types of Gates
3.3 Editing a GatingStrategy & FileGroup

We all know that sometimes not everything goes to plan. This is true for autonomous analysis as it is for any analysis and it is easy for an autonomous gate to misclassify data due to a technical error in our data that we did not anticipate. For this reason it is vital that we can interfere and edit our GatingStrategy and/or FileGroup.

3.3.1 Deleting populations

If we want to delete a population, we can use the delete_populations method of either the underlying FileGroup or the GatingStrategy. We simply provide a list of populations we want to delete:

```python
# These two commands do the same thing:
template.filegroup.delete_populations(['A', 'B'])
template.delete_populations(['A', 'B'])
```

Remember, we can access the FileGroup using the filegroup attribute of the GatingStrategy. You can also load a FileGroup directly from the Experiment:

```python
filegroup = exp.get_sample('297-02')
filegroup.delete_populations(['A', 'B'])
```

**Warning:** When you delete a Population, all Populations that are ‘downstream’ (that is, they are the parent/grandparent/great-grandparent etc) will also be deleted. You cannot delete the ‘root’ population.

**Note:** The removal of population will not be made permanent until the save method has been called either on the FileGroup directly or on the GatingStrategy it is currently loaded into.

3.3.2 Deleting gates

Deleting a gate is simple, we specify the gate name and delete with the delete_gate method:

```python
template.delete_gate('cd3_live_gate')
```

**Warning:** When you delete a gate this does not propagate to Populations that have been generated by this gate. They must be removed manually.

3.3.3 Editing a gate

We can edit a gate using the edit_gate method. We simply specify the name of the gate and update the ‘geometry’ that defines the gate. For a threshold gate this is the x_threshold and also the y_threshold if the gate is applied in two dimensions. For a polygon gate/ellipse gate we specify the new x and y coordiingates in x_values and y_values respectively. Example code for modifying a threshold:

```python
template.edit_gate(gate_name='cd3_live_gate',
                  x_threshold=0.6,
                  y_threshold=0.8)
```
This would update the threshold gate ‘cd3_live_gate’ and resulting Populations for the FileGroup currently associated to our GatingStrategy. The immediate Populations (the ones generated by this gate directly) will be modified to fall within the bounds of the new geometry. Gates that occur downstream of this one (where the parent population is downstream of the populations generated by the modified gate) will be applied statically; the geometries as they were first defined will be applied to the newly generated populations. Therefore it is important to check that the change is not so dramatic that the subsequent gates need changing!

### 3.3.4 Deleting a GatingStrategy

If you wish to delete a GatingStrategy, then you can use the `delete` method. Use the `delete_gates` and `remove_associations` arguments to modify behaviour:

- `delete_gates`: this will also delete all gates associated to this gating strategy from the database (defaults to True)
- `remove_associations`: this will iterate through all the FileGroup objects in your database and delete populations generated from this GatingStrategy (defaults to True)

### 3.4 Using control data

An experimental feature of CytoPy is the ability to use control data to influence autonomous gating. We can specify that the Gate (whether that be a threshold, polygon, or ellipse) be ‘fitted’ to the data from a control sample and then the resulting geometric shape defining the gate applied to the data from the primary staining. In this way we can use FMO or isotype controls to influence our gates.

To do this, we simply specify the name of the control sample we would like to use when we define our Gate object. The control must be available in all future FileGroup’s that the GatingStrategy encounters. To see what controls are available in the current FileGroup loaded into our object, we can do the following:

```python
template.filegroup.controls
>>> ['CD57', 'CD27', 'CD45RA', 'CCR7']
```

A definition like this, would result in a gate that uses the CCR7 FMO control to generate the gate and then apply the resulting gate to the original stainings:

```python
gate = ThresholdGate(gate_name="ccr7_ctrl_gate",
    parent="Live CD3+",
    x="CCR7",
    y="CD45RA",
    ctrl="CCR7",
    transformations={"x": "logicle",
                    "y": "logicle"},
    method="density")
```

The parent population that the gate is applied to is estimated in the control data by fitting a K nearest neighbours classifier to the primary data and using the resulting model to estimate the population in the control data, under the assumption that the stating conditions should be similar. The number of neighbours to use in this process is estimated using grid search hyperparameter tuning.
ASSESSMENT OF INTER-SAMPLE VARIATION

In the previous chapter we introduced autonomous gating as a method for classifying single cells in cytometry data. We also mentioned how this method can fail due to the heterogenity of this data, especially in studies that have involved data acquisition that spans months or even years. For such studies we advise that autonomous gating be used for pre-processing purposes only.

For a detailed classification of the single cell data in a FileGroup you can use the supervised classification tools provided by CellClassifier or high-dimensional clustering (e.g. using FlowSOM or PhenoGraph) using Clustering class.

For supervised classification we want to take into account inter-sample variation when choosing training data; this will be a representative FileGroup that is annotated using gates. Unless all of your biological samples were processed at the same time and all cytometry data acquired in the same run, there will be inter-sample variation introduced not just from biological differences between subjects but also as a result of “batch effect”. We recognised that it is important that we can explore and examine inter-sample variation prior to an extensive analysis. This is what the variance module is designed for.

4.1 Loading data for analysis

The variance module provides a function to load data from an Experiment and downsample the data to make it more manageable. This is aptly named load_and_sample and is used as such:

```python
define
    from CytoPy.data.project import Project
    from CytoPy.flow.variance import load_and_sample
    pd_project = Project.objects(project_id='Peritonitis').get()
    exp = pd_project.load_experiment('PD_T_PDMCs')
    data = load_and_sample(experiment=exp,
                           population='T cells',
                           sample_size=5000,
                           sampling_method='uniform',
                           transform='logicle')
```

This function will sample every FileGroup of the Experiment and returns an ordered dictionary, where the key value is the sample ID and the value a Pandas DataFrame of events sampled from the specified population in the respective FileGroup. Sampling methods can be one of ‘uniform’, ‘density’ or ‘faithful’. See CytoPy.flow.sampling for details.
4.2 Choosing a reference sample

CytoPy offers tools to visualise inter-sample variation. Some of these tools contrast the variation in samples with a chosen reference sample. The reference sample is chosen as the sample that aligns closest to all other samples. We choose this reference sample using the `calculate_ref_sample` function.

This function borrows the idea presented in Li H et al (doi: 10.1093/bioinformatics/btx448). In brief, CytoPy performs a pairwise computation of the Euclidean norm of each sample’s covariance matrix, and selects the sample with the smallest average distance as reference.

To generate a reference sample we simply pass the ordered dictionary generated from `load_and_sample`:

```python
from CytoPy.data.project import Project
from CytoPy.flow.variance import calculate_ref_sample
calculate_ref_sample_fast(data)
# This returns the sample ID '325-01', our reference sample
```

4.3 Visualising univariant differences

The first thing we might want to do is visualise the inter-sample variation for individual cell markers measured. We can do this by comparing the estimated PDF (using a Gaussian KDE) of each marker, for each sample, constrained to the reference sample.

This is greatly simplified in CytoPy, which provides convenience functions for tasks such as this. To produce a grid of KDE plots, with samples overlaid on some reference sample we use the `marker_variance` function:

```python
from CytoPy.flow.variance import marker_variance
marker_variance(data=data,
                 reference='325-01',
                 comparison_samples=comparisons,
                 markers=['CD3', 'CD8', 'CD4', 'CD45RA', 'CD27', 'CCR7'])
```

We provide the ordered dictionary of data, the sample ID for the reference sample (must be present in `data`), a list of samples to be compared to the reference sample (comparisons) and a list of markers to plot. This generates a grid of KDE plots, with the reference sample in the background and other samples overlaid:
The number of markers and/or biological samples that can be included in any single plot are not limited, but the user should be cautious not to cloud plots.

### 4.4 Visualising multivariant differences

If we want to see the multivariant shift in biological samples compared to some reference sample, we can observe this using the `dim_reduction_grid` function. As the name suggests, this applies dimensionality reduction, and the use can use any algorithm from CytoPy.flow.dim_reduction.

The reduced projections of the reference sample are shown in each plot of the grid as a background and then for each plot a different biological sample is overlaid:

```python
from CytoPy.flow.variance import dim_reduction_grid
dim_reduction_grid(data=data,
                   reference_id='325-01,
                   comparison_samples=comparisons,
                   features=markers,
                   method='PCA')
```
4.5 Building a similarity matrix

It is useful to visualise inter-sample variation but it doesn’t quantify it nor does it provide us with a strategy to circumvent its effects. We’re going to want to classify cells by phenotype using supervised methods, which is going to require that we label some training data. If there are extreme batch effects, this will influence how well our model generalises given the training data. For example, say we train the data on a reference sample that is significantly different to all other samples, then the trained model is going to perform poorly.

We have therefore developed something we call the “similarity matrix”. This is a matrix of the pairwise statistical distance for all samples in an Experiment. We use this “similarity matrix” to “group” samples (using single linkage clustering) and then we train an individual model for each group of similar samples.

We generate a similarity matrix using the SimilarityMatrix class like so:

```
from CytoPy.flow.variance import SimilarityMatrix
sim_matrix = SimilarityMatrix(data=data,
                              reference='325-01',
                              verbose=True,
                              )
```

(continues on next page)
By default `similarity_matrix` uses Jenson-Shannon distance and this our recommended metric (as discussed in the manuscript).

This method outputs a few things. The first is a “linkage matrix”, generated from the SciPy.cluster.hierarchy function, and is the result of single linkage clustering on the matrix of pairwise statistical distances. Second is an ordered list of sample IDs that correspond to the linkage matrix. Finally is a Seaborn.ClusterMap object. The plot is printed to stdout.

With this matrix produced we can group samples either heuristically (by visual inspection of the clustermap) or by using the linkage matrix and the `generate_groups` function:
We provide this function with the output of the `SimilarityMatrix` object and it generates a Pandas DataFrame of group assignments. It does this by “cutting” the linkage matrix to form the number of clusters specified by “n_groups”. In the above example it would generate 3 clusters or “groups” of similar samples.
CHAPTER
FIVE

SINGLE CELL PHENOTYPE CLASSIFICATION BY SUPERVISED LEARNING

There are many ways in which we can autonomously classify cells by their phenotype. CytoPy encourages the use of multiple methodologies by creating a single data repository in which the results of multiple methods can be stored and then contrasted.

One such method is supervised machine learning. Here, we label some representative data by manual or autonomous gating, train a classifier, and then predict the classification for all remaining samples.

How do we choose representative data as training data? In the last section we discuss how we can generate a “similarity matrix” for an experiment and group samples according to some statistical distance metric. We then either choose or create a sample for each group to act as training data. A classifier is trained for each group.

We can choose a reference sample using the `calculate_ref_sample` function (see previous section) or we can create a reference sample by taking a uniform sample of events from each member of our group:

```python
from CytoPy.data.project import Project
from CytoPy.flow.ref import create_reference_sample
pd_project = Project.objects(project_id='Peritonitis').get()
exp = pd_project.load_experiment('PD_T_PDMCs')
create_ref_sample(exp, 
    root_population='T cells', 
    samples=group_1, 
    new_file_name='group_1_training_data', 
    sample_n=1000, 
    verbose=True)
```

In the function call above, we pass in an instance of `Experiment` (the experiment we are currently working on). We specify a root population that is true for each biological sample and is the population we will sample vents from. We specify a list of sample IDs to sample events from, here they are contained in the variable “group_1” which corresponds to a group derived as detailed in the previous section.

The function `create_ref_sample` doesn’t return anything, instead it saves the new file to the experiment. It can then be retrieved and manipulated like any file in the experiment. We specify the file name with the argument “new_file_name”. Lastly we specify how many events to sample from each biological sample.

A sample is used as training data by interpreting the populations currently associated to it. So if we take “group_1_training_data” and gate 10 populations using the `GatingStrategy` class (see `Autonomous gating`), we can then specify 10 (or less) of those populations to be “labels” in a classification task. This is all handled by the `CellClassifier` class detailed in the next section.
5.1 Introducing the CellClassifier

The CellClassifier class is the base class that all supervised classifiers inherit from in CytoPy. It handles the retrieval of population data from samples, the conversion of this data into “labels” for classification, and saving predictions. The predictions of a classifier are saved as Population’s no different to a Population defined by a gate.

You might ask, well how do classifiers handle the multi-class structure of population tree’s we see in cytometry data. Take the example below:

This is clearly a very complex population tree. If we wanted a classifier to identify the populations “CD3+” and “T cells”, how would we do so when there are clearly overlaps? (A cell might fall inside the CD3+ gate but then not the T cell gate). CellClassifier supports both multi-class and multi-label prediction (however the choice of algorithm to use may be limited for multi-label prediction, see Scikit-Learn documentation for more details: https://scikit-learn.org/stable/modules/multiclass.html).

For multi-class but single label predictions (cells belong to one population and one population only), CellClassifier
assigns a single class to each cell in the training data. For multi-label prediction, \textbf{CellClassifier} generates a dense binary matrix of shape (n_samples, n_classes). When we call subsequent methods for multi-label prediction we can specify the \textit{threshold} a class must exceed for a positive assignment (defaults to 0.5, i.e. >50% probability of positive outcome to be assigned a class).

When the predicting new \textbf{Populations} for some unclassified \textbf{FileGroup} after training, \textbf{Populations} inherit from the ‘root’ population chosen when initiating the \textbf{CellClassifier}.

The target populations for prediction are given at the point of initialising a \textbf{CellClassifier} object. The user can also specify how to transform the data, whether additional scaling should be applied (e.g. min max normalisation or standard scaling) and specify how to handle issues such as class imbalance; class weights can be provided or a sampling procedure applied (see CytoPy.flow.supervised.cell_clasifier).

There are currently two \textbf{CellClassifier} classes (inheriting the behaviour of \textbf{CellClassifier}), these are: * SklearnCellClassifier - to be used with any supervised classification model from the Scikit-Learn ecosystem (including XGBClassifier from the XGBoost library) * KerasCellClassifier - for the construction of deep neural networks using the Keras sequential API

### 5.1.1 Creating a classifier

The \textbf{CellClassifier} object and the classes that inherit from it follow the conventions of Scikit-Learn and provides a familiar API for training and prediction. Creating a classifier always follows with these steps, shown with an SklearnCellClassifier as an example.

1. We create the \textbf{CellClassifier} object:

   ```python
   xgb = SklearnCellClassifier(name="xgb_classifier",
   multi_class=False,
   features=features,
   target_populations=populations,
   klass="XGBClassifier",
   population_prefix="xgb",
   params={"max_depth": 4, "subsample": 0.05})
   ```

   We provide a name for the classifier, for when we save it to the database. We provide a list of features (column names) to be used for classification and we provide the labels (target populations). The \textit{klass} argument is a string value and should correspond to a valid Scikit-Learn class or a library supported by CytoPy that follows the Scikit-Learn template (currently, beyond Scikit-Learn, we only support XGBoost). We then provide the parameters that would be used to initiate the class in the \textit{params} argument.

2. We load in some training data using the \textit{load_training_data} method:

   ```python
   xgb.load_training_data(experiment=exp,
   reference="group_1_training_data",
   root_population="T cells")
   ```

3. (Optional) It is often the case with single cell data that our data suffers from ‘class imbalance’, that is, some populations are significantly larger than others. We can account for class imbalance by providing class weights. We can use the \textit{auto_class_weights} method to automatically calculate some suitable class weights:

   ```python
   xgb.auto_class_weights()
   ```

   \textbf{Note:} Some algorithms inherently do not support class weights. Make sure to research beforehand and see if your chosen algorithm does.

4. Finally, we build our model. This initiates our model and means we’re ready to start training:
5.2 Training

Taking XGBoostClassifier as an example, training a model is simple, we can just call the `fit` method like you would with any Scikit-Learn model. The `CellClassifier` provides some convenience methods as well however:

- `fit_train_test_split`: fits the model to training data but also keeps a fraction as a ‘holdout’ set (size specified by `test_frac` argument). The training and holdout performance is then measured using a list of metrics (specified in the `metrics` parameter). The function returns a dictionary of training and holdout (testing) performance.

- `fit_cv`: you can provide any cross-validator from the `Scikit-learn library<https://scikit-learn.org/stable/modules/cross_validation.html>` or let it default to simple Kfold cross validation. Training and testing performance across multiple folds is then returned as a list of dictionaries.

In addition to this, the `SklearnCellClassifier` class provides a few additional functions:

- `hyperparameter_tuning`: providing a dictionary of parameters or “parameter grid” the optimal parameters will be chosen by either grid search cross-validation or random search. See specific API for details and consult the Scikit-Learn documentation for a complete guide: https://scikit-learn.org/stable/modules/grid_search.html

- `plot_learning_curve`: this method will generate a learning curve using the scikit-learn utility function `sklearn.model_selection.learning_curve`. This can be performed either with the training data or by providing the ID of some other previously gated `FileGroup`.

- `plot_confusion_matrix`: this will generate a new figure of confusion matrices represented by heatmaps. An example of such is shown below.
5.3 Validating

When working with a new data set it is recommended that you validate the performance of your classifier by manually classifying multiple samples and assessing the performance using `validate_classifier`. This method of `CellClassifier` returns a dictionary of classification performance compared to the already existing populations. In the example below, the samples had already been classified by manual gating:

```python
validation_samples = ['254-05',
                      '325-01',
                      '326-01',
                      '332-01',
                      '338-01']

val_performance = pd.DataFrame()
for v in validation_samples:
    result = xgb.validate_classifier(experiment=exp,
                                       validation_id=v,
                                       metrics=['f1_weighted',
                                                'balanced_accuracy_score'])
```

(continues on next page)
'precision_score',
'recall_score'],
root_population='T cells',
return_predictions=False)

results = pd.DataFrame(results, index=[0])
result['sample_id'] = v
val_performance = pd.concat([val_performance, result])

The dataframe “val_performance” looks like this:

<table>
<thead>
<tr>
<th>f1_score</th>
<th>balanced_accuracy</th>
<th>precision</th>
<th>recall</th>
<th>sample_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.985270</td>
<td>0.939354</td>
<td>0.993578</td>
<td>254-05_pdmc_t</td>
</tr>
<tr>
<td>0</td>
<td>0.602571</td>
<td>0.785370</td>
<td>0.826523</td>
<td>325-01_pdmc_t</td>
</tr>
<tr>
<td>0</td>
<td>0.942474</td>
<td>0.809112</td>
<td>0.964192</td>
<td>326-01_pdmc_t</td>
</tr>
<tr>
<td>0</td>
<td>0.812896</td>
<td>0.854504</td>
<td>0.903763</td>
<td>332-01_pdmc_t</td>
</tr>
<tr>
<td>0</td>
<td>0.931885</td>
<td>0.768352</td>
<td>0.969850</td>
<td>332-01_pdmc_t</td>
</tr>
</tbody>
</table>

Note that metrics can be the name of any valid Scikit-Learn metric function, see Scikit-Learn documentation for details: https://scikit-learn.org/stable/modules/model_evaluation.html

The poor performance of the outlier can be investigated further by passing the feature space and labels for the validation sample to `plot_confusion_matrix`:

```python
x, y = xgb.load_validation(experiment=exp, validation_id='325-01', root_population='T cells')
xgb.plot_confusion_matrix(x=x, y=y)
```

This produces the following confusion matrix, showing that the poor performance stems from misclassification of gamma delta T cells and unclassified events:

```
..........Class Mappings..........
{0: array(['None', dtype='<U4'),
  1: array(['CD4-CD8-', dtype='<U8'),
  2: array(['CD4-CD8+', dtype='<U8'),
  3: array(['CD4+CD8+'], dtype='<U8'),
  4: array(['CD4-CD8-'], dtype='<U8'),
  5: array(['mait'], dtype='<U8'),
  6: array(['gd1'], dtype='<U8'))
```
5.4 Predicting populations and troubleshooting with backgating

When we call the `predict` method, we provide the `Experiment` and the name of the sample (FileGroup) we want to predict populations for. The `predict` method will then use the model to predict the populations and return a modified `FileGroup` with the new populations assigned:

```python
classified_filegroup = xgb.predict(experiment=exp,
sample_id='325-01',
root_population='T cells',
return_predictions=False)
```

To save the results of our classifier to the `FileGroup` we would then call the `save` method on `classified_filegroup`.

We may want to investigate further as to how the cells classified as gamma delta T cells by XGBoost compare to those classified manually. Since the `predict` method returns a modified `FileGroup`, we can use this `CreatePlot` class and inspect the populations. A particularly useful method of this class is the `backgate` method. We can use this to directly compare the “pseudo-gate” (predictions) of the XGBoost classifier with the manual gate, by overlaying both on the parent population, the ‘T cells’.

![Confusion Matrix](image)
from CytoPy.flow.plotting import CreatePlot
plotting = CreatePlot(transform_x="logicle", transform_y="logicle")
# We have to provide the parent population dataframe to backgate,
# this can be retrieved from the filegroup like so...
parent = updated_filegroup.load_population_df("T cells", transform=None)
# Notice how we set 'transform' to None. This is because plotting will
# transform the data for us and we don't want to transform it twice!
# We do the same for the populations we want to overlay on the parent
children = {"gdt": updated_filegroup.load_population_df("gdt", transform=None),
             "XGBoost_gdt": updated_filegroup.load_population_df("XGBoost_gdt",
                        transform=None)}
plotting.backgate(parent=parent,
                   children=children,
                   x="PanGD",
                   y="Vd2",
                   method={"gdt": "polygon", "XGBoost_gdt": "scatter"})

The method specifies how to plot the overlaid populations. We have chose to plot the manual gate as a polygon and the XGBoost generated population as a scatter plot. The above gives us the following that displays how the “poor classification” is a result of this biological sample having reduced numbers of gamma delta T cells:
5.5 Keras

CytoPy extends the functionality if CellClassifier to deep neural networks using Keras through the KerasCellClassifier class. This call inherits all the functionality of CellClassifier but differs slightly in the way that the objects are created.

The KerasCellClassifier requires that an optimizer (see https://keras.io/optimizers), loss function (see https://keras.io/losses) and performance metrics (see https://keras.io/metrics) be provided when initialising the object. Additional compile kwargs can be provided with the compile_kwargs argument.

Layers of the neural network are defined with the Layer class. Individual layers should be defined with the name of the Keras class to user (klass argument; see https://keras.io/api/layers/) and the layer parameters in the argument kwargs.
Layers are then appended to a `KerasCellClassifier` layers attribute.

Keras and deep neural networks are a complex topic and we suggest further reading for a new audience. We recommend “Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow” by Aurelien Geron for further reading.

### 5.6 Saving classifiers

Once we have defined a classifier we can save it’s settings to the database for future use using the `save` method.

**Note:** Saving a CellClassifier to the database does not save the model, but saves the options and parameters used to create the model. When reloading the model, the user will have to call `build_model` again.

For the `SklearnCellClassifier`, the underlying Scikit-Learn model can be saved to and reloaded from disk using the `save_model` and `load_model` methods, respectively.

**Warning:** Be aware of continuity issues of saving Scikit-Learn models. Compatibility with new releases of Scikit-Learn and CytoPy are not guaranteed.
SINGLE-CELL PHENOTYPE CLASSIFICATION BY HIGH-DIMENSIONAL CLUSTERING

CytoPy supports the application of a clustering algorithm to any population, separating cells into clusters of similar phenotype in high dimensional space. Currently CytoPy supports PhenoGraph clustering, FlowSOM, HDBSCAN, and any clustering/probabilistic model currently offered through the Scikit-Learn `cluster` and `mixture` modules.

Clustering is implemented using the `Clustering` class and clustering is performed upon a chosen `Population` of each biological sample of an `Experiment`. The resulting clusters, represented by a `Cluster` object, is saved to the `Population` as an embedded document. Multiple instances of a clustering algorithm can be applied to the same population and the design of the MongoDB database makes it easy to contrast and compare populations (produced from both autonomous/manual gates and from supervised classifiers) with the results of clustering algorithms.

6.1 The Clustering class

The `Clustering` class is agnostic by design and will accept any clustering function from the `clustering` module, which we will discuss shortly. The object of the `Clustering` class is to handle all of the infrastructure around this process i.e. loading and saving data to the database.

We start by creating our `Clustering` object and providing it with an `Experiment` to work with:

```python
from CytoPy.data.project import Project
from CytoPy.flow.ref import create_reference_sample
pd_project = Project.objects(project_id='Peritonitis').get()
exp = pd_project.load_experiment('PD_T_PDMCs')
features=["CXCR3", "CD161",
    "CCR7", "Va7-2",
    "CD8", "Vd2", "CD45RA",
    "PanGD", "CD4", "CD27"]
clusterer = Clustering(experiment=exp,
    tag="My first clusterer",
    features=features,
    root_population="T cells",
    transform="logicle")
```

Few things to speak about here. If we don’t provide the argument `sample_ids`, a list of identifiers for `FileGroup`’s, then `Clustering` will assume we want to study all the samples in this `Experiment`. We also provide the features, names of columns, we want to perform clustering with, what population to cluster on, and how to transform the data prior to clustering analysis. We also have to provide a ‘tag’ which acts as a unique identifier for clustering performed by this `Clustering` object.

This function will then load into memory the ‘T cells’ population from each FileGroup in the Experiment. If clusters already existed for the population ‘T cells’ for one or more FileGroups, then the cluster IDs will also be loaded into memory.
The data is stored within the `data` attribute of `clusterer`. This attribute is a Pandas DataFrame where each row is a single cell, there is a column for each of the chosen features, and there are the following additional columns:

- `sample_id`: the identifier for the FileGroup the cell originated from
- `original_index`: the index the cell was at in the original FileGroup
- `cluster_id`: the cluster identifier, will be None if no prior clustering performed under the given tag
- `meta_label`: any meta-clustering identifiers, again will be None if no prior clustering performed

We interact with the `Clustering` object with the following methods:

- `cluster`: clusters the events in `data` using a given function from the `clustering` module (see below). Functions take an argument `global_clustering`, if True, `data` is treated as a single entity and the clustering algorithm applied to all events, otherwise the data is grouped on `sample_id` and clusters are generated for each unique biological sample
- `meta_cluster`: perform meta-clustering using a given function from the `clustering` module (see below). By meta-clustering, we mean ‘clustering the clusters’. Each cluster is summarised by mapping a function over their values and generating a summary vector that describes the cluster (by default Numpy’s median function is used). The summary vectors are clustered to produce meta-clusters that groups similar clusters together, allowing alike clusters to be matched between biological samples.
- `explore`: generates an `Explorer` object for visualisation and exploratory analysis
- `save`: the results of the clustering are saved to the **FileGroup**(s)

### 6.2 Clustering functions

There are two different signatures for clustering functions in the `clustering` module: `<some identifier>_clustering` and `* <some identifier>_metaclustering`, the difference being that the latter is for, well... meta-clustering.

All of the general clustering functions take a dataframe, `data` and a list of features to cluster on `features`. These two required arguments are passed to the function automatically if called from the `Clustering` `cluster` method. As previously discussed, the `global_clustering` argument can be set to True to treat `data` as a single entity. Additional keyword arguments specific to the method used can then be provided as additional keyword arguments (see specific function API for details). In brief, the clustering functions are as follows:

- `sklearn_clustering`: access to any clustering/probablistic algorithm from Scikit-Learn’s cluster and mixture modules, as well as HDBSCAN. The class name to use should be given as a string value to `method` and the initialising parameters given as keyword arguments.
- `phenograph_clustering`: provides access to the population PhenoGraph algorithm ([https://github.com/dpeerlab/PhenoGraph](https://github.com/dpeerlab/PhenoGraph)), see original documentation for additional keyword arguments to control clustering performance

Meta-clustering is provided by the following functions:

- `sklearn_metaclustering`: as with `sklearn_clustering`, this function provides access to any clustering/probablistic algorithm from Scikit-Learns cluster and mixture modules, as well as HDBSCAN, but performs meta-clustering as opposed to general clustering.
- `phenograph_metaclustering`: same as the `phenograph_clustering` function but for meta-clustering
- `consensus_metacluster`: meta-clustering with the consensus clustering algorithm, as first described here: [https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf](https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf). The optimal number of clusters is taken as a consensus amongst multiple rounds of clustering with random starts. The algorithm used for clustering
should be given with `cluster_class` argument and should have the Scikit-Learn signatures for clustering i.e. `fit_predict` method.

The metaclustering functions additionally take arguments `norm_method` and `norm_kwargs` which use the function `CytoPy.flow.transform.scaler` to normalise data, within each biological sample, prior to summarisation and meta-clustering. By default min max scaling is performed.

### 6.3 Clustering a single sample

To start simple, let’s cluster a single FileGroup with PhenoGraph. Performing clustering is very easy, we start by initiating our clustering object and provide a single sample ID for the argument `sample_ids`:

```python
cclusterer = Clustering(experiment=exp,  
tag="My first clusterer",  
features=features,  
root_population="T cells",  
transform="logicle",  
sample_ids=[experiment.list_samples()[0]])
```

We simply call the `cluster` method to perform clustering whilst providing the `phenograph_clustering` function to do the heavy lifting:

```python
cclusterer.cluster(phenograph_clustering)
```

The `data` attribute of our object will be updated and the cluster IDs stored in the `cluster_id` column.

If we want to jump in and explore our clustering results, we can do so using the `Explorer` class...

### 6.4 Introducing exploratory data analysis with Explorer

The `Explorer` class is the ultimate tool of exploratory data analysis in CytoPy. The `Clustering` has a method called `explore` that generates an `Explorer` object. The `Explorer` can be thought of as a wrapper to a Pandas DataFrame that brings immense data wrangling and visualisation power. `Explorer` houses either single cell data from a single biological sample and/or the results of meta-clustering in it’s `data` attribute. It then contains many methods for visualising this data and exploring it interactively, as well as relating this data to patient metadata.

When the `Explorer` object is generated the data is populated with labels of the clustering results. `Population` labels for each single cell, and identifiers that relate each single cell back to the biological subject it originated from. Let’s see an example of `Explorer` in action:

```python
# Generate the Explorer object
explorer = cclusterer.explorer()
```

We can generate a dimensionality reduction plot using any of the methods in `CytoPy.flow.dim_reduction` (Linear PCA, non-linear PCA, UMAP, t-SNE, Isomap, and PHATE). We can specify to plot two components as a static two dimensional scatter plot or three components that will render automatically as a three-dimensional interactive plot:

```python
explorer.scatter_plot(label='cluster_id',  
features=['CXCR3', 'CD161',  
'CCR7', 'Va7-2',  
'CD8', 'Vd2', 'CD45RA',  
'PanGD', 'CD4', 'CD27'],  
 discrete=True,
```
The results of dimensionality reduction are housed within the Pandas DataFrame as additional columns. The Pandas DataFrame can be saved to hard disk using the `save` method of Explorer and then an Explorer object created from loading that DataFrame:

```python
explorer.save('to_use_later.csv')
explorer = Explorer(data='to_use_later.csv')
```

If we want to contrast the results of our clustering analysis with the results of a supervised classifier like XGBoost, we simply change the variable we want to label data points with:

```python
explorer.scatter_plot(label='population_label',
                      features=['CXCR3', 'CD161',
                                'CCR7', 'Va7-2',
                                'CD8', 'Vd2', 'CD45RA',
                                'PanGD', 'CD4', 'CD27'],
                      discrete=True,
                      n_components=2,
                      dim_reduction_method='PHATE',
                      matplotlib_kwargs={'s': 10, 'linewidth':0.2, 'edgecolor':'black '})
```
The performance is greatly increased because dimensionality reduction is only ever performed once and then the results stored and reused whenever the label is changed.

We can observe the phenotype of each cluster by using the `heatmap` method:

```python
explorer.heatmap(heatmap_var='cluster_id',
                 features=[
                         'CXCR3', 'CD161',
                         'CCR7', 'Va7-2',
                         'CD8', 'Vd2', 'CD45RA',
                         'PanGD', 'CD4', 'CD27'],
                 clustermap=True)
```
It is all well and good clustering one biological sample, but we want to see what is happening across the range of subjects studied. To do this we need to contrast and compare the clustering results of each FileGroup, and that means matching similar clusters from different FileGroups together. This is where we apply meta-clustering.

Let’s create a new `Clustering` object, this time with all of our samples present. Then cluster and meta-cluster using PhenoGraph:

```python
clusterer = Clustering(experiment=exp,
    tag="PhenoGraph meta clustering",
    features=features,
    root_population="T cells",
    transform="logicle")
clusterer.cluster(phenograph_clustering)
clusterer.meta_cluster(phenograph_metaclustering,
    summary_method=numpy.median,
    normalise="norm")
```

The results are stored in the `data` attribute, just like before. The cluster IDs being in the `cluster_id` column but now additionally annotated with meta cluster IDs in the `meta_label` column. We can create an `Explorer` object again but this time explore the results of our meta clustering. Let’s produce a heatmap of the phenotype of our meta-clusters. Remember, these clusters now represent the consensus of all our biological samples:

```python
explore = meta.explorer()
features = [f for f in cd.features if f not in ['FSC-A', 'SSC-A']]
explore.heatmap(heatmap_var='meta_cluster_id',
    normalise=False,
    clustermap=True,
    col_cluster=True,
    features=features,
    figsize=(12,8))
```
It would be great if we could provide our clusters with more familiar names. We can do this using the `rename_meta_clusters` method of our `Clustering` object:

```python
clusterer.rename_meta_clusters({'cluster_4': 'MAITs', 'cluster_9': 'T cells' ...})
```

This can be done for each of our meta clusters. We can save the results of meta clusters to our database. Each cluster, for each biological sample, will have a field called “meta_label” that refers to it’s associated meta cluster.

```python
clusterer.save()
```

Let’s use the `Explorer` class to explore the newly labelled meta clusters:

```python
explore = clusterer.explorer()
explore.heatmap(heatmap_var='meta_cluster_id',
               clustermap=True,
               col_cluster=True,
               features=features,
               figsize=(8,8),
               vmin=0,
               vmax=1)
```
The plots of CytoPy use common libraries: * Heatmaps are produced using Seaborn * Scatterplots in Explorer are produced using Scprep * All other plots use Matplotlib

Additional keyword arguments that are common to these libraries can be given and will be passed to the call to Seaborn/Scprep/Matplotlib.

We can visualise meta clusters as a scatter plot where all clusters from all biological samples are shown after dimensionality reduction. The colour of the data point corresponds to it’s meta cluster assignment and the size of the data point the proportion of cellular events relative to the biological sample the cluster originated from. The size of data points can be controlled using the ‘meta_scale_factor’ argument:

```python
cytoexplore.scatter_plot(label='meta_label',
                        features=features,
                        discrete=True,
                        scale_factor=1000,
                        matplotlib_kwargs={'edgecolors': 'black',
                                            'linewidth': 1},
                        figsize=(15,10),
                        dim_reduction_method='UMAP')
```

---

50 Chapter 6. Single-cell phenotype classification by high-dimensional clustering
The crown jewel of CytoPy is its ability to easily and rapidly relate the results of complex cytometry analysis to the underlying clinical or experimental meta data. In the **Explorer** class we can load meta data using the `load_meta` method. We provide any field name in the **Subject** document that a column is amended to the Pandas DataFrame for that variable. Additionally we can load drug data, infection data, and other embedded data where multiple events of a variable exist for one patient (see CytoPy.flow.clustering.main.Explorer).

Below is an example of loading the peritonitis variables, which specifies if a patient has peritonitis or not. We can then colour clusters according to this variable:

```python
explore.load_meta('peritonitis')
explore.scatter_plot(label='peritonitis',
    features=cd.features,
    discrete=True,
    meta=True,
    meta_scale_factor=4000,
    matplotlib_kwargs={'edgecolors': 'black',
                        'linewidth': 1},
    figsize=(12,10),
    dim_reduction_method='UMAP')
```
Chapter 6. Single-cell phenotype classification by high-dimensional clustering
Once the biological samples of an experiment have been classified into phenotypically similar populations and/or clusters, we want to summarise these ‘features’ of the biological samples so that we can observe difference between clinical/experimental groups. CytoPy offers the feature extraction module for summarising the findings of an experiment and performing feature selection.

7.1 Summarising the proportion of cell populations/clusters

There are multiple functions in this module for extracting and summarising the findings of an Experiment. The first is the experiment_statistics function. This takes an Experiment and returns a Pandas DataFrame of population statistics for each sample in the experiment:

```python
from CytoPy.flow.feature_extraction import experiment_statistics
from CytoPy.data.project import Project
pd_project = Project.objects(project_id='Peritonitis').get()
exp = pd_project.load_experiment('PD_N_PDMCs')
exp_stats = experiment_statistics(experiment=exp, include_subject_id=True)
```

The resulting dataframe will have a column for the subject ID, the sample ID (FileGroup ID), the population ID, and then statistics on the number of cells in that population and the proportion of cells compared to both the immediate parent and the root population.

If we want to then label this dataframe with some meta-data associated to our subjects e.g. disease status, we can use the meta_labelling function. We provide it with the dataframe we have just created and the name of some variable stored in our Subject documents, and it creates a new column for this variable in the dataframe:

```python
from CytoPy.flow.feature_extraction import meta_labelling
exp_stats = meta_labelling(experiment=exp, dataframe=exp_stats,
                           meta_label="peritonitis")
```

The dataframe will now have a column named “peritonitis” containing a boolean value as to whether the patient had peritonitis or not.

We can generate a similar dataframe but instead look at the clustering analysis performed on a particular population. This is achieved with the cluster_statistics function:

```python
cluster_stats = cluster_statistics(experiment=exp, population="T cells")
```

This generates a similar dataframe as before but now each row is a cluster and additional columns are included such as population ID, cluster ID, meta label, and clustering tag. The meta label and tag can be specified as arguments to
this function to filter the clusters you want. Also, population is optional, and if it is not provided then all populations from all FileGroups are parsed for existing clusters.

### 7.2 Dimensionality reduction

A rapid method for detecting if there is a ‘global’ difference between two experimental or clinical groups is by using dimensionality reduction and plotting data points coloured according to their group. In the example below we differentiate patients with and without acute peritonitis. The dataframe ‘summary’ contains the proportion of cell populations identified by XGBoost and the proportion if clusters from all our experiments combined. We can use any method from CytoPy.flow.dim_reduction for the dimensionality reduction and a scatter plot is returned with data points coloured according to some label (here it is whether a patient has peritonitis or not):

```python
from CytoPy.flow.feature_extraction import dim_reduction
dim_reduction(summary=summary, label='peritonitis', scale=True, method='PCA')
```
7.3 Feature selection

A simple approach for eliminating redundant variables is ranking them by their variance. The summary dataframe produced by the functions previously discussed can be passed to `sort variance` which will return a sorted dataframe for convenience.

This often isn’t enough, however. If the number of features is large and we want to narrow down which are of most value to predicting some clinical or experimental endpoint, we can use L1 regularisation in a suitable linear model to do so. L1 regularisation, also known as ‘lasso’ regularisation, shrinks the coefficient of less important variables to zero, producing a more sparse model. By varying the regularisation term and observing the coefficients of all our features, we can see which features shrink more rapidly compared to others. This serves as a helpful feature selection technique, giving us the variables important for predicting some clinical or experimental endpoint.

The feature extraction module contains a function for this called `l1_feature_selection`. This function takes the feature space, a dataframe of features where each row is a different biological sample and a ‘label’ column specifies the label to predict. We specify which features to include in our selection and the name of the label column to predict. The model takes a search space as a tuple. This is passed to Numpy.logspace to generate a range of values to use as the different L1 regularisation terms. The first value specifies the starting value and the second the end. The search space is a \( n \) values (where \( n \) is the third value in this argument) between the start and end on a log scale.

Finally we also provide the model to use. This must be a Scikit-Learn linear classifier that takes an L1 regularisation term as an argument ‘C’. If None is given then a linear support vector machine is used as default:

```python
l1_feature_selection(feature_space=summary,
                     features=features,
                     label='peritonitis',
                     scale=True,
                     search_space=(-2, 0, 50),
                     model=None)
```

We recommend exploring the API documentation for the `feature_extraction` module. Feature selection is a large and complex topic which can be approached many ways. Some additional resources worth checking out are:

7.3. Feature selection
• https://academic.oup.com/bioinformatics/article/23/19/2507/185254
The fcs module houses all functionality for the management and manipulation of data pertaining to a single biological specimen. This might include multiple cytometry files (primary staining and controls) all of which are housed within the FileGroup document. FileGroups should be generated and access through the Experiment class.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

- **FileGroup**(*args, **values*)  
  Document representation of a file group; a selection of related fcs files (e.g.

- **PolygonGeom**(*args, **kwargs*)  
  Polygon shape.

- **Population**(*args, **kwargs*)  
  A population of cells identified by either a gate or supervised algorithm.
apply_transform(data[...])

Apply a transformation to the given dataframe. The features_to_transform specified which columns in the dataframe to transform. This can be given as: * a string value of either ‘all’ or ‘fluorochromes’; transform_method defines which transform to apply to columns * a list of columns to transform; transform_method defines which transform to apply to columns * alternatively, a dictionary where the key is the column name and the value is the transform method to apply to this column; transform_method is ignored.

calculate_optimal_neighbours(x, y, scoring, ...)

Calculate the optimal n_neighbours parameter for KNeighborsClassifier using GridSearchCV.

construct_tree(populations)

Given a list of populations, construct a tree of population hierarchy using the population parent information.

create_convex_hull(x_values, y_values)

Given the x and y coordinates of a cloud of data points, generate a convex hull, returning the x and y coordinates of its vertices.

knn(data, labels, features, n_neighbours[...])

Train a nearest neighbours classifier (scikit-learn implementation) and return the balanced accuracy score for both training and validation.

merge_populations(left, right[...])

Merge two Population’s.

population_stats(filegroup)

Given a FileGroup generate a DataFrame detailing the number of events, proportion of parent population, and proportion of total (root population) for each population in the FileGroup.

uniform_downsampling(data, sample_size, **kwargs)

Uniform downsampling.

vprint(verbose)

Utility function for optional printing.

warn(message[, category, stacklevel, source])

Issue a warning, or maybe ignore it or raise an exception.

class CytoPy.data.fcs.FileGroup(*args, **values)

Bases: mongoengine.document.Document

Document representation of a file group; a selection of related fcs files (e.g. a sample and it’s associated controls)

Parameters

- **primary_id**(str, required) – Unique ID to associate to group
- **files**(EmbeddedDocList) – List of File objects
- **flags**(str, optional) – Warnings associated to file group
- **notes**(str, optional) – Additional free text
- **populations**(EmbeddedDocList) – Populations derived from this file group
- **gates**(EmbeddedDocList) – Gate objects that have been applied to this file group
- **collection_datetime**(DateTime, optional) – Date and time of sample collection
- **processing_datetime**(DateTime, optional) – Date and time of sample processing

Miscellaneous:
### Methods:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_ctrl_file(ctrl_id, data, channels, markers)</code></td>
<td>Add a new control file to this FileGroup.</td>
</tr>
<tr>
<td><code>add_population(population)</code></td>
<td>Add a new Population to this FileGroup.</td>
</tr>
<tr>
<td><code>cascade_save(**kwargs)</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>data(source[, sample_size])</code></td>
<td>Load the FileGroup dataframe for the desired source file.</td>
</tr>
<tr>
<td><code>delete([delete_hdf5_file])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>delete_clusters([tag, meta_label, drop_all])</code></td>
<td>Delete clusters.</td>
</tr>
<tr>
<td><code>delete_populations(populations)</code></td>
<td>Delete given populations.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>estimate_ctrl_population(ctrl, population[,...])</code></td>
<td>Estimate a population for a control sample by training a KNearestNeighbors classifier on the population in the primary data and using this model to predict membership in the control data.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_population(population_name)</code></td>
<td>Given the name of a population associated to the FileGroup, returns the Population object, with index and control index ready loaded.</td>
</tr>
<tr>
<td><code>get_population_by_parent(parent)</code></td>
<td>Given the name of some parent population, return a list of Population object whom’s parent matches</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>list_downstream_populations(population)</code></td>
<td>For a given population find all dependencies</td>
</tr>
<tr>
<td><code>list_gated_controls()</code></td>
<td>List ID of controls that have a cached index in each population of the saved population tree (i.e.</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>list_populations()</code></td>
<td>Yields list of population names :returns: :rtype: Generator</td>
</tr>
<tr>
<td><code>load_ctrl_population_df(ctrl, population[, ...])</code></td>
<td>Load the DataFrame for the events pertaining to a single population from a control.</td>
</tr>
<tr>
<td><code>load_population_df(population[, transform[, ...]])</code></td>
<td>Load the DataFrame for the events pertaining to a single population.</td>
</tr>
<tr>
<td><code>merge_populations(left, right[, ...])</code></td>
<td>Merge two populations present in the current population tree.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>modify(query)</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>population_stats(population)</code></td>
<td>param population</td>
</tr>
<tr>
<td><code>print_population_tree([image, path])</code></td>
<td>Print population tree to stdout or save as an image if ‘image’ is True.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls, ...)</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>save(*args, **kwargs)</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>subtract_populations(left, right[...])</code></td>
<td>Subtract the right population from the left population.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[...])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

**Classes:**

- `my_metaclass` alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaClass`

**Attributes:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pk</code></td>
<td>Get the primary key.</td>
</tr>
</tbody>
</table>

**Exceptions:**

- `DoesNotExist` Bases: `mongoengine.errors.DoesNotExist`

  ```python
  with_traceback()
  Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
  ```

- `MultipleObjectsReturned` Bases: `mongoengine.errors.MultipleObjectsReturned`

  ```python
  with_traceback()
  Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
  ```
add_ctrl_file (ctrl_id: str, data: numpy.array, channels: List[str], markers: List[str])
Add a new control file to this FileGroup.

Parameters
• ctrl_id (str) –
• data (Numpy.Array) –
• channels (list) –
• markers (list) –

Returns
Return type None

add_population (population: CytoPy.data.population.Population)
Add a new Population to this FileGroup.

Parameters population (Population) –

Returns
Return type None

cascade_save (**kwargs)
Recursively save any references and generic references on the document.

clean ()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes ()
Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

classmethod create_index (keys, background=False, **kwargs)
Creates the given indexes if required.

Parameters
• keys – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
• background – Allows index creation in the background

data (source: str, sample_size: int = None) → pandas.core.frame.DataFrame
Load the FileGroup dataframe for the desired source file.

Parameters
• source (str) – Name of the file to load from e.g. either “primary” or the name of a control
• sample_size (int or float (optional)) – Sample the DataFrame

Returns
Return type Pandas.DataFrame

delete (delete_hdf5_file: bool = True, *args, **kwargs)
Delete the Document from the database. This will only take effect if the document has been previously saved.
Parameters

- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(...) w: 2, fsync: True)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument

delete_clusters *(tag: str = None, meta_label: str = None, drop_all: bool = False)*

Parameters

- **tag**
- **meta_label**
- **drop_all**

deletePopover *(populations: list) → None*

Delete given populations. Populations downstream from delete population(s) will also be removed.

Parameters **populations** *(list or str)* – Either a list of populations (list of strings) to remove or a single population as a string. If a value of “all” is given, all populations are dropped.

Returns

Return type: None

classmethod drop_collection()  
Drops the entire collection associated with this Document type from the database.

Raises `OperationError` if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: `OperationError` exception raised if no collection available

classmethod ensure_index *(key_or_list, background=False, **kwargs)*

Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

- **background** – Allows index creation in the background

classmethod ensure_indexes()  
Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting auto_create_index to False in the documents meta data

estimate_ctrl_population *(ctrl: str, population: str, verbose: bool = True, scoring: str = 'balanced_accuracy', **kwargs)*

Estimate a population for a control sample by training a KNearestNeighbors classifier on the population in the primary data and using this model to predict membership in the control data. If n_neighbors parameter of Scikit-Learns KNearestNeighbors class is not given, it will be estimated using grid search cross-validation and optimisation of the given scoring parameter. See CytoPy.flow.neighbours for further details.
CytoPy, Release 1.0.0

Results of the population estimation will be saved to the populations ctrl_index property.

Parameters

- **ctrl**(str) – Control to estimate population for
- **population**(str) – Population to estimate
- **verbose**(bool (default=True)) –
- **scoring**(str (default="balanced_accuracy")) –
- **kwargs**(dict) – Additional keyword arguments passed to initiate KNearestNeighbors

Returns

Return type None

classmethod from_json(json_data, created=False)

Converts json data to a Document instance

Parameters

- **json_data**(str) – The json data to load into the Document
- **created**(bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

get_population(population_name: str) -> CytoPy.data.population.Population

Given the name of a population associated to the FileGroup, returns the Population object, with index and control index ready loaded.

Parameters **population_name**(str) – Name of population to retrieve from database

Returns

Return type Population

generate_population_by_parent(parent: str) -> Generator

Given the name of some parent population, return a list of Population object whom’s parent matches

Parameters **parent**(str) – Name of the parent population to search for

Returns List of Populations

Return type Generator

generate_text_score()

Get text score from text query

list_downstream_populations(population: str) -> list

For a given population find all dependencies

Parameters **population**(str) – population name

Returns List of populations dependent on given population
Return type  list or None

list_gated_controls() → Generator
List ID of controls that have a cached index in each population of the saved population tree (i.e. they have been gated)

Returns  List of control IDs for gated controls

Return type  list

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

list_populations() → iter
Yields list of population names :returns: :rtype: Generator

load_ctrl_population_df(ctrl: str, population: str, transform: str = 'logicle', **kwargs)
Load the DataFrame for the events pertaining to a single population from a control. If the control is absent from this FileGroup it will raise an AssertionError. If the population has not been estimated for the given control, it will attempt to estimate the population using KNearestNeighbours classifier. See estimated_ctrl_population for details.

Parameters

• **ctrl** (str) – Name of the control sample to load

• **population** (str) – Name of the desired population

• **transform** (str or dict (optional)) – If given, transformation method applied to the columns of the DataFrame. If the value given is a string, it should be the name of the transform method applied to ALL columns. If it is a dictionary, keys should correspond to column names and values the transform to apply to said column.

• **kwargs** – Additional keyword arguments passed to estimated_ctrl_population

load_population_df(population: str, transform: str = 'logicle', label_downstream_affiliations: bool = False) → pandas.core.frame.DataFrame
Load the DataFrame for the events pertaining to a single population.

Parameters

• **population** (str) – Name of the desired population

• **transform** (str or dict (optional)) – If given, transformation method applied to the columns of the DataFrame. If the value given is a string, it should be the name of the transform method applied to ALL columns. If it is a dictionary, keys should correspond to column names and values the transform to apply to said column.

• **label_downstream_affiliations** (bool (default=False)) – If True, an additional column will be generated named “population_label” containing the end node membership of each event e.g. if you choose CD4+ population and there are subsequent populations belonging to this CD4+ population in a tree like: “CD4+ -> CD4+CD25+ -> CD4+CD25+CD45RA+” then the population label column will contain the name of the lowest possible “leaf” population that an event is assigned too.

Returns

Return type  Pandas.DataFrame

merge_populations(left: CytoPy.data.population.Population, right: CytoPy.data.population.Population, new_population_name: str = None)
Merge two populations present in the current population tree. The merged population will have the combined index of both populations but will not inherit any clusters and will not be associated to any children.
downstream of either the left or right population. The population will be added to the tree as a descendant of the left populations parent. New population will be added to FileGroup.

**Parameters**
- **left** (*Population*)
- **right** (*Population*)
- **new_population_name** (*str (optional)*)

**Returns**
- **Return type** None

**modify**(query=None, **update**)  
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

**Note:** All unsaved changes that have been made to the document are rejected if the method returns True.

**Parameters**
- **query** – the update will be performed only if the document in the database matches the query
- **update** – Django-style update keyword arguments

**my_metaclass**  
alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`  
**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_auto_id_names(new_class)</code></td>
<td>Find a name for the automatic ID field for the given new class.</td>
</tr>
<tr>
<td><code>mro()</code></td>
<td>Return a type’s method resolution order.</td>
</tr>
</tbody>
</table>

**property pk**  
Get the primary key.

**population_stats**(population: *str*)  

**print_population_tree**(image: *bool = False*, path: *str = None*)  
Print population tree to stdout or save as an image if ‘image’ is True.

**Parameters**
- **image** (*bool (default=False]*) – Save tree as a png image
- **path** (*str (optional)*) – File path for image, ignored if ‘image’ is False. Defaults to working directory.

**Returns**
- **Return type** None

**classmethod register_delete_rule**(document_cls, field_name, rule)  
This method registers the delete rules to apply when removing this object.
reload (*fields, **kwargs)
    Reloads all attributes from the database.

    Parameters
    
    • fields – (optional) args list of fields to reload
    • max_depth – (optional) depth of dereferencing to follow

    New in version 0.1.2.
    Changed in version 0.6: Now chainable
    Changed in version 0.9: Can provide specific fields to reload

save (*args, **kwargs)
    Save the Document to the database. If the document already exists, it will be updated, otherwise it will
    be created. Returns the saved object instance.

    Parameters
    
    • force_insert – only try to create a new document, don’t allow updates of existing
        documents.
    • validate – validates the document; set to False to skip.
    • clean – call the document clean method, requires validate to be True.
    • write_concern – Extra keyword arguments are passed down to save() OR
        insert() which will be used as options for the resultant getLastError command.
        For example, save(..., write_concern={w: 2, fsync: True}, ...)
        will wait until at least two servers have recorded the write and will force an fsync on the
        primary server.
    • cascade – Sets the flag for cascading saves. You can set a default by setting “cascade”
        in the document __meta__
    • cascade_kwargs – (optional) kwargs dictionary to be passed throw to cascading saves.
        Implies cascade=True.
    • __refs – A list of processed references used in cascading saves
    • save_condition – only perform save if matching record in db satisfies condition(s)
        (e.g. version number). Raises OperationError if the conditions are not satisfied
    • signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.

    Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are
    cascaded and any DBRef objects that have changes are saved as well.
    Changed in version 0.6: Added cascading saves
    Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control
    then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the
    cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.
    Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition
    is satisfied in the current db record.
    Changed in version 0.10: OperationError exception raised if save_condition fails.
    Changed in version 0.10.1: :class: save_condition failure now raises a SaveConditionError
    Changed in version 0.10.7: Add signal_kwargs argument
**select_related***(max_depth=1)*

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.


Subtract the right population from the left population. The right population must either have the same parent as the left population or be downstream of the left population. The new population will descend from the same parent as the left population. The new population will have a PolygonGeom geom. New population will be added to FileGroup.

**Parameters**

- **left** (Population)
- **right** (Population)
- **new_population_name** (str (optional))

**switch_collection**(collection_name, keep_created=True)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

- **collection_name** (str) – The database alias to use for saving the document
- **keep_created** (bool) – keep self._created value after switching collection, else is reset to True

**See also:**

Use switch_db if you need to read from another database

**switch_db**(db_alias, keep_created=True)

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**Parameters**

- **db_alias** (str) – The database alias to use for saving the document
- **keep_created** (bool) – keep self._created value after switching db, else is reset to True

**See also:**

Use switch_collection if you need to read from another collection
to_dbref()
Returns an instance of DBRef useful in __raw__ queries.

to_json(*args, **kwargs)
Convert this document to JSON.

Parameters use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
Return as SON data ready for use with MongoDB.

update(**kwargs)
Performs an update on the Document A convenience wrapper to update().

Raises OperationError if called on an object that has not yet been saved.

validate(clean=True)
Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

CytoPy.data.fcs.population_stats(filegroup: CytoPy.data.fcs.FileGroup) → pandas.core.frame.DataFrame
Given a FileGroup generate a DataFrame detailing the number of events, proportion of parent population, and proportion of total (root population) for each population in the FileGroup.

Parameters filegroup (FileGroup) –

Returns

Return type Pandas.DataFrame

8.2 CytoPy.data.experiment

The experiment module houses the Experiment class, used to define cytometry based experiments that can consist of one or more biological specimens. An experiment should be defined for each cytometry staining panel used in your analysis and the single cell data (contained in *.fcs files) added to the experiment using the ‘add_new_sample’ method. All functionality for experiments and Panels are housed within this module.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelMap(*args, **kwargs)</td>
<td>Defines channel/marker mapping.</td>
</tr>
<tr>
<td>Counter(**kwds)</td>
<td>Dict subclass for counting hashable items.</td>
</tr>
</tbody>
</table>
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Class/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Experiment(*args, **kwargs)</code></td>
<td>Container for Cytometry experiment.</td>
</tr>
<tr>
<td><code>FCSFile(filepath[, comp_matrix])</code></td>
<td>Utilising FlowIO to generate an object for representing an FCS file</td>
</tr>
<tr>
<td><code>FileGroup(*args, **values)</code></td>
<td>Document representation of a file group; a selection of related fcs files (e.g.)</td>
</tr>
<tr>
<td><code>NormalisedName(*args, **kwargs)</code></td>
<td>Defines a standardised name for a channel or marker and provides method for testing if a channel/marker should be associated to standard</td>
</tr>
<tr>
<td><code>Panel(*args, **values)</code></td>
<td>Document representation of channel/marker definition for an experiment.</td>
</tr>
<tr>
<td><code>Subject(*args, **values)</code></td>
<td>Document based representation of subject meta-data.</td>
</tr>
<tr>
<td><code>datetime(year, month, day[, hour[, minute[, ...]])</code></td>
<td>The year, month and day arguments are required.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>check_excel_template(path)</code></td>
<td>Check excel template and if valid return pandas dataframes</td>
</tr>
<tr>
<td><code>standardise_names(channel_marker,...)</code></td>
<td>Given a dictionary detailing a channel/marker pair ({“channel”: str, “marker”: str}) standardise its contents using the reference material provided.</td>
</tr>
<tr>
<td><code>vprint(verbose)</code></td>
<td>Utility function for optional printing.</td>
</tr>
<tr>
<td><code>warn(message[, category, stacklevel, source])</code></td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

**class CytoPy.data.experiment.Experiment(*args, **kwargs)**

Container for Cytometry experiment. The correct way to generate and load these objects is using the `Project.add_experiment` method (see CytoPy.data.project.Project). This object provides access to all experiment-wide functionality. New files can be added to an experiment using the `add_new_sample` method.

```python
experiment_id
    Unique identifier for experiment
    Type str, required

panel
    Panel object describing associated channel/marker pairs
    Type ReferenceField, required

definition
    Reference field for associated files
    Type ListField

flags
    Warnings associated to experiment
    Type str, optional

notes
    Additional free text comments
    Type str, optional

gating_templates
    Reference to gating templates associated to this experiment
```
**Type** ListField

**Miscellaneous:**

- **DoesNotExist**
- **MultipleObjectsReturned**

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_new_sample</code></td>
<td>Add a new sample (FileGroup) to this experiment</td>
</tr>
<tr>
<td><code>cascade_save</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>Delete Experiment.</td>
</tr>
<tr>
<td><code>delete_all_populations</code></td>
<td>Delete population data associated to experiment.</td>
</tr>
<tr>
<td><code>drop_collection</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>from_json</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>generate_panel</code></td>
<td>Associate a panel to this Experiment, either by fetching an existing panel using the given panel name or by generating a new panel using the panel definition provided (path to a valid template).</td>
</tr>
<tr>
<td><code>get_sample</code></td>
<td>Given a sample ID, return the corresponding FileGroup object</td>
</tr>
<tr>
<td><code>get_sample_mid</code></td>
<td>Given a sample ID (for a sample belonging to this experiment) return it’s mongo ObjectID as a string</td>
</tr>
<tr>
<td><code>get_text_score</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>list_indexes</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>list_invalid</code></td>
<td>Generate list of sample IDs for samples that have the ‘invalid’ flag in their flag attribute</td>
</tr>
<tr>
<td><code>list_samples</code></td>
<td>Generate a list IDs of file groups associated to experiment</td>
</tr>
<tr>
<td><code>modify</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>register_delete_rule</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>remove_sample</code></td>
<td>Remove sample (FileGroup) from experiment.</td>
</tr>
<tr>
<td><code>sample_exists</code></td>
<td>Returns True if the given sample_id exists in Experiment</td>
</tr>
<tr>
<td><code>save</code></td>
<td>Save the Document to the database.</td>
</tr>
</tbody>
</table>

continues on next page
Table 11 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of <code>DBRef</code> objects to a maximum depth in order to cut down the number queries to <code>mongodb</code>.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[, ...])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of <code>DBRef</code> useful in <code>__raw__</code> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update_data_directory(new_path[, move])</code></td>
<td>Update the data directory associated to this experiment.</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

- `my_metaclass` alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

Attributes:

- `pk` Get the primary key.

Exception `DoesNotExist`

Bases: `mongoengine.errors.DoesNotExist`

```python
with_traceback()  
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```

Exception `MultipleObjectsReturned`

Bases: `mongoengine.errors.MultipleObjectsReturned`

```python
with_traceback()  
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```

Method `add_new_sample(sample_id: str, primary_path: str, controls_path: dict = None, subject_id: str = None, comp_matrix: str = None, compensate: bool = True, verbose: bool = True, processing_datetime: str = None, collection_datetime: str = None, missing_error: str = 'raise')`

Add a new sample (FileGroup) to this experiment

Parameters

- `sample_id(str)` – Primary ID for identification of sample (FileGroup.primary_id)
- `subject_id(str, optional)` – ID for patient to associate sample too
• **primary_path**(str or FCSFile) – The path to the primary file or FCSFile object of primary file

• **controls_path**(dict) – Dictionary of control ID and relative path to file for those files to be treated as “control” files e.g. they are ‘FMO’ or ‘isotype’ controls. Alternatively the value can be an FCSFile object containing the respective control file

• **comp_matrix**(str, optional) – Path to csv file for spillover matrix for compensation calculation; if not supplied the matrix linked within the fcs file will be used, if not present will present an error

• **compensate**(bool, (default=True)) – Boolean value as to whether compensation should be applied before data entry (default=True)

• **verbose**(bool, (default=True)) – If True function will provide feedback in the form of print statements (default=True)

• **missing_error**(str (default="raise")) – How to handle missing channels (that is, channels that appear in the associated staining panel but cannot be found in the FCS data. Either “raise” to raise AssertionError or “warn” to flag a UserWarning but continue execution.

• **processing_datetime**(str, optional) –

• **collection_datetime**(str, optional) –

Returns

Return type None

cascade_save(**kwargs)
Recursively save any references and generic references on the document.

clean()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()
Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

classmethod create_index(keys, background=False, **kwargs)
Creates the given indexes if required.

Parameters

• **keys** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

• **background** – Allows index creation in the background

delete(*args, **kwargs)
Delete Experiment.

Parameters

• **args**(list) –

• **kwargs**(dict) –

Returns

Return type None
**delete_all_populations** *(sample_id: str) → None*

Delete population data associated to experiment. Give a value of ‘all’ for sample_id to remove all population data for every sample.

- **Parameters**
  - sample_id *(str)* – Name of sample to remove populations from; give a value of ‘all’ for sample_id to remove all population data for every sample.

- **Returns**
  - None

**classmethod drop_collection()**

Drops the entire collection associated with this Document type from the database.

- Raises *OperationError* if the document has no collection set (i.e. if it is *abstract*).
- Changed in version 0.10.7: *OperationError* exception raised if no collection available

**classmethod ensure_index(key_or_list, background=False, **kwargs)**

Ensure that the given indexes are in place. Deprecated in favour of *create_index*.

- **Parameters**
  - key_or_list – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
  - background – Allows index creation in the background

**classmethod ensure_indexes()**

Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

---

**Note:** You can disable automatic index creation by setting *auto_create_index* to False in the documents meta data

**classmethod from_json(json_data, created=False)**

Converts json data to a Document instance

- **Parameters**
  - json_data *(str)* – The json data to load into the Document
  - created *(bool)* – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**generate_panel(panel_definition: str, panel_name: str)**

Associate a panel to this Experiment, either by fetching an existing panel using the given panel name or by generating a new panel using the panel definition provided (path to a valid template).

- **Parameters**
  - panel_name *(str or None)* – Name of an existing panel
- `panel_definition (str or None)` – Path to a panel definition

Returns

Return type `Panel`

`get_sample(sample_id: str) → CytoPy.data.fcs.FileGroup`

Given a sample ID, return the corresponding FileGroup object

Parameters `sample_id (str)` – Sample ID for search

Returns

Return type `FileGroup`

`get_sample_mid(sample_id: str) → str`

Given a sample ID (for a sample belonging to this experiment) return it’s mongo ObjectID as a string

Parameters `sample_id (str)` – Sample ID for sample of interest

Returns `string value for ObjectID`

Return type `str or None`

`get_text_score()`

Get text score from text query

`classmethod list_indexes()`

Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

`list_invalid() → Generator`

Generate list of sample IDs for samples that have the ‘invalid’ flag in their flag attribute

Returns `List of sample IDs for invalid samples`

Return type `Generator`

`list_samples(valid_only: bool = True) → Generator`

Generate a list IDs of file groups associated to experiment

Parameters `valid_only (bool)` – If True, returns only valid samples (samples without ‘invalid’ flag)

Returns `List of IDs of file groups associated to experiment`

Return type `Generator`

`modify(query=None, **update)`

Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

- `query` – the update will be performed only if the document in the database matches the query
- `update` – Django-style update keyword arguments
my_metaclass
   alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Methods:

get_auto_id_names(new_class)  Find a name for the automatic ID field for the
given new class.
mro()  Return a type’s method resolution order.

property pk
   Get the primary key.

classmethod register_delete_rule(document_cls, field_name, rule)
   This method registers the delete rules to apply when removing this object.

reload(*fields, **kwargs)
   Reloads all attributes from the database.

   Parameters
   • fields – (optional) args list of fields to reload
   • max_depth – (optional) depth of dereferencing to follow

   New in version 0.1.2.
   Changed in version 0.6: Now chainable
   Changed in version 0.9: Can provide specific fields to reload

remove_sample(sample_id: str)
   Remove sample (FileGroup) from experiment.

   Parameters sample_id(str) – ID of sample to remove

   Returns
   Return type None

sample_exists(sample_id: str) → bool
   Returns True if the given sample_id exists in Experiment

   Parameters sample_id(str) – Name of sample to search for

   Returns True if exists, else False

   Return type bool

save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)
   Save the Document to the database. If the document already exists, it will be updated, otherwise it will
   be created. Returns the saved object instance.

   Parameters
   • force_insert – only try to create a new document, don’t allow updates of existing
documents.
   • validate – validates the document; set to False to skip.
   • clean – call the document clean method, requires validate to be True.
   • write_concern – Extra keyword arguments are passed down to save() OR
     insert() which will be used as options for the resultant getLastError command.
For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.

- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document `__meta__`
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies `cascade=True`.
- **_refs** – A list of processed references used in cascading saves
- **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises `OperationError` if the conditions are not satisfied
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any `DBRef` objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using `cascade_kwargs` which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: `OperationError` exception raised if save_condition fails.

Changed in version 0.10.1: `:class:` save_condition failure now raises a `SaveConditionError`

Changed in version 0.10.7: Add signal_kwargs argument

**select_related** *(max_depth=1)*

Handles dereferencing of `DBRef` objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

**switch_collection** *(collection_name, keep_created=True)*

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

- **collection_name** *(str)* – The database alias to use for saving the document
- **keep_created** *(bool)* – keep self._created value after switching collection, else is reset to True

**See also:**

Use `switch_db` if you need to read from another database

**switch_db** *(db_alias, keep_created=True)*

Temporarily switch the database for a document instance.
Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

- `db_alias` *(str)* – The database alias to use for saving the document
- `keep_created` *(bool)* – keep self._created value after switching db, else is reset to True

See also:

Use `switch_collection` if you need to read from another collection

```python
to_dbref()
```

Returns an instance of `DBRef` useful in `__raw__` queries.

```python
to_json(*args, **kwargs)
```

Convert this document to JSON.

  Parameters `use_db_field` – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

```python
to_mongo(*args, **kwargs)
```

Return as SON data ready for use with MongoDB.

```python
update(**kwargs)
```


Raises `OperationError` if called on an object that has not yet been saved.

```python
update_data_directory(new_path: str, move: bool = True)
```

Update the data directory associated to this experiment. This will propagate to all associated FileGroup’s. WARNING: this function will move the existing data directory and all of it’s contents to the new given path.

  Parameters

  - `new_path` *(str)* –
  - `move` *(bool (default=True))* – If True, the data is assumed to be present at the old path and will be moved over to the new path by CytoPy

  Returns

  Return type `None`

```python
validate(clean=True)
```

Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.

```python
class CytoPy.data.experiment.NormalisedName(*args, **kwargs)
```

Bases: `mongoengine.document.EmbeddedDocument`

Defines a standardised name for a channel or marker and provides method for testing if a channel/marker should be associated to standard

```python
standard
```

the “standard” name i.e. the nomenclature we used for a channel/marker in this panel
CytoPy, Release 1.0.0

Type str, required

regex_str
    regular expression used to test if a term corresponds to this standard
    Type str

permutations
    String values that have direct association to this standard (comma separated values)
    Type str

case_sensitive
    is the nomenclature case sensitive? This would be false for something like ‘CD3’ for example, where ‘cd3’ and ‘CD3’ are synonymous
    Type bool, (default=False)

Methods:

clean()          Hook for doing document level data cleaning before validation is run.

from_json(json_data[, created])          Converts json data to a Document instance

get_text_score()          Get text score from text query

query(x)          Given a term `x`, determine if `x` is synonymous to this standard.

to_json(*args, **kwargs)          Convert this document to JSON.

to_mongo(*args, **kwargs)          Return as SON data ready for use with MongoDB.

validate([clean])          Ensure that all fields’ values are valid and that required fields are present.

Classes:

my_metaclass          alias of mongoengine.base.metaclasses.DocumentMetaclass

---
clean()
    Hook for doing document level data cleaning before validation is run.

    Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod from_json (json_data, created=False)
    Converts json data to a Document instance

Parameters

- **json_data** (str) – The json data to load into the Document
- **created** (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
– Defaults to False.

get_text_score()
Get text score from text query

my_metaclass
alias of mongoengine.base.metaclasses.DocumentMetaclass
Methods:

mro()
Return a type’s method resolution order.

query (x: str) → str
Given a term ‘x’, determine if ‘x’ is synonymous to this standard. If so, return the standardised name.

Parameters x (str) – search term

Returns Standardised name if synonymous to standard, else None

Return type str or None

to_json (*args, **kwargs)
Convert this document to JSON.

Parameters use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo (*args, **kwargs)
Return as SON data ready for use with MongoDB.

validate (clean=True)
Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

class CytoPy.data.experiment.Panel (*args, **values)
Bases: mongoengine.document.Document
Document representation of channel/marker definition for an experiment. A panel, once associated to an experiment will standardise data upon input; when an fcs file is created in the database, it will be associated to an experiment and the channel/marker definitions in the fcs file will be mapped to the associated panel.

panel_name
unique identifier for the panel

Type str, required

markers
list of marker names; see NormalisedName

Type EmbeddedDocListField

channels
list of channels; see NormalisedName

Type EmbeddedDocListField

mappings
list of channel/marker mappings; see ChannelMap

Type EmbeddedDocListField

initiation_date
date of creation files['controls']

Type DateTime
### Miscellaneous:

**DoesNotExist**  
**MultipleObjectsReturned**

### Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cascade_save(**kwargs)</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_from_dict(x)</code></td>
<td>Populate panel attributes from a python dictionary</td>
</tr>
<tr>
<td><code>create_from_excel(path)</code></td>
<td>Populate panel attributes from an excel template</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete([signal_kwargs])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensures that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_channels()</code></td>
<td>Yields list of channels associated to panel</td>
</tr>
<tr>
<td><code>get_markers()</code></td>
<td>Yields list of channels associated to panel</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls, . . . )</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean, . . . ])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[, . . . ])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

### Classes:

80 Chapter 8. API Reference
my_metaclass

**my_metaclass**

alias of `mongoengine.base.metaclasses.TopLevelDocumentMeta
class`

**Attributes:**

**pk**

Get the primary key.

class **Exception**

Bases: `mongoengine.errors.DoesNotExist`

**args**

with_traceback() – set self.__traceback__ to tb and return self.

class **MultipleObjectsReturned**

Bases: `mongoengine.errors.MultipleObjectsReturned`

**args**

with_traceback() – set self.__traceback__ to tb and return self.

class **cascade_save**(**kwargs**)

Recursively save any references and generic references on the document.

**clean**()

Hook for doing document level data cleaning before validation is run.

Any `ValidationError` raised by this method will not be associated with a particular field; it will have a
special-case association with the field defined by `NON_FIELD_ERRORS`.

**classmethod compare_indexes**()

Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

create_from_dict**(x: dict)**

Populate panel attributes from a python dictionary

Parameters:

- **x (dict)** – dictionary object containing panel definition

Returns

Return type: `None`

create_from_excel**(path: str)** → `None`

Populate panel attributes from an excel template

Parameters:

- **path (str)** – path of file

Returns

Return type: `None`

**classmethod create_index**(keys, background=False, **kwargs**)

Creates the given indexes if required.

Parameters:

- **keys** – a single index key or a list of index keys (to construct a multi-field index); keys may
  be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background
**delete** *(signal_kwargs=None, **write_concern)*

Delete the Document from the database. This will only take effect if the document has been previously saved.

**Parameters**

- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.
- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(..., w: 2, fsync: True)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument

**classmethod drop_collection()**

Drops the entire collection associated with this Document type from the database.

Raises `OperationError` if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: `OperationError` exception raised if no collection available

**classmethod ensure_index(key_or_list, background=False, **kwargs)**

Ensure that the given indexes are in place. Deprecated in favour of `create_index`.

**Parameters**

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background

**classmethod ensure_indexes()**

Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

**Note:** You can disable automatic index creation by setting `auto_create_index` to False in the documents meta data

**classmethod from_json(json_data, created=False)**

Converts json data to a Document instance

**Parameters**

- **json_data** *(str)* – The json data to load into the Document
- **created** *(bool)* – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_channels()** → iter

Yields list of channels associated to panel
CytoPy, Release 1.0.0

Returns

Return type Generator

get_markers() → iter
Yields list of channels associated to panel

Returns

Return type Generator

get_text_score()
Get text score from text query

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

modify(query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

• query – the update will be performed only if the document in the database matches the query

• update – Django-style update keyword arguments

my_metaclass
alias of mongoengine.base.metasclasses.TopLevelDocumentMetacllass Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_auto_id_names(new_class)</td>
<td>Find a name for the automatic ID field for the given new class.</td>
</tr>
<tr>
<td>mro()</td>
<td>Return a type’s method resolution order.</td>
</tr>
</tbody>
</table>

property pk
Get the primary key.

classmethod register_delete_rule(document_cls, field_name, rule)
This method registers the delete rules to apply when removing this object.

reload(*fields, **kwargs)
Reloads all attributes from the database.

Parameters

• fields – (optional) args list of fields to reload

• max_depth – (optional) depth of dereferencing to follow

New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

8.2. CytoPy.data.experiment 83
CytoPy, Release 1.0.0

**save** (force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)

Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters

- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to False to skip.
- **clean** – call the document clean method, requires validate to be True.
- **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern={w: 2, fsync: True}, ...) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.
- **_refs** – A list of processed references used in cascading saves
- **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: OperationError exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a SaveConditionError

Changed in version 0.10.7: Add signal_kwargs argument

**select_related** (max_depth=1)

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

**switch_collection** (collection_name, keep_created=True)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```
Parameters

• **collection_name** *(str)* – The database alias to use for saving the document

• **keep_created** *(bool)* – keep self._created value after switching collection, else is reset to True

See also:
Use switch_db if you need to read from another database

```python
switch_db(db_alias, keep_created=True)
```

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

• **db_alias** *(str)* – The database alias to use for saving the document

• **keep_created** *(bool)* – keep self._created value after switching db, else is reset to True

See also:
Use switch_collection if you need to read from another collection

to_dbref()

Returns an instance of DBRef useful in __raw__ queries.

to_json(*args, **kwargs)

Convert this document to JSON.

Parameters **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)

Return as SON data ready for use with MongoDB.

update(**kwargs)


Raises `OperationError` if called on an object that has not yet been saved.

validate(clean=True)

Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.

```python
CytoPy.data.experiment.check_excel_template(path: str) -> (class 'pandas.core.frame.DataFrame', class 'pandas.core.frame.DataFrame')
```

Check excel template and if valid return pandas dataframes

Parameters **path** *(str)* – file path for excel template

Returns tuple of pandas dataframes (nomenclature, mappings) or None

Return type (Pandas.DataFrame, Pandas.DataFrame) or None
CytoPy.data.experiment.standardise_names(channel_marker: dict, ref_channels: List[CytoPy.data.experiment.NormalisedName], ref_markers: List[CytoPy.data.experiment.NormalisedName], ref_mappings: List[CytoPy.data.mapping.ChannelMap])

Given a dictionary detailing a channel/marker pair ("channel": str, “marker”: str) standardise its contents using the reference material provided.

Parameters

- channel_marker (dict) –
- ref_channels (list) –
- ref_markers (list) –
- ref_mappings (list) –

Returns

Return type  dict

### 8.3 CytoPy.data.gate

Gates are traditionally used to subset single cell data in one or two dimensional space by hand-drawn polygons in a manual and laborious process. CytoPy attempts to emulate this using autonomous gates, driven by unsupervised learning algorithms. The gate module contains the classes that provide the infrastructure to apply these algorithms to the context of single cell data whilst interacting with the underlying database that houses our analysis.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AffinityPropagation(**kwargs)</td>
<td>Perform Affinity Propagation Clustering of data.</td>
</tr>
<tr>
<td>AgglomerativeClustering(**kwargs)</td>
<td>Agglomerative Clustering</td>
</tr>
<tr>
<td>BayesianGaussianMixture(**kwargs)</td>
<td>Variational Bayesian estimation of a Gaussian mixture.</td>
</tr>
<tr>
<td>Birch(**kwargs)</td>
<td>Implements the Birch clustering algorithm.</td>
</tr>
<tr>
<td>Child(*args, **kwargs)</td>
<td>Base class for a gate child population.</td>
</tr>
<tr>
<td>ChildPolygon(*args, **kwargs)</td>
<td>Child population of a Polygon or Ellipse gate.</td>
</tr>
<tr>
<td>ChildThreshold(*args, **kwargs)</td>
<td>Child population of a Threshold gate.</td>
</tr>
<tr>
<td>Counter(**kwds)</td>
<td>Dict subclass for counting hashable items.</td>
</tr>
<tr>
<td>DBSCAN(**kwargs)</td>
<td>Perform DBSCAN clustering from vector array or distance matrix.</td>
</tr>
<tr>
<td>EllipseGate(*args, **values)</td>
<td>EllipseGate inherits from PolygonGate.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th><strong>Class</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FFTKDE([kernel, bw, norm])</td>
<td>This class implements a convolution (FFT) based computation of a KDE.</td>
</tr>
<tr>
<td>FeatureAgglomeration(<strong>kwargs)</strong></td>
<td>Agglomerate features.</td>
</tr>
<tr>
<td>Gate(**args, <strong>values</strong>)</td>
<td>Base class for a Gate.</td>
</tr>
<tr>
<td>GaussianMixture(<strong>kwargs</strong>)</td>
<td>Gaussian Mixture.</td>
</tr>
<tr>
<td>HDBSCAN([min_cluster_size, min_samples, ...])</td>
<td>Perform HDBSCAN clustering from vector array or distance matrix.</td>
</tr>
<tr>
<td>KMeans(<strong>kwargs</strong>)</td>
<td>K-Means clustering.</td>
</tr>
<tr>
<td>MeanShift(<strong>kwargs</strong>)</td>
<td>Mean shift clustering using a flat kernel.</td>
</tr>
<tr>
<td>MiniBatchKMeans(<strong>kwargs</strong>)</td>
<td>Mini-Batch K-Means clustering.</td>
</tr>
<tr>
<td>OPTICS(<strong>kwargs</strong>)</td>
<td>Estimate clustering structure from vector array.</td>
</tr>
<tr>
<td>PolygonGate(**args, <strong>values</strong>)</td>
<td>PolygonGate inherits from Gate.</td>
</tr>
<tr>
<td>PolygonGeom(**args, <strong>kwargs</strong>)</td>
<td>Polygon shape.</td>
</tr>
<tr>
<td>Population(**args, <strong>kwargs</strong>)</td>
<td>A population of cells identified by either a gate or supervised algorithm.</td>
</tr>
<tr>
<td>ShapelyPoly</td>
<td>alias of shapely.geometry.polygon. Polygon</td>
</tr>
<tr>
<td>SpectralBiclustering(<strong>kwargs</strong>)</td>
<td>Spectral biclustering (Kluger, 2003).</td>
</tr>
<tr>
<td>SpectralClustering(<strong>kwargs</strong>)</td>
<td>Apply clustering to a projection of the normalized Laplacian.</td>
</tr>
<tr>
<td>SpectralCoclustering(<strong>kwargs</strong>)</td>
<td>Spectral Co-Clustering algorithm (Dhillon, 2001).</td>
</tr>
<tr>
<td>ThresholdGate(**args, <strong>values</strong>)</td>
<td>ThresholdGate inherits from Gate.</td>
</tr>
<tr>
<td>ThresholdGeom(**args, <strong>kwargs</strong>)</td>
<td>Threshold shape.</td>
</tr>
</tbody>
</table>

**Functions:**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>affinity_propagation(S, *[, preference, ...])</td>
<td>Perform Affinity Propagation Clustering of data</td>
</tr>
<tr>
<td>apply_threshold(data, x, x_threshold[, y, ...])</td>
<td>Simple wrapper for threshold_1d and threshold_2d</td>
</tr>
<tr>
<td>apply_transform(data[, ...])</td>
<td>Apply a transformation to the given dataframe. The features_to_transform specified which columns in the dataframe to transform. This can be given as: * a string value of either ‘all’ or ‘fluorochromes’; transform_method defines which transform to apply to columns * a list of columns to transform; transform_method defines which transform to apply to columns * alternatively, a dictionary where the key is the column name and the value is the transform method to apply to this column; transform_method is ignored.</td>
</tr>
<tr>
<td>cascaded_union(geoms)</td>
<td>Returns the union of a sequence of geometries</td>
</tr>
<tr>
<td>cluster_optics_dbscan(*, reachability,...)</td>
<td>Performs DBSCAN extraction for an arbitrary epsilon.</td>
</tr>
<tr>
<td>cluster_optics_xi(*, reachability,...[, ...])</td>
<td>Automatically extract clusters according to the Xi-steep method.</td>
</tr>
<tr>
<td>compute_optics_graph(X,*, min_samples,...)</td>
<td>Computes the OPTICS reachability graph.</td>
</tr>
<tr>
<td>create_convex_hull(x_values, y_values)</td>
<td>Given the x and y coordinates of a cloud of data points, generate a convex hull, returning the x and y coordinates of its vertices.</td>
</tr>
<tr>
<td>create_polygon(x, y)</td>
<td>Given a list of x coordinates and a list of y coordinates, generate a shapely Polygon</td>
</tr>
<tr>
<td>create_signature(data[, idx, summary_method])</td>
<td>Given a dataframe of FCS events, generate a signature of those events; that is, a summary of the dataframes columns using the given summary method.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbscan(X[, eps, min_samples, metric, ...])</code></td>
<td>Perform DBSCAN clustering from vector array or distance matrix.</td>
</tr>
<tr>
<td><code>density_dependent_downsampling(data[, ...])</code></td>
<td>Perform density dependent down-sampling to remove risk of under-sampling rare populations; adapted from SPADE*</td>
</tr>
<tr>
<td><code>detect_peaks(x[, mph, mpd, threshold, edge, ...])</code></td>
<td>Detect peaks in data based on their amplitude and other features.</td>
</tr>
<tr>
<td><code>dimensionality_reduction(data, features, ...)</code></td>
<td>Perform dimensionality reduction using either UMAP, PCA, tSNE, or PHATE.</td>
</tr>
<tr>
<td><code>ellipse_to_polygon(centroid, width, height, ...)</code></td>
<td>Convert an ellipse to a shapely Polygon object.</td>
</tr>
<tr>
<td><code>estimate_bandwidth(X, *[quantile, ...])</code></td>
<td>Estimate the bandwidth to use with the mean-shift algorithm.</td>
</tr>
<tr>
<td><code>euclidean(u, v[, w])</code></td>
<td>Computes the Euclidean distance between two 1-D arrays.</td>
</tr>
<tr>
<td><code>faithful_downsampling(data, h)</code></td>
<td>An implementation of faithful downsampling as described in: Zare H, Shooestani P, Gupta A, Brinkman R.</td>
</tr>
<tr>
<td><code>find_inflection_point(x, p, peak_idx[, ...])</code></td>
<td>Given some probability vector and grid space that represents a PDF as calculated by KDE, and assuming this vector has a single peak of highest density, calculate the inflection point at which the peak flattens.</td>
</tr>
<tr>
<td><code>find_local_minima(p, x, peaks)</code></td>
<td>Find local minima between the two highest peaks in the density distribution provided</td>
</tr>
<tr>
<td><code>find_peaks(p, min_peak_threshold, peak_boundary)</code>, <code>get_bin_seeds(X, bin_size[, min_bin_freq])</code>, <code>inside_polygon(df, x, y, poly[, njobs])</code></td>
<td>Find peaks using the detecta package (see detecta.detect_peaks for details). Find seeds for mean_shift. Return rows in dataframe who’s values for x and y are contained in some polygon coordinate shape</td>
</tr>
<tr>
<td><code>k_means(X, n_clusters, *[sample_weight, ...])</code></td>
<td>K-means clustering algorithm.</td>
</tr>
<tr>
<td><code>linkage_tree(X[, connectivity, n_clusters, ...])</code></td>
<td>Linkage agglomerative clustering based on a Feature matrix.</td>
</tr>
<tr>
<td><code>mean_shift(X, *[bandwidth, seeds, ...])</code></td>
<td>Perform mean shift clustering of data using a flat kernel.</td>
</tr>
<tr>
<td><code>merge_children(children)</code></td>
<td>Given a list of Child objects, merge and return single child</td>
</tr>
<tr>
<td><code>merge_multiple_populations(populations[, ...])</code></td>
<td>Merge multiple Population’s.</td>
</tr>
<tr>
<td><code>polygon_overlap(poly1, poly2[, threshold])</code></td>
<td>Compare the area of two polygons and give the fraction overlap.</td>
</tr>
<tr>
<td><code>probabilistic_ellipse(covariances, conf)</code></td>
<td>Given the covariance matrix of a mixture component, calculate an elliptical shape that represents a probabilistic confidence interval.</td>
</tr>
<tr>
<td><code>savgol_filter(x, window_length, polyorder[, ...])</code></td>
<td>Apply a Savitzky-Golay filter to an array.</td>
</tr>
<tr>
<td><code>smoothed_peak_finding(p[, ...])</code></td>
<td>Given the grid space and probability vector of some PDF calculated using KDE, first attempt to smooth the probability vector using a Savitzky-Golay filter (see scipy.signal.savgol_filter) and then perform peak finding until the number of peaks is less than 3.</td>
</tr>
<tr>
<td><code>spectral_clustering(affinity, *[...])</code></td>
<td>Apply clustering to a projection of the normalized Laplacian.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>threshold_1d</code></td>
<td>Apply the given threshold (x_threshold) to the x-axis variable (x) and return the resulting dataframes corresponding to the positive and negative populations.</td>
</tr>
<tr>
<td><code>threshold_2d</code></td>
<td>Apply the given threshold (x_threshold) to the x-axis variable (x) and the given threshold (y_threshold) to the y-axis variable (y), and return the resulting dataframes as a dictionary: ‘++’: Greater than or equal to threshold for both x and y ‘+-’: Greater than or equal to threshold for x but less than threshold for y ‘-+’: Greater than or equal to threshold for y but less than threshold for x ‘–’: Less than threshold for both x and y</td>
</tr>
<tr>
<td><code>update_polygon</code></td>
<td>Given an existing population and some new definition for it’s polygon gate (different to what is already associated to the Population), update the Population index and geom accordingly.</td>
</tr>
<tr>
<td><code>update_threshold</code></td>
<td>Given an existing population and some new threshold(s) (different to what is already associated to the Population), update the Population index and geom accordingly.</td>
</tr>
<tr>
<td><code>upsample_knn</code></td>
<td>Given some sampled dataframe and the original dataframe from which it was derived, use the given labels (which should correspond to the sampled dataframe row index) to fit a nearest neighbours model to the sampled data and predict the assignment of labels in the original data.</td>
</tr>
<tr>
<td><code>ward_tree</code></td>
<td>Ward clustering based on a Feature matrix.</td>
</tr>
<tr>
<td><code>warn</code></td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

```python
class CytoPy.data.gate.Child(*args, **kwargs):
    Bases: mongoengine.document.EmbeddedDocument

    Base class for a gate child population. This is representative of the ‘population’ of cells identified when a gate is first defined and will be used as a template to annotate the populations identified in new data.

    Methods:
    
    clean()  
    from_json(json_data[, created])  
    get_text_score()  
    to_json(*args, **kwargs)  
    to_mongo(*args, **kwargs)  
    validate([clean])
```

Classes:

```python
my_metaclass
```

```python
clean()
```
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod from_json (json_data, created=False)

Converts json data to a Document instance

Parameters

• json_data (str) – The json data to load into the Document

• created (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data
  it’s loaded with (i.e. even if an ID is loaded).
  – If False and an ID is NOT provided, consider the document as brand new.
  – If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  – Defaults to False.

get_text_score ()

Get text score from text query

my_metaclass

alias of mongoengine.base.metaclasses.DocumentMetaClass

Methods:

mro ()

Return a type’s method resolution order.

to_json (*args, **kwargs)

Convert this document to JSON.

Parameters use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo (*args, **kwargs)

Return as SON data ready for use with MongoDB.

validate (clean=True)

Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

class CytoPy.data.gate.ChildPolygon (*args, **kwargs)

Bases: CytoPy.data.gate.Child

Child population of a Polgon or Ellipse gate. This is representative of the ‘population’ of cells identified when a gate is first defined and will be used as a template to annotate the populations identified in new data.

name

Name of the child

Type str

geom

Geometric definition for this child population

Type ChildPolygon
signature
Average of a population feature space (median of each channel); used to match children to newly identified populations for annotating

Type  dict

Methods:

clean()  Hook for doing document level data cleaning before validation is run.

from_json(json_data[, created])  Converts json data to a Document instance

get_text_score()  Get text score from text query

to_json(*args, **kwargs)  Convert this document to JSON.

to_mongo(*args, **kwargs)  Return as SON data ready for use with MongoDB.

validate(clean)  Ensure that all fields’ values are valid and that required fields are present.

Classes:

my_metaclass  alias of mongoengine.base.metaclasses.DocumentMetaclass

clean()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod from_json (json_data, created=False)
Converts json data to a Document instance

Parameters

- json_data (str) – The json data to load into the Document
- created (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

get_text_score()
Get text score from text query

my_metaclass  alias of mongoengine.base.metaclasses.DocumentMetaclass

Methods:

mro()  Return a type’s method resolution order.

to_json(*args, **kwargs)
Convert this document to JSON.
Parameters *use_db_field* – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

**to_mongo** (*args, **kwargs*)
Return as SON data ready for use with MongoDB.

**validate** (*clean=True*)
Ensure that all fields’ values are valid and that required fields are present.
Raises `ValidationError` if any of the fields’ values are found to be invalid.

```python
class CytoPy.data.gate.ChildThreshold (*args, **kwargs*)
Bases: CytoPy.data.gate.Child
Child population of a Threshold gate. This is representative of the ‘population’ of cells identified when a gate is first defined and will be used as a template to annotate the populations identified in new data.

**name**
Name of the child
*Type* `str`

**definition**
Definition of population e.g “+” or “-” for 1 dimensional gate or “++” etc for 2 dimensional gate
*Type* `str`

**geom**
Geometric definition for this child population
*Type* `ThresholdGeom`

**signature**
Average of a population feature space (median of each channel); used to match children to newly identified populations for annotating
*Type* `dict`

**Methods:**

- `clean()` Hook for doing document level data cleaning before validation is run.
- `from_json(json_data[, created])` Converts json data to a Document instance
- `get_text_score()` Get text score from text query
- `match_definition(definition)` Given a definition, return True or False as to whether it matches this ChildThreshold’s definition.
- `to_json(*args, **kwargs)` Convert this document to JSON.
- `to_mongo(*args, **kwargs)` Return as SON data ready for use with MongoDB.
- `validate([clean])` Ensure that all fields’ values are valid and that required fields are present.

**Classes:**

- `my_metaclass` alias of mongoengine.base.metaclasses.DocumentMetaClass

- `clean()` Hook for doing document level data cleaning before validation is run.
Any `ValidationError` raised by this method will not be associated with a particular field; it will have a
special-case association with the field defined by NON_FIELD_ERRORS.

**classmethod from_json** *(json_data, created=False)*  
Converts json data to a Document instance

**Parameters**

- **json_data** *(str)* – The json data to load into the Document
- **created** *(bool)* – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_text_score** ()  
Get text score from text query

**match_definition** *(definition: str)*  
Given a definition, return True or False as to whether it matches this ChildThreshold’s definition. If definition contains multiples separated by a comma, or the ChildThreshold’s definition contains multiple, first split and then compare. Return True if matches any.

**Parameters**

- **definition** *(str)* –

**Returns**

**Return type** bool

**my_metaclass**  
alias of mongoengine.base.metaclasses.DocumentMetaclsass

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>mro()</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
</tbody>
</table>
| to_json**(*args, **kwargs)** | Convert this document to JSON.  

**Parameters**

- **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

| to_mongo**(*args, **kwargs)** | Return as SON data ready for use with MongoDB. |
|------------------------------------------|
| validate**(clean=True)** | Ensure that all fields’ values are valid and that required fields are present.  

**Raises** ValidationError if any of the fields’ values are found to be invalid.

**class** *CytoPy.data.gate.EllipseGate**(*args, **values)*  
**Bases:** *CytoPy.data.gate.PolygonGate*

EllipseGate inherits from PolygonGate. A Gate attempts to separate single cell data in one or two-dimensional space using unsupervised learning algorithms. The algorithm is fitted to example data to generate “children”; the populations of cells a user expects to identify. These children are stored and then when the gate is ‘fitted’ to new data, the resulting populations are matched to the expected children.
The EllipseGate uses probabilistic mixture models to subset data into “populations”. For each component of the mixture model the covariance matrix is used to generate a confidence ellipse, surrounding data and emulating a gate. EllipseGate can use any of the methods from the Scikit-Learn mixture module. Keyword arguments for the initiation of a class from this module can be given in “method_kwargs”.

**gate_name**
Name of the gate
- **Type**: str (required)

**parent**
Parent population that this gate is applied to
- **Type**: str (required)

**x**
Name of the x-axis variable forming the one/two dimensional space this gate is applied to
- **Type**: str (required)

**y**
Name of the y-axis variable forming the two dimensional space this gate is applied to
- **Type**: str (required)

**transformations**
Transform method to be applied to each dimension, should be a dictionary with keys corresponding to each variable (e.g. “x” and/or “y”) and values the transform to apply (e.g. {“x”: “logicle”} for logicle transform of x-axis)
- **Type**: dict (optional)

**sampling**
Options for downsampling data prior to application of gate. Should contain a key/value pair for desired method e.g ({“method”: “uniform”}). Available methods are: ‘uniform’, ‘density’ or ‘faithful’. See CytoPy.flow.sampling for details. Additional keyword arguments should be provided in the sampling dictionary.
- **Type**: dict (optional)

**dim_reduction**
Experimental feature. Allows for dimension reduction to be performed prior to applying gate. Gate will be applied to the resulting embeddings. Provide a dictionary with a key “method” and the value as any supported method in CytoPy.flow.dim_reduction. Additional keyword arguments should be provided in this dictionary.
- **Type**: dict (optional)

**method**
Name of the underlying algorithm to use. Should have a value of: “manual”, or correspond to the name of an existing class in Scikit-Learn mixture module. If you have a method that follows the Scikit-Learn template but isn’t currently present in CytoPy and you would like it to be, please contribute to the repository on GitHub or contact burtonrj@cardiff.ac.uk
- **Type**: str (required)

**method_kwargs**
Keyword arguments for initiation of the above method.
- **Type**: dict

Miscellaneous:
Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_child(child)</code></td>
<td>Add a new child for this gate.</td>
</tr>
<tr>
<td><code>cascade_save(**kwargs)</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete([signal_kwargs])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>fit(data)</code></td>
<td>Fit the gate using a given dataframe.</td>
</tr>
<tr>
<td><code>fit_predict(data)</code></td>
<td>Fit the gate using a given dataframe and then associate predicted Population objects to existing children.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>label_children(labels[, drop])</code></td>
<td>Rename children using a dictionary of labels where the key correspond to the existing child name and the value is the new desired population name.</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>predict(data)</code></td>
<td>Using existing children associated to this gate, the previously calculated polygons of these children will be applied to the given data and then Population objects created and labelled to match the children of this gate.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls,...)</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>reset_gate()</code></td>
<td>Removes existing children and resets all parameters.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean,...])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongod.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[, ...])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
</tbody>
</table>
Table 35 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of <code>DBRef</code> useful in <code>_raw_</code> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

- `my_metaclass` alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

Attributes:

- `pk` Get the primary key.

exception `DoesNotExist`
- `args`
- `with_traceback()`
  - Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `MultipleObjectsReturned`
- Bases: `CytoPy.data.gate.MultipleObjectsReturned`, `CytoPy.data.gate.MultipleObjectsReturned`
- `args`
- `with_traceback()`
  - Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`add_child(child: CytoPy.data.gate.ChildPolygon) → None`
- Add a new child for this gate. Checks that child is valid and overwrites geom with gate information.
  - Parameters `child (ChildPolygon)`
  - Returns
  - `Return type` None

`cascade_save(**kwargs)`
- Recursively save any references and generic references on the document.

`clean()`
- Hook for doing document level data cleaning before validation is run.
  - Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`classmethod compare_indexes()`
- Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.
classmethod create_index(keys, background=False, **kwargs)
    Creates the given indexes if required.

    Parameters
    • keys – a single index key or a list of index keys (to construct a multi-field index); keys
      may be prefixed with a + or a - to determine the index ordering
    • background – Allows index creation in the background

delete(signal_kwargs=None, **write_concern)
    Delete the Document from the database. This will only take effect if the document has been previously
    saved.

    Parameters
    • signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.
    • write_concern – Extra keyword arguments are passed down which will be used as
      options for the resultant getLastError command. For example, save(..., w: 2, fsync: True) will wait until
      at least two servers have recorded the write and will force an fsync on the primary server.

    Changed in version 0.10.7: Add signal_kwargs argument

classmethod drop_collection()
    Drops the entire collection associated with this Document type from the database.
    Raises OperationError if the document has no collection set (i.e. if it is abstract)

    Changed in version 0.10.7: OperationError exception raised if no collection available

classmethod ensure_index(key_or_list, background=False, **kwargs)
    Ensure that the given indexes are in place. Deprecated in favour of create_index.

    Parameters
    • key_or_list – a single index key or a list of index keys (to construct a multi-field
      index); keys may be prefixed with a + or a - to determine the index ordering
    • background – Allows index creation in the background

classmethod ensure_indexes()
    Checks the document meta data and ensures all the indexes exist.
    Global defaults can be set in the meta - see guide/defining-documents

    Note: You can disable automatic index creation by setting auto_create_index to False in the documents
    meta data

fit(data: pandas.core.frame.DataFrame) → None
    Fit the gate using a given dataframe. This will generate new children using the calculated polygons. If
    children already exist will raise an AssertionError and notify user to call fit_predict.

    Parameters data (Pandas.DataFrame) – Population data to fit gate to

    Returns

    Return type None

fit_predict(data: pandas.core.frame.DataFrame) → List[CytoPy.data.population.Population]
    Fit the gate using a given dataframe and then associate predicted Population objects to existing children.
    If no children exist, an AssertionError will be raised prompting the user to call ‘fit’ method.
Parameters **data** (*Pandas.DataFrame*) – Population data to fit gate to

Returns List of predicted Population objects, labelled according to the gates child objects

Return type List

classmethod **from_json** (*json_data, created=False*)

Converts json data to a Document instance

Parameters

- **json_data** (*str*) – The json data to load into the Document
- **created** (*bool*) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_text_score**()

Get text score from text query

**label_children** (*labels: dict, drop: bool = True*) → None

Rename children using a dictionary of labels where the key correspond to the existing child name and the value is the new desired population name. If the same population name is given to multiple children, these children will be merged. If drop is True, then children that are absent from the given dictionary will be dropped.

Parameters

- **labels** (*dict*) – Mapping for new children name
- **drop** (*bool (default=True)* – If True, children absent from labels will be dropped

Returns

Return type None

classmethod **list_indexes**()

Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

**modify** (*query=None, **update*)

Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

**Note**: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

- **query** – the update will be performed only if the document in the database matches the query
• **update** – Django-style update keyword arguments

```python
my_metaclass
    alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass
```

Methods:

- `get_auto_id_names(new_class)`
  Find a name for the automatic ID field for the given new class.
- `mro()`
  Return a type's method resolution order.

```python
property pk
    Get the primary key.
```

```python
predict(data: pandas.core.frame.DataFrame) → List[CytoPy.data.population.Population]
```

Using existing children associated to this gate, the previously calculated polygons of these children will be applied to the given data and then Population objects created and labelled to match the children of this gate. NOTE: the data will not be fitted and polygons applied will be STATIC not data driven. For data driven gates call `fit_predict` method.

- **Parameters**
  - `data` *(Pandas.DataFrame)* – Data to apply static polygons to
- **Returns**
  - List of Population objects

```python
classmethod register_delete_rule(document_cls, field_name, rule)
```

This method registers the delete rules to apply when removing this object.

```python
reload(*fields, **kwargs)
```

Reloads all attributes from the database.

- **Parameters**
  - `fields` *(optional)* args list of fields to reload
  - `max_depth` *(optional)* depth of dereferencing to follow

  New in version 0.1.2.

  Changed in version 0.6: Now chainable

  Changed in version 0.9: Can provide specific fields to reload

```python
reset_gate() → None
```

Removes existing children and resets all parameters.

- **Returns**
  - None

```python
save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)
```

Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

- **Parameters**
  - `force_insert` – only try to create a new document, don’t allow updates of existing documents.
  - `validate` – validates the document; set to `False` to skip.
  - `clean` – call the document clean method, requires `validate` to be True.
• **write_concern** – Extra keyword arguments are passed down to `save()` OR `insert()` which will be used as options for the resultant `getLastError` command. For example, `save(..., write_concern={"w": 2, "fsync": True}, ...)` will wait until at least two servers have recorded the write and will force an `fsync` on the primary server.

• **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document `__meta__`

• **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies `cascade=True`.

• **_refs** – A list of processed references used in cascading saves

• **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises `OperationError` if the conditions are not satisfied

• **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any `DBRef` objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta[‘cascade’] = True. Also you can pass different kwargs to the cascade save using `cascade_kwargs` which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional `save_condition` that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: `OperationError` exception raised if `save_condition` fails.

Changed in version 0.10.1: :class: `save_condition` failure now raises a `SaveConditionError`

Changed in version 0.10.7: Add `signal_kwargs` argument

**select_related** (`max_depth=1`)

Handles dereferencing of `DBRef` objects to a maximum depth in order to cut down the number queries to `mongodb`.

New in version 0.5.

**switch_collection** (`collection_name`, `keep_created=True`)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

• **collection_name** (`str`) – The database alias to use for saving the document

• **keep_created** (`bool`) – keep self._created value after switching collection, else is reset to True

**See also:**

Use `switch_db` if you need to read from another database
switch_db(db_alias, keep_created=True)

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

- **db_alias** (str) – The database alias to use for saving the document
- **keep_created** (bool) – keep self._created value after switching db, else is reset to True

See also:

Use switch_collection if you need to read from another collection

to_dbref()

Returns an instance of DBRef useful in __raw__ queries.

to_json(*args, **kwargs)

Convert this document to JSON.

Parameters **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)

Return as SON data ready for use with MongoDB.

update(**kwargs)

Performs an update on the Document A convenience wrapper to update().

Raises OperationError if called on an object that has not yet been saved.

validate(clean=True)

Ensure that all fields’ values are valid and that required fields are present.

Raises Validation Error if any of the fields’ values are found to be invalid.

class CytoPy.data.gate.Gate(*args, **values)

Bases: mongoengine.document.Document

Base class for a Gate. A Gate attempts to separate single cell data in one or two-dimensional space using unsupervised learning algorithms. The algorithm is fitted to example data to generate “children”; the populations of cells a user expects to identify. These children are stored and then when the gate is ‘fitted’ to new data, the resulting populations are matched to the expected children.

gate_name

Name of the gate

    Type  str (required)

parent

Parent population that this gate is applied to

    Type  str (required)

x

Name of the x-axis variable forming the one/two dimensional space this gate is applied to

    Type  str (required)
y
Name of the y-axis variable forming the two dimensional space this gate is applied to

Type str (optional)

transformations
Transform method to be applied to each dimension, should be a dictionary with keys corresponding to
each variable (e.g. “x” and/or “y”) and values the transform to apply (e.g. {“x”: “logicle”} for logicle
transform of x-axis)

Type dict (optional)
sampling
Options for downsampling data prior to application of gate. Should contain a key/value pair for desired
method e.g ({“method”: “uniform”}). Available methods are: ‘uniform’, ‘density’ or ‘faithful’. See Cy-
toPy.flow.sampling for details. Additional keyword arguments should be provided in the sampling dictio-
nary.

Type dict (optional)
dim_reduction
Experimental feature. Allows for dimension reduction to be performed prior to applying gate. Gate will
be applied to the resulting embeddings. Provide a dictionary with a key “method” and the value as any
supported method in CytoPy.flow.dim_reduction. Additional keyword arguments should be provided in
this dictionary.

Type dict (optional)
ctrl
If a value is given here it should be the name of a control specimen commonly associated to the samples in
an Experiment. When given this signals that the gate is designed to be applied to the control data not the
primary data.

Type str (optional)

method
Name of the underlying algorithm to use. Should have a value of: “manual”, “density”, “quantile” or
correspond to the name of an existing class in Scikit-Learn or HDBSCAN. If you have a method that
follows the Scikit-Learn template but isn’t currently present in CytoPy and you would like it to be, please
contribute to the repository on GitHub or contact burtonrj@cardiff.ac.uk

Type str (required)

method_kwargs
Keyword arguments for initiation of the above method.

Type dict

Miscellaneous:

DoNotExist
MultipleObjectsReturned

Methods:

cascade_save(**kwargs)
Recursively save any references and generic refer-
ences on the document.

clean()
Hook for doing document level data cleaning before validation is run.

continues on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete([signal_kwargs])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance.</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query.</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls, ...)</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>reset_gate()</code></td>
<td>Removes existing children and resets all parameters.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean, ...])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[, ...])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <em>raw</em> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_metaclass</td>
<td>alias of mongoengine.base.metaclasses. TopLevelDocumentMetaclass</td>
</tr>
</tbody>
</table>

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pk</td>
<td>Get the primary key.</td>
</tr>
</tbody>
</table>

**Exception:**

`DoesNotExist`

- **Bases:** `mongoengine.errors.DoesNotExist`
- **args**: 
- **with_traceback()**: 

8.3. CytoPy.data.gate
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned
    Bases: mongoengine.errors.MultipleObjectsReturned
    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

cascade_save(**kwargs)
    Recursively save any references and generic references on the document.

clean()
    Hook for doing document level data cleaning before validation is run.
    Any ValidationError raised by this method will not be associated with a particular field; it will have a
    special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()
    Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any miss-
    ing extra indexes.

classmethod create_index(keys, background=False, **kwargs)
    Creates the given indexes if required.
    Parameters
    • keys – a single index key or a list of index keys (to construct a multi-field index); keys
        may be prefixed with a + or a - to determine the index ordering
    • background – Allows index creation in the background

delete(signal_kwargs=None, **write_concern)
    Delete the Document from the database. This will only take effect if the document has been previously
    saved.
    Parameters
    • signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.
    • write_concern – Extra keyword arguments are passed down which will be used as
        options for the resultant getLastError command. For example, save(..., w: 2, fsync: True) will wait until at least two servers have recorded the write and
        will force an fsync on the primary server.
        Changed in version 0.10.7: Add signal_kwars argument

classmethod drop_collection()
    Drops the entire collection associated with this Document type from the database.
    Raises OperationError if the document has no collection set (i.e. if it is abstract)
    Changed in version 0.10.7: OperationError exception raised if no collection available

classmethod ensure_index(key_or_list, background=False, **kwargs)
    Ensure that the given indexes are in place. Deprecated in favour of create_index.
    Parameters
    • key_or_list – a single index key or a list of index keys (to construct a multi-field
        index); keys may be prefixed with a + or a - to determine the index ordering
    • background – Allows index creation in the background

classmethod ensure_indexes()
Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting auto_create_index to False in the documents meta data

classmethod from_json (json_data, created=False)
Converts json data to a Document instance

Parameters

- **json_data** (str) – The json data to load into the Document
- **created** (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data
  it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this
    has an impact on the subsequent call to .save()).
  - Defaults to False.

get_text_score()
Get text score from text query

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super-
and sub-classes.

modify (query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated
version.

Returns True if the document has been updated or False if the document in the database doesn’t match the
query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

- **query** – the update will be performed only if the document in the database matches the
  query
- **update** – Django-style update keyword arguments

my_metaclass
alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Methods:

get_auto_id_names(new_class)
Find a name for the automatic ID field for the given new class.
property pk
Get the primary key.

classmethod register_delete_rule(document_cls, field_name, rule)
This method registers the delete rules to apply when removing this object.

reload(*fields, **kwargs)
Reloads all attributes from the database.

Parameters
- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.
Changed in version 0.6: Now chainable
Changed in version 0.9: Can provide specific fields to reload

reset_gate() → None
Removes existing children and resets all parameters.

Returns
- **Return type**: None

save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)
Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters
- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to False to skip.
- **clean** – call the document clean method, requires validate to be True.
- **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern=(w: 2, fsync: True), ... ) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.
- **_refs** – A list of processed references used in cascading saves
- **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.
Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: OperationError exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a SaveConditionError

Changed in version 0.10.7: Add signal_kwargs argument

select_related(max_depth=1)
Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

switch_collection(collection_name, keep_created=True)
Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

Parameters

- **collection_name** (str) – The database alias to use for saving the document
- **keep_created** (bool) – keep self._created value after switching collection, else is reset to True

See also:

Use switch_db if you need to read from another database

switch_db(db_alias, keep_created=True)
Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

- **db_alias** (str) – The database alias to use for saving the document
- **keep_created** (bool) – keep self._created value after switching db, else is reset to True

See also:

Use switch_collection if you need to read from another collection
to_dbref()
Returns an instance of DBRef useful in __raw__ queries.

to_json(*args, **kwargs)
Convert this document to JSON.

Parameters

use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
Return as SON data ready for use with MongoDB.

update(**kwargs)
Performs an update on the Document. A convenience wrapper to update().

Raises OperationError if called on an object that has not yet been saved.

validate(clean=True)
Ensure that all fields’ values are valid and that required fields are present.

Raises Validation Error if any of the fields’ values are found to be invalid.

class CytoPy.data.gate.PolygonGate(*args, **values)
Bases: CytoPy.data.gate.Gate

PolygonGate inherits from Gate. A Gate attempts to separate single cell data in one or two-dimensional space using unsupervised learning algorithms. The algorithm is fitted to example data to generate “children”; the populations of cells a user expects to identify. These children are stored and then when the gate is ‘fitted’ to new data, the resulting populations are matched to the expected children.

The PolygonGate subsets data based on the results of an unsupervised learning algorithm such a clustering algorithm. PolygonGate supports any clustering algorithm from the Scikit-Learn machine learning library. Support is extended to any clustering library that follows the Scikit-Learn template, but currently this only includes HDBSCAN. Contributions to extend to other libraries are welcome. The name of the class to use should be provided in “method” along with keyword arguments for initiating this class in “method_kwargs”.

Alternatively the “method” can be “manual” for a static gate to be applied; user should provide x_values and y_values (if two-dimensional) to “method_kwargs” as two arrays, this will be interpreted as the x and y coordinates of the polygon to fit to the data.

gate_name
Name of the gate

    Type str (required)

parent
Parent population that this gate is applied to

    Type str (required)

x
Name of the x-axis variable forming the one/two dimensional space this gate is applied to

    Type str (required)

y
Name of the y-axis variable forming the two dimensional space this gate is applied to

    Type str (required)

transformations
Transform method to be applied to each dimension, should be a dictionary with keys corresponding to each variable (e.g. “x” and/or “y”) and values the transform to apply (e.g. {“x”: “logicle”} for logicle transform of x-axis)
**Type** dict (optional)

**sampling**
Options for downsampling data prior to application of gate. Should contain a key/value pair for desired method e.g ("method": "uniform"). Available methods are: ‘uniform’, ‘density’ or ‘faithful’. See CytoPy.flow.sampling for details. Additional keyword arguments should be provided in the sampling dictionary.

**Type** dict (optional)

**dim_reduction**
Experimental feature. Allows for dimension reduction to be performed prior to applying gate. Gate will be applied to the resulting embeddings. Provide a dictionary with a key “method” and the value as any supported method in CytoPy.flow.dim_reduction. Additional keyword arguments should be provided in this dictionary.

**Type** dict (optional)

**ctrl**
If a value is given here it should be the name of a control specimen commonly associated to the samples in an Experiment. When given this signals that the gate is designed to be applied to the control data not the primary data.

**Type** str (optional)

**method**
Name of the underlying algorithm to use. Should have a value of: “manual”, or correspond to the name of an existing class in Scikit-Learn or HDBSCAN. If you have a method that follows the Scikit-Learn template but isn’t currently present in CytoPy and you would like it to be, please contribute to the repository on GitHub or contact burtonrj@cardiff.ac.uk

**Type** str (required)

**method_kwargs**
Keyword arguments for initiation of the above method.

**Type** dict

---

**Miscellaneous:**

- **DoesNotExist**
- **MultipleObjectsReturned**

---

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_child(child)</code></td>
<td>Add a new child for this gate.</td>
</tr>
<tr>
<td><code>cascade_save(**kwargs)</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete([signal_kwargs])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
</tbody>
</table>

---

8.3. CytoPy.data.gate 109
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>fit(data)</code></td>
<td>Fit the gate using a given dataframe.</td>
</tr>
<tr>
<td><code>fit_predict(data)</code></td>
<td>Fit the gate using a given dataframe and then associate predicted Population objects to existing children.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>label_children(labels[, drop])</code></td>
<td>Rename children using a dictionary of labels where the key correspond to the existing child name and the value is the new desired population name.</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>predict(data)</code></td>
<td>Using existing children associated to this gate, the previously calculated polygons of these children will be applied to the given data and then Population objects created and labelled to match the children of this gate.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls, . . .)</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>reset_gate()</code></td>
<td>Removes existing children and resets all parameters.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean, . . .])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[, . . .])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

my_metaclass | alias of mongoengine.base.metaclasses. TopLevelDocumentMetaclass |

Attributes:

pk | Get the primary key.
exception DoesNotExist
    Bases: CytoPy.data.gate.DoesNotExist
    args

    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned
    Bases: CytoPy.data.gate.MultipleObjectsReturned
    args

    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

add_child(child: CytoPy.data.gate.ChildPolygon) → None
    Add a new child for this gate. Checks that child is valid and overwrites geom with gate information.

    Parameters
    child (ChildPolygon) –

    Returns

    Return type None

cascade_save(**kwargs)
    Recursively save any references and generic references on the document.

clean()
    Hook for doing document level data cleaning before validation is run.

    Any ValidationError raised by this method will not be associated with a particular field; it will have a
special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()
    Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any miss-
ing/extra indexes.

classmethod create_index(keys, background=False, **kwargs)
    Creates the given indexes if required.

    Parameters
    • keys – a single index key or a list of index keys (to construct a multi-field index); keys
      may be prefixed with a + or a - to determine the index ordering
    • background – Allows index creation in the background

delete(signal_kwargs=None, **write_concern)
    Delete the Document from the database. This will only take effect if the document has been previously
    saved.

    Parameters
    • signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.
    • write_concern – Extra keyword arguments are passed down which will be used as
      options for the resultant getLastError command. For example, save(..., w: 2, fsync: True) will wait until at least two servers have recorded the write and
      will force an fsync on the primary server.

        Changed in version 0.10.7: Add signal_kwargs argument

classmethod drop_collection()
    Drops the entire collection associated with this Document type from the database.
Raises `OperationError` if the document has no collection set (i.e. if it is `abstract`)

Changed in version 0.10.7: `OperationError` exception raised if no collection available

classmethod `ensure_index`(key_or_list, background=False, **kwargs)
Ensure that the given indexes are in place. Deprecated in favour of `create_index`.

Parameters

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background

classmethod `ensure_indexes`()
Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting `auto_create_index` to False in the documents meta data

```python
fit(data: pandas.core.frame.DataFrame) -> None
Fit the gate using a given dataframe. This will generate new children using the calculated polygons. If children already exist will raise an AssertionError and notify user to call `fit_predict`.

Parameters data (Pandas.DataFrame) – Population data to fit gate to

Returns

Return type None
```

```python
fit_predict(data: pandas.core.frame.DataFrame) -> List[CytoPy.data.population.Population]
Fit the gate using a given dataframe and then associate predicted Population objects to existing children. If no children exist, an AssertionError will be raised prompting the user to call ‘fit’ method.

Parameters data (Pandas.DataFrame) – Population data to fit gate to

Returns List of predicted Population objects, labelled according to the gates child objects

Return type List
```

classmethod `from_json`(json_data, created=False)
Converts json data to a Document instance

Parameters

- **json_data** (str) – The json data to load into the Document
- **created** (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  
  – If False and an ID is NOT provided, consider the document as brand new.
  
  – If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to `.save()`).
  
  – Defaults to False.
get_text_score()
Get text score from text query

label_children (labels: dict, drop: bool = True) → None
rename children using a dictionary of labels where the key correspond to the existing child name and the
value is the new desired population name. If the same population name is given to multiple children, these
children will be merged. If drop is True, then children that are absent from the given dictionary will be
dropped.

Parameters

- labels (dict) – Mapping for new children name
- drop (bool (default=True)) – If True, children absent from labels will be dropped

Returns

Return type None

classmethod list_indexes ()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super-
and sub-classes.

modify (query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated
version.

Returns True if the document has been updated or False if the document in the database doesn’t match the
query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

- query – the update will be performed only if the document in the database matches the
  query
- update – Django-style update keyword arguments

my_metaclass
alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Methods:

get_auto_id_names(new_class)
Find a name for the automatic ID field for the
given new class.

mro()
Return a type’s method resolution order.

property pk
Get the primary key.

predict (data: pandas.core.frame.DataFrame) → List[CytoPy.data.population.Population]
Using existing children associated to this gate, the previously calculated polygons of these children will
be applied to the given data and then Population objects created and labelled to match the children of this
gate. NOTE: the data will not be fitted and polygons applied will be STATIC not data driven. For data
driven gates call fit_predict method.

Parameters data (Pandas.DataFrame) – Data to apply static polygons to

Returns List of Population objects
classmethod register_delete_rule(document_cls, field_name, rule)
This method registers the delete rules to apply when removing this object.

reload(*fields, **kwargs)
Reloads all attributes from the database.

Parameters

• **fields** – (optional) args list of fields to reload
• **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.
Changed in version 0.6: Now chainable
Changed in version 0.9: Can provide specific fields to reload

reset_gate() → None
Removes existing children and resets all parameters.

Returns

Return type None

save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwags=None, _refs=None, save_condition=None, signal_kwags=None, **kwargs)
Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters

• **force_insert** – only try to create a new document, don’t allow updates of existing documents.
• **validate** – validates the document; set to False to skip.
• **clean** – call the document clean method, requires validate to be True.
• **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern={w: 2, fsync: True}, ... ) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
• **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
• **cascade_kwags** – (optional) kwags dictionary to be passed throw to cascading saves. Implies cascade=True.
• **_refs** – A list of processed references used in cascading saves
• **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied
• **signal_kwags** – (optional) kwags dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves
Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: `OperationError` exception raised if save_condition fails.

Changed in version 0.10.1: `class` save_condition failure now raises a `SaveConditionError`

Changed in version 0.10.7: Add signal_kwargs argument

```py
select_related(max_depth=1)
```
Handles dereferencing of `DBRef` objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

```py
switch_collection(collection_name, keep_created=True)
```
Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling `save()`:

```py
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

- `collection_name` *(str)* – The database alias to use for saving the document
- `keep_created` *(bool)* – keep self._created value after switching collection, else is reset to True

**See also:**

Use `switch_db` if you need to read from another database

```py
switch_db(db_alias, keep_created=True)
```
Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling `save()`:

```py
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**Parameters**

- `db_alias` *(str)* – The database alias to use for saving the document
- `keep_created` *(bool)* – keep self._created value after switching db, else is reset to True

**See also:**

Use `switch_collection` if you need to read from another collection

```py
to_dbref()
```
Returns an instance of `DBRef` useful in `__raw__` queries.
to_json(*args, **kwargs)
    Convert this document to JSON.

Parameters
    use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
    Return as SON data ready for use with MongoDB.

update(**kwargs)
    Performs an update on the Document. A convenience wrapper to update().
    Raises OperationError if called on an object that has not yet been saved.

validate(clean=True)
    Ensure that all fields’ values are valid and that required fields are present.
    Raises ValidationError if any of the fields’ values are found to be invalid.

class CytoPy.data.gate.ThresholdGate(*args, **values)
    Bases: CytoPy.data.gate.Gate

ThresholdGate inherits from Gate. A Gate attempts to separate single cell data in one or two-dimensional space using unsupervised learning algorithms. The algorithm is fitted to example data to generate “children”; the populations of cells a user expects to identify. These children are stored and then when the gate is ‘fitted’ to new data, the resulting populations are matched to the expected children.

The ThresholdGate subsets data based on the properties of the estimated probability density function of the underlying data. For each axis, kernel density estimation (KDEpy.FFTKDE) is used to estimate the PDF and a straight line “threshold” applied to the region of minimum density to separate populations. This is achieved using a peak finding algorithm and a smoothing procedure, until either:

- Two predominant “peaks” are found and the threshold is taken as the local minima between there peaks
- A single peak is detected and the threshold is applied as either the quantile given in method_kwars or the inflection point on the descending curve.

Alternatively the “method” can be “manual” for a static gate to be applied; user should provide x_threshold and y_threshold (if two-dimensional) to “method_kwars”, or “method” can be “quantile”, where the threshold will be drawn at the given quantile, defined by “q” in “method_kwars”.

Additional kwars to control behaviour of ThresholdGate when method is “density” can be given in method_kwars:

- kernel (default="guassian") - kernel used for KDE calculation (see KDEpy.FFTKDE for avialable kernels)
- bw (default="silverman") - bandwidth to use for KDE calculation, can either be “silverman” or “ISJ” or a float value (see KDEpy)
- min_peak_threshold (default=0.05) - percentage of highest recorded peak below which peaks are ignored. E.g. 0.05 would mean any peak less than 5% of the highest peak would be ignored.
- peak_boundary (default=0.1) - bounding window around which only the highest peak is considered. E.g. 0.1 would mean that peaks are assessed within a window the size of peak_boundary * length of probability vector and only highest peak within window is kept.
- inflection_point_kwars - dictionary; see CytoPy.data.gate.find_inflection_point
- smoothed_peak_finding_kwars - dictionary; see CytoPy.data.gate.smoothed_peak_finding

gate_name
    Name of the gate
    
    Type  str (required)
parent
   Parent population that this gate is applied to
   
   **Type** str (required)

\( x \)
   Name of the x-axis variable forming the one/two dimensional space this gate is applied to
   
   **Type** str (required)

\( y \)
   Name of the y-axis variable forming the two dimensional space this gate is applied to
   
   **Type** str (optional)

transformations
   Transform method to be applied to each dimension, should be a dictionary with keys corresponding to
   each variable (e.g. “x” and/or “y”) and values the transform to apply (e.g. {“x”: “logicle”} for logicle
   transform of x-axis)
   
   **Type** dict (optional)

control
   If a value is given here it should be the name of a control specimen commonly associated to the samples in
   an Experiment. When given this signals that the gate is designed to be applied to the control data not the
   primary data.
   
   **Type** str (optional)

controls
   Options for downsampling data prior to application of gate. Should contain a key/value pair for desired
   method e.g. (“method”: “uniform”). Available methods are: ‘uniform’, ‘density’ or ‘faithful’. See CytoPy.flow.sampling for details. Additional keyword arguments should be provided in the sampling dictionary.
   
   **Type** dict (optional)

dim_reduction
   Experimental feature. Allows for dimension reduction to be performed prior to applying gate. Gate will
   be applied to the resulting embeddings. Provide a dictionary with a key “method” and the value as any
   supported method in CytoPy.flow.dim_reduction. Additional keyword arguments should be provided in
   this dictionary.
   
   **Type** dict (optional)

method
   Name of the underlying algorithm to use. Should have a value of: “manual”, “density”, or “quantile”
   
   **Type** str (required)

method_kwargs
   Keyword arguments for initiation of the above method.
   
   **Type** dict

Miscellaneous:

Methods:

---

**DoesNotExist**
**MultipleObjectsReturned**

8.3. CytoPy.data.gate
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_child(child)</code></td>
<td>Add a new child for this gate.</td>
</tr>
<tr>
<td><code>cascade_save(**kwargs)</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete([signal_kwargs])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>fit(data)</code></td>
<td>Fit the gate using a given dataframe.</td>
</tr>
<tr>
<td><code>fit_predict(data)</code></td>
<td>Fit the gate using a given dataframe and then associate predicted Population objects to existing children.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query.</td>
</tr>
<tr>
<td><code>label_children(labels[, drop])</code></td>
<td>Rename children using a dictionary of labels where the key correspond to the existing child name and the value is the new desired population name.</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>predict(data)</code></td>
<td>Using existing children associated to this gate, the previously calculated thresholds of these children will be applied to the given data and then Population objects created and labelled to match the children of this gate.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls,...)</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields,**kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>reset_gate()</code></td>
<td>Removes existing children and resets all parameters.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean,...])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[,...])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><code>to_json(*args,**kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args,**kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document. A convenience wrapper to update().</td>
</tr>
</tbody>
</table>

continues on next page
Table 50 – continued from previous page

validate([clean])
Ensure that all fields’ values are valid and that required fields are present.

Classes:

my_metaclass
alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Attributes:

pk
Get the primary key.

exception DoesNotExist
Bases: CytoPy.data.gate.DoesNotExist
args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned
Bases: CytoPy.data.gate.MultipleObjectsReturned
args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

add_child(child: CytoPy.data.gate.ChildThreshold) → None
Add a new child for this gate. Checks that definition is valid and overwrites geom with gate information.

Parameters child (ChildThreshold) –

Returns

Return type None

cascade_save(**kwargs)
Recursively save any references and generic references on the document.
clean()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.
classmethod compare_indexes()
Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.
classmethod create_index(keys, background=False, **kwargs)
Creates the given indexes if required.

Parameters

- keys – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

- background – Allows index creation in the background
**delete** *(signal_kwargs=None, **write_concern)*

Delete the Document from the database. This will only take effect if the document has been previously saved.

**Parameters**

- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.
- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant getLastError command. For example, `save(..., w: 2, fsync: True)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument

**classmethod drop_collection**

Drops the entire collection associated with this Document type from the database.

Raises OperationError if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: OperationError exception raised if no collection available

**classmethod ensure_index** *(key_or_list, background=False, **kwargs)*

Ensure that the given indexes are in place. Deprecated in favour of create_index.

**Parameters**

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background

**classmethod ensure_indexes**

Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

---

**Note:** You can disable automatic index creation by setting `auto_create_index` to False in the documents meta data

**fit** *(data: pandas.core.frame.DataFrame) → None*

Fit the gate using a given dataframe. If children already exist will raise an AssertionError and notify user to call `fit_predict`.

**Parameters**

- **data** *(Pandas.DataFrame)* – Population data to fit threshold too

**Returns**

**Return type** None

**fit_predict** *(data: pandas.core.frame.DataFrame) → list*

Fit the gate using a given dataframe and then associate predicted Population objects to existing children. If no children exist, an AssertionError will be raised prompting the user to call `fit` method.

**Parameters**

- **data** *(Pandas.DataFrame)* – Population data to fit threshold too

**Returns**

List of predicted Population objects, labelled according to the gates child objects

**Return type** List

**classmethod from_json** *(json_data, created=False)*

Converts json data to a Document instance

**Parameters**
• **json_data**(str) – The json data to load into the Document

• **created**(bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).

  - If False and an ID is NOT provided, consider the document as brand new.

  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).

  - Defaults to False.

get_text_score()
Get text score from text query

label_children(labels: dict, drop: bool = True) → None
Rename children using a dictionary of labels where the key correspond to the existing child name and the value is the new desired population name. If the same population name is given to multiple children, these children will be merged. If drop is True, then children that are absent from the given dictionary will be dropped.

Parameters
• **labels**(dict) – Mapping for new children name

• **drop**(bool (default=True)) – If True, children absent from labels will be dropped

Returns

Return type None

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

modify(query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters
• **query** – the update will be performed only if the document in the database matches the query

• **update** – Django-style update keyword arguments

my_metaclass
alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Methods:
get_auto_id_names(new_class)  Find a name for the automatic ID field for the given new class.

mro()  Return a type’s method resolution order.

**property pk**
Get the primary key.

**predict** *(data: pandas.core.frame.DataFrame) → list*
Using existing children associated to this gate, the previously calculated thresholds of these children will be applied to the given data and then Population objects created and labelled to match the children of this gate. NOTE: the data will not be fitted and thresholds applied will be STATIC not data driven. For data driven gates call `fit_predict` method.

Parameters  
**data** *(Pandas.DataFrame)* – Data to apply static thresholds too

Returns  
List of Population objects

Return type  
List

**classmethod register_delete_rule** *(document_cls, field_name, rule)*
This method registers the delete rules to apply when removing this object.

**reload** *(**fields**, **kwargs)**
Reloads all attributes from the database.

Parameters  
- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

**reset_gate** () → None
Removes existing children and resets all parameters.

Returns  
Return type  
None

**save** *(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwars=None, _refs=None, save_condition=None, signal_kwars=None, **kwargs)*
Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters  
- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to *False* to skip.
- **clean** – call the document clean method, requires *validate* to be True.
- **write_concern** – Extra keyword arguments are passed down to *save() OR insert()* which will be used as options for the resultant getLastError command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
• **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__

• **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.

• **_refs** – A list of processed references used in cascading saves

• **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied

• **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: OperationError exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a SaveConditionError

Changed in version 0.10.7: Add signal_kwargs argument

**select_related** *(max_depth=1)*

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

**switch_collection**(collection_name, keep_created=True)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

Parameters

• **collection_name**(str) – The database alias to use for saving the document

• **keep_created**(bool) – keep self._created value after switching collection, else is reset to True

See also:

Use switch_db if you need to read from another database

**switch_db**(db_alias, keep_created=True)

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()

Parameters

- **db_alias**(str) – The database alias to use for saving the document
- **keep_created**(bool) – keep self._created value after switching db, else is reset to True

See also:

Use switch_collection if you need to read from another collection

to_dbref()
Returns an instance of DBRef useful in __raw__ queries.

to_json(*args, **kwargs)
Convert this document to JSON.

Parameters use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
Return as SON data ready for use with MongoDB.

update(**kwargs)
Performs an update on the Document A convenience wrapper to update().

Raises OperationError if called on an object that has not yet been saved.

validate(clean=True)
Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

CytoPy.data.gate.apply_threshold(data: pandas.core.frame.DataFrame, x: str, x_threshold: float, y: str = None, y_threshold: float = None) → Dict[str, pandas.core.frame.DataFrame]
Simple wrapper for threshold_1d and threshold_2d

Parameters

- **data**(Pandas.DataFrame)–
- **x**(str)–
- **x_threshold**(float)–
- **y**(str (optional))–
- **y_threshold**(float (optional))–

Returns

Return type  dict

CytoPy.data.gate.find_inflection_point(x: nump.array, p: nump.array, peak_idx: int, incline: bool = False, window_size: int = None, polyorder: int = 3, **kwargs)

Given some probability vector and grid space that represents a PDF as calculated by KDE, and assuming this vector has a single peak of highest density, calculate the inflection point at which the peak flattens. Probability vector is first smoothed using Savitzky-Golay filter.
Parameters

- \(x (np.array)\) – Grid space for the probability vector
- \(p (np.array)\) – Probability vector as calculated by KDE
- \(peak_idx (int)\) – Index of the peak
- \(incline (bool (default=False))\) – If true, calculates the inflection point of the incline towards the peak as opposed to the decline away from the peak
- \(window_size (int (optional))\) – Window length of filter (must be an odd number). If not given then it is calculated as an odd integer nearest to a 10th of the grid length
- \(polyorder (int (default=3))\) – Polynomial order for Savitzky-Golay filter
- \(kwargs (dict)\) – Additional keyword argument to pass to scipy.signal.savgol_filter

Returns Value of \(x\) at which the inflection point occurs

Return type float

CytoPy.data.gate.find_local_minima \((p: numpy.array, x: numpy.array, peaks: numpy.array) \rightarrow float\)

Find local minima between the two highest peaks in the density distribution provided

Parameters

- \(p (Numpy.array)\) – probability vector as generated from KDE
- \(x (Numpy.array)\) – Grid space for probability vector
- \(peaks (Numpy.array)\) – array of indices for identified peaks

Returns local minima between highest peaks

Return type float

CytoPy.data.gate.find_peaks \((p: numpy.array, min_peak_threshold: float, peak_boundary: float) \rightarrow numpy.array\)

Perform peak finding using the detecta package (see detecta.detect_peaks for details).

Parameters

- \(p (np.array)\) – Probability vector as generated from KDE
- \(min_peak_threshold (float)\) – Percentage of highest recorded peak below which peaks are ignored. E.g. 0.05 would mean any peak less than 5% of the highest peak would be ignored.
- \(peak_boundary (float)\) – Bounding window around which only the highest peak is considered. E.g. 0.1 would mean that peaks are assessed within a window the size of peak_boundary * length of probability vector and only highest peak within window is kept.

Returns Index of peaks

Return type Numpy.array

CytoPy.data.gate.merge_children \((children: list) \rightarrow CytoPy.data.gate.Child\)

Given a list of Child objects, merge and return single child

Parameters children\((list)\) –

Returns

Return type Child or ChildThreshold or ChildPolygon
CytoPy.data.gate.smoothed_peak_finding(p: numpy.array, starting_window_length: int = 11, polyorder: int = 3, min_peak_threshold: float = 0.05, peak_boundary: float = 0.1, **kwargs) -> (numpy.array, numpy.array)

Given the grid space and probability vector of some PDF calculated using KDE, first attempt to smooth the probability vector using a Savitzky-Golay filter (see scipy.signal.savgol_filter) and then perform peak finding until the number of peaks is less than 3. Window size will be incremented until the number of peaks is reduced. If window size exceeds half the length of the probability vector, will raise an AssertionError to avoid misrepresentation of the data.

Parameters

- **p** (numpy.array) – Probability vector resulting from KDE calculation
- **starting_window_length** (int (default=11)) – Window length of filter (must be > length of p, < length of p * 0.5, and an odd number)
- **polyorder** (int (default=3)) – Order of polynomial for filter
- **min_peak_threshold** (float (default=0.05)) – See CytoPy.data.gate.find_peaks
- **peak_boundary** (float (default=0.1)) – See CytoPy.data.gate.find_peaks
- **kwargs** (dict) – Additional keyword arguments to pass to scipy.signal.savgol_filter

Returns

Smooth probability vector and index of peaks

Return type

numpy.array, numpy.array

CytoPy.data.gate.threshold_1d(data: pandas.core.frame.DataFrame, x: str, x_threshold: float) -> Dict[str, pandas.core.frame.DataFrame]

Apply the given threshold (x_threshold) to the x-axis variable (x) and return the resulting dataframes corresponding to the positive and negative populations. Returns a dictionary of dataframes: {'-': Pandas.DataFrame, '+': Pandas.DataFrame}

Parameters

- **data** (Pandas.DataFrame) –
- **x** (str) –
- **x_threshold** (float) –

Returns

Negative population (less than threshold) and positive population (greater than or equal to threshold) in a dictionary as so: {'-': Pandas.DataFrame, '+': Pandas.DataFrame}

Return type

dict

CytoPy.data.gate.threshold_2d(data: pandas.core.frame.DataFrame, x: str, y: str, x_threshold: float, y_threshold: float) -> Dict[str, pandas.core.frame.DataFrame]

Apply the given threshold (x_threshold) to the x-axis variable (x) and the given threshold (y_threshold) to the y-axis variable (y), and return the resulting dataframes as a dictionary:

‘++’: Greater than or equal to threshold for both x and y ‘+-’: Greater than or equal to threshold for x but less than threshold for y ‘-+’: Greater than or equal to threshold for y but less than threshold for x ‘–’: Less than threshold for both x and y

Parameters

- **data** (Pandas.DataFrame) –
• `x (str)` –
• `y (str)` –
• `x_threshold (float)` –
• `y_threshold (float)` –

Returns

Return type `dict`

`CytoPy.data.gate.update_polygon(population: CytoPy.data.population.Population, parent_data: pandas.core.frame.DataFrame, x_values: list, y_values: list)`

Given an existing population and some new definition for it’s polygon gate (different to what is already associated to the Population), update the Population index and geom accordingly. Any associated clusters will be removed and any controls will have to be estimated again.

:param population:
:param parent_data:
:param x_values:
:param y_values:

`CytoPy.data.gate.update_threshold(population: CytoPy.data.population.Population, parent_data: pandas.core.frame.DataFrame, x_threshold: float, y_threshold: float = None)`

Given an existing population and some new threshold(s) (different to what is already associated to the Population), update the Population index and geom accordingly. Any associated clusters will be removed and any controls will have to be estimated again.

Parameters

• `population (Population)` –
• `parent_data (Pandas.DataFrame)` –
• `x_threshold (float)` –
• `y_threshold (float (optional))` – Required if 2D threshold geometry

Returns

Return type `None`

### 8.4 CytoPy.data.gating_strategy

In a traditional analysis, an immunologist would apply a ‘gating strategy’; a series of ‘gates’ that separate single cell data into the populations of interest. CytoPy provides autonomous gates (see CytoPy.data.gate) to emulate this process and these gates can be packaged together for bulk analysis using the GatingStrategy class, housed within this module.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
### Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Action</code> (*args, *<em>kwargs)</em></td>
<td>An Action represents a process applied to the gates/populations in some gating strategy that is independent of the gates themselves. At the moment this includes merging populations or subtracting one population from another. These actions can appear in a gating strategy and will be applied to new data in an autonomous fashion.</td>
</tr>
<tr>
<td><code>CreatePlot</code> (transform_x, transform_y, xlabel, ...)</td>
<td>Generate 1D or 2D histograms of cell populations as identified by cytometry.</td>
</tr>
<tr>
<td><code>EllipseGate</code> (*args, **values)</td>
<td>EllipseGate inherits from PolygonGate.</td>
</tr>
<tr>
<td><code>Experiment</code> (*args, **kwargs)</td>
<td>Container for Cytometry experiment.</td>
</tr>
<tr>
<td><code>FileGroup</code> (*args, **values)</td>
<td>Document representation of a file group; a selection of related fcs files (e.g.)</td>
</tr>
<tr>
<td><code>Gate</code> (*args, **values)</td>
<td>Base class for a Gate.</td>
</tr>
<tr>
<td><code>GatingStrategy</code> (*args, **values)</td>
<td>A GatingTemplate is synonymous to what an immunologist would classically consider a “gating template”; it is a collection of ‘gates’ (Gate objects, in the case of CytoPy) that can be applied to multiple fcs files or an entire experiment in bulk.</td>
</tr>
<tr>
<td><code>PolygonGate</code> (*args, **values)</td>
<td>PolygonGate inherits from Gate.</td>
</tr>
<tr>
<td><code>PolygonGeom</code> (*args, **values)</td>
<td>Polygon shape.</td>
</tr>
<tr>
<td><code>ThresholdGate</code> (*args, **values)</td>
<td>ThresholdGate inherits from Gate.</td>
</tr>
<tr>
<td><code>ThresholdGeom</code> (*args, **kwargs)</td>
<td>Threshold shape.</td>
</tr>
<tr>
<td><code>datetime</code> (year, month, day[, hour[, minute[, ...]])</td>
<td>The year, month and day arguments are required.</td>
</tr>
</tbody>
</table>

### Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gate_stats</code> (gate, populations, parent_data)</td>
<td>Print the statistics of populations generated from a Gate</td>
</tr>
<tr>
<td><code>hyperparameter_gate</code> (gate, grid, cost, parent)</td>
<td>Fit a Gate to some parent data whilst searching the hyperparameter space (grid) for the optimal ‘fit’ as defined by minimising some cost (e.g.)</td>
</tr>
<tr>
<td><code>progress_bar</code> (x[, verbose])</td>
<td>Generate a progress bar using the tqdm library.</td>
</tr>
<tr>
<td><code>update_polygon</code> (population, parent_data, ...)</td>
<td>Given an existing population and some new definition for it’s polygon gate (different to what is already associated to the Population), update the Population index and geom accordingly.</td>
</tr>
<tr>
<td><code>update_threshold</code> (population, parent_data, ...)</td>
<td>Given an existing population and some new threshold(s) (different to what is already associated to the Population), update the Population index and geom accordingly.</td>
</tr>
<tr>
<td><code>vprint</code> (verbose)</td>
<td>Utility function for optional printing.</td>
</tr>
</tbody>
</table>

```python
class CytoPy.data.gating_strategy.Action(*args, **kwargs)
    Bases: mongoengine.document.EmbeddedDocument

    An Action represents a process applied to the gates/populations in some gating strategy that is independent of the gates themselves. At the moment this includes merging populations or subtracting one population from another. These actions can appear in a gating strategy and will be applied to new data in an autonomous fashion.
```

**action_name**

- **Name of the action**
  - **Type** str

**method**
Should have a value of “merge” or “subtract”

Type str

left
The population to merge on or subtract from

Type str

right
The population to merge with or be subtracted from ‘left’

Type str

new_population_name
Name of the new population generated from this action

Type str

Methods:

**clean()**
Hook for doing document level data cleaning before validation is run.

**from_json(json_data[, created])**
Converts json data to a Document instance

**get_text_score()**
Get text score from text query

**to_json(**args, **kwargs)**
Convert this document to JSON.

**to_mongo(**args, **kwargs)**
Return as SON data ready for use with MongoDB.

**validate([clean])**
Ensure that all fields’ values are valid and that required fields are present.

Classes:

**my_metaclass**
alias of mongoengine.base.metaclasses.DocumentMetaclass

**clean()**
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

**classmethod from_json(json_data, created=False)**
Converts json data to a Document instance

Parameters

- **json_data (str)** – The json data to load into the Document
- **created (bool)** – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is PROVIDED, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.
get_text_score()
    Get text score from text query

my_metaclass
    alias of mongoengine.base.metaclasses.DocumentMetaClass

Methods:

mro()
    Return a type’s method resolution order.

to_json(*args, **kwargs)
    Convert this document to JSON.

    Parameters
    use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
    Return as SON data ready for use with MongoDB.

validate(clean=True)
    Ensure that all fields’ values are valid and that required fields are present.

    Raises ValueError if any of the fields’ values are found to be invalid.

class CytoPy.data.gating_strategy.GatingStrategy(*args, **values)
    Bases: mongoengine.document.Document

A GatingTemplate is synonymous to what an immunologist would classically consider a “gating template”; it is a collection of ’gates’ (Gate objects, in the case of CytoPy) that can be applied to multiple fcs files or an entire experiment in bulk. A user defines a GatingTemplate using a single example from an experiment, uses the object to preview gates and label child populations, and when satisfied with the performance save the GatingStrategy to the database to be applied to the remaining samples in the Experiment.

template_name
    unique identifier for template

    Type str, required

gates
    list of Gate documents

    Type EmbeddedDocumentList

creation_date
    date of creation

    Type DateTime

last_edit
    date of last edit

    Type DateTime

flags
    warnings associated to this gating template

    Type str, optional

notes
    free text comments

    Type str, optional

Miscellaneous:
Methods:

```
add_hyperparameter_grid(gate_name, params[, ...])
```

Add a hyperparameter grid to search which applying the given gate to new data. This hyperparameter grid should correspond to valid hyperparameters for the corresponding gate. Invalid parameters will be ignored. Choice of the cost parameter to be minimised is dependent on the type of gate: * ThresholdGate: - “manhattan” (default): optimal parameters are those that result in the population whom’s signature is of minimal distance to the original data used to define the gate. The manhattan distance is used as the distance metric. - “euclidean”: optimal parameters are those that result in the population whom’s signature is of minimal distance to the original data used to define the gate. The euclidean distance is used as the distance metric. - “threshold_dist”: optimal parameters are those that result in the threshold whom’s distance to the original threshold defined are smallest * PolygonGate & EllipseGate: - “hausdorff” (optional): parameters chosen that minimise the hausdorff distance between the polygon generated from new data and the original polgon gate created when the gate was defined - “manhattan” (default): optimal parameters are those that result in the population whom’s signature is of minimal distance to the original data used to define the gate. The manhattan distance is used as the distance metric. - “euclidean”: optimal parameters are those that result in the population whom’s signature is of minimal distance to the original data used to define the gate. The euclidean distance is used as the distance metric.

```
apply_action(action[, print_stats, ...])
```

Apply an action, that is, a merge or subtraction:

```
apply_all([verbose])
```

Apply all the gates associated to this GatingStrategy

```
apply_gate(gate[, plot, verbose, ...])
```

Apply a gate to the associated FileGroup.

```
cascade_save(**kwargs)
```

Recursively save any references and generic references on the document.

```
clean()
```

Hook for doing document level data cleaning before validation is run.

```
compare_indexes()
```

Compares the indexes defined in MongoEngine with the ones existing in the database.

```
create_index(keys[, background])
```

Creates the given indexes if required.

```
delete([delete_gates, remove_associations])
```

Delete gating strategy.

```
delete_actions(action_name)
```

Delete an action associated to this GatingStrategy

```
delete_gate(gate_name)
```

Remove a gate from this GatingStrategy.

```
delete_populations(populations)
```

Delete given populations.

```
drop_collection()
```

Drops the entire collection associated with this Document type from the database.

continues on next page
Table 60 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>edit_gate</strong> (gate_name[, x_threshold, ...])</td>
<td>Edit an existing gate (i.e.</td>
</tr>
<tr>
<td><strong>ensure_index</strong> (key_or_list[, background])</td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><strong>ensure_indexes()</strong></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><strong>from_json</strong> (json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><strong>get_gate</strong> (gate)</td>
<td>Given the name of a gate, return the Gate object</td>
</tr>
<tr>
<td><strong>get_text_score()</strong></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><strong>list_gates()</strong></td>
<td>List name of existing Gates</td>
</tr>
<tr>
<td><strong>list_indexes()</strong></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><strong>list_populations()</strong></td>
<td>Wrapper to FileGroup list_populations.</td>
</tr>
<tr>
<td><strong>load_data</strong> (experiment, sample_id)</td>
<td>Load a FileGroup into the GatingStrategy ready for gating.</td>
</tr>
<tr>
<td><strong>modify</strong> ([query])</td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><strong>plot_backgate</strong> (parent, overlay, x[, y, ...])</td>
<td>Given some population as the backdrop (parent) and a list of one or more populations that occur downstream of the parent (overlay), plot the downstream populations as scatter plots over the top of the parent.</td>
</tr>
<tr>
<td><strong>plot_gate</strong> (gate[, create_plot_kwars])</td>
<td>Plot a gate.</td>
</tr>
<tr>
<td><strong>plot_population</strong> (population, x[, y, ...])</td>
<td>Plot an existing population in the associate FileGroup.</td>
</tr>
<tr>
<td><strong>preview_gate</strong> (gate[, create_plot_kwars, ...])</td>
<td>Preview the results of some given Gate</td>
</tr>
<tr>
<td><strong>print_population_tree</strong> (<strong>kwargs</strong>)</td>
<td>Print the population tree to stdout.</td>
</tr>
<tr>
<td><strong>register_delete_rule</strong> (document_cls, ...)</td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><strong>reload</strong> ([fields, **kwargs])</td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><strong>save</strong> ([save_strategy, save_filegroup])</td>
<td>Save GatingStrategy and the populations generated for the associated FileGroup.</td>
</tr>
<tr>
<td><strong>select_related</strong> ([max_depth])</td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><strong>switch_collection</strong> (collection_name[, ...])</td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><strong>switch_db</strong> (db_alias[, keep_created])</td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><strong>to_dbref()</strong></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><strong>to_json</strong> (*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><strong>to_mongo</strong> (*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><strong>update</strong> (**kwargs)</td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><strong>validate</strong> ([clean])</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>my_metaclass</strong></td>
<td>alias of mongoengine.base.metaclasses.</td>
</tr>
<tr>
<td><strong>TopLevelDocumentMetaclass</strong></td>
<td>TopLevelDocumentMetaclass</td>
</tr>
</tbody>
</table>
Attributes:

```
pk
```

Get the primary key.

```
exception DoesNotExist
    Bases: mongoengine.errors.DoesNotExist

    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned
    Bases: mongoengine.errors.MultipleObjectsReturned

    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

add_hyperparameter_grid
    (gate_name: str, params: dict, cost: str = None)

Add a hyperparameter grid to search which applying the given gate to new data. This hyperparameter
grid should correspond to valid hyperparameters for the corresponding gate. Invalid parameters will be
ignored. Choice of the cost parameter to be minimised is dependent on the type of gate:

* **ThresholdGate**:
  - “manhattan” (default): optimal parameters are those that result in the population whom’s signature is
    of minimal distance to the original data used to define the gate. The manhattan distance is used as the
distance metric.
  - “euclidean”: optimal parameters are those that result in the population whom’s signature is of minimal
distance to the original data used to define the gate. The euclidean distance is used as the distance
metric.
  - “threshold_dist”: optimal parameters are those that result in the threshold
    whom’s distance to
    the original threshold defined are smallest

* **PolygonGate & EllipseGate**:
  - “hausdorff” (optional): parameters chosen that minimise the hausdorff distance between the
    polygon generated from new data and the original polgon gate created when the gate was
    defined
  - “manhattan” (default): optimal parameters are those that result in the population whom’s signature is
    of minimal distance to the original data used to define the gate. The manhattan distance is used as the
distance metric.
  - “euclidean”: optimal parameters are those that result in the population whom’s signature is of
    minimal distance to the original data used to define the gate. The euclidean distance is used as the
distance metric.

Parameters

* **gate_name** *(str)* – Gate to define hyperparameter grid for
* **params** *(dict)* – Grid of hyperparameters to be searched
* **cost** *(str)* – What to be minimised to choose optimal hyperparameters

Returns

Return type: None
apply_action (action: CytoPy.data.gating_strategy.Action, print_stats: bool = True, add_to_strategy: bool = True)

Apply an action, that is, a merge or subtraction:

- Merge: merge two populations present in the current population tree.
  The merged population will have the combined index of both populations but will not inherit any clusters and will not be associated to any children downstream of either the left or right population.
  The population will be added to the tree as a descendant of the left populations parent.
- Subtraction: subtract the right population from the left population. The right population must either have the same parent as the left population or be downstream of the left population. The new population will descend from the same parent as the left population. The new population will have a PolygonGeom geom.

Parameters

- action (Action) –
- print_stats (bool (default=True)) – Print population statistics to stdout
- add_to_strategy (bool (default=True)) – Add action to this GatingStrategy

Returns

Return type None

apply_all (verbose: bool = True)

Apply all the gates associated to this GatingStrategy

Parameters verbose (bool (default=True)) – If True, print feedback to stdout

Returns

Return type None

apply_gate (gate: str, plot: bool = True, verbose: bool = True, add_to_strategy: bool = True, create_plot_kwargs: dict = None, plot_gate_kwargs: dict = None, hyperparam_search: bool = True, overwrite_method_kwargs: dict = None)

Apply a gate to the associated FileGroup. The gate must be previously defined; children associated and labeled. Either a Gate object can be provided or the name of an existing gate saved to this GatingStrategy.

Parameters

- gate (str or Gate or ThresholdGate or PolygonGate or EllipseGate) – Name of an existing Gate or a Gate object
- plot (bool (default=True)) – If True, returns a Matplotlib.Axes object of plotted gate
- verbose (bool (default=True)) – If True, print gating statistics to stdout and provide feedback
- add_to_strategy (bool (default=True)) – If True, append the Gate to the GatingStrategy
- create_plot_kwargs (dict (optional)) – Additional arguments passed to CreatePlot
- plot_gate_kwargs (dict (optional)) – Additional arguments passed to plot_gate call of CreatePlot
- hyperparam_search (bool (default=True)) – If True and hyperparameter grid has been defined for the chosen gate, then hyperparameter search is performed to find the optimal fit for the newly encountered data.
- **overwrite_method_kwargs** (dict, optional) – If a dictionary is provided (and hyperparameter search isn’t defined for this gate) then method parameters are overwritten with these new parameters.

Returns

**Return type** Matplotlib.Axes or None

cascade_save (**kwargs)
Recursively save any references and generic references on the document.

clean ()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes ()
Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

classmethod create_index (keys, background=False, **kwargs)
Creates the given indexes if required.

Parameters

- **keys** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background

delete (delete_gates: bool = True, remove_associations: bool = True, *args, **kwargs)
Delete gating strategy. If delete_gates is True, then associated Gate objects will also be deleted. If remove_associations is True, then populations generated from this gating strategy will also be deleted.

Parameters

- **delete_gates** (bool (default=True))–
- **remove_associations** (bool (default=True))–
- **args** – Positional arguments for mongoengine.document.delete call
- **kwargs** – Keyword arguments for mongoengine.document.delete call

delete_actions (action_name: str)
Delete an action associated to this GatingStrategy

Parameters **action_name** (str)–

Returns

**Return type** None

delete_gate (gate_name: str)
Remove a gate from this GatingStrategy. Note: populations generated from this gate will not be deleted. These populations must be deleted separately by calling the ‘delete_population’ method.

Parameters **gate_name** (str) – Name of the gate for removal

Returns

**Return type** None

delete_populations (populations: str)
Delete given populations. Populations downstream from delete population(s) will also be removed.
Parameters populations (list or str) – Either a list of populations (list of strings) to remove or a single population as a string. If a value of “all” is given, all populations are dropped.

Returns
Return type None

classmethod drop_collection()  
Drops the entire collection associated with this Document type from the database.

Raises OperationError if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: OperationError exception raised if no collection available

edit_gate (gate_name: str, x_threshold: float = None, y_threshold: float = None, x_values: list = None, y_values: list = None)
Edit an existing gate (i.e. the polygon or threshold shape that generates the resulting populations). The altered geometry will be applied to the parent population resulting this gate acts upon, resulting in new data. Populations downstream of this edit will also be effected but gates will not adapt dynamically, instead the static results of gating algorithms will still apply, but to a new dataset. For this reason, gates should be checked (similar to the effects of moving a gate in FlowJo).

Parameters
• gate_name (str) –
• x_threshold (float (optional)) – Required for threshold geometries
• y_threshold (float (optional)) –
• for 2D threshold geometries (Required) –
• x_values (list) – Required for Polygon geometries
• y_values (list) – Required for Polygon geometries

Returns
Return type None

classmethod ensure_index (key_or_list, background=False, **kwargs)
Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters
• key_or_list – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
• background – Allows index creation in the background

classmethod ensure_indexes ()
Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting auto_create_index to False in the documents meta data

classmethod from_json (json_data, created=False)
Converts json data to a Document instance

Parameters
• json_data (str) – The json data to load into the Document
• **created (bool)** – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  – If False and an ID is NOT provided, consider the document as brand new.
  – If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  – Defaults to False.

`get_gate (gate: str) → CytoPy.data.gate.Gate`

Given the name of a gate, return the Gate object.

**Parameters**

- **gate (str)** –

**Returns**

**Return type** *Gate*

`get_text_score ()`

Get text score from text query.

`list_gates () → list`

List name of existing Gates.

**Returns**

**Return type** *list*

`classmethod list_indexes ()`

Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

`list_populations () → list`

Wrapper to FileGroup list_populations. Lists populations in associated FileGroup.

**Returns**

**Return type** *list*

`load_data (experiment: CytoPy.data.experiment.Experiment, sample_id: str)`

Load a FileGroup into the GatingStrategy ready for gating.

**Parameters**

- **experiment (Experiment)** –
- **sample_id (str)** –

**Returns**

**Return type** *None*

`modify (query=None, **update)`

Perform an atomic update of the document in the database and reload the document object using updated version.

**Returns** True if the document has been updated or False if the document in the database doesn’t match the query.
Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

- **query** – the update will be performed only if the document in the database matches the query
- **update** – Django-style update keyword arguments

**my_metaclass**
alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

**Methods:**

- `get_auto_id_names(new_class)`
  Find a name for the automatic ID field for the given new class.

- `mro()`
  Return a type’s method resolution order.

**property pk**
Get the primary key.

**plot_backgate** (parent: str, overlay: list, x: str, y: str=None, create_plot_kwargs: dict = None, **backgate_kwargs)
Given some population as the backdrop (parent) and a list of one or more populations that occur downstream of the parent (overlay), plot the downstream populations as scatter plots over the top of the parent.

**Parameters**

- **parent** (str)
- **overlay** (list)
- **x** (str)
- **y** (str)-
- **create_plot_kwargs** – Additional keyword arguments passed to CytoPy.plotting.CreatePlot
- **backgate_kwargs** – Additional keyword arguments passed to CytoPy.plotting.CreatePlot.backgate

**Returns**

**Return type** Matplotlib.Axes

**plot_gate** (gate: str, create_plot_kwargs: dict = None, **kwargs)
Plot a gate. Must provide the name of a Gate currently associated to this GatingStrategy. This will plot the parent population this gate acts on along with the geometries that define the child populations the gate generates.

**Parameters**

- **gate** (str or Gate or EllipseGate or ThresholdGate or PolygonGate)-

- create_plot_kwargs (dict) – Keyword arguments for CreatePlot object. See CytoPy.plotting.CreatePlot for details.

- **kwargs** – Keyword arguments for plot_gate call. See CytoPy.plotting.CreatePlot.plot_population_geom for details.

**Returns**
Return type  Matplotlib.Axes

```
plot_population (population: str, x: str, y: str = None, transform_x: str = 'logicle', transform_y: str = 'logicle', create_plot_kwargs: dict = None, **plot_kwargs)
```

Plot an existing population in the associate FileGroup.

Parameters

- **population** *(str)*
- **x** *(str)*
- **y** *(str (optional))*
- **transform_x** *(str (optional; default="logicle"))*
- **transform_y** *(str (optional; default="logicle"))*
- **create_plot_kwargs** – Additional keyword arguments passed to CytoPy.flow.plotting.CreatePlot
- **plot_kwargs** – Additional keyword arguments passed to CytoPy.flow.plotting.CreatePlot.plot

Returns

Return type  Matplotlib.Axes

```
preview_gate (gate: str, create_plot_kwargs: dict = None, plot_gate_kwargs: dict = None)
```

Preview the results of some given Gate

Parameters

- **gate** *(str or Gate or ThresholdGate or PolygonGate or EllipseGate)* – Name of an existing Gate or a Gate object
- **create_plot_kwargs** *(dict (optional))* – Additional arguments passed to CreatePlot
- **plot_gate_kwargs** *(dict (optional))* – Additional arguments passed to plot_gate call of CreatePlot

Returns

Return type  Matplotlib.Axes

```
print_population_tree (**kwargs)
```

Print the population tree to stdout. Wraps CytoPy.data.fcs.FileGroup.print_population_tree

Parameters **kwargs – See keyword arguments for CytoPy.data.fcs.FileGroup.print_population_tree

Returns

Return type  None

```
classmethod register_delete_rule (document_cls, field_name, rule)
```

This method registers the delete rules to apply when removing this object.

```
reload (*fields, **kwargs)
```

Reloads all attributes from the database.

Parameters

- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow
New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

**save** *(save_strategy: bool = True, save_filegroup: bool = True, *args, **kwargs)*

Save GatingStrategy and the populations generated for the associated FileGroup.

**Parameters**

- **save_filegroup** *(bool (default=True))*
- **save_strategy** *(bool (default=True))*
- **args** – Positional arguments for mongoengine.document.save call
- **kwargs** – Keyword arguments for mongoengine.document.save call

**Returns**

**Return type** None

**select_related** *(max_depth=1)*

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

**switch_collection** *(collection_name, keep_created=True)*

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

- **collection_name** *(str)* – The database alias to use for saving the document
- **keep_created** *(bool)* – keep self._created value after switching collection, else is reset to True

**See also:**

Use switch_db if you need to read from another database

**switch_db** *(db_alias, keep_created=True)*

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**Parameters**

- **db_alias** *(str)* – The database alias to use for saving the document
- **keep_created** *(bool)* – keep self._created value after switching db, else is reset to True
See also:

Use `switch_collection` if you need to read from another collection

**to_dbref()**

Returns an instance of `DBRef` useful in `__raw__` queries.

**to_json(**args, **kwargs)**

Convert this document to JSON.

Parameters `use_db_field` – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

**to_mongo(**args, **kwargs)**

Return as SON data ready for use with MongoDB.

**update(**kwargs)**


Raises `OperationError` if called on an object that has not yet been saved.

**validate(**clean=True)**

Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.

CytoPy.data.gating_strategy.gate_stats(**gate: CytoPy.data.gate.Gate, populations: list, parent_data: Pandas.DataFrame**)

Print the statistics of populations generated from a Gate

Parameters

- `gate (Gate)` –
- `populations (list)` – List of populations generated from fit_predict method of a Gate
- `parent_data (Pandas.DataFrame)` – Parent data that the gate is applied to

Returns

Return type None

8.5 CytoPy.data.setup

This module contains `global_init` which establishes a connection to the database and should be called at the start of each script.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Functions:

```
_global_init(database_name[, alias])          Global initializer for mongogengine ORM.
```

CytoPy.data.setup.global_init(database_name: str, alias: str = 'core', **kwargs) → None

Global initializer for mongogengine ORM. See mongoengine.register_connection for additional keyword arguments and mongoengine documentation for extensive details about registering connections. In brief, database connections are registered globally and refered to using an alias. By default CytoPy uses the alias ‘core’.

The database is assumed to be hosted locally, but if a remote server is used the user should provide the host address and port. If authentication is needed then a username and password should also be provided.

Parameters

- **database_name (str)** – name of database to establish connection with
- **alias (str (default="core"))** – name of connection to generate

Returns

Return type None

### 8.6 CytoPy.data.geometry

For the purpose of cytometry analysis we often think of a population of cells as having a particular phenotype that can be identified by sub-setting cells in one or two dimensional space. This results in geometric objects that define a population. This module houses the functionality around those geometric objects.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

```
ConvexHull(points[, incremental, qhull_options])  Convex hulls in N dimensions.
Ellipse(xy, width, height[, angle])             A scale-free ellipse.
Point(*args)                                   A zero dimensional feature
Polygon([shell, holes])                        A two-dimensional figure bounded by a linear ring
```

```
PolygonGeom(*args, **kwargs)                    Polygon shape.
PopulationGeometry(*args, **kwargs)            Geometric shape generated by non-threshold generating
                                             Gate
ThresholdGeom(*args, **kwargs)                Threshold shape.
```
Table 65 – continued from previous page

<table>
<thead>
<tr>
<th>partial</th>
<th>partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.</th>
</tr>
</thead>
</table>

Functions:

| Pool([processes, initializer, initargs, ...]) | Returns a process pool object |
| cpu_count() | Returns the number of CPUs in the system |
| create_convex_hull(x_values, y_values) | Given the x and y coordinates of a cloud of data points, generate a convex hull, returning the x and y coordinates of its vertices. |
| create_polygon(x, y) | Given a list of x coordinated and a list of y coordinates, generate a shapely Polygon |
| ellipse_to_polygon(centroid, width, height, ...) | Convert an ellipse to a shapely Polygon object. |
| inside_ellipse(data, center, width, height, ...) | Return mask of two dimensional matrix specifying if a data point (row) falls within an ellipse |
| inside_polygon(df, x, y, poly[, njobs]) | Return rows in dataframe who’s values for x and y are contained in some polygon coordinate shape |
| polygon_overlap(poly1, poly2[, threshold]) | Compare the area of two polygons and give the fraction overlap. |
| probabilistic_ellipse(covariances, conf) | Given the covariance matrix of a mixture component, calculate an elliptical shape that represents a probabilistic confidence interval. |
| warn(message[, category, stacklevel, source]) | Issue a warning, or maybe ignore it or raise an exception. |

Exceptions:

QhullError

class CytoPy.data.geometry.PolygonGeom(*args, **kwargs)
   Bases: CytoPy.data.geometry.PopulationGeometry
   Polygon shape. Inherits from PopulationGeometry.

   x_values
   X-axis coordinates
   Type list

   y_values
   Y-axis coordinates
   Type list

Methods:

   clean() | Hook for doing document level data cleaning before validation is run. |
   from_json(json_data[, created]) | Converts json data to a Document instance |
   get_text_score() | Get text score from text query |
   to_json(*args, **kwargs) | Convert this document to JSON. |
   to_mongo(*args, **kwargs) | Return as SON data ready for use with MongoDB. |

8.6. CytoPy.data.geometry 143

continues on next page
validate(clean= True)
Ensure that all fields’ values are valid and that required fields are present.

Classes:

my_metaclass
alias of mongoengine.base.metaclasses.DocumentMetaclass

Methods:

mro()
Return a type’s method resolution order.

to_json(*args, **kwargs)
Convert this document to JSON.

Parameters
use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
Return as SON data ready for use with MongoDB.

validate(clean=True)
Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

class CytoPy.data.geometry.PopulationGeometry(*args, **kwargs)
Bases: mongoengine.document.EmbeddedDocument
Geometric shape generated by non-threshold generating Gate

\( x \)
Name of the X-dimension e.g. CD3, FSC-A etc
Type str

\( y \)
Name of the Y-dimension e.g. CD3, FSC-A etc
Type str

\textit{transform_x}
Transformation method applied to the x-axis
Type str

\textit{transform_y}
Transformation method applied to the x-axis
Type str

Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean()</td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td>from_json(json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td>get_text_score()</td>
<td>Get text score from text query</td>
</tr>
<tr>
<td>to_json(*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td>to_mongo(*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td>validate([clean])</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

my_metaclass
alias of mongoengine.base.metaclasses.DocumentMetaclass

\texttt{clean()}
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON\_FIELD\_ERRORS.

\texttt{classmethod from_json(json_data, created=False)}
Converts json data to a Document instance

Parameters

- \texttt{json_data (str)} – The json data to load into the Document
- \texttt{created (bool)} – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  
  - If False and an ID is NOT provided, consider the document as brand new.
  
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
- Defaults to False.

```python
get_text_score()
Get text score from text query
```

```python
my_metaclass
alias of mongoengine.base.metaclasses.DocumentMetaclass Methods:
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mro()</td>
<td>Return a type’s method resolution order.</td>
</tr>
<tr>
<td>to_json(*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td>Parameters use_db_field</td>
<td>Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.</td>
</tr>
<tr>
<td>to_mongo(*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td>validate(clean=True)</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
<tr>
<td>Raises ValidationError if any of the fields’ values are found to be invalid.</td>
<td></td>
</tr>
</tbody>
</table>

```python
class CytoPy.data.geometry.ThresholdGeom(*args, **kwargs)
Bases: CytoPy.data.geometry.PopulationGeometry
Threshold shape. Inherits from PopulationGeometry.
```

```python
x_threshold
Threshold applied to the X-axis
Type float
```

```python
y_threshold
Threshold applied to the Y-axis
Type float
```

```python
Methods:
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean()</td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td>from_json(json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td>get_text_score()</td>
<td>Get text score from text query</td>
</tr>
<tr>
<td>to_json(*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td>to_mongo(*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td>validate([clean])</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

```python
my_metaclass
alias of mongoengine.base.metaclasses.DocumentMetaclass
```

```python
clean()
Hook for doing document level data cleaning before validation is run.
```
Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

**classmethod from_json**(json_data, created=False)

Converts json data to a Document instance

**Parameters**

- **json_data**(str) – The json data to load into the Document
- **created**(bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_text_score**()

Get text score from text query

**my_metaclass**

alias of mongoengine.base.metaclasses.DocumentMetaclass

**Methods:**

- **mro()**
  Return a type’s method resolution order.

**to_json**(*args, **kwargs)

Convert this document to JSON.

**Parameters** **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

**to_mongo**(*args, **kwargs)

Return as SON data ready for use with MongoDB.

**validate**(clean=True)

Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

CytoPy.data.geometry.create_convex_hull(x_values: numpy.array, y_values: numpy.array)

Given the x and y coordinates of a cloud of data points, generate a convex hull, returning the x and y coordinates of its vertices.

**Parameters**

- **x_values**(Numpy.array) –
- **y_values**(Numpy.array) –

**Returns**

**Return type** Numpy.array, Numpy.array

CytoPy.data.geometry.create_polygon(x: list, y: list)

Given a list of x coordinated and a list of y coordinates, generate a shapely Polygon

**Parameters**
CytoPy, Release 1.0.0

- \(x\) (list)
- \(y\) (list)

Returns

Return type: Polygon

CytoPy.data.geometry.ellipse_to_polygon(centroid: ((float, float), (float, float)), width: float, height: float, angle: float, ellipse: matplotlib.patches.Ellipse = None)

Convert an ellipse to a shapely Polygon object.

Parameters

- centroid ((float, float))-
- width (float)-
- height (float)-
- angle (float)-
- ellipse (Ellipse (optional))-  

Returns

Return type: Polygon

CytoPy.data.geometry.inside_ellipse(data: numpy.array, center: tuple, width: int, height: int, angle: int) → object

Return mask of two dimensional matrix specifying if a data point (row) falls within an ellipse

Parameters

- data (Numpy.array) – two dimensional matrix (x,y)
- center (tuple) – x,y coordinate corresponding to center of ellipse
- width (int or float) – semi-major axis of ellipse
- height (int or float) – semi-minor axis of ellipse
- angle (int or float) – angle of ellipse

Returns numpy array of indices for values inside specified ellipse

Return type: Numpy.array

CytoPy.data.geometry.inside_polygon(df: pandas.core.frame.DataFrame, x: str, y: str, poly: shapely.geometry.polygon.Polygon, njobs: int = -1)

Return rows in dataframe who’s values for \(x\) and \(y\) are contained in some polygon coordinate shape

Parameters

- df (Pandas.DataFrame) – Data to query
- \(x\) (str) – name of \(x\)-axis plane
- \(y\) (str) – name of \(y\)-axis plane
- poly (shapely.geometry.Polygon) – Polygon object to search
- njobs (int) – Number of jobs to run in parallel, by default uses all available cores

Returns Masked DataFrame containing only those rows that fall within the Polygon

Return type: Pandas.DataFrame
CytoPy.data.geometry.polygon_overlap(poly1: shapely.geometry.polygon.Polygon, poly2: shapely.geometry.polygon.Polygon, threshold: float = 0.0)

Compare the area of two polygons and give the fraction overlap. If fraction overlap does not exceed given threshold or the polygon’s do not overlap, return 0.0

Parameters
  • poly1 (Polygon) –
  • poly2 (Polygon) –
  • threshold(float (default = 0.0)) –

Returns
Return type float

CytoPy.data.geometry.probabilistic_ellipse(covariances: numpy.array, conf: float)

Given the covariance matrix of a mixture component, calculate a elliptical shape that represents a probabilistic confidence interval.

Parameters
  • covariances (np.array) – Covariance matrix
  • conf (float) – The confidence interval (e.g. 0.95 would give the region of 95% confidence)

Returns Width, Height and Angle of ellipse
Return type float, float, float

8.7 CytoPy.data.mapping

Each staining panel will have a combination of detection channel and the marker associated with that channel. This module houses ChannelMap, a simple class that keeps track of these mappings.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

ChannelMap(*args, **kwargs) Defines channel/marker mapping.

```python
class CytoPy.data.mapping.ChannelMap(*args, **kwargs)
Bases: mongoengine.document.EmbeddedDocument
```

8.7. CytoPy.data.mapping
Defines channel/marker mapping. Each document will contain a single value for channel and a single value for marker; these two values are treated as a pair within the panel.

**channel**
- Name of channel (e.g. fluorochrome)
  - **Type**: str

**marker**
- Name of marker (e.g. protein)
  - **Type**: str

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>check_matched_pair(channel, marker)</code></td>
<td>Check a channel/marker pair for resemblance</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query.</td>
</tr>
<tr>
<td><code>to_dict()</code></td>
<td>Convert object to python dictionary</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields' values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
</table>

**check_matched_pair** (channel: str, marker: str) → bool

Check a channel/marker pair for resemblance

**Parameters**

- **channel (str)** – channel to check
- **marker (str)** – marker to check

**Returns** True if equal, else False

**Return type** bool

**clean()**

Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

**classmethod from_json**(json_data, created=False)

Converts json data to a Document instance

**Parameters**

- **json_data (str)** – The json data to load into the Document
- **created (bool)** – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data
it’s loaded with (i.e. even if an ID is loaded).

– If False and an ID is NOT provided, consider the document as brand new.
– If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
– Defaults to False.

get_text_score()
Get text score from text query

my_metaclass
alias of mongoengine.base.metaclasses.DocumentMetaclass

Methods:

mro() Return a type’s method resolution order.

to_dict () → dict
Convert object to python dictionary

Returns

Return type  dict
to_json (*args, **kwargs)
Convert this document to JSON.

Parameters use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.
to_mongo (*args, **kwargs)
Return as SON data ready for use with MongoDB.

validate (clean=True)
Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

8.8 CytoPy.data.population

When analysing single cell data we are ultimately interested in populations of cells. This module contains the Population class, which controls the data attaining to a single cell population. A FileGroup (see CytoPy.data.fcs) can contain many Populations (which are embedded within the FileGroup). These Populations can also contain many Clusters, generated from a high-dimensional clustering algorithm applied to a population of cells e.g. FlowSOM or Phenograph. The cluster results are embedded within the Population as a collection of Clusters.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster(*args, **kwargs)</td>
<td>Represents a single cluster generated by a clustering experiment on a single file.</td>
</tr>
<tr>
<td>PolygonGeom(*args, **kwargs)</td>
<td>Polygon shape.</td>
</tr>
<tr>
<td>Population(*args, **kwargs)</td>
<td>A population of cells identified by either a gate or supervised algorithm.</td>
</tr>
<tr>
<td>PopulationGeometry(*args, **kwargs)</td>
<td>Geometric shape generated by non-threshold generating Gate.</td>
</tr>
<tr>
<td>ThresholdGeom(*args, **kwargs)</td>
<td>Threshold shape.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_signature(data[, idx, summary_method])</td>
<td>Given a dataframe of FCS events, generate a signature of those events; that is, a summary of the dataframes columns using the given summary method.</td>
</tr>
<tr>
<td>merge_multiple_populations(populations[, ...])</td>
<td>Merge multiple Population's.</td>
</tr>
<tr>
<td>merge_populations(left, right[, ...])</td>
<td>Merge two Population's.</td>
</tr>
<tr>
<td>reduce(function, sequence[, initial])</td>
<td>Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value.</td>
</tr>
<tr>
<td>scaler(data, scale_method[, return_scaler])</td>
<td>Wrapper for Sklearn transformation methods</td>
</tr>
<tr>
<td>unary_union(geoms)</td>
<td>Returns the union of a sequence of geometries</td>
</tr>
<tr>
<td>warn(message[, category, stacklevel, source])</td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

```python
class CytoPy.data.population.Cluster(*args, **kwargs)

    Bases: mongoengine.document.EmbeddedDocument

    Represents a single cluster generated by a clustering experiment on a single file. Clusters are generated by a Clustering object (see CytoPy.flow.clustering.main.Clustering for more info).

    cluster_id
        name associated to cluster (must be unique to population)

        Type  str, required

    meta_label
        name of associated meta-cluster to which this cluster belongs

        Type  str

    index
        index of cell events associated to cluster (very large array)

        Type  Numpy.Array

    n
        number of events in cluster

        Type  int, required

    prop_of_events
        proportion of events in cluster relative to root population
```
**Type** float, required

**tag**
identifier for grouping clusters derived from the same analysis/algorithm

**Type** str

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean()</td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td>from_json(json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td>get_text_score()</td>
<td>Get text score from text query</td>
</tr>
<tr>
<td>to_json(*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td>to_mongo(*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td>validate([clean])</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_metaclass</td>
<td>alias of mongoengine.base.metaclasses.DocumentMetaclass</td>
</tr>
</tbody>
</table>

**clean()**
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

**classmethod from_json**(json_data[, created])
Converts json data to a Document instance

**Parameters**

- **json_data** *(str)* – The json data to load into the Document
- **created** *(bool)* – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_text_score()**
Get text score from text query

**my_metaclass**
alias of mongoengine.base.metaclasses.DocumentMetaclass

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mro()</td>
<td>Return a type’s method resolution order.</td>
</tr>
</tbody>
</table>

8.8. CytoPy.data.population
Convert this document to JSON.

**Parameters**

- **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

**to_mongo** (*args, **kwargs)

Return as SON data ready for use with MongoDB.

**validate** (*clean=True*)

Ensure that all fields’ values are valid and that required fields are present.

Raises **ValidationError** if any of the fields’ values are found to be invalid.

**class**  
CytoPy.data.population.Population(*args, **kwargs)

**Bases:** mongoengine.document.EmbeddedDocument

A population of cells identified by either a gate or supervised algorithm. Stores the index of events corresponding to a single population, where the index relates back to the primary data in the FileGroup in which a population is embedded.

Populations also store Clusters generated from high dimensional clustering algorithms such as FlowSOM or PhenoGraph. These clusters are derived from this population.

**Parameters**

- **population_name** (*str, required*) – name of population
- **n** (*int*) – number of events associated to this population
- **parent** (*str, required, (default: "root")) – name of parent population
- **prop_of_parent** (*float, required*) – proportion of events as a percentage of parent population
- **prop_of_total** (*float, required*) – proportion of events as a percentage of all events
- **warnings** (*list, optional*) – list of warnings associated to population
- **geom** (*PopulationGeometry*) – PopulationGeometry (see CytoPy.data.geometry) that defines the gate that captures this population.
- **clusters** (*EmbeddedDocListField*) – list of associated Cluster documents
- **definition** (*str*) – relevant for populations generated by a ThresholdGate; defines the source of this population e.g. “+” for a 1D threshold or “+-” for a 2D threshold
- **index** (*Numpy.Array*) – numpy array storing index of events that belong to population
- **signature** (*dict*) – average of a population feature space (median of each channel); used to match children to newly identified populations for annotating

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_cluster(cluster)</td>
<td>Add a new cluster generated from CytoPy.flow.clustering.main.Clustering.</td>
</tr>
<tr>
<td>clean()</td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td>delete_all_clusters([clusters])</td>
<td>Provide either a list of cluster IDs for deletion or give value of “all” to delete all clusters.</td>
</tr>
<tr>
<td>delete_cluster([cluster_id, tag, meta_label])</td>
<td>Delete cluster using either cluster ID, tag, or meta label</td>
</tr>
</tbody>
</table>
Table 86 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_clusters([cluster_id, tag, meta_label])</code></td>
<td>Returns list of cluster objects by either cluster IDs, tag or meta label</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>list_clusters([tag, meta_label])</code></td>
<td>List cluster IDs associated to a given tag or meta label</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>my_metaclass</code></td>
<td>alias of <code>mongoengine.base.metaclasses.DocumentMetaclass</code></td>
</tr>
</tbody>
</table>

**add_cluster** *(cluster: CytoPy.data.population.Cluster)*

Add a new cluster generated from CytoPy.flow.clustering.main.Clustering.

- **Parameters**
  - `cluster` *(Cluster)*

- **Returns**
  - None

**clean()**

Hook for doing document level data cleaning before validation is run.

Any `ValidationError` raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

**delete_all_clusters** *(clusters: list = 'all')*

Provide either a list of cluster IDs for deletion or give value of “all” to delete all clusters.

- **Parameters**
  - `clusters` *(list or str (default="all"))-

- **Returns**
  - None

**delete_cluster** *(cluster_id: str = None, tag: str = None, meta_label: str = None)*

Delete cluster using either cluster ID, tag, or meta label

- **Parameters**
  - `cluster_id` *(str)-
  - `tag` *(str)-
  - `meta_label` *(str)-

- **Returns**
  - None

**classmethod from_json**(json_data, created=False)

Converts json data to a Document instance

- **Parameters**
  - `json_data` *(str)* – The json data to load into the Document
• **created (bool)** – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: *
  - If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_clusters (cluster_id: list = None, tag: str = None, meta_label: str = None) → List[CytoPy.data.population.Cluster]**
Returns list of cluster objects by either cluster IDs, tag or meta label

**Parameters**

• **cluster_id (list)** –
• **tag (str)** –
• **meta_label (str)** –

**Returns**

**Return type** List

**get_text_score ()**
Get text score from text query

**list_clusters (tag: str = None, meta_label: str = None) → List[str]**
List cluster IDs associated to a given tag or meta label

**Parameters**

• **tag (str)** –
• **meta_label (str)** –

**Returns**

**Return type** List

**my_metaclass**
alias of mongoengine.base.metaclasses.DocumentMetaclass

**Methods:**

• **mro()**
  Return a type’s method resolution order.

• **to_json (**args, **kwargs **)**
  Convert this document to JSON.

  **Parameters**
  **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

• **to_mongo (**args, **kwargs **)**
  Return as SON data ready for use with MongoDB.

• **validate (clean=True)**
  Ensure that all fields’ values are valid and that required fields are present.

  Raises ValidationError if any of the fields’ values are found to be invalid.
CytoPy.data.population.create_signature(data: pandas.core.frame.DataFrame, idx: numpy.array = None, summary_method: callable = None) → dict

Given a dataframe of FCS events, generate a signature of those events; that is, a summary of the dataframes columns using the given summary method.

Parameters
- **data** (Pandas.DataFrame) –
- **idx** (Numpy.array (optional)) – Array of indexes to be included in this operation, if None, the whole dataframe is used
- **summary_method** (callable (optional)) – Function to use to summarise columns, defaults is Numpy.median

Returns Dictionary representation of signature; {column name: summary statistic}

Return type dict

CytoPy.data.population.merge_multiple_populations(populations: List[CytoPy.data.population.Population], new_population_name: str = None)

Merge multiple Population’s. The indexes and signatures of these populations will be merged. The populations must have the same geometries.

Parameters
- **populations** (list) –
- **new_population_name** (str) –

Returns

Return type Population

CytoPy.data.population.merge_populations(left: CytoPy.data.population.Population, right: CytoPy.data.population.Population, new_population_name: str = None)

Merge two Population’s. The indexes and signatures of these populations will be merged. The populations must have the same geometries.

Parameters
- **left** (Population) –
- **right** (Population) –
- **new_population_name** (str) –

Returns

Return type Population
8.9 CytoPy.data.project

Every analysis is controlled using the Project class, the highest structure in the hierarchy of documents in the central MongoDB database. You can create multiple experiments for a Project, each attaining to a different staining panel. Experiments are accessed and managed through the Project class.

Projects also house the subjects (represented by the Subject class; see CytoPy.data.subject) of an analysis which can contain multiple meta-data.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

- Experiment(*args, **kwargs)
  Container for Cytometry experiment.

- Project(*args, **values)
  A project is the highest controlling structure of an analysis and houses all the experiments, their associated FileGroups and the populations contained in each FileGroup and the populations clusters.

- Subject(*args, **values)
  Document based representation of subject meta-data.

class CytoPy.data.project.Project(*args, **values)
  Bases: mongoengine.document.Document

  A project is the highest controlling structure of an analysis and houses all the experiments, their associated FileGroups and the populations contained in each FileGroup and the populations clusters.

  Project can be used to create new experiments and to load existing experiments to interact with.

  **project_id**
  unique identifier for project

  Type str, required

  **subjects**
  List of references for associated subjects; see Subject

  Type list

  **start_date**
  date of creation

  Type DateTime

  **owner**
  user name of owner

  Type str, required
**experiments**
List of references for associated fcs files

**Type** list

**Miscellaneous:**

*DoesNotExist*

*MultipleObjectsReturned*

**Methods:**

- `add_experiment(experiment_id, data_directory)`: Add new experiment to project.
- `add_subject(subject_id[, drug_data, ...])`: Create a new subject and associated to project; a subject is an individual element of a study e.g.
- `cascade_save(**kwargs)`: Recursively save any references and generic references on the document.
- `clean()`: Hook for doing document level data cleaning before validation is run.
- `compare_indexes()`: Compares the indexes defined in MongoEngine with the ones existing in the database.
- `create_index(keys[, background])`: Creates the given indexes if required.
- `delete(*args, **kwargs)`: Delete project (wrapper function of mongoengine.Document.delete)
- `drop_collection()`: Drops the entire collection associated with this Document type from the database.
- `ensure_index(key_or_list[, background])`: Ensure that the given indexes are in place.
- `ensure_indexes()`: Checks the document meta data and ensures all the indexes exist.
- `from_json(json_data[, created])`: Converts json data to a Document instance
- `get_subject(subject_id)`: Given a subject ID associated to Project, return the Subject document
- `get_text_score()`: Get text score from text query
- `list_experiments()`: Generate a list of associated flow cytometry experiments
- `list_indexes()`: Lists all of the indexes that should be created for given collection.
- `list_subjects()`: Generate a list of subject ID for subjects associated to this project
- `load_experiment(experiment_id)`: Load the experiment object for a given experiment ID
- `modify([query])`: Perform an atomic update of the document in the database and reload the document object using updated version.
- `register_delete_rule(document_cls, ...)`: This method registers the delete rules to apply when removing this object.
- `reload(*fields, **kwargs)`: Reloads all attributes from the database.
- `save([force_insert, validate, clean, ...])`: Save the Document to the database.
- `select_related([max_depth])`: Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

continues on next page
Table 91 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>switch_collection</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref</code></td>
<td>Returns an instance of DBRef useful in <code>__raw__</code> queries.</td>
</tr>
<tr>
<td><code>to_json</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update</code></td>
<td>Performs an update on the Document A convenience wrapper to <code>update()</code>.</td>
</tr>
<tr>
<td><code>validate</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

- `my_metaclass` alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

Attributes:

- `pk` Get the primary key.

**exception** `DoesNotExist`  
*Bases:* `mongoengine.errors.DoesNotExist`  
*args*  
*with_traceback*()  
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

**exception** `MultipleObjectsReturned`  
*Bases:* `mongoengine.errors.MultipleObjectsReturned`  
*args*  
*with_traceback*()  
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

**add_experiment**(experiment_id: str, data_directory: str, panel_name: str = None, panel_definition: str = None) \(\rightarrow\) `CytoPy.data.experiment.Experiment`  
Add new experiment to project. Note you must provide either a path to an excel template for the panel definition (panel_definition) or the name of an existing panel (panel_name). If panel_definition is provided, then the panel_name will be used to name the new Panel document associated to this experiment. If no panel_name is provided, then the panel name will default to “{experiment_id}.panel”.

**Parameters**

- `experiment_id` (str) – experiment name
- `data_directory` (str) – Path where experiment events data files will be stored
- `panel_name` (str (optional)) – Name of panel to associate to experiment
- `panel_definition` (str (optional)) – Path to excel template for generating the panel

**Returns** Newly created FCSExperiment
Return type  Experiment

add_subject(subject_id: str, drug_data: list = None, infection_data: list = None, patient_biology: list = None, **kwargs) → CytoPy.data.subject.Subject
Create a new subject and associated to project; a subject is an individual element of a study e.g. a patient or a mouse

Parameters

• subject_id(str) – subject ID for the new subject

• drug_data(list, optional) – list of Drug documents to associated to subject (see cytopy.data.subject.Drug)

• infection_data(list, optional) – list of Bug documents to associated to subject (see cytopy.data.subject.Bug)

• patient_biology(list, optional) – list of Biology documents to associated to subject (see cytopy.data.subject.Biology)

• kwargs – Additional keyword arguments to pass to Subject initialisation (see cytopy.data.subject.Subject)

Returns

Return type  None

cascade_save(**kwargs)
Recursively save any references and generic references on the document.

clean()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()  
Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing(extra indexes).

classmethod create_index(keys, background=False, **kwargs)
Creates the given indexes if required.

Parameters

• keys – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

• background – Allows index creation in the background

delete(*args, **kwargs) → None
Delete project (wrapper function of mongoengine.Document.delete)

Parameters

• args – positional arguments to pass to parent call (see mongoengine.Document.delete)

• kwargs – keyword arguments to pass to parent call (see mongoengine.Document.delete)

Returns

Return type  None

classmethod drop_collection()
Drops the entire collection associated with this Document type from the database.

Raises OperationError if the document has no collection set (i.e. if it is abstract)
Changed in version 0.10.7: `OperationError` exception raised if no collection available

**classmethod** ensure_index(key_or_list, background=False, **kwargs)

Ensure that the given indexes are in place. Deprecated in favour of `create_index`.

**Parameters**

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a `+` or a `-` to determine the index ordering
- **background** – Allows index creation in the background

**classmethod** ensure_indexes()

Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

**Note:** You can disable automatic index creation by setting `auto_create_index` to False in the documents meta data

**classmethod** from_json(json_data, created=False)

Converts json data to a Document instance

**Parameters**

- **json_data** (*str*) – The json data to load into the Document
- **created** (*bool*) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to `.save()`).
  - Defaults to False.

**get_subject**(subject_id: str) → `CytoPy.data.subject.Subject`

Given a subject ID associated to Project, return the Subject document

**Parameters** **subject_id** (*str*) – subject ID to pull

**Returns**

**Return type** `Subject`

**get_text_score()**

Get text score from text query

**list_experiments()** → Generator

Generate a list of associated flow cytometry experiments

**Returns** list of experiment IDs

**Return type** Generator

**classmethod** list_indexes()

Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.
list_subjects() → Generator
Generate a list of subject ID for subjects associated to this project

Returns List of subject IDs
Return type Generator

load_experiment (experiment_id: str) → CytoPy.data.experiment.Experiment
Load the experiment object for a given experiment ID

Parameters experiment_id (str) – experiment to load
Returns
Return type Experiment

modify (query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters

• query – the update will be performed only if the document in the database matches the query
• update – Django-style update keyword arguments

my_metaclass
alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Methods:

get_auto_id_names(new_class) Find a name for the automatic ID field for the given new class.
mro() Return a type’s method resolution order.

property pk
Get the primary key.

classmethod register_delete_rule (document_cls, field_name, rule)
This method registers the delete rules to apply when removing this object.

reload (*fields, **kwargs)
Reloads all attributes from the database.

Parameters

• fields – (optional) args list of fields to reload
• max_depth – (optional) depth of dereferencing to follow

New in version 0.1.2.
Changed in version 0.6: Now chainable
Changed in version 0.9: Can provide specific fields to reload
save (force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)

Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters

- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to False to skip.
- **clean** – call the document clean method, requires validate to be True.
- **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern={w: 2, fsync: True}, ...) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.
- **_refs** – A list of processed references used in cascading saves
- **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: OperationError exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a SaveConditionError

Changed in version 0.10.7: Add signal_kwars argument

select_related (max_depth=1)

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

switch_collection (collection_name, keep_created=True)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```
Parameters

- **collection_name (str)** – The database alias to use for saving the document
- **keep_created (bool)** – keep self._created value after switching collection, else is reset to True

See also:

Use switch_db if you need to read from another database

```python
switch_db(db_alias, keep_created=True)
```
Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

- **db_alias (str)** – The database alias to use for saving the document
- **keep_created (bool)** – keep self._created value after switching db, else is reset to True

See also:

Use switch_collection if you need to read from another collection

```python
to_dbref()
```
Returns an instance of DBRef useful in __raw__ queries.

```python
to_json(*args, **kwargs)
```
Convert this document to JSON.

Parameters **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

```python
to_mongo(*args, **kwargs)
```
Return as SON data ready for use with MongoDB.

```python
update(**kwargs)
```
Performs an update on the Document A convenience wrapper to `update()`.

Raises `OperationError` if called on an object that has not yet been saved.

```python
validate(clean=True)
```
Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.
8.10 CytoPy.data.subject

Each subject in your analysis (human, mouse, cell line etc) can be represented by a Subject document that can then be associated to specimens in an Experiment. This Subject document is dynamic and can house any relating meta-data.

Projects also house the subjects (represented by the Subject class; see CytoPy.data.subject) of an analysis which can contain multiple meta-data.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology(*args, **kwargs)</td>
<td>Document representation of biological test (blood pathology).</td>
</tr>
<tr>
<td>Bug(*args, **kwargs)</td>
<td>Document representation of isolated pathogen.</td>
</tr>
<tr>
<td>Drug(*args, **kwargs)</td>
<td>Document representation of drug administration.</td>
</tr>
<tr>
<td>FileGroup(*args, **values)</td>
<td>Document representation of a file group; a selection of related fcs files (e.g.)</td>
</tr>
<tr>
<td>MetaDataDictionary(*args, **values)</td>
<td>Model for a custom dictionary that can be used for given descriptions to meta-data.</td>
</tr>
<tr>
<td>Subject(*args, **values)</td>
<td>Document based representation of subject meta-data.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>biology(subject_id, test_name, method)</td>
<td>Given some test name, return a summary statistic of all results for a given patient ID</td>
</tr>
<tr>
<td>bugs(subject, multi_org[, short_name])</td>
<td>Fetch the name of isolated organisms for each patient.</td>
</tr>
<tr>
<td>gram_status(subject)</td>
<td>Given an instance of Subject, return the gram status of isolated organisms.</td>
</tr>
<tr>
<td>hmbpp_ribo(subject, field)</td>
<td>Given a value of either ‘hmbpp’ or ‘ribo’ for ‘field’ argument, return True if any Bug has a positive status for the given patient ID.</td>
</tr>
<tr>
<td>org_type(subject)</td>
<td>Parse all infectious isolates for each patient and return the organism type isolated, one of either: ‘gram positive’, ‘gram negative’, ‘virus’, ‘mixed’ or ‘fungal’</td>
</tr>
</tbody>
</table>

class CytoPy.data.subject.Biology(*args, **kwargs)

    Bases: mongoengine.document.EmbeddedDocument


    test_date
date that test was performed
    Type  DateTime

test
    name of pathology test
    Type  str

result
    value of pathology test
    Type  float

unit
    units reported
    Type  str

ref_range
    reported reference range
    Type  str

test_category
    category of test
    Type  str

Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean()</td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td>from_json(json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td>get_text_score()</td>
<td>Get text score from text query</td>
</tr>
<tr>
<td>to_json(*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td>to_mongo(*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td>validate([clean])</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_metaclass</td>
<td>alias of mongoengine.base.metaclasses.DocumentMetaclass</td>
</tr>
</tbody>
</table>

**clean()**

Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

**classmethod from_json (json_data, created=False)**

Converts json data to a Document instance

Parameters

- **json_data (str)** – The json data to load into the Document
- **created (bool)** – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data
it’s loaded with (i.e. even if an ID is loaded).

– If False and an ID is NOT provided, consider the document as brand new.
– If False and an ID is provided, assume that the object has already been persisted (this
  has an impact on the subsequent call to .save()).
– Defaults to False.

get_text_score()
Get text score from text query

my_metaclass
    alias of mongoengine.base.metaclasses.DocumentMetaClass

Methods:

mro()  Return a type’s method resolution order.

to_json(*args, **kwargs)
Convert this document to JSON.

    Parameters use_db_field – Serialize field names as they appear in MongoDB (as opposed
to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)
Return as SON data ready for use with MongoDB.

validate(clean=True)
Ensure that all fields’ values are valid and that required fields are present.

    Raises ValidationError if any of the fields’ values are found to be invalid.

class CytoPy.data.subject.Bug(*args, **kwargs)
    Bases: mongoengine.document.EmbeddedDocument

Document representation of isolated pathogen. Single document instance represents one pathogen.

gram_status
    value of organisms gram status, valid choices are [‘P+ve’, ‘N-ve’, ‘Unknown’]

        Type str, optional

hmbpp_status
    value of hmbpp status, valid choices are [‘P+ve’, ‘N-ve’, ‘Unknown’]

        Type str, optional

ribo_status
    value of organisms ribo status, valid choices are [‘P+ve’, ‘N-ve’, ‘Unknown’]

        Type str, optional

org_name
    name of the organism

        Type str

id_method
    method used to identify organism

        Type str, optional

culture_source
    site of isolated organism
Type  str, optional

organism_type  
  type of organism isolated, valid choices are ['bacteria', 'fungi', 'virus']
  Type  str, optional

report_date  
  date that organism was reported
  Type  DateTime, optional

notes  
  string value for free text notes
  Type  str, optional

Methods:

- clean()  
  Hook for doing document level data cleaning before validation is run.

- from_json(json_data[, created])  
  Converts json data to a Document instance

- get_text_score()  
  Get text score from text query

- to_json(*args, **kwargs)  
  Convert this document to JSON.

- to_mongo(*args, **kwargs)  
  Return as SON data ready for use with MongoDB.

- validate([clean])  
  Ensure that all fields’ values are valid and that required fields are present.

Classes:

- my_metaclass  
  alias of mongoengine.base.metaclasses.DocumentMetaclass

- clean()  
  Hook for doing document level data cleaning before validation is run.

  Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

- classmethod from_json(json_data, created=False)  
  Converts json data to a Document instance

  Parameters

  - json_data (str) – The json data to load into the Document
  - created (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.
**get_text_score()**
Get text score from text query

**my_metaclass**
alias of `mongoengine.base.metaclasses.DocumentMetaclass` Methods:

- **mro()**
  Return a type’s method resolution order.

**to_json(*args, **kwargs)**
Convert this document to JSON.

  **Parameters**
  - `use_db_field` – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

**to_mongo(*args, **kwargs)**
Return as SON data ready for use with MongoDB.

**validate(clean=True)**
Ensure that all fields’ values are valid and that required fields are present.

  Raises `ValidationError` if any of the fields’ values are found to be invalid.

**class CytoPy.data.subject.Drug(*args, **kwargs)**
Bases: `mongoengine.document.EmbeddedDocument`

Document representation of drug administration. Single document instance represents one event.

- **name**
  name of therapy/drug
  Type: str

- **init_date**
  date that therapy/drug started
  Type: DateTime

- **end_data**
  date that therapy/drug ended
  Type: DateTime

Methods:

- **clean()**
  Hook for doing document level data cleaning before validation is run.

- **from_json(json_data[, created])**
  Converts json data to a Document instance

- **get_text_score()**
  Get text score from text query

- **to_json(*args, **kwargs)**
  Convert this document to JSON.

- **to_mongo(*args, **kwargs)**
  Return as SON data ready for use with MongoDB.

- **validate([clean])**
  Ensure that all fields’ values are valid and that required fields are present.

Classes:

- **my_metaclass**
  alias of `mongoengine.base.metaclasses.DocumentMetaclass`

  **clean()**
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod from_json (json_data, created=False)

Converts json data to a Document instance

Parameters

- **json_data** (str) – The json data to load into the Document
- **created** (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

get_text_score()

Get text score from text query

my_metaclass

alias of mongoengine.base.metaclasses.DocumentMetaclass

Methods:
mro() Return a type’s method resolution order.

to_json(*args, **kwargs)

Convert this document to JSON.

Parameters **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

to_mongo(*args, **kwargs)

Return as SON data ready for use with MongoDB.

validate(clean=True)

Ensure that all fields’ values are valid and that required fields are present.

Raises ValidationError if any of the fields’ values are found to be invalid.

class CytoPy.data.subject.MetaDataDictionary(*args, **values)

Bases: mongoengine.document.Document

Model for a custom dictionary that can be used for given descriptions to meta-data. Helpful when exploring single cell data that has been associated to meta-data in the Explorer object; see flow.clustering.main.Explorer

key

name of meta-data (column name)

Type str

desc

string value of written description

Type str
Miscellaneous:

- DoesNotExist
- MultipleObjectsReturned

Methods:

- `cascade_save(**kwargs)` Recursively save any references and generic references on the document.
- `clean()` Hook for doing document level data cleaning before validation is run.
- `compare_indexes()` Compares the indexes defined in MongoEngine with the ones existing in the database.
- `create_index(keys[, background])` Creates the given indexes if required.
- `delete([signal_kwargs])` Delete the Document from the database.
- `drop_collection()` Drops the entire collection associated with this Document type from the database.
- `ensure_index(key_or_list[, background])` Ensure that the given indexes are in place.
- `ensure_indexes()` Checks the document meta data and ensures all the indexes exist.
- `from_json(json_data[, created])` Converts json data to a Document instance
- `get_text_score()` Get text score from text query
- `list_indexes()` Lists all of the indexes that should be created for given collection.
- `modify([query])` Perform an atomic update of the document in the database and reload the document object using updated version.
- `register_delete_rule(document_cls,...)` This method registers the delete rules to apply when removing this object.
- `reload(*fields, **kwargs)` Reloads all attributes from the database.
- `save([force_insert, validate, clean,...])` Save the Document to the database.
- `select_related([max_depth])` Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.
- `switch_collection(collection_name[,...])` Temporarily switch the collection for a document instance.
- `switch_db(db_alias[, keep_created])` Temporarily switch the database for a document instance.
- `to_dbref()` Returns an instance of DBRef useful in __raw__ queries.
- `to_json(*args, **kwargs)` Convert this document to JSON.
- `to_mongo(*args, **kwargs)` Return as SON data ready for use with MongoDB.
- `update(**kwargs)` Performs an update on the Document A convenience wrapper to update().
- `validate([clean])` Ensure that all fields’ values are valid and that required fields are present.

Classes:
my_metaclass

my_metaclass is an alias of mongoengine.base.metaclasses.TopLevelDocumentMetaClass.

Attributes:

pk

Get the primary key.

exception DoesNotExist

Bases: mongoengine.errors.DoesNotExist

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned

Bases: mongoengine.errors.MultipleObjectsReturned

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

cascade_save(**kwargs)

Recursively save any references and generic references on the document.

clean()

Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a

special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()

Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any miss-

ing/extra indexes.

classmethod create_index(keys, background=False, **kwargs)

Creates the given indexes if required.

Parameters

- keys – a single index key or a list of index keys (to construct a multi-field index); keys
  may be prefixed with a + or a - to determine the index ordering

- background – Allows index creation in the background

delete(signal_kwargs=None, **write_concern)

Delete the Document from the database. This will only take effect if the document has been previously

saved.

Parameters

- signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.

- write_concern – Extra keyword arguments are passed down which will be used as
  options for the resultant getLastError command. For example, save(..., w:
  2, fsync: True) will wait until at least two servers have recorded the write and
  will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument
classmethod drop_collection()
Drops the entire collection associated with this Document type from the database.
Raises OperationError if the document has no collection set (i.e. if it is abstract)
Changed in version 0.10.7: OperationError exception raised if no collection available

classmethod ensure_index(key_or_list, background=False, **kwargs)
Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters
- key_or_list – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- background – Allows index creation in the background

classmethod ensure_indexes()
Checks the document meta data and ensures all the indexes exist.
Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting auto_create_index to False in the documents meta data

classmethod from_json(json_data, created=False)
Converts json data to a Document instance

Parameters
- json_data (str) – The json data to load into the Document
- created (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

get_text_score()
Get text score from text query

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

modify(query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.
Parameters

- **query** – the update will be performed only if the document in the database matches the query
- **update** – Django-style update keyword arguments

```python
my_metaclass
    alias of mongoengine.base.metaclases.TopLevelDocumentMetacllass
```

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_auto_id_names(new_class)</code></td>
<td>Find a name for the automatic ID field for the given new class.</td>
</tr>
<tr>
<td><code>mro()</code></td>
<td>Return a type’s method resolution order.</td>
</tr>
</tbody>
</table>

**property pk**

Get the primary key.

```python
classmethod register_delete_rule(document_cls, field_name, rule)
```

This method registers the delete rules to apply when removing this object.

```python
reload(*fields, **kwargs)
```

Reloads all attributes from the database.

**Parameters**

- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

```python
save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)
```

Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

**Parameters**

- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to False to skip.
- **clean** – call the document clean method, requires validate to be True.
- **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern=(w: 2, fsync: True), ...) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.
- **_refs** – A list of processed references used in cascading saves.
• **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises `OperationError` if the conditions are not satisfied

• **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any `DBRef` objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: `OperationError` exception raised if save_condition fails.

Changed in version 0.10.1: `class: save_condition` failure now raises a `SaveConditionError`

Changed in version 0.10.7: Add signal_kwargs argument

**select_related** *(max_depth=1)*

Handles dereferencing of `DBRef` objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

**switch_collection**(collection_name, keep_created=True)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

• **collection_name**(str) – The database alias to use for saving the document

• **keep_created**(bool) – keep self.created value after switching collection, else is reset to True

**See also:**

Use `switch_db` if you need to read from another database

**switch_db**(db_alias, keep_created=True)

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**Parameters**

• **db_alias**(str) – The database alias to use for saving the document
- **keep_created**(bool) – keep self._created value after switching db, else is reset to True

  See also:
  
  Use `switch_collection` if you need to read from another collection

**to_dbref()**

Returns an instance of DBRef useful in `__raw__` queries.

**to_json(**`*args`, **`**kwargs**`)**

Convert this document to JSON.

  **Parameters** use_db_field – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

**to_mongo(**`*args`, **`**kwargs**`)**

Return as SON data ready for use with MongoDB.

**update(**`**kwargs**`)**


  Raises `OperationError` if called on an object that has not yet been saved.

**validate**(clean=True)

Ensure that all fields’ values are valid and that required fields are present.

  Raises `ValidationError` if any of the fields’ values are found to be invalid.

**class** `CytoPy.data.subject.Subject(**`*args`, **`**values**`)`

**Bases:** `mongoengine.document.DynamicDocument`

Document based representation of subject meta-data. Subjects are stored in a dynamic document, meaning new properties can be added ad-hoc.

**subject_id**

Unique identifier for subject

  **Type** str, required

**files**

List of references to files associated to subject

  **Type** ListField

**drug_data**

Associated drug data

  **Type** EmbeddedDocListField

**infection_data**

Associated infection data

  **Type** EmbeddedDocListField

**patient_biology**

Associated biological data

  **Type** EmbeddedDocListField

**notes**

Additional notes

  **Type** str
CytoPy, Release 1.0.0

## Miscellaneous:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoesNotExist</td>
<td></td>
</tr>
<tr>
<td>MultipleObjectsReturned</td>
<td></td>
</tr>
</tbody>
</table>

## Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cascade_save(**kwargs)</code></td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td><code>clean()</code></td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td><code>compare_indexes()</code></td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete(*args, **kwargs)</code></td>
<td>Delete the Subject.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls, . . . )</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean, . . . ])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[, . . . ])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
</tbody>
</table>

## Classes:

### 178 Chapter 8. API Reference
my_metaclass

alias of mongoengine.base.metaclasses.

TopLevelDocumentMetaclass

Attributes:

pk

Get the primary key.

exception DoesNotExist

Bases: mongoengine.errors.DoesNotExist

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned

Bases: mongoengine.errors.MultipleObjectsReturned

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

cascade_save (**kwargs)

Recursively save any references and generic references on the document.

clean()

Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a
special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()

Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any miss-
ing/extra indexes.

classmethod create_index (keys, background=False, **kwargs)

Creates the given indexes if required.

Parameters

• keys – a single index key or a list of index keys (to construct a multi-field index); keys
  may be prefixed with a + or a - to determine the index ordering

• background – Allows index creation in the background

delete (*args, **kwargs)

Delete the Subject. The subject will automatically be pulled from associated Projects (reference field in
Project model has reverse_delete_rule=4; see mongoengine API for info).

WARNING: deletion of a subject will result in the automatic removal of all associated FCS data!

Parameters

• signal_kwargs (optional) – kwargs dictionary to be passed to the signal calls.

• write_concern – Extra keyword arguments are passed down which will be used as
  options for the resultant getLastError command. For example, save(..., w: 2, fsync: True)
  will wait until at least two servers have recorded the write and will force an fsync on the
  primary server.

Returns
Return type: None

classmethod drop_collection()
Drops the entire collection associated with this Document type from the database.

Raises OperationError if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: OperationError exception raised if no collection available

classmethod ensure_index(key_or_list, background=False, **kwargs)
Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

- **background** – Allows index creation in the background

classmethod ensure_indexes()
Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting auto_create_index to False in the documents meta data

classmethod from_json(json_data, created=False)
Converts json data to a Document instance

Parameters

- **json_data** (str) – The json data to load into the Document

- **created** (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).

  - If False and an ID is NOT provided, consider the document as brand new.

  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).

  - Defaults to False.

get_text_score()
Get text score from text query

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

modify(query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.
**Note:** All unsaved changes that have been made to the document are rejected if the method returns True.

**Parameters**

- **query** – the update will be performed only if the document in the database matches the query
- **update** – Django-style update keyword arguments

**my_metaclass**

Alias of `mongoengine.base.metaclasess.TopLevelDocumentMetaclass` Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_auto_id_names(new_class)</code></td>
<td>Find a name for the automatic ID field for the given new class.</td>
</tr>
<tr>
<td><code>mro()</code></td>
<td>Return a type's method resolution order.</td>
</tr>
</tbody>
</table>

**property pk**

Get the primary key.

**classmethod register_delete_rule(document_cls, field_name, rule)**

This method registers the delete rules to apply when removing this object.

**reload(*fields, **kwargs)**

Reloads all attributes from the database.

**Parameters**

- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

**save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)**

Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

**Parameters**

- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to False to skip.
- **clean** – call the document clean method, requires validate to be True.
- **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern={w: 2, fsync: True}, ...) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
• **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.

• **_refs** – A list of processed references used in cascading saves

• **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied

• **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

**Changed in version 0.5:** In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

**Changed in version 0.6:** Added cascading saves

**Changed in version 0.8:** Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

**Changed in version 0.8.5:** Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

**Changed in version 0.10:** OperationError exception raised if save_condition fails.

**Changed in version 0.10.1:** :class: save_condition failure now raises a SaveConditionError

**Changed in version 0.10.7:** Add signal_kwargs argument

**select_related** (**max_depth=1**)  
Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

**New in version 0.5.**

**switch_collection** (**collection_name**, **keep_created=True**)  
Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

• **collection_name** (**str**) – The database alias to use for saving the document

• **keep_created** (**bool**) – keep self._created value after switching collection, else is reset to True

**See also:**

Use switch_db if you need to read from another database

**switch_db** (**db_alias**, **keep_created=True**)  
Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```
Parameters

- **db_alias** *(str)* – The database alias to use for saving the document
- **keep_created** *(bool)* – keep self._created value after switching db, else is reset to True

See also:
Use switch_collection if you need to read from another collection

to_dbref()  
Returns an instance of DBRef useful in __raw__ queries.
to_json(*args, **kwargs)*  
Convert this document to JSON.

Parameters **use_db_field** – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.
to_mongo(*args, **kwargs)*  
Return as SON data ready for use with MongoDB.
update(**kwargs)*  
Performs an update on the Document. A convenience wrapper to update().
Raises OperationError if called on an object that has not yet been saved.
validate(clean=True)*  
Ensure that all fields’ values are valid and that required fields are present.
Raises ValidationException if any of the fields’ values are found to be invalid.

CytoPy.data.subject.biology(subject_id: str, test_name: str, method: str) → float
Given some test name, return a summary statistic of all results for a given patient ID

Parameters

- **subject_id** *(str)* – patient identifier
- **test_name** *(str)* – name of test to search for
- **method** *(str)* – summary statistic to use

Returns Summary statistic (numpy float) or None if test does not exist

Return type Numpy.float or None

CytoPy.data.subject.bugs(subject: CytoPy.data.subject.Subject, multi_org: str, short_name: bool = False) → str
Fetch the name of isolated organisms for each patient.

Parameters

- **subject** *(Subject)* –
- **short_name** *(bool)* – If True, the shortened name rather than whole latin name is returned
- **multi_org** *(str)* – If ‘multi_org’ equals ‘list’ then multiple organisms will be stored as a comma separated list without duplicates, whereas if the value is ‘mixed’ then multiple organisms will result in a value of ‘mixed’.

Returns string of isolated organisms comma separated, or ‘mixed’ if multi_org == ‘mixed’ and multiple organisms listed for patient

Return type str
CytoPy.data.subject.gram_status (subject: CytoPy.data.subject.Subject) → str
Given an instance of Subject, return the gram status of isolated organisms. Where multiple organisms are found, if gram status differs amongst orgs, returns ‘mixed’

Parameters subject (Subject) –

Returns String value for gram status

Return type str

CytoPy.data.subject.hmbpp_ribo (subject: CytoPy.data.subject.Subject, field: str) → str
Given a value of either ‘hmbpp’ or ‘ribo’ for ‘field’ argument, return True if any Bug has a positive status for the given patient ID.

Parameters

• subject (Subject) –

• field (str) – field name to search for; expecting either ‘hmbpp_status’ or ‘ribo_status’

Returns common value of hmbpp_status/ribo_status

Return type str

CytoPy.data.subject.org_type (subject: CytoPy.data.subject.Subject) → str
Parse all infectious isolates for each patient and return the organism type isolated, one of either: ‘gram positive’, ‘gram negative’, ‘virus’, ‘mixed’ or ‘fungal’

Parameters subject (Subject) –

Returns common organism type isolated for patient

Return type str

8.11 CytoPy.data.read_write

The read_write module contains tools for accessing *.fcs files and relies on the Python library FlowIO by Scott White. This is used by Experiment to population FileGroups.

Projects also house the subjects (represented by the Subject class; see CytoPy.data.subject) of an analysis which can contain multiple meta-data.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:
**FCSFile**(filepath[, comp_matrix])

Utilising FlowIO to generate an object for representing an FCS file

### Functions:

**Pool**([processes, initializer, initargs, ...])

Returns a process pool object

**chunks**(df_list, n)

Yield successive n-sized chunks from l.

**cpu_count**()

Returns the number of CPUs in the system

**explore_channel_mappings**(fcs_dir[, ...])

Given a directory, explore all fcs files and find all permutations of channel/marker mappings

**fcs_mappings**(path)

Fetch channel mappings from fcs file.

**filter_fcs_files**(fcs_dir[, exclude_comps, ...])

Given a directory, return file paths for all fcs files in directory and subdirectories contained within

**get_fcs_file_paths**(fcs_dir, control_names, ...)

Generate a standard dictionary object of fcs files in given directory.

- **Parameters**
  - `fcs_dir`: target directory for search
  - `control_names`: names of expected control files (names must appear in filenames)
  - `ctrl_id`: global identifier for control file e.g.

---

**class CytoPy.data.read_write.FCSFile(filepath, comp_matrix=None)**

Bases: object

Utilising FlowIO to generate an object for representing an FCS file

**Parameters**

- **filepath** *(str)* – location of fcs file to parse
- **comp_matrix** *(str)* – csv file containing compensation matrix (optional, not required if a spillover matrix is already linked to the file)

**Methods:**

**compensate()**

Apply compensation to event data

**compensate**()

Apply compensation to event data

Returns

Return type None

**CytoPy.data.read_write.chunks**(df_list, n: int) \(\rightarrow\) pandas.core.frame.DataFrame

Yield successive n-sized chunks from l. ref: https://stackoverflow.com/questions/312443/how-do-you-split-a-list-into-evenly-sized-chunks

**Parameters**

- **df_list** *(list)* – list of DataFrames to generated ‘chunks’ from
- **n** *(int)* – number of chunks to generate

**Returns** Yields successive n-sized DataFrames

**Return type** generator
CytoPy.data.read_write.explore_channel_mappings (fcs_dir: str, exclude_comps: bool = True) → list

Given a directory, explore all fcs files and find all permutations of channel/marker mappings

Parameters

• fcs_dir (str) – root directory to search

• exclude_comps (bool, (default=True)) – exclude compentation files (must have ‘comp’ in filename)

Returns list of all unique channel/marker mappings

Return type List

CytoPy.data.read_write.fcs_mappings (path: str) → list

Fetch channel mappings from fcs file.

Parameters path (str) – path to fcs file

Returns List of channel mappings. Will return None if file fails to load.

Return type List or None

CytoPy.data.read_write.filter_fcs_files (fcs_dir: str, exclude_comps: bool = True, exclude_dir: str = ‘DUPLICATES’) → list

Given a directory, return file paths for all fcs files in directory and subdirectories contained within

Parameters

• fcs_dir (str) – path to directory for search

• exclude_comps (bool) – if True, compensation files will be ignored (note: function searches for ‘comp’ in file name for exclusion)

• exclude_dir (str (default = ‘DUPLICATES’)) – Will ignore any directories with this name

Returns list of fcs file paths

Return type list

CytoPy.data.read_write.get_fcs_file_paths (fcs_dir: str, control_names: list, ctrl_id: str, ignore_comp: bool = True) → dict

Generate a standard dictionary object of fcs files in given directory :param fcs_dir: target directory for search :type fcs_dir: str :param control_names: names of expected control files (names must appear in filenames) :type control_names: list :param ctrl_id: global identifier for control file e.g. ‘FMO’ (must appear in filenames) :type ctrl_id: str :param ignore_comp: If True, files with ‘compensation’ in their name will be ignored (default = True) :type ignore_comp: bool, (default=True)

Returns standard dictionary of fcs files contained in target directory

Return type dict
8.12 CytoPy.data.supervised_classifier

Cells can be classified by following a traditional approach of separating data in one or two dimensional space using “gates” which can be automated in CytoPy using the Gate and GatingStrategy classes. CytoPy also offers an alternative approach through the CellClassifier, contained here. The CellClassifier allows you to take gated examples, train a supervised classification algorithm and then annotate the remaining data using the trained model. The apparatus of CellClassifier means that the resulting populations can be stored to a FileGroup in Populations and handled in all other subsequent analysis.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axes(fig, rect[, facecolor, frameon, ...])</td>
<td>The Axes contains most of the figure elements: ~.axis.Axis, ~.axis.Tick, ~.lines.Line2D, ~.text.Text, ~.patches.Polygon, etc., and sets the coordinate system.</td>
</tr>
<tr>
<td>BaseCrossValidator()</td>
<td>Base class for all cross-validators</td>
</tr>
<tr>
<td>CellClassifier(*args, **values)</td>
<td>Supervised classification of cells using a gated example.</td>
</tr>
<tr>
<td>Experiment(*args, **kwargs)</td>
<td>Container for Cytometry experiment.</td>
</tr>
<tr>
<td>FileGroup(*args, **values)</td>
<td>Document representation of a file group; a selection of related fcs files (e.g.</td>
</tr>
<tr>
<td>GridSearchCV(**kwargs)</td>
<td>Exhaustive search over specified parameter values for an estimator.</td>
</tr>
<tr>
<td>History()</td>
<td>Callback that records events into a History object.</td>
</tr>
<tr>
<td>KFold(**kwargs)</td>
<td>K-Folds cross-validator</td>
</tr>
<tr>
<td>KerasCellClassifier(*args, **values)</td>
<td>Supervised classification of cells using a gated example.</td>
</tr>
<tr>
<td>Layer(*args, **kwargs)</td>
<td>Neural networks consist of layers and in the Keras framework these can be represented by individual objects.</td>
</tr>
<tr>
<td>Population(*args, **kwargs)</td>
<td>A population of cells identified by either a gate or supervised algorithm.</td>
</tr>
<tr>
<td>RandomOverSampler(**kwargs)</td>
<td>Class to perform random over-sampling.</td>
</tr>
<tr>
<td>RandomizedSearchCV(**kwargs)</td>
<td>Randomized search on hyper parameters.</td>
</tr>
<tr>
<td>SklearnCellClassifier(*args, **values)</td>
<td>Supervised classification of cells using a gated example.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_signature(data[, idx, summary_method])</td>
<td>Given a dataframe of FCS events, generate a signature of those events; that is, a summary of the dataframes columns using the given summary method.</td>
</tr>
</tbody>
</table>
Table 120 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>learning_curve</td>
<td>Learning curve.</td>
</tr>
<tr>
<td>progress_bar</td>
<td>Generate a progress bar using the tqdm library.</td>
</tr>
<tr>
<td>scaler</td>
<td>Wrapper for Sklearn transformation methods.</td>
</tr>
<tr>
<td>signature</td>
<td>Get a signature object for the passed callable.</td>
</tr>
<tr>
<td>train_test_split</td>
<td>Split arrays or matrices into random train and test subsets.</td>
</tr>
<tr>
<td>vprint</td>
<td>Utility function for optional printing.</td>
</tr>
<tr>
<td>warn</td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

Class: CytoPy.data.supervised_classifier.CellClassifier(*args, **values)

Supervised classification of cells using a gated example. The user provides an Experiment and the name of a sample which has existing Populations; we call this the ‘reference sample’. Note, a CellClassifier should be defined for a single Experiment and should not be applied to multiple Experiments. These Populations are used as labels in training data, to train a supervised classification model. This class is the parent of two classes: SklearnCellClassifier and KerasCellClassifier. Please consult the documentation for these classes for more information.

**name**
Name of the CellClassifier to save to the database. Must be unique

- **Type**: str, required

**feature**
List of markers used as input variables for classification

- **Type**: list, required

**multi_class**
If True, the classification problem will be treated as a multi-class problem; that is, a single cell can belong to multiple populations and the classifier will be trained to attribute multiple classes to a cell. This requires that the user provides a threshold for positivity which defaults to 0.5.

- **Type**: bool (default=False)

**target_populations**
List of populations to search for in the reference sample and use as target labels

- **Type**: list, required

**transform**
Transformation to apply to data prior to training/prediction

- **Type**: str (optional; default="logicle")

**scale**
Value of “standard” or “norm” should be provided if you wish to scale the data prior to training/prediction. Recommended for most methods except tree-based classifiers.

- **Type**: str (optional; default=None)

**scale_kwargs**
Keyword arguments passed to scaling method, see CytoPy.flow.transform

- **Type**: dict

**downsample**
Value of ‘uniform’, ‘density’ or ‘faithful’ should be provided if the user wishes to downsample the data prior to training. For downsampling methods see CytoPy.flow.sampling.
Type  str (optional, default=None)

downsampling_kwargs
Keyword arguments passed to sampling method, see CytoPy.flow.sampling
Type  dict

class_weights
Can be used to handle class imbalance by passing a dictionary of weights to associate to each population
class. Alternatively user can use the “auto_class_weights” method to calculate balanced weights using the
compute_class_weight function from Scikit-Learn.
Type  dict (optional)

population_prefix
Prefix added to the name of predicted populations
Type  str (default="sml")

verbose
Provide feedback to stdout
Type  bool (default=True)

model
Read-only attribute housing the classification model
Type  object

x
Training feature space
Type  Pandas.DataFrame

y
Training labels
Type  Numpy.Array

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Methods:

auto_class_weights()
Compute optimal class weights using the compute_class_weights function from Scikit-Learn.
cascade_save(**kwargs)
Recursively save any references and generic references on the document.
clean()
Hook for doing document level data cleaning before validation is run.
compare_indexes()
Compares the indexes defined in MongoEngine with the ones existing in the database.
create_index(keys[, background])
Creates the given indexes if required.
delete([signal_kwargs])
Delete the Document from the database.
drop_collection()
Drops the entire collection associated with this Document type from the database.
ensure_index(key_or_list[, background])
Ensure that the given indexes are in place.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensure_indexes()</td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td>fit(**kwargs)</td>
<td>Fits the model to loaded training data.</td>
</tr>
<tr>
<td>fit_cv([cross_validator, metrics, ...])</td>
<td>Fits the model to training data using cross-validation.</td>
</tr>
<tr>
<td>fit_train_test_split([test_frac, metrics, ...])</td>
<td>Fit the model and test on holdout data using a test-train split approach.</td>
</tr>
<tr>
<td>from_json(json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td>get_text_score()</td>
<td>Get text score from text query</td>
</tr>
<tr>
<td>list_indexes()</td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td>load_training_data(experiment, reference, ...)</td>
<td>Loads training data from the given Experiment.</td>
</tr>
<tr>
<td>load_validation(experiment, validation_id, ...)</td>
<td>Load a FileGroup from the same Experiment as the training data (reference sample) and use this as validation data to test the model.</td>
</tr>
<tr>
<td>modify([query])</td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td>predict(experiment, sample_id, root_population)</td>
<td>Predict the population labels for cells in a FileGroup (specified by “sample_id”) in the same Experiment as the training data (reference sample).</td>
</tr>
<tr>
<td>register_delete_rule(document_cls, ...)</td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td>reload(*fields, **kwargs)</td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td>save([force_insert, validate, clean, ...])</td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td>select_related([max_depth])</td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td>switch_collection(collection_name[,...])</td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td>switch_db(db_alias[, keep_created])</td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td>to_dbref()</td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td>to_json(*args, **kwargs)</td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td>to_mongo(*args, **kwargs)</td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td>update(**kwargs)</td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td>validate([clean])</td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
<tr>
<td>validate_classifier(experiment,...[, ...])</td>
<td>Validate the model on a FileGroup in the same Experiment as the training data (reference sample).</td>
</tr>
</tbody>
</table>

Classes:

```python
my_metaclass
```

*alias of mongoengine.base.metaclases.

TopLevelDocumentMetacllass

Attributes:
Get the primary key.

```python
exception DoesNotExist
    Bases: mongoengine.errors.DoesNotExist
    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MultipleObjectsReturned
    Bases: mongoengine.errors.MultipleObjectsReturned
    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

auto_class_weights()
    Compute optimal class weights using the compute_class_weights function from Scikit-Learn. Weights are stored as a dictionary in self.class_weights.

    Returns
    Return type  None

cascade_save(**kwargs)
    Recursively save any references and generic references on the document.

clean()
    Hook for doing document level data cleaning before validation is run.

    Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()
    Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

classmethod create_index(keys, background=False, **kwargs)
    Creates the given indexes if required.

    Parameters

    • keys – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

    • background – Allows index creation in the background

delete(signal_kwargs=None, **write_concern)
    Delete the Document from the database. This will only take effect if the document has been previously saved.

    Parameters

    • signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.

    • write_concern – Extra keyword arguments are passed down which will be used as options for the resultant getLastError command. For example, save(..., w: 2, fsync: True) will wait until at least two servers have recorded the write and will force an fsync on the primary server.

    Changed in version 0.10.7: Add signal_kwargs argument
```
classmethod drop_collection()
Drops the entire collection associated with this Document type from the database.
Raises OperationError if the document has no collection set (i.e., if it is abstract)
Changed in version 0.10.7: OperationError exception raised if no collection available

classmethod ensure_index(key_or_list, background=False, **kwargs)
Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters

• key_or_list – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
• background – Allows index creation in the background

classmethod ensure_indexes()
Checks the document meta data and ensures all the indexes exist.
Global defaults can be set in the meta - see guide/defining-documents

Note: You can disable automatic index creation by setting auto_create_index to False in the documents meta data

fit(**kwargs)
Fits the model to loaded training data. If “load_data” is not called prior to this method, will raise AssertionError.

Parameters kwargs – Additional keyword arguments pass to “fit”

Returns

Return type None

fit_cv(cross_validator: sklearn.model_selection._split.BaseCrossValidator = None, metrics: list = None, threshold: float = 0.5, split_kwargs: dict = None, fit_kwargs: dict = None)
Fit the model to training data using cross-validation. The model will be fitted to training data generated when “load_data” is called. If “load_data” is not previously called will raise an AssertionException.

Any Scikit-Learn cross-validator object can be used to perform cross validation, but if not given, will default to a basic K-fold cross-validation.

Parameters

• cross_validator(BaseCrossValidator (default=KFold)) –
• metrics(list (optional)) –
  List of metrics to assess performance with. Default to:
  – Balanced accuracy
  – Weighted F1 score
  – ROC AUC score
• threshold(float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class
• split_kwargs(dict (optional)) – Additional keyword arguments passed to “split” call of cross_validator
• fit_kwargs(dict (optional)) – Additional keyword arguments passed to “fit” call of model
Returns List of dictionaries detailing training performance on each round List of dictionaries
detailing testing performance on each round

Return type List, List

\texttt{fit\_train\_test\_split (test\_frac: float = 0.3, metrics: list = None, return\_predictions: bool = True, threshold: float = 0.5, train\_test\_split\_kwargs: dict = None, fit\_kwargs: dict = None)}

Fit the model and test on holdout data using a test-train split approach. The model will be fitted to
training data generated when “load_data” is called. If “load_data” is not previously called will raise an
AssertionError.

Parameters

\begin{itemize}
  \item \texttt{test\_frac (float (default=0.3))} – Proportion of training data to be kept
    as holdout data for testing
  \item \texttt{metrics (list (optional))} – List of metrics to assess performance with. Default to:
    \begin{itemize}
      \item Balanced accuracy
      \item Weighted F1 score
      \item ROC AUC score
    \end{itemize}
  \item \texttt{return\_predictions (bool (default=True))} – If True, vector of pre-
    dicted labels returned along with dictionary of classification performance
  \item \texttt{threshold (float (default=0.5))} – If multi_class is True, this will be used
to determine positive association to a class
  \item \texttt{train\_test\_split\_kwargs (dict (optional))} – Additional keyword ar-
    guments passed to call to Scikit-Learn’s train\_test\_split function
  \item \texttt{fit\_kwargs (dict (optional))} – Additional keyword arguments passed to models
    fit method call
\end{itemize}

Returns Dictionary of training and testing performance. If return\_predictions is True, will
also return an array of cell class predictions.

Return type dict or (dict, Numpy.Array)

\texttt{classmethod from\_json (json\_data, created=False)}

Converts json data to a Document instance

Parameters

\begin{itemize}
  \item \texttt{json\_data (str)} – The json data to load into the Document
  \item \texttt{created (bool)} – Boolean defining whether to consider the newly instantiat-
ed document as brand new or as persisted already: * If True, consider the document as
brand new, no matter what data
  it’s loaded with (i.e. even if an ID is loaded).
  \begin{itemize}
    \item If False and an ID is NOT provided, consider the document as brand new.
    \item If False and an ID is provided, assume that the object has already been persisted
(this has an impact on the subsequent call to .save()).
    \item Defaults to False.
  \end{itemize}
\end{itemize}
get_text_score()
Get text score from text query

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

load_training_data(experiment: CytoPy.data.experiment.Experiment, reference: str, root_population: str)
Loads training data from the given Experiment. User must provide the name of an existing FileGroup associated to this Experiment (reference) that has the desired Populations previously created. The Populations should be downstream of a given “root_population”, which acts as the starting point of analysis.
This method populates the attributes “x” (the feature space) and “y” (the training labels). The training labels, y, will either be a 1-dimensional array, if multi_class if False, or a multi-dimensional array where each column is a population label, if multi_class is True.

Parameters
• experiment (Experiment)
• reference (str)
• root_population (str)

Returns
None

load_validation(experiment: CytoPy.data.experiment.Experiment, validation_id: str, root_population: str)
Load a FileGroup from the same Experiment as the training data (reference_sample) and use this as validation data to test the model. The validation FileGroup must contain all of the Population labels in target_populations and these populations must be downstream of ‘root_population’.

Parameters
• experiment (Experiment)
• validation_id (str)
• root_population (str)

Returns Feature space and population labels for validation FileGroup

Return type Pandas.DataFrame, Numpy.Array

modify(query=None, **update)
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

Note: All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters
• query – the update will be performed only if the document in the database matches the query
• update – Django-style update keyword arguments
my_metcaclass
    alias of mongoengine.base.metaclasses.TopLevelDocumentMetaclass

Methods:

get_auto_id_names(new_class)
    Find a name for the automatic ID field for the given new class.

mro()
    Return a type’s method resolution order.

property pk
    Get the primary key.

predict(experiment: CytoPy.data.experiment.Experiment, sample_id: str, root_population: str, threshold: float = 0.5, return_predictions: bool = True)
    Predict the population labels for cells in a FileGroup (specified by “sample_id”) in the same Experiment as the training data (reference sample). Populations will be generated with the “root_population” as their parent. The modified FileGroup with newly predicted Populations will be returned. To save the Populations, call the “save” method of the returned FileGroup.

Parameters

- experiment (Experiment) – Must be the same Experiment that the training data was from
- sample_id (str) – Name of the FileGroup to predict populations for
- root_population (str) – Root of the analysis; will be the immediate parent of any generated Populations
- threshold (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class
- return_predictions (bool (default=True)) – If True, vector of predicted labels returned along with the modified FileGroup object

Returns
    Modified FileGroup with newly predicted Populations If return_predictions is True, will return a dictionary of the following format: {“y_pred”: predicted labels, “y_score”: confidence scores}

Return type
    FileGroup or (FileGroup, dict)

classmethod register_delete_rule(document_cls, field_name, rule)
    This method registers the delete rules to apply when removing this object.

reload(*fields, **kwargs)
    Reloads all attributes from the database.

Parameters

- fields – (optional) args list of fields to reload
- max_depth – (optional) depth of dereferencing to follow

New in version 0.1.2.
Changed in version 0.6: Now chainable
Changed in version 0.9: Can provide specific fields to reload

save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)
    Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters
• **force_insert** – only try to create a new document, don’t allow updates of existing documents.

• **validate** – validates the document; set to False to skip.

• **clean** – call the document clean method, requires validate to be True.

• **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern={w: 2, fsync: True}, ...) will wait until at least two servers have recorded the write and will force an fsync on the primary server.

• **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__

• **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.

• **_refs** – A list of processed references used in cascading saves

• **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied

• **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: OperationError exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a SaveConditionError

Changed in version 0.10.7: Add signal_kwargs argument

select_related(max_depth=1)

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

switch_collection(collection_name, keep_created=True)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling save():

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

Parameters

• **collection_name**(str) – The database alias to use for saving the document
• `keep_created (bool)` – keep self._created value after switching collection, else is reset to True

See also:
Use `switch_db` if you need to read from another database

```python
switch_db (db_alias, keep_created=True)
```
Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

- `db_alias (str)` – The database alias to use for saving the document
- `keep_created (bool)` – keep self._created value after switching db, else is reset to True

See also:
Use `switch_collection` if you need to read from another collection

```python
to_dbref ()
```
Returns an instance of `DBRef` useful in `__raw__` queries.

```python
to_json (*args, **kwargs)
```
Convert this document to JSON.

**Parameters**

- `use_db_field` – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

```python
to_mongo (*args, **kwargs)
```
Return as SON data ready for use with MongoDB.

```python
update (**kwargs)
```

Raises `OperationError` if called on an object that has not yet been saved.

```python
validate (clean=True)
```
Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.

```python
validate_classifier (experiment: CytoPy.data.supervised_classifier.Experiment, validation_id: str, root_population: str, threshold: float = 0.5, metrics: list = None, return_predictions: bool = True)
```
Validate the model on a FileGroup in the same Experiment as the training data (reference sample). The validation FileGroup must contain all of the Population labels in `target_populations` and these populations must be downstream of ‘root_population’. This method returns the performance of this classifier on the validation sample.

**Parameters**

- `experiment (Experiment)` –
- `validation_id (str)` –
- `root_population (str)` –
metrics (list (optional)) –
List of metrics to assess performance with. Default to:
- Balanced accuracy
- Weighted F1 score
- ROC AUC score

threshold (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class

return_predictions (bool (default=True)) – If True, vector of predicted labels returned along with dictionary of classification performance

Returns Dictionary of validation performance If return_predictions is True, will return a dictionary of the following format: {“y_pred”: predicted labels, “y_score”: confidence scores}

Return type dict or (dict, dict)

class CytoPy.data.supervised_classifier.KerasCellClassifier(*args, **values)

Bases: CytoPy.data.supervised_classifier.CellClassifier

Supervised classification of cells using a gated example. The user provides an Experiment and the name of a sample which has existing Populations; we call this the ‘reference sample’. Note, a CellClassifier should be defined for a single Experiment and should not be applied to multiple Experiments. These Populations are used as labels in training data, to train a supervised classification model. This class inherits from CellClassifier and provides functionality to use deep neural networks implemented with Keras. CytoPy uses the Keras sequential API and layers are defined and stored as Layer documents embedded within this class. Specify the name of a valid Keras layer, along with initialisation parameters, and each time the KerasCellClassifier is initialised, the Keras model will be created and compiled. Layers should be added in the order in which they should occur in the neural network.

name
Name of the CellClassifier to save to the database. Must be unique

Type str, required

layers
List of Layer objects

Type list

optimizer
Optimisation function to use (see https://keras.io/optimizers)

Type str

loss
Loss function (see https://keras.io/losses)

Type str

metrics
List of metrics to use to measure performance (see https://keras.io/metrics)

Type list

compile_kwargs
Additional keyword arguments to pass when ‘compile’ method is called

Type dict
feature
   List of markers used as input variables for classification
   
   Type  list, required

multi_class
   If True, the classification problem will be treated as a multi-class problem; that is, a single cell can belong
to multiple populations and the classifier will be trained to attribute multiple classes to a cell. This requires
that the user provides a threshold for positivity which defaults to 0.5.
   
   Type  bool (default=False)

target_populations
   List of populations to search for in the reference sample and use as target labels
   
   Type  list, required

transform
   Transformation to apply to data prior to training/prediction
   
   Type  str (optional; default="logicle")

scale
   Value of “standard” or “norm” should be provided if you wish to scale the data prior to training/prediction. Recommended for most methods except tree-based classifiers.
   
   Type  str (optional; default=None)

scale_kwargs
   Keyword arguments passed to scaling method, see CytoPy.flow.transform
   
   Type  dict

downsample
   Value of ‘uniform’, ‘density’ or ‘faithful’ should be provided if the user wishes to downsample the data
prior to training. For downsampling methods see CytoPy.flow.sampling.
   
   Type  str (optional, default=None)

downsample_kwargs
   Keyword arguments passed to sampling method, see CytoPy.flow.sampling
   
   Type  dict

class_weights
   Can be used to handle class imbalance by passing a dictionary of weights to associate to each population
class. Alternatively user can use the “auto_class_weights” method to calculate balanced weights using
the compute_class_weight function from Scikit-Learn.
   
   Type  dict (optional)

population_prefix
   Prefix added to the name of predicted populations
   
   Type  str (default=”sml”)

verbose
   Provide feedback to stdout
   
   Type  bool (default=True)

model
   Read-only attribute housing the classification model
   
   Type  object
×
Training feature space

**Type** Pandas.DataFrame

Y
Training labels

**Type** Numpy.Array

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_class_weights()</td>
<td>Compute optimal class weights using the compute_class_weights function from Scikit-Learn.</td>
</tr>
<tr>
<td>build_model()</td>
<td>Build and compile Keras model.</td>
</tr>
<tr>
<td>cascade_save(<strong>kwargs</strong>)</td>
<td>Recursively save any references and generic references on the document.</td>
</tr>
<tr>
<td>clean()</td>
<td>Hook for doing document level data cleaning before validation is run.</td>
</tr>
<tr>
<td>compare_indexes()</td>
<td>Compares the indexes defined in MongoEngine with the ones existing in the database.</td>
</tr>
<tr>
<td>create_index(keys[, background])</td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td>delete([signal_kwargs])</td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td>drop_collection()</td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td>ensure_index(key_or_list[, background])</td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td>ensure_indexes()</td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td>fit([validation_frac, ...])</td>
<td>Fit the Keras model to the associated training data.</td>
</tr>
<tr>
<td>fit_cv([cross_validator, metrics, ...])</td>
<td>Fit the model to training data using cross-validation.</td>
</tr>
<tr>
<td>fit_train_test_split([test_frac, metrics, ...])</td>
<td>Fit the model and test on holdout data using a test-train split approach.</td>
</tr>
<tr>
<td>from_json(json_data[, created])</td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td>get_text_score()</td>
<td>Get text score from text query</td>
</tr>
<tr>
<td>list_indexes()</td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td>load_training_data(experiment, reference, ...)</td>
<td>Loads training data from the given Experiment.</td>
</tr>
<tr>
<td>load_validation(experiment, validation_id, ...)</td>
<td>Load a FileGroup from the same Experiment as the training data (reference_sample) and use this as validation data to test the model.</td>
</tr>
<tr>
<td>modify([query])</td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td>plot_learning_curve([history, ax, figsize, ...])</td>
<td>This method will generate a learning curve using the History object generated from the fit method from the Keras sequential API.</td>
</tr>
</tbody>
</table>
### Table 127 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>predict</strong></td>
<td>Predict the population labels for cells in a FileGroup (specified by “sample_id”) in the same Experiment as the training data (reference sample).</td>
</tr>
<tr>
<td><strong>register_delete_rule</strong></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><strong>reload</strong></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><strong>save</strong></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><strong>select_related</strong></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><strong>switch_collection</strong></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><strong>switch_db</strong></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><strong>to_dbref</strong></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><strong>to_json</strong></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><strong>to_mongo</strong></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
<tr>
<td><strong>update</strong></td>
<td>Performs an update on the Document A convenience wrapper to update().</td>
</tr>
<tr>
<td><strong>validate</strong></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
<tr>
<td><strong>validate_classifier</strong></td>
<td>Validate the model on a FileGroup in the same Experiment as the training data (reference sample).</td>
</tr>
</tbody>
</table>

#### Classes:

- **my_metaclass**
  - alias of mongoengine.base.metaclasses.
  - TopLevelDocumentMetaclass

#### Attributes:

- **pk**
  - Get the primary key.

#### Exceptions:

- **DoesNotExist**
  - Bases: CytoPy.data.supervised_classifier.DoesNotExist
  - Args
    - with_traceback() Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

- **MultipleObjectsReturned**
  - Bases: CytoPy.data.supervised_classifier.MultipleObjectsReturned
  - Args
    - with_traceback() Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

- **auto_class_weights()**
  - Compute optimal class weights using the compute_class_weights function from Scikit-Learn. Weights are stored as a dictionary in self.class_weights.
  - Returns
Return type  None

build_model()
Build and compile Keras model. Model is then associated to self.model.

Returns
Return type  None

cascade_save(**kwargs)
Recursively save any references and generic references on the document.

clean()
Hook for doing document level data cleaning before validation is run.

Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

classmethod compare_indexes()
Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

classmethod create_index(keys, background=False, **kwargs)
Creates the given indexes if required.

Parameters

• keys – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

• background – Allows index creation in the background

delete(signal_kwargs=None, **write_concern)
Delete the Document from the database. This will only take effect if the document has been previously saved.

Parameters

• signal_kwargs – (optional) kwargs dictionary to be passed to the signal calls.

• write_concern – Extra keyword arguments are passed down which will be used as options for the resultant getLastError command. For example, save(..., w: 2, fsync: True) will wait until at least two servers have recorded the write and will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument

classmethod drop_collection()
Drops the entire collection associated with this Document type from the database.

Raises OperationError if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: OperationError exception raised if no collection available

classmethod ensure_index(key_or_list, background=False, **kwargs)
Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters

• key_or_list – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering

• background – Allows index creation in the background
classmethod ensure_indexes()
Checks the document meta data and ensures all the indexes exist.
Global defaults can be set in the meta - see guide/defining-documents

**Note:** You can disable automatic index creation by setting auto_create_index to False in the documents meta data.

`fit(validation_frac: float = 0.3, train_test_split_kwargs: dict = None, epochs: int = 100, **kwargs)`
Fit the Keras model to the associated training data. If `validation_frac` is provided, then a given proportion of the training data will be set aside and given to the `validation_data` parameter to Keras fit method of the sequential API. The validation data is created using the `train_test_split` function from Scikit-Learn; additional keyword arguments can be provided as a dictionary with `train_test_split_kwargs`.

**Parameters**

- `validation_frac` *(float (optional; default=0.3)) – Proportion of training data to set aside for validation*
- `train_test_split_kwargs` *(dict (optional)) – Additional keyword arguments for train_test_split function from Scikit-Learn*
- `epochs` *(int (default=100)) – Number of training rounds*
- `kwargs` – Additional keyword arguments passed to fit method of keras sequential API

**Returns** Keras History object

**Return type** Keras.callbacks.History

`fit_cv(cross_validator: sklearn.model_selection._split.BaseCrossValidator = None, metrics: list = None, threshold: float = 0.5, split_kwargs: dict = None, fit_kwargs: dict = None)`
Fit the model to training data using cross-validation. The model will be fitted to training data generated when “load_data” is called. If “load_data” is not previously called will raise an AssertionError.

Any Scikit-Learn cross-validator object can be used to perform cross validation, but if not given, will default to a basic K-fold cross-validation.

**Parameters**

- `cross_validator` *(BaseCrossValidator (default=KFold)) –*
- `metrics` *(list (optional)) – List of metrics to assess performance with. Default to: Balanced accuracy – Weighted F1 score – ROC AUC score*
- `threshold` *(float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class*
- `split_kwargs` *(dict (optional)) – Additional keyword arguments passed to “split” call of cross_validator*
- `fit_kwargs` *(dict (optional)) – Additional keyword arguments passed to “fit” call of model*

**Returns** List of dictionaries detailing training performance on each round List of dictionaries detailing testing performance on each round
**Return type**  List, List

```
fit_train_test_split (test_frac: float = 0.3, metrics: list = None, return_predictions: bool = True, threshold: float = 0.5, train_test_split_kwargs: dict = None, fit_kwargs: dict = None)
```

Fit the model and test on holdout data using a test-train split approach. The model will be fitted to training data generated when “load_data” is called. If “load_data” is not previously called will raise an AssertionError.

**Parameters**

- **test_frac** (float (default=0.3)) – Proportion of training data to be kept as holdout data for testing
- **metrics** (list (optional)) – List of metrics to assess performance with. Default to:
  - Balanced accuracy
  - Weighted F1 score
  - ROC AUC score
- **return_predictions** (bool (default=True)) – If True, vector of predicted labels returned along with dictionary of classification performance
- **threshold** (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class
- **train_test_split_kwargs** (dict (optional)) – Additional keyword arguments passed to call to Scikit-Learn’s train_test_split function
- **fit_kwargs** (dict (optional)) – Additional keyword arguments passed to models fit method call

**Returns**  Dictionary of training and testing performance. If return_predictions is True, will also return an array of cell class predictions.

**Return type**  dict or (dict, Numpy.Array)

**classmethod from_json (json_data, created=False)**

Converts json data to a Document instance

**Parameters**

- **json_data** (str) – The json data to load into the Document
- **created** (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).
  - If False and an ID is NOT provided, consider the document as brand new.
  - If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).
  - Defaults to False.

**get_text_score ()**

Get text score from text query
**classmethod list_indexes()**

Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

**load_training_data(experiment: CytoPy.data.experiment.Experiment, reference: str, root_population: str)**

Loads training data from the given Experiment. User must provide the name of an existing FileGroup associated to this Experiment (reference) that has the desired Populations previously created. The Populations should be downstream of a given “root_population”, which acts as the starting point of analysis.

This method populates the attributes “x” (the feature space) and “y” (the training labels). The training labels, y, will either be a 1-dimensional array, if multi_class if False, or a multi-dimensional array where each column is a population label, if multi_class is True.

**Parameters**

- **experiment (Experiment)**
- **reference (str)**
- **root_population (str)**

**Returns**

**Return type** None

**load_validation(experiment: CytoPy.data.experiment.Experiment, validation_id: str, root_population: str)**

Load a FileGroup from the same Experiment as the training data (reference_sample) and use this as validation data to test the model. The validation FileGroup must contain all of the Population labels in target_populations and these populations must be downstream of ‘root_population’.

**Parameters**

- **experiment (Experiment)**
- **validation_id (str)**
- **root_population (str)**

**Returns** Feature space and population labels for validation FileGroup

**Return type** Pandas.DataFrame, Numpy.Array

**modify(query=None, **update)**

Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

**Note:** All unsaved changes that have been made to the document are rejected if the method returns True.
my_metaclass
  alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

  Methods:
  get_auto_id_names(new_class)
    Find a name for the automatic ID field for the given new class.
  mro()
    Return a type’s method resolution order.

property pk
  Get the primary key.

plot_learning_curve
  (history: `tensorflow.python.keras.callbacks.History` = None, ax: `matplotlib.axes.Axes` = None, figsize: `tuple` = (10, 10, plot_kwargs: `dict`) = None, **fit_kwargs)
  This method will generate a learning curve using the History object generated from the fit method from the Keras sequential API.

  Parameters
  history (History (optional)) – If not given, then the ‘fit’ method will be called and use the associated training data.
  ax (Matplotlib.Axes) –
  figsize (tuple (default=(10,10)) –
  plot_kwargs (dict (optional)) – Keyword arguments passed to Pandas.DataFrame.plot method
  fit_kwargs – Keyword arguments passed to fit method if ‘history’ is not given

  Returns
  Return type Matplotlib.Axes

predict
  (experiment: `CytoPy.data.experiment.Experiment`, sample_id: `str`, root_population: `str`, threshold: `float` = 0.5, return_predictions: `bool` = True)
  Predict the population labels for cells in a FileGroup (specified by “sample_id”) in the same Experiment as the training data (reference sample). Populations will be generated with the “root_population” as their parent. The modified FileGroup with newly predicted Populations will be returned. To save the Populations, call the “save” method of the returned FileGroup.

  Parameters
  experiment (Experiment) – Must be the same Experiment that the training data was from
  sample_id (str) – Name of the FileGroup to predict populations for
  root_population (str) – Root of the analysis; will be the immediate parent of any generated Populations
  threshold (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class
  return_predictions (bool (default=True)) – If True, vector of predicted labels returned along with the modified FileGroup object

  Returns
  Modified FileGroup with newly predicted Populations If return_predictions is True, will return a dictionary of the following format: {“y_pred”: predicted labels, “y_score”: confidence scores}

  Return type FileGroup or (FileGroup, dict)
classmethod register_delete_rule(document_cls, field_name, rule)
This method registers the delete rules to apply when removing this object.

reload(*fields, **kwargs)
Reloads all attributes from the database.

Parameters
- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.
Changed in version 0.6: Now chainable
Changed in version 0.9: Can provide specific fields to reload

save(force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs)
Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

Parameters
- **force_insert** – only try to create a new document, don’t allow updates of existing documents.
- **validate** – validates the document; set to False to skip.
- **clean** – call the document clean method, requires validate to be True.
- **write_concern** – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, save(..., write_concern={w: 2, fsync: True}, ...) will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies cascade=True.
- **_refs** – A list of processed references used in cascading saves
- **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises OperationError if the conditions are not satisfied
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta[‘cascade’] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: OperationError exception raised if save_condition fails.
Changed in version 0.10.1: class: save_condition failure now raises a `SaveConditionError`

Changed in version 0.10.7: Add signal_kwargs argument

```python
select_related(max_depth=1)
```
Handles dereferencing of `DBRef` objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

```python
switch_collection(collection_name, keep_created=True)
```
Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

- `collection_name` (str) – The database alias to use for saving the document
- `keep_created` (bool) – keep self._created value after switching collection, else is reset to True

**See also:**

Use `switch_db` if you need to read from another database

```python
switch_db(db_alias, keep_created=True)
```
Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**Parameters**

- `db_alias` (str) – The database alias to use for saving the document
- `keep_created` (bool) – keep self._created value after switching db, else is reset to True

**See also:**

Use `switch_collection` if you need to read from another collection

```python
to_dbref()
```
Returns an instance of `DBRef` useful in `__raw__` queries.

```python
to_json(*args, **kwargs)
```
Convert this document to JSON.

- **Parameters** `use_db_field` – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

```python
to_mongo(*args, **kwargs)
```
Return as SON data ready for use with MongoDB.
update(**kwargs)
Performs an update on the Document. A convenience wrapper to update().
Raises OperationError if called on an object that has not yet been saved.

validate(clean=True)
Ensure that all fields’ values are valid and that required fields are present.
Raises ValidationError if any of the fields’ values are found to be invalid.

validate_classifier(experiment: CytoPy.data.experiment.Experiment, validation_id: str, root_population: str, threshold: float = 0.5, metrics: list = None, return_predictions: bool = True)
Validate the model on a FileGroup in the same Experiment as the training data (reference sample). The validation FileGroup must contain all of the Population labels in target_populations and these populations must be downstream of ‘root_population’. This method returns the performance of this classifier on the validation sample.

Parameters
• experiment (Experiment) –
• validation_id (str) –
• root_population (str) –
• metrics (list (optional)) –
  List of metrics to assess performance with. Default to:
  – Balanced accuracy
  – Weighted F1 score
  – ROC AUC score
• threshold (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class
• return_predictions (bool (default=True)) – If True, vector of predicted labels returned along with dictionary of classification performance

Returns Dictionary of validation performance If return_predictions is True, will return a dictionary of the following format: {“y_pred”: predicted labels, “y_score”: confidence scores}

Return type dict or (dict, dict)

class CytoPy.data.supervised_classifier.Layer(*args, **kwargs)
Bases: mongoengine.document.EmbeddedDocument

Neural networks consist of layers and in the Keras framework these can be represented by individual objects. The Layer document is embedded in the KerasCellClassifier to define the types of layers to use when initialising and compiling the Keras model.

klass
Name of the keras class to use; see Keras layers API for valid class names

Type str

kwargs
Parameters to use when initialising Keras layer

Type dict
CytoPy, Release 1.0.0

Methods:

- **clean()**
  Hook for doing document level data cleaning before validation is run.

- **from_json(json_data[, created])**
  Converts json data to a Document instance

- **get_text_score()**
  Get text score from text query

- **to_json(*args, **kwargs)**
  Convert this document to JSON.

- **to_mongo(*args, **kwargs)**
  Return as SON data ready for use with MongoDB.

- **validate([clean])**
  Ensure that all fields’ values are valid and that required fields are present.

Classes:

- **my_metaclass**
  alias of mongoengine.base.metaclasses.DocumentMetaclass

  *Methods:*

  - **mro()**
    Return a type’s method resolution order.

  - **to_json(*args, **kwargs)**
    Convert this document to JSON.

    **Parameters**
    - **use_db_field** — Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

    - **to_mongo(*args, **kwargs)**
Return as SON data ready for use with MongoDB.

**validate** *(clean=True)*

Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.

**class** `CytoPy.data.supervised_classifier.SklearnCellClassifier(*args, **values)`

**Bases:** `CytoPy.data.supervised_classifier.CellClassifier`

Supervised classification of cells using a gated example. The user provides an Experiment and the name of a sample which has existing Populations; we call this the ‘reference sample’. Note, a CellClassifier should be defined for a single Experiment and should not be applied to multiple Experiments. These Populations are used as labels in training data, to train a supervised classification model. This class inherits from CellClassifier and provides functionality to use any Scikit-Learn classifier from the modules discriminant_analysis, neighbors, ensemble, or svm. This class also supports the use of XGBClассifier from XGBoost. If you have another classifier that follows the Scikit-Learn template and you would like to use this with CellClassifier, please raise an issue on GitHub or contact us at burtonrj@cardiff.ac.uk.

**name**

Name of the CellClassifier to save to the database. Must be unique

**Type** str, required

**klass**

Name of the Scikit-Learn (or Scikit-Learn ‘like’) class to use for classification

**Type** str, required

**params**

Parameters used when initialising the model

**Type** dict, optional

**feature**

List of markers used as input variables for classification

**Type** list, required

**multi_class**

If True, the classification problem will be treated as a multi-class problem; that is, a single cell can belong to multiple populations and the classifier will be trained to attribute multiple classes to a cell. This requires that the user provides a threshold for positivity which defaults to 0.5.

**Type** bool (default=False)

**target_populations**

List of populations to search for in the reference sample and use as target labels

**Type** list, required

**transform**

Transformation to apply to data prior to training/prediction

**Type** str (optional; default=“logicle”)
Type dict
downsample
Value of ‘uniform’, ‘density’ or ‘faithful’ should be provided if the user wishes to downsample the data prior to training. For downsampling methods see CytoPy.flow.sampling.

Type str (optional, default=None)
downsample_kwargs
Keyword arguments passed to sampling method, see CytoPy.flow.sampling

Type dict
class_weights
Can be used to handle class imbalance by passing a dictionary of weights to associate to each population class. Alternatively user can use the “auto_class_weights” method to calculate balanced weights using the compute_class_weight function from Scikit-Learn.

Type dict (optional)
population_prefix
Prefix added to the name of predicted populations

Type str (default="sml")
verbose
Provide feedback to stdout

Type bool (default=True)
model
Read-only attribute housing the classification model

Type object
x
Training feature space

Type Pandas.DataFrame
Y
Training labels

Type Numpy.Array

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Methods:

auto_class_weights() Compute optimal class weights using the compute_class_weights function from Scikit-Learn.

build_model() Call prior to fit or predict.

cascade_save(**kwargs) Recursively save any references and generic references on the document.

clean() Hook for doing document level data cleaning before validation is run.

compare_indexes() Compares the indexes defined in MongoEngine with the ones existing in the database.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create_index(keys[, background])</code></td>
<td>Creates the given indexes if required.</td>
</tr>
<tr>
<td><code>delete([signal_kwargs])</code></td>
<td>Delete the Document from the database.</td>
</tr>
<tr>
<td><code>drop_collection()</code></td>
<td>Drops the entire collection associated with this Document type from the database.</td>
</tr>
<tr>
<td><code>ensure_index(key_or_list[, background])</code></td>
<td>Ensure that the given indexes are in place.</td>
</tr>
<tr>
<td><code>ensure_indexes()</code></td>
<td>Checks the document meta data and ensures all the indexes exist.</td>
</tr>
<tr>
<td><code>fit(**kwargs)</code></td>
<td>Fits the model to loaded training data.</td>
</tr>
<tr>
<td><code>fit_cv([cross_validator, metrics, ...])</code></td>
<td>Fit the model to training data using cross-validation.</td>
</tr>
<tr>
<td><code>fit_train_test_split([test_frac, metrics, ...])</code></td>
<td>Fit the model and test on holdout data using a test-train split approach.</td>
</tr>
<tr>
<td><code>from_json(json_data[, created])</code></td>
<td>Converts json data to a Document instance</td>
</tr>
<tr>
<td><code>get_text_score()</code></td>
<td>Get text score from text query</td>
</tr>
<tr>
<td><code>hyperparameter_tuning(param_grid[, method])</code></td>
<td>Perform hyperparameter tuning using either exhaustive grid search or randomised grid search.</td>
</tr>
<tr>
<td><code>list_indexes()</code></td>
<td>Lists all of the indexes that should be created for given collection.</td>
</tr>
<tr>
<td><code>load_model(path, **kwargs)</code></td>
<td>Load a pickled model from disk.</td>
</tr>
<tr>
<td><code>load_training_data(experiment, reference, ...)</code></td>
<td>Loads training data from the given Experiment.</td>
</tr>
<tr>
<td><code>load_validation(experiment, validation_id, ...)</code></td>
<td>Load a FileGroup from the same Experiment as the training data (reference_sample) and use this as validation data to test the model.</td>
</tr>
<tr>
<td><code>modify([query])</code></td>
<td>Perform an atomic update of the document in the database and reload the document object using updated version.</td>
</tr>
<tr>
<td><code>plot_confusion_matrix([cmap, figsize, x, y])</code></td>
<td>Wraps CytoPy.flow.supervised.confusion_matrix_plots (see for more details).</td>
</tr>
<tr>
<td><code>plot_learning_curve([experiment, ...])</code></td>
<td>This method will generate a learning curve using the Scikit-Learn utility function sklearn.model_selection.learning_curve.</td>
</tr>
<tr>
<td><code>predict(experiment, sample_id, root_population)</code></td>
<td>Predict the population labels for cells in a FileGroup (specified by “sample_id”) in the same Experiment as the training data (reference sample).</td>
</tr>
<tr>
<td><code>register_delete_rule(document_cls, ...)</code></td>
<td>This method registers the delete rules to apply when removing this object.</td>
</tr>
<tr>
<td><code>reload(*fields, **kwargs)</code></td>
<td>Reloads all attributes from the database.</td>
</tr>
<tr>
<td><code>save([force_insert, validate, clean, ...])</code></td>
<td>Save the Document to the database.</td>
</tr>
<tr>
<td><code>save_model(path, **kwargs)</code></td>
<td>Pickle the associated model and save to disk.</td>
</tr>
<tr>
<td><code>select_related([max_depth])</code></td>
<td>Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.</td>
</tr>
<tr>
<td><code>switch_collection(collection_name[...])</code></td>
<td>Temporarily switch the collection for a document instance.</td>
</tr>
<tr>
<td><code>switch_db(db_alias[, keep_created])</code></td>
<td>Temporarily switch the database for a document instance.</td>
</tr>
<tr>
<td><code>to_dbref()</code></td>
<td>Returns an instance of DBRef useful in <strong>raw</strong> queries.</td>
</tr>
<tr>
<td><code>to_json(*args, **kwargs)</code></td>
<td>Convert this document to JSON.</td>
</tr>
<tr>
<td><code>to_mongo(*args, **kwargs)</code></td>
<td>Return as SON data ready for use with MongoDB.</td>
</tr>
</tbody>
</table>
Table 135 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update(**kwargs)</code></td>
<td>Performs an update on the Document as a convenience wrapper to <code>update()</code></td>
</tr>
<tr>
<td><code>validate([clean])</code></td>
<td>Ensure that all fields’ values are valid and that required fields are present.</td>
</tr>
<tr>
<td><code>validate_classifier(experiment, ...[, ...])</code></td>
<td>Validate the model on a FileGroup in the same Experiment as the training data (reference sample).</td>
</tr>
</tbody>
</table>

Classes:

- `my_metaclass`  
  alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

Attributes:

- `pk`  
  Get the primary key.

Exception `DoesNotExist`

Bases: `CytoPy.data.supervised_classifier.DoesNotExist`

- `args`  
  `with_traceback()`  
  Exception.with_traceback(tb) – set self._traceback__ to tb and return self.

Exception `MultipleObjectsReturned`

Bases: `CytoPy.data.supervised_classifier.MultipleObjectsReturned`

- `args`  
  `with_traceback()`  
  Exception.with_traceback(tb) – set self._traceback__ to tb and return self.

Method `auto_class_weights()`

Compute optimal class weights using the compute_class_weights function from Scikit-Learn. Weights are stored as a dictionary in self.class_weights.

- **Returns**  
  None

Method `build_model()`

Call prior to fit or predict. Initiates model and associates to self.model.

- **Returns**  
  None

Method `cascade_save(**kwargs)`

Recursively save any references and generic references on the document.

Method `clean()`

Hook for doing document level data cleaning before validation is run.

Any ValidationException raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

Method `compare_indexes()`

Compares the indexes defined in MongoDB with the ones existing in the database. Returns any missing/extra indexes.
**classmethod create_index** *(keys, background=False, **kwargs)*

Creates the given indexes if required.

**Parameters**

- **keys** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background

**delete** *(signal_kwargs=None, **write_concern)*

Delete the Document from the database. This will only take effect if the document has been previously saved.

**Parameters**

- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.
- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant getLastError command. For example, save(..., w: 2, fsync: True) will wait until at least two servers have recorded the write and will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument

**classmethod drop_collection** *

Drops the entire collection associated with this Document type from the database.

Raises OperationError if the document has no collection set (i.e. if it is abstract)

Changed in version 0.10.7: OperationError exception raised if no collection available

**classmethod ensure_index** *(key_or_list, background=False, **kwargs)*

Ensure that the given indexes are in place. Deprecated in favour of create_index.

**Parameters**

- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a + or a - to determine the index ordering
- **background** – Allows index creation in the background

**classmethod ensure_indexes** *

Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see guide/defining-documents

---

**Note:** You can disable automatic index creation by setting auto_create_index to False in the documents meta data

**fit** *(**kwargs)*

Fits the model to loaded training data. If “load_data” is not called prior to this method, will raise AssertionError.

**Parameters** **kwargs** – Additional keyword arguments pass to “fit”

**Returns** None

**fit_cv** *(cross_validator: sklearn.model_selection._split.BaseCrossValidator = None, metrics: list = None, threshold: float = 0.5, split_kwargs: dict = None, fit_kwargs: dict = None)*

Fit the model to training data using cross-validation. The model will be fitted to training data generated when “load_data” is called. If “load_data” is not previously called will raise an AssertionError.
Any Scikit-Learn cross-validator object can be used to perform cross validation, but if not given, will default to a basic K-fold cross-validation.

**Parameters**

- `cross_validator` ([BaseCrossValidator](default=KFold))
- `metrics` (list (optional))

**List of metrics to assess performance with. Default to:**

- Balanced accuracy
- Weighted F1 score
- ROC AUC score

- `threshold` (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class

- `split_kwars` (dict (optional)) – Additional keyword arguments passed to “split” call of cross_validator

- `fit_kwars` (dict (optional)) – Additional keyword arguments passed to “fit” call of model

**Returns** List of dictionaries detailing training performance on each round List of dictionaries detailing testing performance on each round

**Return type** List, List

fit_train_test_split (test_frac: float = 0.3, metrics: list = None, return_predictions: bool = True, threshold: float = 0.5, train_test_split_kwars: dict = None, fit_kwars: dict = None)

Fit the model and test on holdout data using a test-train split approach. The model will be fitted to training data generated when “load_data” is called. If “load_data” is not previously called will raise an AssertionError.

**Parameters**

- `test_frac` (float (default=0.3)) – Proportion of training data to be kept as holdout data for testing
- `metrics` (list (optional))

**List of metrics to assess performance with. Default to:**

- Balanced accuracy
- Weighted F1 score
- ROC AUC score

- `return_predictions` (bool (default=True)) – If True, vector of predicted labels returned along with dictionary of classification performance

- `threshold` (float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class

- `train_test_split_kwars` (dict (optional)) – Additional keyword arguments passed to call to Scikit-Learn’s train_test_split function

- `fit_kwars` (dict (optional)) – Additional keyword arguments passed to models fit method call

**Returns** Dictionary of training and testing performance. If return_predictions is True, will also return an array of cell class predictions.
Return type dict or (dict, Numpy.Array)

classmethod from_json(json_data, created=False)
Converts json data to a Document instance

Parameters

• json_data (str) – The json data to load into the Document

• created (bool) – Boolean defining whether to consider the newly instantiated document as brand new or as persisted already: * If True, consider the document as brand new, no matter what data it’s loaded with (i.e. even if an ID is loaded).

– If False and an ID is NOT provided, consider the document as brand new.

– If False and an ID is provided, assume that the object has already been persisted (this has an impact on the subsequent call to .save()).

– Defaults to False.

get_text_score()
Get text score from text query

hyperparameter_tuning(param_grid: dict, method: str = 'grid_search', **kwargs)
Perform hyperparameter tuning using either exhaustive grid search or randomised grid search.

Parameters

• param_grid (dict) – Search space

• method (str (default="grid_search")) – Should either be “grid_search” or “random”

• kwargs – Keyword arguments passed to grid search method

Returns

Return type GridSearchCV or RandomizedSearchCV

classmethod list_indexes()
Lists all of the indexes that should be created for given collection. It includes all the indexes from super-
and sub-classes.

load_model(path: str, **kwargs)
Load a pickled model from disk. WARNING: be aware of continuity issues. Compatibility with new re-
leases of Scikit-Learn and CytoPy are not guaranteed. The loaded model must correspond to the expected method for this CellClassifier.

Parameters

• path (str) – Where to save on disk

• kwargs – Additional keyword arguments passed to pickle.dump call

Returns

Return type None

load_training_data(experiment: CytoPy.data.experiment_experiment, reference: str, root_population: str)
Loads training data from the given Experiment. User must provide the name of an existing FileGroup associated to this Experiment (reference) that has the desired Populations previously created. The Popu-
lations should be downstream of a given “root_population”, which acts as the starting point of analysis.
This method populates the attributes “x” (the feature space) and “y” (the training labels). The training labels, y, will either be a 1-dimensional array, if multi_class if False, or a multi-dimensional array where each column is a population label, if multi_class is True.

**Parameters**

- experiment (Experiment)
- reference (str)
- root_population (str)

**Returns**

**Return type** None

### load_validation

`load_validation(experiment: CytoPy.data.experiment.Experiment, validation_id: str, root_population: str)`  
Load a FileGroup from the same Experiment as the training data (reference_sample) and use this as validation data to test the model. The validation FileGroup must contain all of the Population labels in target_populations and these populations must be downstream of ‘root_population’.

**Parameters**

- experiment (Experiment)
- validation_id (str)
- root_population (str)

**Returns** Feature space and population labels for validation FileGroup

**Return type** Pandas.DataFrame, Numpy.Array

### modify

`modify(query=None, **update)`  
Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn’t match the query.

**Note:** All unsaved changes that have been made to the document are rejected if the method returns True.

**Parameters**

- query – the update will be performed only if the document in the database matches the query
- update – Django-style update keyword arguments

### my_metaclass

alias of `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

**Methods:**

- `get_auto_id_names(new_class)` Find a name for the automatic ID field for the given new class.
- `mro()` Return a type’s method resolution order.

**property pk**  
Get the primary key.
plot_confusion_matrix(cmap: str = None, figsize: tuple = 10, 5, x: pandas.core.frame.DataFrame = None, y: numpy.ndarray = None, **kwargs)
Wraps CytoPy.flow.supervised.confusion_matrix_plots (see for more details). Given some feature space and target labels, use the model to generate a confusion matrix heatmap. If x and y are not provided, will use associated training data.

Parameters

- **cmap** *(str (optional)) – Colour scheme*
- **figsize** *(tuple (default=(10, 5)) – Figure size*
- **x** *(Pandas.DataFrame (optional)) – Feature space. If not given, will use associated training data. To use a validation dataset, use the ‘load_validation’ method to get relevant data.*
- **y** *(Numpy.Array (optional)) – Target labels. If not given, will use associated training data. To use a validation dataset, use the ‘load_validation’ method to get relevant data.*
- **kwargs** – Additional keyword arguments passed to CytoPy.flow.supervised.confusion_matrix_plots

plot_learning_curve(experiment: CytoPy.data.experiment.Experiment = None, validation_id: str = None, root_population: str = None, ax: matplotlib.axes._axes.Axes = None, x_label: str = 'Training examples', y_label: str = 'Score', train_sizes: numpy.array = None, **kwargs)
This method will generate a learning curve using the Scikit-Learn utility function sklearn.model_selection.learning_curve. Either use the associated training data or a validation FileGroup by providing the Experiment object and the ID for the validation sample (validation_id). This validation sample should contain the same populations as the training data, which must be downstream of the ‘root_population’.

Parameters

- **experiment** *(Experiment (optional)) – If provided, should be the same Experiment training data was derived from*
- **validation_id** *(str (optional)) – Name of the sample to use for validation*
- **root_population** *(str (optional)) – If not given, will use the same root_population as training data*
- **ax** *(Matplotlib.Axes (optional)) – Axes object to use to draw plot*
- **x_label** *(str (default="Training examples")) – X-axis labels*
- **y_label** *(str (default="Score")) – Y-axis labels*
- **train_sizes** *(Numpy.Array (optional)) – Defaults to linear range between 0.1 and 1.0, with 10 steps*
- **kwargs** – Additional keyword arguments passed to sklearn.model_selection.learning_curve

Returns

Return type Matplotlib.Axes

predict(experiment: CytoPy.data.experiment.Experiment, sample_id: str, root_population: str, threshold: float = 0.5, return_predictions: bool = True)
Predict the population labels for cells in a FileGroup (specified by “sample_id”) in the same Experiment as the training data (reference sample). Populations will be generated with the “root_population” as
their parent. The modified FileGroup with newly predicted Populations will be returned. To save the Populations, call the “save” method of the returned FileGroup.

**Parameters**

- `experiment` ([Experiment]): Must be the same Experiment that the training data was from
- `sample_id` (str): Name of the FileGroup to predict populations for
- `root_population` (str): Root of the analysis; will be the immediate parent of any generated Populations
- `threshold` (float, default=0.5): If multi_class is True, this will be used to determine positive association to a class
- `return_predictions` (bool, default=True): If True, vector of predicted labels returned along with the modified FileGroup object

**Returns** Modified FileGroup with newly predicted Populations If return_predictions is True, will return a dictionary of the following format: `{“y_pred”: predicted labels, “y_score”: confidence scores}`

**Return type** `FileGroup` or `FileGroup, dict`

**classmethod register_delete_rule** (`document_cls, field_name, rule`)

This method registers the delete rules to apply when removing this object.

**reload** (*fields, **kwargs*)

Reloads all attributes from the database.

**Parameters**

- `fields` – (optional) args list of fields to reload
- `max_depth` – (optional) depth of dereferencing to follow

New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

**save** (`force_insert=False, validate=True, clean=True, write_concern=None, cascade=None, cascade_kwargs=None, _refs=None, save_condition=None, signal_kwargs=None, **kwargs`) Save the Document to the database. If the document already exists, it will be updated, otherwise it will be created. Returns the saved object instance.

**Parameters**

- `force_insert` – only try to create a new document, don’t allow updates of existing documents.
- `validate` – validates the document; set to False to skip.
- `clean` – call the document clean method, requires validate to be True.
- `write_concern` – Extra keyword arguments are passed down to save() OR insert() which will be used as options for the resultant getLastError command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- `cascade` – Sets the flag for cascading saves. You can set a default by setting “cascade” in the document __meta__
• `cascade_kwargs` – (optional) kwargs dictionary to be passed throw to cascading saves. Implies `cascade=True`.

• `_refs` – A list of processed references used in cascading saves

• `save_condition` – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises `OperationError` if the conditions are not satisfied

• `signal_kwargs` – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any DBRef objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: `OperationError` exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a `SaveConditionError`

Changed in version 0.10.7: Add signal_kwargs argument

`save_model` *(path: str, **kwargs)*

Pickle the associated model and save to disk. WARNING: be aware of continuity issues. Compatibility with new releases of Scikit-Learn and CytoPy are not guaranteed.

**Parameters**

• `path` *(str)* – Where to save on disk

• `kwargs` – Additional keyword arguments passed to pickle.dump call

**Returns**

Return type None

`select_related` *(max_depth=1)*

Handles dereferencing of DBRef objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

`sytch_collection` *(collection_name, keep_created=True)*

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**Parameters**

• `collection_name` *(str)* – The database alias to use for saving the document

• `keep_created` *(bool)* – keep self._created value after switching collection, else is reset to True
See also:

Use `switch_db` if you need to read from another database

```python
switch_db(db_alias, keep_created=True)
```

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling `save()`:

```python
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters

- `db_alias (str)` – The database alias to use for saving the document
- `keep_created (bool)` – keep self.created value after switching db, else is reset to True

See also:

Use `switch_collection` if you need to read from another collection

```python
to_dbref()
```

Returns an instance of `DBRef` useful in `__raw__` queries.

```python
to_json(*args, **kwargs)
```

Convert this document to JSON.

Parameters `use_db_field` – Serialize field names as they appear in MongoDB (as opposed to attribute names on this document). Defaults to True.

```python
to_mongo(*args, **kwargs)
```

Return as SON data ready for use with MongoDB.

```python
update(**kwargs)
```


Raises `OperationError` if called on an object that has not yet been saved.

```python
validate(clean=True)
```

Ensure that all fields’ values are valid and that required fields are present.

Raises `ValidationError` if any of the fields’ values are found to be invalid.

```python
validate_classifier(experiment: CytoPy.data.experiment.Experiment, validation_id: str, root_population: str, threshold: float = 0.5, metrics: list = None, return_predictions: bool = True)
```

Validate the model on a FileGroup in the same Experiment as the training data (reference sample). The validation FileGroup must contain all of the Population labels in target_populations and these populations must be downstream of ‘root_population’. This method returns the performance of this classifier on the validation sample.

Parameters

- `experiment (Experiment)` –
- `validation_id (str)` –
- `root_population (str)` –
- `metrics (list (optional))` –

List of metrics to assess performance with. Default to:
- Balanced accuracy
- Weighted F1 score
- ROC AUC score

- **threshold** *(float (default=0.5)) – If multi_class is True, this will be used to determine positive association to a class*

- **return_predictions** *(bool (default=True)) – If True, vector of predicted labels returned along with dictionary of classification performance*

Returns Dictionary of validation performance If return_predictions is True, will return a dictionary of the following format: {“y_pred”: predicted labels, “y_score”: confidence scores}

Return type dict or (dict, dict)

### 8.13 CytoPy.flow.clustering

Here you will find CytoPy’s implementation of the FlowSOM algorithm, which relies on the MiniSOM library for self-organising maps. The work was adapted from [https://github.com/Hatchin/FlowSOM](https://github.com/Hatchin/FlowSOM) for integration with CytoPy and the database architecture.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsensusCluster(cluster,...)</td>
<td>Implementation of Consensus clustering, following the paper <a href="https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf">https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf</a> Code is adapted from <a href="https://github.com/ZigaSajovic/Consensus_Clustering">https://github.com/ZigaSajovic/Consensus_Clustering</a></td>
</tr>
<tr>
<td>FlowSOM(data, features,...)</td>
<td>Python implementation of FlowSOM algorithm, adapted from <a href="https://github.com/Hatchin/FlowSOM">https://github.com/Hatchin/FlowSOM</a> This class implements MiniSOM in an almost identical manner to the work by Hatchin, but removed all of the data handling steps seen in Hatchin’s original library, since these are handled by the infrastructure in CytoPy.</td>
</tr>
<tr>
<td>MinMaxScaler(**kwargs)</td>
<td>Transform features by scaling each feature to a given range.</td>
</tr>
</tbody>
</table>

**Functions:**
progress_bar(x[, verbose])  Generate a progress bar using the tqdm library.

vprint(verbos3e)  Utility function for optional printing.

warn(message[, category, stacklevel, source])  Issue a warning, or maybe ignore it or raise an exception.

class CytoPy.flow.clustering.flowsom.FlowSOM

    FlowSOM(data: pandas.core.frame.DataFrame, features: list, neighborhood_function: str = 'gaussian', normalisation: bool = False, verbose: bool = True)

Bases: object

Python implementation of FlowSOM algorithm, adapted from https://github.com/Hatchin/FlowSOM This class implements MiniSOM in an almost identical manner to the work by Hatchin, but removed all the of the data handling steps seen in Hatchin’s original library, since these are handled by the infrastructure in CytoPy. The FlowSOM algorithm is implemented here in such a way that it requires only a Pandas DataFrame, like that typically produced when retrieving data from the CytoPy database, and gives access to methods of clustering and meta-clustering. In addition to Hatchin’s work, the CytoPy implementation has improved error handling and integrates better with the CytoPy workflow.

Parameters

- **data**(Pandas.DataFrame) – training data
- **features**(List) – list of columns to include
- **neighborhood_function**(str) – name of distribution for initialising weights
- **normalisation**(bool) – if True, min max normalisation applied prior to computation

Methods:

- **meta_cluster**(cluster_class[, min_n, max_n, ...])  Perform meta-clustering. Implementation of Consensus clustering, following the paper https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf :param cluster_class: clustering object (must follow Sklearn standard; needs fit_predict method called with parameter n_clusters) :param min_n: the min proposed number of clusters :param max_n: the max proposed number of clusters :param iter_n: the iteration times for each number of clusters :param resample_proportion: within (0, 1), the proportion of re-sampling when computing clustering :type resample_proportion: float, (Default value = 0.5).

- **predict**()  Predict the cluster allocation for each cell in the associated dataset.
Table 141 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>train([som_dim, sigma, learning_rate, ...])</code></td>
<td>Train self-organising map.</td>
<td><code>som_dim</code>: dimensions of SOM embedding (number of nodes), <code>sigma</code>: the radius of the different neighbors in the SOM, <code>learning_rate</code>: alters the rate at which weights are updated, <code>batch_size</code>: size of batches used in training (alters number of total iterations), <code>seed</code>: random seed, <code>weight_init</code>: how to initialise weights: either ‘random’ or ‘pca’ (Initializes the weights to span the first two principal components)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>meta_cluster(cluster_class: callable, min_n: int = 5, max_n: int = 50, iter_n: int = 10, resample_proportion: float = 0.5)</code></td>
<td>Perform meta-clustering. Implementation of Consensus clustering, following the paper <a href="https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf">https://link.springer.com/content/pdf/10.1023%2FA%3A1023949509487.pdf</a></td>
<td><code>cluster_class</code>: clustering object (must follow Sklearn standard; needs fit_predict method called with parameter n_clusters)</td>
</tr>
</tbody>
</table>

**Parameters**

- `-min_n (int)`: the min proposed number of clusters
- `-max_n (int)`: the max proposed number of clusters
- `-iter_n (int)`: the iteration times for each number of clusters
- `-resample_proportion (float, (Default value = 0.5))`: within (0, 1), the proportion of re-sampling when computing clustering

**Returns**

- **Return type**: None

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>predict()</code></td>
<td>Predict the cluster allocation for each cell in the associated dataset. (Requires that train and meta_cluster have been called previously)</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

- **Return type**: Numpy.array

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>train(som_dim: tuple = 50, 50, sigma: float = 1.0, learning_rate: float = 0.5, batch_size: int = 500, seed: int = 42, weight_init: str = 'random')</code></td>
<td>Train self-organising map.</td>
<td><code>som_dim</code>: dimensions of SOM embedding (number of nodes), <code>sigma</code>: the radius of the different neighbors in the SOM, <code>learning_rate</code>: alters the rate at which weights are updated, <code>batch_size</code>: size of batches used in training (alters number of total iterations), <code>seed</code>: random seed, <code>weight_init</code>: how to initialise weights: either ‘random’ or ‘pca’ (Initializes the weights to span the first two principal components)</td>
</tr>
</tbody>
</table>

**Returns**
This module houses an adaption of the consensus clustering method first described in [1]. Python implementation is adapted from Žiga Sajovic with the original source code found here: [2]. Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

```python
ConsensusCluster(cluster, ...) Implementation of Consensus clustering, following the paper [1]. Code is adapted from [2].
```

```python
combinations combinations(iterable, r) -> combinations object
```

Functions:

```python
progress_bar(x[, verbose]) Generate a progress bar using the tqdm library.
```

```python
class CytoPy.flow.clustering.consensus.ConsensusCluster(cluster: callable, smallest_cluster_n: int, largest_cluster_n: int, n_resamples: int, resample_proportion: float = 0.5, verbose: bool = True)
```

Bases: object

Implementation of Consensus clustering, following the paper [1]. Code is adapted from [2].

Parameters

- `cluster` – clustering class (must be an instance of clustering algorithm with fit/fit_predict method (e.g. scikit-learn)
- `L` – smallest number of clusters to try
- `K` – biggest number of clusters to try
- `H` – number of resamplings for each cluster number
- `resample_proportion` – percentage to sample
- `Mk` – consensus matrices for each k (OTE: every consensus matrix is retained)
• **Ak** – area under CDF for each number of clusters (see paper)

• **deltaK** – changes in area under CDF (see paper)

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fit</strong>(data)</td>
<td>Fits a consensus matrix for each number of clusters: param data: numpy array</td>
</tr>
<tr>
<td><strong>predict</strong></td>
<td>Predicts on the consensus matrix, for best found cluster number: returns:</td>
</tr>
<tr>
<td><strong>predict_data</strong>(data)</td>
<td>Predicts on the data, for best found cluster number: param data: data to make predictions: type data: np.array</td>
</tr>
</tbody>
</table>

**fit**(data: numpy.array) → None

Fits a consensus matrix for each number of clusters: param data: numpy array to fit clustering algorithm
too: type data: Numpy.array

Returns

Return type None

**predict**() Predicts on the consensus matrix, for best found cluster number: returns: type: Clustering predictions

**predict_data**(data) Predicts on the data, for best found cluster number: param data: data to make predictions: type data: np.array:

Returns

Return type Clustering predictions

### 8.14 CytoPy.flow.variance

Before we perform any detailed analysis and/or classification of our single cell data, it is valuable to assess the intersample variation that could be arising from biological differences, but also technical variation introduced by batch effects. This module contains multiple functions for visualising univariate and multivariate differences between FileGroups in the same experiment. Additionally we have the SimilarityMatrix class, that generates a heatmap of pairwise statistical distance’s, allow us to group similar FileGroups.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:
Experiment(*args, **kwargs)  Container for Cytometry experiment.

FFTKDE([*kernel, bw, norm])      This class implements a convolution (FFT) based computation of a KDE.

FileGroup(*args, **values)       Document representation of a file group; a selection of related fcs files (e.g.

GridSearchCV(**kwargs)          Exhaustive search over specified parameter values for an estimator.


OrderedDict                   Dictionary that remembers insertion order

SimilarityMatrix(data, reference[, verbose, ...])      Class for assessing the degree of variation observed in a single experiment.

defaultdict                       defaultdict(default_factory[, ...])  -> dict with default factory

Functions:

  bw_optimisation(data, features[, kernel, ...]) Using GridSearchCV and the Scikit-Learn implementation of KDE, find the optimal bandwidth for the given data using grid search cross-validation

  calculate_ref_sample(data[, verbose]) This is performed as described in Li et al paper (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5860171/) on DeepCyTOF: for every 2 samples i, j compute the euclidean norm of the difference between their covariance matrices and then select the sample with the smallest average distance to all other samples.

  density_dependent_downsampling(data[, ...]) Perform density dependent down-sampling to remove risk of under-sampling rare populations; adapted from SPADE*

  dim_reduction_grid(data, reference, features) Generate a grid of embeddings using a valid dimensionality reduction technique, in each plot a reference sample is shown in blue and a comparison sample in red.

  dimensionality_reduction(data, features,...) Perform dimensionality reduction using either UMAP, PCA, tSNE, or PHATE.

  faithful_downsampling(data, h) An implementation of faithful downsampling as described in: Zare H, Shooshtari P, Gupta A, Brinkman R.

  generate_groups(linkage_matrix, sample_ids, ...) Given the output of SimilarityMatrix (that is the linkage matrix and ordered list of sample IDs) and a desired number of groups, return a Pandas DataFrame of sample IDs and assigned group ID, generated by cutting the linkage matrix in such a way that the desired number of groups are generated.

  jsd(p, q[, base]) Compute the Jensen-Shannon distance (metric) between two 1-D probability arrays.

  kld(pk[, qk, base, axis]) Calculate the entropy of a distribution for given probability values.

  load_and_sample(experiment, population,...) Load sample data from experiment and return as a dictionary of Pandas DataFrames.

  marker_variance(data, reference[, ...]) Compare the kernel density estimates for each marker in the associated experiment for the given comparison samples.

continues on next page
Table 146 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>progress_bar(x[, verbose])</td>
<td>Generate a progress bar using the tqdm library.</td>
</tr>
<tr>
<td>scale_data(data[, method])</td>
<td>Given a dictionary of events data as generated by load_and_sample, scale the data using a valid scale method (see CytoPy.flow.transforms.scaler)</td>
</tr>
<tr>
<td>scaler(data, scale_method[, return_scaler])</td>
<td>Wrapper for Sklearn transformation methods</td>
</tr>
<tr>
<td>uniform_downsampling(data, sample_size, **kwargs)</td>
<td>Uniform downsampling.</td>
</tr>
<tr>
<td>vprint(verbose)</td>
<td>Utility function for optional printing.</td>
</tr>
<tr>
<td>warn(message[, category, stacklevel, source])</td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

```python
class CytoPy.flow.variance.SimilarityMatrix(

Bases: object

Class for assessing the degree of variation observed in a single experiment. This can be useful for determining the influence of batch effects in your cytometry experiment.

data
Ordered dictionary as produced by load_and_sample function

Type OrderedDict

reference
Reference sample; this will be the dataframe used to establish the embedded space upon which data is projected to reduce dimensionality

Type str

verbose
Whether to provide feedback

Type bool (default=True)

njobs
Number of parallel jobs to run

Type int (default=-1)

kde_kernel
Kernel to use for KDE, for options see KDEpy.FFTKDE

Type str (default="gaussian")

kde_bw
Bandwidth/bandwidth estimation method to use for KDE. See KDEpy for options. Defaults too improved Sheather Jones (ISJ) algorithm, which does not assume normality and is robust to multimodal distributions. If you need to speed up results, change this to ‘silvermans’ which is less accurate but less computationally intensive.

Type str or float (default="ISJ")

kde_norm
p-norm for high-dimensional KDE calculation

Type int (default=2)
```

Methods:
clean_cache()  
Clears the KDE cached results

Returns

Return type  None

matrix(distance_metric: str = 'jsd', features: list = None, dim_reduction_method: str = 'PCA',  
dim_reduction_kwargs: dict = None, bw_optimisaiton_kwargs: dict = None)  →  pandas.core.frame.DataFrame

Generate a Pandas DataFrame containing a symmetrical matrix of pairwise statistical distances for every  
sample in self.data

Parameters

• **distance_metric** (callable or str (default='jsd'))–

  Either a callable function to calculate the statistical distance or a string value; options are:

  - jsd: Jensson-shannon distance  
  - kl:Kullback-Leibler divergence (entropy)

• **features** (list (optional))– List of markers to use in analysis. If not given,  
  will use all available markers.

• **dim_reduction_method** (str (default="PCA"))– Dimension reduction  
  method, see CytoPy.flow.dim_reduction. Set to None to not reduce first

• **dim_reduction_kwarggs** (dict)– Keyword arguments for dimension reduction  
  method, see CytoPy.flow.dim_reduction

• **bw_optimisaiton_kwarggs** (dict)– Additional keyword arguments passed to  
  CytoPy.flow.variance.bw_optimisation call

Returns

Return type  Pandas.DataFrame

CytoPy.flow.variance.bw_optimisation(data: pandas.core.frame.DataFrame, features: list, kernel: str = 'gaussian', bandwidth: tuple = 0.01, 0.1, 10,  
cv: int = 10, verbose: int = 0)  →  float

Using GridSearchCV and the Scikit-Learn implementation of KDE, find the optimal bandwidth for the given  
data using grid search cross-validation

Parameters

• **data** (pd.DataFrame)–

• **features** (features)–

• **kernel** (str (default="gaussian"))–

• **bandwidth** (tuple (default=(0.01, 0.1, 20)))– Linear search space for  
  bandwidth (min, max, increments)

• **cv** (int (default=10))– Number of k-folds
verbose (int (default=0)) –

Returns

Return type float

CytoPy.flow.variance.calculate_ref_sample (data: collections.OrderedDict, verbose: bool = True) → str

This is performed as described in Li et al paper (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5860171/) on DeepCyTOF: for every 2 samples i, j compute the euclidean norm of the difference between their covariance matrices and then select the sample with the smallest average distance to all other samples.

This is an optimised version of supervised.ref.calculate_red_sample that leverages the multi-processing library to speed up operations

Parameters

• data (dict) –

• verbose (bool, (default=True)) – Feedback

Returns Sample ID of reference sample

Return type str


Generate a grid of embeddings using a valid dimensionality reduction technique, in each plot a reference sample is shown in blue and a comparison sample in red. The reference sample is conserved across all plots.

Parameters

• data (OrderedDict) – Ordered dictionary as generated from load_and_sample

• reference (str) – Reference sample to plot in the background

• comparison_samples (list) – List of samples to compare to reference (blue)

• features (list) – List of features to use for dimensionality reduction

• figsize (tuple, (default=(10,10))) – Size of figure

• method (str, (default='PCA')) – Method to use for dimensionality reduction (see flow.dim_reduction)

• dim_reduction_kwargs (dict) – Additional keyword arguments passed to CytoPy.dim_reduction.dimensionality_reduction

• kde (bool, (default=False)) – If True, overlay with two-dimensional PDF estimated by KDE

• verbose (bool (default=True)) –

Returns Plot printed to stdout

Return type None

CytoPy.flow.variance.generate_groups (linkage_matrix: numpy.array, sample_ids: list, n_groups: int)

Given the output of SimilarityMatrix (that is the linkage matrix and ordered list of sample IDs) and a desired number of groups, return a Pandas DataFrame of sample IDs and assigned group ID, generated by cutting the linkage matrix in such a way that the desired number of groups are generated. :param linkage_matrix: Linkage matrix generated from EvaluateBatchEffects.similarity_matrix (using SciPy.cluster.hierarchy.linkage)
:type linkage_matrix: np.array :param sample_ids: Ordered list of sample IDs generated from EvaluateBatchEffects.similarity_matrix :type sample_ids: list or np.array :param n_groups: Desired number of groups :type n_groups: int

Returns

Return type Pandas.DataFrame


Load sample data from experiment and return as a dictionary of Pandas DataFrames.

Parameters

• experiment (Experiment) –
• sample_ids (list) –
• sample_size (int or float (optional)) – Total number of events to sample from each file
• sampling_method (str) –
• transform (str (optional)) –
• population (str) –
• kwargs – Additional keyword arguments for sampling method

Returns

Return type OrderedDict


Compare the kernel density estimates for each marker in the associated experiment for the given comparison samples. The estimated distributions of the comparison samples will be plotted against the reference sample.

Parameters

• data (OrderedDict) – Ordered dictionary as generated from load_and_sample
• reference (str) – Reference sample to plot in the background
• comparison_samples (list) – List of valid sample IDs for the associated experiment
• markers (list (optional)) – List of markers to include (defaults to all available markers)
• figsize (figsize (default=(10, 10))) –
• xlim (tuple (optional)) – x-axis limits
• verbose (bool (default=True)) –
• kernel (str (default="gaussian")) –
• kde_bw (str or float (default="silverman")) –
• kwargs (dict) – Additional kwargs passed to Matplotlib.Axes.plot call
CytoPy, Release 1.0.0

Returns

Return type  matplotlib.Figure

CytoPy.flow.variance.scale_data(data: collections.OrderedDict, method: str = 'standard', **kwargs) \rightarrow collections.OrderedDict

Given a dictionary of events data as generated by load_and_sample, scale the data using a valid scale method (see CytoPy.flow.transforms.scaler)

Parameters

- data (dict)-
- method (str (default="standard")-)  
- kwargs (dict) – Keywords passed to scaler

Returns

Return type  dict

8.15 CytoPy.flow.explore

8.16 CytoPy.flow.neighbours

This module houses some two convenient functions for wrapping the Scikit-Learn implementation of K nearest neighbours classification algorithm.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

- GridSearchCV(**kwargs) Exhaustive search over specified parameter values for an estimator.
- KNeighborsClassifier(**kwargs) Classifier implementing the k-nearest neighbors vote.

Functions:

- balanced_accuracy_score(y_true, y_pred, *, ...) Compute the balanced accuracy
- calculate_optimal_neighbours(x, y, scoring, ...) Calculate the optimal n_neighbours parameter for KNeighborsClassifier using GridSearchCV.

continues on next page
### Table 149 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>knn(data, labels, features, n_neighbours[, ...])</code></td>
<td>Train a nearest neighbours classifier (scikit-learn implementation) and return the balanced accuracy score for both training and validation.</td>
</tr>
<tr>
<td><code>train_test_split(*arrays, **options)</code></td>
<td>Split arrays or matrices into random train and test subsets</td>
</tr>
</tbody>
</table>

CytoPy.flow.neighbours.**calculate_optimal_neighbours** *(x: `pandas.core.frame.DataFrame`, y: `numpy.array`, scoring: `str`, **kwargs)*

Calculate the optimal n_neighbours parameter for KNeighborsClassifier using GridSearchCV. Returns optimal n and highest score

**Parameters**
- `x` *(Pandas.DataFrame)*
- `y` *(np.array)*
- `scoring` *(str)*
- `kwargs` *(dict)*

**Returns**
- `int, float`

CytoPy.flow.neighbours.**knn** *(data: `pandas.core.frame.DataFrame`, labels: `numpy.array`, features: `list`, n_neighbours: `int`, holdout_size: `float = 0.2`, random_state: `int = 42`, return_model: `bool = False`, **kwargs)*

Train a nearest neighbours classifier (scikit-learn implementation) and return the balanced accuracy score for both training and validation.

**Parameters**
- `data` *(Pandas.DataFrame)*
- `labels` *(Numpy.Array)*
- `features` *(list)*
- `n_neighbours` *(int)*
- `holdout_size` *(float (default=0.2))*
- `random_state` *(int (default=42))*
- `return_model` *(bool (default=False))*
- `kwargs` *(dict)*

**Returns** Training balanced accuracy score, Validation balanced accuracy score, Classifier (if return_model is True)

**Return type** *(float, float) or (float, float, object)*
Central to the analysis of Cytometry data is visualisation. For exploratory analysis and ‘gating’ this normally comes in the form of bi-axial plots of different cell surface markers or intracellular stains. This module contains the CreatePlot class which houses the functionality for all one and two dimensional plotting of cytometry data. This class interacts with Population objects to present the data in multiple ways. This can be as standard 2D histograms as is common in software like FlowJo, but also allows for plotting of Population geometries (the shapes that define the gates that generated a Population) or overlaying downstream populations for ‘back-gating’ purposes.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvexHull</td>
<td>Convex hulls in N dimensions.</td>
</tr>
<tr>
<td>CreatePlot</td>
<td>Generate 1D or 2d histograms of cell populations as identified by cytometry.</td>
</tr>
<tr>
<td>EllipseGate</td>
<td>EllipseGate inherits from PolygonGate.</td>
</tr>
<tr>
<td>Gate</td>
<td>Base class for a Gate.</td>
</tr>
<tr>
<td>LogNorm</td>
<td>Normalize a given value to the 0-1 range on a log scale.</td>
</tr>
<tr>
<td>PolygonGate</td>
<td>PolygonGate inherits from Gate.</td>
</tr>
<tr>
<td>PolygonGeom</td>
<td>Polygon shape.</td>
</tr>
<tr>
<td>Population</td>
<td>A population of cells identified by either a gate or supervised algorithm.</td>
</tr>
<tr>
<td>ThresholdGate</td>
<td>ThresholdGate inherits from Gate.</td>
</tr>
<tr>
<td>ThresholdGeom</td>
<td>Threshold shape.</td>
</tr>
<tr>
<td>cycle</td>
<td>cycle(Iterable) -&gt; cycle object</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_transform</td>
<td>Apply a transformation to the given dataframe. The features_to_transform specified which columns in the dataframe to transform. This can be given as: * a string value of either ‘all’ or ‘fluorochromes’; transform_method defines which transform to apply to columns * a list of columns to transform; transform_method defines which transform to apply to columns * alternatively, a dictionary where the key is the column name and the value is the transform method to apply to this column; transform_method is ignored.</td>
</tr>
</tbody>
</table>

continues on next page
Table 151 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warn(message[, category, stacklevel, source])</td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

```python

Bases: object

Generate 1D or 2D histograms of cell populations as identified by cytometry. Supports plotting of individual populations, single or multiple gates, “backgating” (plotting child populations overlaid on parent) and overlaying populations from control samples on their equivalent in the primary sample.

**transform_x**

How to transform the x-axis. Method ‘plot_gate’ overwrites this value with the value associated with the gate

*Type* str (default = “logicle”)

**transform_y**

How to transform the y-axis. Method ‘plot_gate’ overwrites this value with the value associated with the gate

*Type* str (default = “logicle”)

**xlabel**

x-axis label

*Type* str, optional

**ylabel**

y-axis label

*Type* str, optional

**xlim**

x-axis limits; if not given defaults to the range of 0.001 quantile to 0.999 quantile

*Type* (float, float), optional

**ylim**

y-axis limits; if not given defaults to the range of 0.001 quantile to 0.999 quantile

*Type* (float, float), optional

**ax**

If not given, an axes object will be generated with the given figsize. Access the figure object through ‘fig’ attribute

*Type* matplotlib.pyplot.axes, optional

**figsize**

Ignored if an Axes object is given

*Type* (int, int)
### CytoPy, Release 1.0.0

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bins</td>
<td>How many bins to use for 2D histogram. Can also provide a value of \“scotts\”, \“sturges\”, \“rice\” or \“sqrt\” to use a given method to estimate suitable bin size</td>
<td>int or str (default=\”scotts&quot;)</td>
</tr>
<tr>
<td>cmap</td>
<td>Colormap for 2D histogram</td>
<td>str, (default=&quot;jet&quot;)</td>
</tr>
<tr>
<td>style</td>
<td>Plotting style (passed to seaborn.set_style)</td>
<td>str, optional (default=&quot;white&quot;)</td>
</tr>
<tr>
<td>autoscale</td>
<td>Allow matplotlib to calculate optimal view (<a href="https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.autoscale.html">https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.autoscale.html</a>)</td>
<td>bool (default=True)</td>
</tr>
<tr>
<td>font_scale</td>
<td>Font scale (passed to seaborn.set_context)</td>
<td>float, optional (default=1.2)</td>
</tr>
<tr>
<td>bw</td>
<td>Bandwidth for 1D KDE (see seaborn.kdeplot)</td>
<td>str or float, (default=&quot;scott&quot;)</td>
</tr>
<tr>
<td>axis_ticks</td>
<td>Show axis ticks with axis labels</td>
<td>bool (default=True)</td>
</tr>
</tbody>
</table>

#### Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgate</td>
<td>Plot one or more populations as an overlay atop a given parent population.</td>
<td>parent: pandas.core.frame.DataFrame, children: Dict[str, pandas.core.frame.DataFrame], x: str, y: str = None, colours: str = \’pastel\’, alpha: float = 0.75, size: float = 5, method: str = \’scatter\’, shade: bool = True, plot_kwars: dict = None, overlay_kwars: dict = None, legend_kwars: dict = None</td>
</tr>
<tr>
<td>overlay_plot</td>
<td>Plot data2 overlaid a 2D histogram or 1D KDE plot of data1.</td>
<td>data1, data2, x[, y, colour, ...]</td>
</tr>
<tr>
<td>plot</td>
<td>Plot a single population as either a 2D histogram or 1D KDE</td>
<td>data, x[, y]</td>
</tr>
<tr>
<td>plot_gate_children</td>
<td>Plot a Gate object.</td>
<td>gate, parent[, lw, y, ...]</td>
</tr>
<tr>
<td>plot_population_geoms</td>
<td>This will plot the geometric shapes from the list of child populations generated from a single Gate, overlaid on the parent population upon which the Gate has been applied.</td>
<td>parent, children[,...]</td>
</tr>
</tbody>
</table>
• **children** (*dict*) – Dictionary of Pandas DataFrames, where the key corresponds to the population name and the value the Pandas DataFrame of single cell events

• **x** (*str*) – X-axis variable

• **y** (*str (optional)*) – Y-axis variable

• **colours** (*str (defaults to Seaborn pastel palette)*) – Name of the palette to use for colouring overlaid populations

• **alpha** (*float (default=0.75)*) – If method is ‘scatter’, controls transparency of markers

• **size** (*float (default=5)*) – If method is ‘scatter’, controls size of markers

• **method** (*str or dict (default="scatter")*) – Method should be either “scatter” (default), “polygon”, or “kde”, which controls how overlaid populations will appear. If a dictionary is provided, then it should have keys matching that of ‘children’ and values being the methods to use for each overlay.

• **shade** (*bool (default=True)*) – If method is ‘kde’, specifies whether to shade in the contours

• **plot_kwargs** (*dict (optional)*) – Keyword arguments passed to CreatePlot.plot

• **overlay_kwargs** (*dict (optional)*) – Keyword arguments passed to plt.scatter or seaborn.kdeplot cals

• **legend_kwargs** (*dict (optional)*) – Additional keyword arguments to pass to axis legend. Defaults: * bbox_to_anchor = (0.5, 1.05) * loc = “upper center” * ncol = 3 * fancybox = True * shadow = False

**Returns**

**Return type** Matplotlib.axes


Plot data2 overlaid a 2D histogram or 1D KDE plot of data1.

**Parameters**

• **data1** (Pandas.DataFrame) –

• **data2** (Pandas.DataFrame) –

• **x** (*str*) –

• **y** (*str, optional*) –

• **colour** (*str (default="#db4b6a")*) –

• **alpha** (*float (default=0.75)*) –

• **size** (*float (default=5.)*) –

• **method** (*str (default="scatter")*) –

• **shade** (*bool (default=True)*) –

• **plot_kwargs** (*dict*) –

• **overlay_kwargs** (*dict*) –
Returns

Return type  None

plot (data: pandas.core.frame.DataFrame, x: str, y: str = None, **kwargs)
Plot a single population as either a 2D histogram or 1D KDE

Parameters

• data (Pandas.DataFrame) – Population dataframe
• x (str) – Channel to plot on the x-axis
• y (str, optional) – Channel to plot on the y-axis
• kwargs – Keyword arguments to be passed to matplotlib.pyplot.axes.hist2d or seaborn.kdeplot (depending on whether a y-axis variable is given)

Returns  Axis object

Return type  Matplotlib.pyplot.axes

plot_gate_children (gate: CytoPy.data.gate.Gate, parent: pandas.core.frame.DataFrame, lw: float = 2.5, y: str = None, transform_x: str = None, transform_y: str = None, plot_kwargs: dict = None, legend_kwargs: dict = None)
Plot a Gate object. This will plot the geometric shapes generated from a single Gate, overlaid on the parent population given as Pandas.DataFrame. It should be noted, this will plot the geometric definitions of a gates children, i.e. the expected populations. If you have generated new populations from new data using a Gate you should plot with the `plot_population_geoms` method

Parameters

• gate (Gate or ThresholdGate or EllipseGate or PolygonGate) –
• parent (Pandas.DataFrame) – Parent DataFrame
• lw (float (default = 2.5)) – Linewidth for shapes to plot
• plot_kwargs – Additional keyword arguments to pass to plot_population (generates the plot of parent population)
• legend_kwargs – Additional keyword arguments to pass to axis legend. Defaults:
  * bbox_to_anchor = (0.5, 1.05) * loc = “upper center” * ncol = 3 * fancybox = True
  * shadow = False
• y (str (optional)) – Overrides the plotting configurations for the gate if y is missing and allows user to plot a two-dimensional instead of one dimensional plot. Only value for ThresholdGate.
• transform_x (str (optional)) – Overrides the transformation to the x-axis variable
• transform_y (str (optional)) – Overrides the transformation to the x-axis variable

Returns  Axis object

Return type  Matplotlib.pyplot.axes

This will plot the geometric shapes from the list of child populations generated from a single Gate, overlaid
on the parent population upon which the Gate has been applied. The parent data should be provided as a
Pandas DataFrame of single cell data and the Geoms of the resulting Populations in the list ‘children’.

Parameters

• **parent** *(Pandas.DataFrame)* – Parent DataFrame
• **children** *(list)* – List of Population objects that derive from the parent. Popula-
tion geometries will be overlaid on the parent population.
• **lw** *(float (default = 2.5))* – Linewidth for shapes to plot
• **plot_kwargs** – Additional keyword arguments to pass to plot_population (gener-
ates the plot of parent population)
• **legend_kwargs** –
  
  Additional keyword arguments to pass to axis legend. Defaults:
  
  – bbox_to_anchor = (0.5, 1.05)
  – loc = “upper center”
  – ncol = 3
  – fancybox = True
  – shadow = False
• **y** *(str (optional))* – Overrides the plotting configurations for the gate if y is
missing and allows user to plot a two-dimensional instead of one dimensional plot.
  Only value for ThresholdGate.
• **transform_x** *(str (optional))* – Overrides the transformation to the x-axis vari-
able
• **transform_y** *(str (optional))* – Overrides the transformation to the x-axis vari-
able

Returns Axis object

Return type Matplotlib.pyplot.axes

8.18 CytoPy.flow.ref

When creating training data for supervised classification it can be useful to generate a new example FileGroup by
sampling many or all the FileGroups present in an Experiment. This can also be useful if we have suitable data to be
modelled as after concatenation of all available events (say the data was all measured within the same batch). This
module contains the create_ref_sample function for merging multiple FileGroups to form a new FileGroup saved to
the experiment.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documen-
tation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom
the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission
notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PAR-
TICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment(*args, **kwargs)</td>
<td>Container for Cytometry experiment.</td>
</tr>
<tr>
<td>FileGroup(*args, **values)</td>
<td>Document representation of a file group; a selection of related fcs files (e.g.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_ref_sample(experiment, ...)</td>
<td>Given some experiment and a root population that is common to all fcs file groups within this experiment, take a sample from each and create a new file group from the concatenation of these data.</td>
</tr>
<tr>
<td>load_and_sample(experiment, population, ...)</td>
<td>Load sample data from experiment and return as a dictionary of Pandas DataFrames.</td>
</tr>
<tr>
<td>vprint(verbos)</td>
<td>Utility function for optional printing.</td>
</tr>
</tbody>
</table>

CytoPy.flow.ref.create_ref_sample(experiment: CytoPy.data.experiment.Experiment, sample_size: int = 2500, sampling_method: str = 'uniform', sampling_kwars: dict = None, root_population='root', sample_ids: list = None, new_file_name: str = None, verbose: bool = True, save_sample_id: bool = True) → None

Given some experiment and a root population that is common to all fcs file groups within this experiment, take a sample from each and create a new file group from the concatenation of these data. New file group will be created and associated to the given FileExperiment object. If no file name is given it will default to '{Experiment Name}_sampled_data'

Parameters

- **experiment** (FCSExperiment) – FCSExperiment object for corresponding experiment to sample
- **root_population** (str) – if the files in this experiment have already been gated, you can specify to sample from a particular population e.g. Live CD3+ cells or Live CD45- cells
- **sample_ids** (list, optional) – list of sample IDs for samples to be included (default = all samples in experiment)
- **new_file_name** (str) – name of file group generated
- **sampling_method** (str, (default='uniform')) – method to use for sampling files (currently only supports ‘uniform’)
- **sample_size** (int or float, (default=1000)) – number or fraction of events to sample from each file
- **sampling_kwars** (dict) – Additional keyword arguments passed to sampling method
- **verbose** (bool, (default=True)) – Whether to provide feedback
- **save_sample_id** (bool (default=True)) – If True, the sample ID that each cell originates from is saved to the FileGroup cell_meta_labels attribute

Returns
8.19 CytoPy.flow.sampling

For manageable analysis sampling is unavoidable. This module contains all the functionality for downsampling and subsequent upsampling in CytoPy. CytoPy supports uniform sampling that wraps the Pandas DataFrame sample method. In addition we provide support for density dependent downsampling (adapted from SPADE; https://www.nature.com/articles/nbt.1991) and faithful downsampling (adapted from SamSPECTRAL; https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-403).

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

BallTree(X[, leaf_size, metric]) BallTree for fast generalized N-point problems
KDTree(X[, leaf_size, metric]) KDTree for fast generalized N-point problems
partial partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

Functions:

Pool([processes, initializer, initargs, ...]) Returns a process pool object
calculate_optimal_neighbours(x, y, scoring, ...) Calculate the optimal n_neighbours parameter for KNeighborsClassifier using GridSearchCV.
cpu_count() Returns the number of CPUs in the system
density_dependent_downsampling(data[, ...]) Perform density dependent down-sampling to remove risk of under-sampling rare populations; adapted from SPADE*
density_probability_assignment(sample, data) Generate an estimation of local density amongst single cell population using the KDTree algorithm from Scikit-Learn.
faithful_downsampling(data, h) An implementation of faithful downsampling as described in: Zare H, Shoshtari P, Gupta A, Brinkman R.
knn(data, labels, features, n_neighbours[, ...]) Train a nearest neighbours classifier (scikit-learn implementation) and return the balanced accuracy score for both training and validation.

density_probability_assignment(sample, data) Generate an estimation of local density amongst single cell population using the KDTree algorithm from Scikit-Learn.
faithful_downsampling(data, h) An implementation of faithful downsampling as described in: Zare H, Shoshtari P, Gupta A, Brinkman R.
knn(data, labels, features, n_neighbours[, ...]) Train a nearest neighbours classifier (scikit-learn implementation) and return the balanced accuracy score for both training and validation.
Table 156 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>prob_downsample</code></td>
<td>Given local, target and outlier density (as estimated by KNN) calculate the probability of retaining the event.</td>
</tr>
<tr>
<td><code>uniform_downsampling</code></td>
<td>Uniform downsampling.</td>
</tr>
<tr>
<td><code>upsample_density</code></td>
<td>Perform upsampling in a density dependent manner; neighbourhoods of cells of low density will have a high probability of being upsampled versus dense neighbourhoods.</td>
</tr>
<tr>
<td><code>upsample_knn</code></td>
<td>Given some sampled dataframe and the original dataframe from which it was derived, use the given labels (which should correspond to the sampled dataframe row index) to fit a nearest neighbours model to the sampled data and predict the assignment of labels in the original data.</td>
</tr>
<tr>
<td><code>vprint</code></td>
<td>Utility function for optional printing.</td>
</tr>
<tr>
<td><code>warn</code></td>
<td>Issue a warning, or maybe ignore it or raise an exception.</td>
</tr>
</tbody>
</table>

CytoPy.flow.sampling.density_dependent_downsampling

```python
from pandas.core.frame import DataFrame
features = list = None, sample_size: int = 0.1, alpha: int = 5, distance_metric: str = 'manhattan', tree_sample: float = 0.1, outlier_dens: int = 1, target_dens: int = 5, njobs: int = -1)
```

Perform density dependent down-sampling to remove risk of under-sampling rare populations; adapted from SPADE*.

- Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE


**Parameters**

- `data` (*Pandas.DataFrame*) – Data to sample
- `features` (*list (defaults to all columns]*) – Name of columns to be used as features in down-sampling algorithm
- `sample_size` (*int or float (default=0.1]*) – number of events to return in sample, either as an integer of fraction of original sample size
- `alpha` (*int, (default=5]*) – used for estimating distance threshold between cell and nearest neighbour (default = 5 used in original paper)
- `distance_metric` (*str (default="manhattan")*) – Metric used for neighbour assignment
- `tree_sample` (*float or int, (default=0.1]*) – proportion/number of cells to sample for generation of KD tree
- `outlier_dens` (*float, (default=1]*) – used to exclude cells with the lowest local densities; int value as a percentile of the lowest local densities e.g. 1 (the default value) means the bottom 1% of cells with lowest local densities are regarded as noise

8.19. CytoPy.flow.sampling
• **target_dens** *(float, (default=5)) –* determines how many cells will survive the down-sampling process; int value as a percentile of the lowest local densities e.g. 5 (the default value) means the density of bottom 5% of cells will serve as the density threshold for rare cell populations

• **njobs** *(int (default=-1)) –* Number of jobs to run in unison when calculating weights (defaults to all available cores)

**Returns**  
Down-sampled pandas dataframe

**Return type**  
Pandas.DataFrame

---

**CytoPy.flow.sampling.density_probability_assignment** *(sample: pandas.core.frame.DataFrame, data: pandas.core.frame.DataFrame, distance_metric: str = 'manhattan', alpha: int = 5, outlier_dens: int = 1, target_dens: int = 5, njobs: int = -1)*

Generate an estimation of local density amongst single cell population using the KDTree algorithm from Scikit-Learn. Using this representation return the probability assignment for retention of each event using `prob_downsample`. adapted from SPADE*

• Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE


**Parameters**

• **sample** *(Pandas.DataFrame) –* Downsampler data to use for generating nearest neighbours tree graph

• **data** *(Pandas.DataFrame) –* Original dataframe

• **distance_metric** *(str (default="manhattan")) –* Metric used for neighbour assignment

• **alpha** *(int) –* Used for estimating distance threshold between cell and nearest neighbour (default = 5 used in original paper)

• **outlier_dens** *(int, (default=1)) –* used to exclude cells with the lowest local densities; float value as a percentile of the lowest local densities e.g. 1 (the default value) means the bottom 1% of cells with lowest local densities are regarded as noise

• **target_dens** *(int, (default=5)) –* determines how many cells will receive a probability > 0; int value as a percentile of the lowest local densities e.g. 5 (the default value) means the density of bottom 5% of cells will serve as the density threshold for rare cell populations

• **njobs** *(int (default=-1)) –* Controls how many parallel processed to run in KDTree search. Default is -1, which will use all available cores.

**Returns**

**Return type**  
Numpy.Array

---

**CytoPy.flow.sampling.faithful_downsampling** *(data: numpy.array, h: float)*

An implementation of faithful downsampling as described in: Zare H, Shooshtari P, Gupta A, Brinkman R. Data reduction for spectral clustering to analyze high throughput flow cytometry data. BMC Bioinformatics 2010;11:403
Parameters

• data (Numpy.array) – numpy array to be down-sampled
• h (float) – radius for nearest neighbours search

Returns Down-sampled array

Return type Numpy.array

CytoPy.flow.sampling.prob_downsample (local_d: int, target_d: int, outlier_d: int)

Given local, target and outlier density (as estimated by KNN) calculate the probability of retaining the event. If local density is less than or equal to the outlier density, returns a probability of 0 (event will be discarded). If the local density is greater than the outlier density but less than the target density, return a value of 1 (absolutely keep this event). If the local density is greater than the target density, then the probability of retention is the ratio between the target and local density.

Parameters

• local_d (int) –
• target_d (int) –
• outlier_d (int) –

Returns Value between 0 and 1

Return type float

CytoPy.flow.sampling.uniform_downsampling (data: pandas.core.frame.DataFrame, sample_size: int, **kwargs)

Uniform downsampling. Wraps the Pandas DataFrame sample method with some additional error handling for when the requested sample size is invalid.

Parameters

• data (Pandas.DataFrame) –
• sample_size (int or float) – Size of sample required. If a float is given will return a sample of this proportion.
• kwargs – Additional keyword arguments passed to Pandas.DataFrame.sample

Returns

Return type Pandas.DataFrame

CytoPy.flow.sampling.upsample_density (data: pandas.core.frame.DataFrame, features: list = None, upsample_factor: int = 2, sample_size: int = None, tree_sample: int = 0.1, distance_metric: str = 'manhattan', alpha: int = 5, outlier_dens: int = 1, target_dens: int = 5, njobs: int = -1)

Perform upsampling in a density dependent manner; neighbourhoods of cells of low density will have a high probability of being upsampled versus dense neighbourhoods. Ignores outliers. adapted from SPADE*

- Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE

Parameters

• data (Pandas.DataFrame) – Data to sample
• features (list (defaults to all columns)) – Name of columns to be used as features in down-sampling algorithm
**CytoPy, Release 1.0.0**

- **sample_size** *(int or float (default=0.1)) – number of events to return in sample, either as an integer or fraction of original sample size*
- **alpha** *(int, (default=5)) – used for estimating distance threshold between cell and nearest neighbour (default = 5 used in original paper)*
- **distance_metric** *(str (default="manhattan")) – Metric used for neighbour assignment*
- **upsample_factor** *(int (default=2)) – Factor to upsample by (e.g. default=2 would double the observations)*
- **tree_sample** *(float or int, (default=0.1)) – proportion/number of cells to sample for generation of KD tree*
- **outlier_dens** *(float, (default=1)) – used to exclude cells with the lowest local densities; int value as a percentile of the lowest local densities e.g. 1 (the default value) means the bottom 1% of cells with lowest local densities are regarded as noise*
- **target_dens** *(float, (default=5)) – determines how many cells will survive the down-sampling process; int value as a percentile of the lowest local densities e.g. 5 (the default value) means the density of bottom 5% of cells will serve as the density threshold for rare cell populations*
- **njobs** *(int (default=-1)) – Number of jobs to run in unison when calculating weights (defaults to all available cores)*

**CytoPy.flow.sampling.upsample_knn**(sample: pandas.core.frame.DataFrame, original_data: pandas.core.frame.DataFrame, labels: list, features: list, verbose: bool = True, scoring: str = 'balanced_accuracy', **kwargs)

Given some sampled dataframe and the original dataframe from which it was derived, use the given labels (which should correspond to the sampled dataframe row index) to fit a nearest neighbours model to the sampled data and predict the assignment of labels in the original data. Uses sklearn.neighbors.KNeighborsClassifier for KNN implementation. If n_neighbors parameter is not provided, will estimate using grid search cross validation. The scoring parameter can be tuned by changing the scoring input (default="balanced_accuracy")

**Parameters**

- **sample** *(Pandas.DataFrame) – Sampled dataframe that has been classified/gated/etc*
- **original_data** *(Pandas.DataFrame) – Original dataframe prior to sampling (unlabeled)*
- **labels** *(list) – List of labels (should correspond to the label for each row)*
- **features** *(list) – List of features (column names)*
- **verbose** *(bool (default=True)) – If True, will provide feedback to stdout*
- **scoring** *(str (default="balanced_accuracy")) – Scoring parameter to use for GridSearchCV. Only relevant is n_neighbors parameter is not provided*
- **kwargs** *(dict) – Additional keyword arguments passed to Scikit-Learn’s KNeighborsClassifier*

**Returns** Array of labels for original data

**Return type** numpy.Array
8.20 CytoPy.flow.supervised

This module houses many of the utility functions for supervised classification.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

- BaggingClassifier(**kwargs): A Bagging classifier.
- BaggingRegressor(**kwargs): A Bagging regressor.
- BallTree(X[, leaf_size, metric]): BallTree for fast generalized N-point problems
- BaseEnsemble(base_estimator, *[...]): Base class for all ensemble classes.
- DistanceMetric: DistanceMetric class
- FileGroup(*args, **values): Document representation of a file group; a selection of related fcs files (e.g.
- GradientBoostingClassifier(**kwargs): Gradient Boosting for classification.
- GradientBoostingRegressor(**kwargs): Gradient Boosting for regression.
- IsolationForest(**kwargs): Isolation Forest Algorithm.
- KDTree(X[, leaf_size, metric]): KDTree for fast generalized N-point problems
- KNeighborsClassifier(**kwargs): Classifier implementing the k-nearest neighbors vote.
- KNeighborsRegressor(**kwargs): Regression based on k-nearest neighbors.
- KNeighborsTransformer(**kwargs): Transform X into a (weighted) graph of k nearest neighbors
- LinearDiscriminantAnalysis(**kwargs): Linear Discriminant Analysis
- LinearSVC(**kwargs): Linear Support Vector Classification.
- LinearSVR(**kwargs): Linear Support Vector Regression.
- LocalOutlierFactor(**kwargs): Unsupervised Outlier Detection using Local Outlier Factor (LOF)
- NeighborhoodComponentsAnalysis(**kwargs): Neighborhood Components Analysis
- NuSVC(**kwargs): Nu-Support Vector Classification.
- NuSVR(**kwargs): Nu Support Vector Regression.
- OneClassSVM(**kwargs): Unsupervised Outlier Detection.

continues on next page
Table 157 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert_population_labels(ref, expected_labels)</td>
<td>Given some reference FileGroup and the expected population labels, check the validity of the labels and return list of valid populations only.</td>
</tr>
<tr>
<td>auto_weights(y)</td>
<td>Estimate optimal weights from a list of class labels.</td>
</tr>
<tr>
<td>build_keras_model(layers, optimizer, loss, ...)</td>
<td>Create and compile a Keras Sequential model using the given KerasClassifier object</td>
</tr>
<tr>
<td>build_sklearn_model(klass, **params)</td>
<td>Initiate a SklearnClassifier object using Classes in the global environment</td>
</tr>
<tr>
<td>calc_metrics(metrics, y_true[, y_pred, y_score])</td>
<td>Given a list of Scikit-Learn supported metrics (<a href="https://scikit-learn.org/stable/modules/model_evaluation.html">https://scikit-learn.org/stable/modules/model_evaluation.html</a>) return a dictionary of results after checking that the required inputs are provided.</td>
</tr>
<tr>
<td>check_downstream_populations(ref, ...)</td>
<td>Check that in the ordered list of population labels, all populations are downstream of the given ‘root’ population.</td>
</tr>
<tr>
<td>compute_class_weight(class_weight, *, classes, y)</td>
<td>Estimate class weights for unbalanced datasets.</td>
</tr>
<tr>
<td>confusion_matrix_plots(classifier, x, y, ...)</td>
<td>Generate a figure of two heatmaps showing a confusion matrix, one normalised by support one showing raw values, displaying a classifiers performance.</td>
</tr>
<tr>
<td>kneighbors_graph(X, n_neighbors, *[mode, ...])</td>
<td>Computes the (weighted) graph of k-Neighbors for points in X</td>
</tr>
<tr>
<td>l1_min_c(X, y[, loss, fit_intercept, ...])</td>
<td>Return the lowest bound for C such that for C in (l1_min_C, infinity) the model is guaranteed not to be empty.</td>
</tr>
<tr>
<td>multilabel(ref, root_population, ...)</td>
<td>Load the root population DataFrame from the reference FileGroup (assumed to be the first population in ‘population_labels’).</td>
</tr>
</tbody>
</table>

Functions:
CytoPy, Release 1.0.0

Table 158 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>radius_neighbors_graph</code></td>
<td>Computes the (weighted) graph of Neighbors for points in X</td>
</tr>
<tr>
<td><code>singlelabel</code></td>
<td>Load the root population DataFrame from the reference FileGroup (assumed to be the first population in 'population_labels').</td>
</tr>
</tbody>
</table>

**CytoPy.flow.supervised.assert_population_labels**

Given some reference FileGroup and the expected population labels, check the validity of the labels and return list of valid populations only.

**Parameters**
- `ref` (FileGroup) -
- `expected_labels` (list) -

**Returns**
- **Return type** List

**CytoPy.flow.supervised.auto_weights**

Estimate optimal weights from a list of class labels.

**Parameters**
- `y` (Numpy.Array) -

**Returns**
- Dictionary of class weights {label: weight}
- **Return type** dict

**CytoPy.flow.supervised.build_keras_model**

Create and compile a Keras Sequential model using the given KerasClassifier object

**Parameters**
- `metrics` (list) – See https://keras.io/api/metrics/
- `loss` (str) – See https://keras.io/api/losses/
- `optimizer` (str) – See https://keras.io/api/optimizers/
- `layers` (list) – List of Layer objects (see https://keras.io/api/layers/)

**Returns**
- **Return type** object

**CytoPy.flow.supervised.build_sklearn_model**

Initiate a SklearnClassifier object using Classes in the global environment

**Parameters**
- `klass` (str, **params) -

**Returns**
- **Return type** object

**CytoPy.flow.supervised.calc_metrics**

Given a list of Scikit-Learn supported metrics (https://scikit-learn.org/stable/modules/model_evaluation.html) return a dictionary of results after checking that the required inputs are provided.

**Parameters**
- `metrics` (list) – List of string values; names of required metrics
- **y_true** (*Numpy.Array*) – True labels or binary label indicators. The binary and multiclass cases expect labels with shape (n_samples,) while the multilabel case expects binary label indicators with shape (n_samples, n_classes).

- **y_pred** (*Numpy.Array*) – Estimated targets as returned by a classifier

- **y_score** (*Numpy.Array*) – Target scores. In the binary and multilabel cases, these can be either probability estimates or non-thresholded decision values (as returned by decision_function on some classifiers). In the multiclass case, these must be probability estimates which sum to 1. The binary case expects a shape (n_samples,), and the scores must be the scores of the class with the greater label. The multiclass and multilabel cases expect a shape (n_samples, n_classes). In the multiclass case, the order of the class scores must correspond to the order of labels, if provided, or else to the numerical or lexicographical order of the labels in y_true.

**Returns**

Dictionary of performance metrics

**Return type**

dict

**CytoPy.flow.supervised.check_downstream_populations**

```
CytoPy.flow.supervised.check_downstream_populations(ref: CytoPy.data.fcs.FileGroup, root_population: str, population_labels: list) → None
```

Check that in the ordered list of population labels, all populations are downstream of the given 'root' population.

**Parameters**

- **ref** (*FileGroup*) –

- **root_population** (*str*) –

- **population_labels** (*list*) –

**Returns**

None

**CytoPy.flow.supervised.confusion_matrix_plots**

```
CytoPy.flow.supervised.confusion_matrix_plots(classifier, x: pandas.core.frame.DataFrame, y: numpy.ndarray, class_labels: list, cmap: str = None, figsize: tuple = (10, 5), **kwargs)
```

Generate a figure of two heatmaps showing a confusion matrix, one normalised by support one showing raw values, displaying a classifier's performance. Returns Matplotlib.Figure object.

**Parameters**

- **classifier** (*object*) – Scikit-Learn classifier

- **x** (*Pandas.DataFrame*) – Feature space

- **y** (*Numpy.Array*) – Labels

- **class_labels** (*list*) – Class labels (as they should be displayed on the axis)

- **cmap** (*str*) – Colour scheme, defaults to Matplotlib Blues

- **figsize** (*tuple (default=(10, 5))*) – Size of the figure

- **kwargs** – Additional keyword arguments passed to sklearn.metrics.plot_confusion_matrix

**Returns**

**Return type** Matplotlib.Figure
CytoPy.flow.supervised.multilabel (ref: CytoPy.data.fcs.FileGroup, root_population: str, population_labels: list, transform: str, features: list) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Load the root population DataFrame from the reference FileGroup (assumed to be the first population in `population_labels`). Then iterate over the remaining population creating a dummy matrix of population affiliations for each row of the root population.

Parameters

- ref (FileGroup)
- population_labels (list)
- transform (str)
- features (list)

Returns Root population fluorescent intensity values, population affiliations (dummy matrix)

Return type (Pandas.DataFrame, Pandas.DataFrame)

CytoPy.flow.supervised.singlelabel (ref: CytoPy.data.fcs.FileGroup, root_population: str, population_labels: list, transform: str, features: list) -> (<class 'pandas.core.frame.DataFrame'>, <class 'numpy.ndarray'>)

Load the root population DataFrame from the reference FileGroup (assumed to be the first population in `population_labels`). Then iterate over the remaining population creating a Array of population affiliations; each cell (row) is associated to their terminal leaf node in the FileGroup population tree.

Parameters

- root_population
- ref (FileGroup)
- population_labels (list)
- transform (str)
- features (list)

Returns Root population fluorescent intensity values, labels

Return type (Pandas.DataFrame, Numpy.Array)

8.21 CytoPy.flow.dim_reduction

CytoPy supports the following dimension reduction methods: UMAP, tSNE, PCA, Kernel PCA, and PHATE. These are implemented through the dim_reduction function. This takes a dataframe of single cell events and generates the desired number of embeddings. These are returned as a matrix or as appended columns to the given dataframe.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KernelPCA(**kwargs)</td>
<td>Kernel Principal component analysis (KPCA)</td>
</tr>
<tr>
<td>PCA(**kwargs)</td>
<td>Principal component analysis (PCA).</td>
</tr>
<tr>
<td>TSNE(**kwargs)</td>
<td>t-distributed Stochastic Neighbor Embedding.</td>
</tr>
<tr>
<td>UMAP([n_neighbors, n_components, metric, ...])</td>
<td>Uniform Manifold Approximation and Projection</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimensionality_reduction</td>
<td>Perform dimensionality reduction using either UMAP, PCA, tSNE, or PHATE.</td>
</tr>
</tbody>
</table>

**dimensionality_reduction**

CytoPy.flow.dim_reduction.dimensionality_reduction(data, features,...) Perform dimensionality reduction using either UMAP, PCA, tSNE, or PHATE.

Parameters

- **data** (Pandas.DataFrame) – Events to perform dim reduction on
- **features** (list) – column names for feature space
- **method** (str) – method to use; either UMAP, PCA, tSNE, or PHATE
- **n_components** (int) – number of components to generate
- **return_embeddings_only** (bool, (default=True)) – if True, the embeddings are returned as a numpy array, otherwise original dataframe is returned modified with new columns, one for each embedding (column name of format {Method}_{i} where i = 0 to n_components)
- **return_reducer** (bool, (default=False)) – If True, returns instance of dimensionality reduction object
- **kwargs** – keyword arguments to pass to chosen dim reduction method

Returns  Embeddings as numpy array or original DataFrame with new columns for embeddings

Return type  (Pandas.DataFrame or Numpy.array) or (Pandas.DataFrame or Numpy.array, Reducer)
8.22 CytoPy.flow.feature_extraction

For studies where the objective is the prediction of some endpoint and characterisation of phenotypes that contribute to that prediction, it is valuable to have tools for generating summaries of our cell populations to serve as variables in differential analysis or modelling tasks. This module provides the tools to summarise the populations generated and has numerous utility functions for ‘feature selection’.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment</td>
<td>Container for Cytometry experiment.</td>
</tr>
<tr>
<td>FileGroup</td>
<td>Document representation of a file group; a selection of related fcs files (e.g. FileGroup)</td>
</tr>
<tr>
<td>LinearSVC</td>
<td>Linear Support Vector Classification.</td>
</tr>
<tr>
<td>Population</td>
<td>A population of cells identified by either a gate or supervised algorithm.</td>
</tr>
<tr>
<td>StandardScaler</td>
<td>Standardize features by removing the mean and scaling to unit variance.</td>
</tr>
<tr>
<td>Subject</td>
<td>Document based representation of subject meta-data.</td>
</tr>
<tr>
<td>defaultdict</td>
<td>defaultdict(default_factory[, . . . ]) --&gt; dict with default factory</td>
</tr>
<tr>
<td>partial</td>
<td>partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_statistics(experiment[, population, ...])</td>
<td>Given an Experiment and the name of a Population known to contain clusters from some high-dimensional clustering algorithm, this function generates a dataframe of statistics.</td>
</tr>
<tr>
<td>experiment_statistics(experiment[, ...])</td>
<td>Given an Experiment, generate a Pandas DataFrame detailing statistics for every population captured in all FileGroups contained within the Experiment.</td>
</tr>
</tbody>
</table>

continues on next page
Table 162 – continued from previous page

**l1_feature_selection**(feature_space, ...[, ...])

Perform L1 regularised classification over a defined search space for the L1 parameter and plot the coefficient of each feature in respect to the change in L1 parameter.

- **param feature_space**: A dataframe of features where each column is a feature, each row a subject, and a column whom’s name is equal to the value of the label argument is the target label for prediction.

  - **type feature_space**: Pandas.DataFrame

- **param features**: List of features to include in model.

  - **type features**: List

- **param label**: The target label to predict.

  - **type label**: str

- **param scale**: if True, features are scaled (standard scale) prior to analysis.

  - **type scale**: bool, (default=True)

- **param search_space**: Search range for L1 parameter

  - **type search_space**: tuple, (default=(-2, 0, 50))

- **param model**: Must be a Scikit-Learn classifier that accepts an L1 regularisation parameter named ‘C’. If left as None, a linear SVM is used.

  - **type model**: callable, optional

  - **param figsize**: tuple, (default=(10,5)).

**meta_labelling**(experiment, meta_label)

Given a Pandas DataFrame containing a column of sample IDs from an Experiment (column should be named ‘sample_id’) search the related Subject of the samples and create a new column for the chosen ‘meta_label’ contained in the related Subject.

**population_stats**(filegroup)

Given a FileGroup generate a DataFrame detailing the number of events, proportion of parent population, and proportion of total (root population) for each population in the FileGroup.

**radar_plot**(summary, features[, figsize])

Given a Pandas DataFrame where columns are features and each row is a different subject (indexed by a column named ‘subject_id’), generate a radar plot of all the features.

- **param summary**: Pandas.DataFrame

- **param features**: Features to be included in the plot.

- **param figsize**: tuple, (default=(10,10))

**sort_variance**(summary, identifier_columns[, ...])

Given a dataframe generated by one of the many functions in this module, sort that dataframe by variance.

- **param summary**: Pandas.DataFrame

- **param identifier_columns**: A list of column names to sort by.

**warn**(message[, category, stacklevel, source])

Issue a warning, or maybe ignore it or raise an exception.

---

**CytoPy.flow.feature_extraction.cluster_statistics**(experiment:

- **CytoPy.data.experiment.Experiment**, population: str = None, meta_label: str = None, tag: str = None, include_subject_id: bool = True)

Given an Experiment and the name of a Population known to contain clusters from some high-dimensional clustering algorithm, this function generates a dataframe of statistics. Details include the number of events within the cluster and what proportion of the total events in the Population this number represents.

**Parameters**

- **experiment (Experiment)** –

- **population (str (optional))** – If not population is provided, will search all possible populations for clusters
- **meta_label** *(str (optional)) - If given, will filter results to include only those clusters with this meta ID*

- **tag** *(str (optional)) - If given, will filter results to include only those clusters with this tag*

- **include_subject_id** *(bool (default=True)) - If True, includes a column for the subject ID in the resulting dataframe*

**Returns**

**Return type** Pandas.DataFrame

```python
CytoPy.flow.feature_extraction.experiment_statistics(experiment: CytoPy.data.experiment.Experiment, include_subject_id: bool = True)
```

Given an Experiment, generate a Pandas DataFrame detailing statistics for every population captured in all FileGroups contained within the Experiment.

**Parameters**

- **experiment** *(Experiment)*

- **include_subject_id** *(bool (default=True))*

**Returns**

**Return type** Pandas.DataFrame

```python
CytoPy.flow.feature_extraction.ll_feature_selection(feature_space: pandas.core.frame.DataFrame, features: list, label: str, scale: bool = True, search_space: tuple = (-2, 0, 50), model: callable = None, figsize: tuple = (10, 5))
```

Perform L1 regularised classification over a defined search space for the L1 parameter and plot the coefficient of each feature in respect to the change in L1 parameter.  

- **param feature_space**: A dataframe of features where each column is a feature, each row a subject, and a column whom’s name is equal to the value of the label argument is the target label for prediction

**Parameters**

- **features** *(List)* - List of features to include in model

- **label** *(str)* - The target label to predict

- **scale** *(bool, (default=True)) - if True, features are scaled (standard scale) prior to analysis

- **search_space** *(tuple, (default=(-2, 0, 50))) - Search range for L1 parameter

- **model** *(callable, optional)* - Must be a Scikit-Learn classifier that accepts an L1 regularisation parameter named ‘C’. If left as None, a linear SVM is used

- **figsize** *(tuple, (default=(10,5))) -

**Returns**

**Return type** Matplotlib.axes
CytoPy, Release 1.0.0

CytoPy.flow.feature_extraction.meta_labelling(experiment: CytoPy.data.experiment.Experiment,
dataframe: pandas.core.frame.DataFrame, meta_label: str)

Given a Pandas DataFrame containing a column of sample IDs from an Experiment (column should be named ‘sample_id’) search the related Subject of the samples and create a new column for the chosen ‘meta_label’ contained in the related Subject. If a sample does not have a related Subject or the ‘meta_label’ cannot be found, the row will be populated with None. Returns a mutated DataFrame.

Parameters

- **experiment** (Experiment)
- **dataframe** (Pandas.DataFrame)
- **meta_label** (str)

Returns

Return type  Pandas.DataFrame

CytoPy.flow.feature_extraction.radar_plot(summary: pandas.core.frame.DataFrame, features: list, figsize: tuple = 10, 10)

Given a Pandas DataFrame where columns are features and each row is a different subject (indexed by a column named ‘subject_id’), generate a radar plot of all the features.

:param features: Features to be included in the plot :type features: List :param figsize: :type figsize: tuple, (default=(10,10))

Returns

Return type  Matplotlib.axes

CytoPy.flow.feature_extraction.sort_variance(summary: pandas.core.frame.DataFrame, identifier_columns: list, value_name: str = 'summary_stat', var_name: str = 'population')

Given a dataframe generated by one of the many functions in this module, sort that dataframe by variance.

Parameters

- **summary** (Pandas.DataFrame) – Dataframe of summary statistics
- **identifier_columns** (list) – Columns to use as identifier(s) e.g. sample_id
- **value_name** (str (default="summary_stat"))–
- **var_name** (str (default="population"))–

Returns

Return type  Pandas.DataFrame
8.23 CytoPy.flow.transforms

Cytometry data has to be transformed prior to analysis. There are multiple techniques for transformation of data, the most popular being the biexponential transform. CytoPy employs multiple methods using the FlowUtils package (https://github.com/whitews/FlowUtils), including the Logicle transform, a modified version of the biexponential transform.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinMaxScaler(**kwargs)</td>
<td>Transform features by scaling each feature to a given range.</td>
</tr>
<tr>
<td>PowerTransformer(**kwargs)</td>
<td>Apply a power transform featurewise to make data more Gaussian-like.</td>
</tr>
<tr>
<td>RobustScaler(**kwargs)</td>
<td>Scale features using statistics that are robust to outliers.</td>
</tr>
<tr>
<td>StandardScaler(**kwargs)</td>
<td>Standardize features by removing the mean and scaling to unit variance.</td>
</tr>
</tbody>
</table>

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_transform(data[, . . .])</td>
<td>Apply a transformation to the given dataframe. The features_to_transform specified which columns in the dataframe to transform. This can be given as: * a string value of either ‘all’ or ‘fluorochromes’; transform_method defines which transform to apply to columns * a list of columns to transform; transform_method defines which transform to apply to columns * alternatively, a dictionary where the key is the column name and the value is the transform method to apply to this column; transform_method is ignored.</td>
</tr>
<tr>
<td>asinh(data, columns, pre_scale)</td>
<td>return asinh transformed points (after pre-scaling) for indices listed</td>
</tr>
<tr>
<td>hyperlog(data, channels[, t, m, w, a])</td>
<td>return hyperlog transformed points for channels listed</td>
</tr>
<tr>
<td>individual_transforms(data, **kwargs)</td>
<td>Given a Pandas DataFrame and a dictionary of transformations to apply, where the key is the column to transform and the value the method for transformation, apply transforms to each specified column.</td>
</tr>
<tr>
<td>logicle(data, channels[, t, m, r, w, a, r_quant])</td>
<td>return logicle transformed points for channels listed</td>
</tr>
<tr>
<td>percentile_rank_transform(data, ...)</td>
<td>Calculate percentile rank transform of data-frame. continues on next page</td>
</tr>
</tbody>
</table>
Table 164 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scaler(data, scale_method[, return_scaler])</code></td>
<td>Wrapper for Sklearn transformation methods</td>
</tr>
</tbody>
</table>

CytoPy.flow.transforms.apply_transform

```python
apply_transform(data: pandas.core.frame.DataFrame, features_to_transform: list = 'all', transform_method: str = 'logicle', **kwargs) → pandas.core.frame.DataFrame
```

Apply a transformation to the given dataframe. The `features_to_transform` specified which columns in the dataframe to transform. This can be given as:

- a string value of either ‘all’ or ‘fluorochromes’; `transform_method` defines which transform to apply to columns
- a list of columns to transform; `transform_method` defines which transform to apply to columns
- alternatively, a dictionary where the key is the column name and the value is the transform method to apply to this column; `transform_method` is ignored

**Parameters**

- `data` (Pandas.DataFrame)
- `features_to_transform` (list or str or dict (default="all"))
- `transform_method` (str or None)

**Returns**

Return type: Pandas.DataFrame

CytoPy.flow.transforms.individual_transforms

```python
individual_transforms(data: pandas.core.frame.DataFrame, transforms: dict, **kwargs)
```

Given a Pandas DataFrame and a dictionary of transformations to apply, where the key is the column to transform and the value the method for transformation, apply transforms to each specified column.

**Parameters**

- `data` (Pandas.DataFrame)
- `transforms` (dict)
- `kwargs` - Additional keyword arguments passed to transform function

**Returns**

Return type: Pandas.DataFrame

CytoPy.flow.transforms.percentile_rank_transform

```python
percentile_rank_transform(data: pandas.core.frame.DataFrame, features_to_transform: list) → pandas.core.frame.DataFrame
```

Calculate percentile rank transform of data-frame. Each event is ranked as the average according to the column, then divided by the total number of events and multiplied by 100 to give the percentile.

**Parameters**

- `data` (Pandas.DataFrame) - Pandas DataFrame of events
- `features_to_transform` (list) - features to perform transformation on

**Returns**

Transformed DataFrame

Return type: Pandas.DataFrame
8.24 CytoPy.flow.tree

CytoPy tracks the population “tree” of a FileGroup when a FileGroup is loaded into memory and is being analysed. This module handles the creation and modification of this “tree” using the anytree library.

Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Classes:

| Population(*args, **kwargs) | A population of cells identified by either a gate or supervised algorithm. |

Functions:

| construct_tree(populations) | Given a list of populations, construct a tree of population hierarchy using the population parent information. |


Given a list of populations, construct a tree of population hierarchy using the population parent information.

Parameters populations (List[Population]) – List of Population objects

Returns Dictionary of Node objects
8.25 CytoPy.feedback

Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_ipython()</code></td>
<td>Get the global InteractiveShell instance.</td>
</tr>
<tr>
<td><code>progress_bar(x[, verbose])</code></td>
<td>Generate a progress bar using the tqdm library. If execution environment is Jupyter, return tqdm_notebook otherwise used tqdm.</td>
</tr>
<tr>
<td><code>tqdm_notebook(*args, **kwargs)</code></td>
<td>See tqdm.notebook.tqdm for full documentation</td>
</tr>
<tr>
<td><code>vprint(verbose)</code></td>
<td>Utility function for optional printing.</td>
</tr>
<tr>
<td><code>which_environment()</code></td>
<td>Test if module is being executed in the Jupyter environment.</td>
</tr>
</tbody>
</table>

Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tqdm(*args, **kwargs)</code></td>
<td>Decorate an iterable object, returning an iterator which acts exactly like the original iterable, but prints a dynamically updating progressbar every time a value is requested.</td>
</tr>
</tbody>
</table>

CytoPy.feedback.progress_bar(x: iter, verbose: bool = True, **kwargs) → callable

Generate a progress bar using the tqdm library. If execution environment is Jupyter, return tqdm_notebook otherwise used tqdm.

Parameters

- `x (iterable)` – some iterable to pass to tqdm function
- `verbose (bool, (default=True))` – Provide feedback (if False, no progress bar produced)
- `**kwargs` – additional keyword arguments for tqdm

?param : :type : return: tqdm or tqdm_notebook, depending on environment

CytoPy.feedback.vprint(verbosel bool)

Utility function for optional printing.

Parameters `verbose (bool)` – If True, returns print function, else False

Returns

Return type callable

CytoPy.feedback.which_environment() → str

Test if module is being executed in the Jupyter environment.

Returns ‘jupyter’, ‘ipython’ or ‘terminal’

Return type str
Copyright 2020 Ross Burton

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
INDICES AND TABLES

• genindex
• modindex
• search
C
CytoPy.data.experiment, 68
CytoPy.data.fcs, 57
CytoPy.data.gate, 86
CytoPy.data.gating_strategy, 127
CytoPy.data.geometry, 142
CytoPy.data.mapping, 149
CytoPy.data.population, 151
CytoPy.data.project, 158
CytoPy.data.read_write, 184
CytoPy.data.setup, 141
CytoPy.data.subject, 166
CytoPy.data.supervised_classifier, 187
CytoPy.feedback, 260
CytoPy.flow.clustering.consensus, 226
CytoPy.flow.clustering.flowsom, 223
CytoPy.flow.dim_reduction, 251
CytoPy.flow.feature_extraction, 253
CytoPy.flow.neighbours, 233
CytoPy.flow.plotting, 235
CytoPy.flow.ref, 240
CytoPy.flow.sampling, 242
CytoPy.flow.supervised, 247
CytoPy.flow.transforms, 257
CytoPy.flow.tree, 259
CytoPy.flow.variance, 227
INDEX

A

Action (class in CytoPy.data.gating_strategy), 128
action_name (CytoPy.data.gating_strategy.Action attribute), 128
add_child() (CytoPy.data.gate.EllipseGate method), 96
add_child() (CytoPy.data.gate.PolygonGate method), 111
add_child() (CytoPy.data.gate.ThresholdGate method), 119
add_cluster() (CytoPy.data.population.Population method), 155
add_ctrl_file() (CytoPy.data.fcs.FileGroup method), 60
add_experiment() (CytoPy.data.project.Project method), 160
add_hyperparameter_grid() (CytoPy.data.gating_strategy.GatingStrategy method), 133
add_new_sample() (CytoPy.data.experiment.Experiment method), 71
add_population() (CytoPy.data.fcs.FileGroup method), 61
add_subject() (CytoPy.data.project.Project method), 161
apply_action() (CytoPy.data.gating_strategy.GatingStrategy method), 133
apply_all() (CytoPy.data.gating_strategy.GatingStrategy method), 134
apply_gate() (CytoPy.data.gating_strategy.GatingStrategy method), 134
apply_threshold() (in module CytoPy.data.gate), 124
apply_transform() (in module CytoPy.flow.transforms), 258
args (CytoPy.data.experiment.Experiment.DoesNotExist attribute), 71
args (CytoPy.data.experiment.Experiment.MultipleObjectsReturned attribute), 81
args (CytoPy.data.experiment.Panel.DoesNotExist attribute), 81
args (CytoPy.data.experiment.Panel.MultipleObjectsReturned attribute), 81
args (CytoPy.data.fcs.FileGroup.DoesNotExist attribute), 60
args (CytoPy.data.fcs.FileGroup.MultipleObjectsReturned attribute), 60
args (CytoPy.data.gate.EllipseGate.DoesNotExist attribute), 96
args (CytoPy.data.gate.EllipseGate.MultipleObjectsReturned attribute), 96
args (CytoPy.data.gate.Gate.DoesNotExist attribute), 103
args (CytoPy.data.gate.Gate.MultipleObjectsReturned attribute), 104
args (CytoPy.data.gate.PolygonGate.DoesNotExist attribute), 111
args (CytoPy.data.gate.PolygonGate.MultipleObjectsReturned attribute), 111
args (CytoPy.data.gate.ThresholdGate.DoesNotExist attribute), 119
args (CytoPy.data.gate.ThresholdGate.MultipleObjectsReturned attribute), 119
args (CytoPy.data.gating_strategy.GatingStrategy.DoesNotExist attribute), 133
args (CytoPy.data.gating_strategy.GatingStrategy.MultipleObjectsReturned attribute), 133
args (CytoPy.data.project.Project.DoesNotExist attribute), 160
args (CytoPy.data.project.Project.MultipleObjectsReturned attribute), 160
args (CytoPy.data.subject.MetaDataDictionary.DoesNotExist attribute), 173
args (CytoPy.data.subject.MetaDataDictionary.MultipleObjectsReturned attribute), 173
args (CytoPy.data.subject.Subject.DoesNotExist attribute), 179
args (CytoPy.data.subject.Subject.MultipleObjectsReturned attribute), 179
args (CytoPy.data.supervised_classifier.CellClassifier.DoesNotExist attribute), 191
args (CytoPy.data.supervised_classifier.CellClassifier.MultipleObjectsReturned attribute), 191
args (CytoPy.data.experiment.Panel.DoesNotExist attribute), 81
args (CytoPy.data.experiment.Panel.MultipleObjectsReturned attribute), 81
args (CytoPy.data.fcs.FileGroup.DoesNotExist attribute), 60
args (CytoPy.data.fcs.FileGroup.MultipleObjectsReturned attribute), 60
args (CytoPy.data.gate.EllipseGate.DoesNotExist attribute), 96
args (CytoPy.data.gate.EllipseGate.MultipleObjectsReturned attribute), 96
args (CytoPy.data.gate.Gate.DoesNotExist attribute), 103
args (CytoPy.data.gate.Gate.MultipleObjectsReturned attribute), 104
args (CytoPy.data.gate.PolygonGate.DoesNotExist attribute), 111
args (CytoPy.data.gate.PolygonGate.MultipleObjectsReturned attribute), 111
args (CytoPy.data.gate.ThresholdGate.DoesNotExist attribute), 119
args (CytoPy.data.gate.ThresholdGate.MultipleObjectsReturned attribute), 119
args (CytoPy.data.gating_strategy.GatingStrategy.DoesNotExist attribute), 133
args (CytoPy.data.gating_strategy.GatingStrategy.MultipleObjectsReturned attribute), 133
args (CytoPy.data.project.Project.DoesNotExist attribute), 160
args (CytoPy.data.project.Project.MultipleObjectsReturned attribute), 160
args (CytoPy.data.subject.MetaDataDictionary.DoesNotExist attribute), 173
args (CytoPy.data.subject.MetaDataDictionary.MultipleObjectsReturned attribute), 173
args (CytoPy.data.subject.Subject.DoesNotExist attribute), 179
args (CytoPy.data.subject.Subject.MultipleObjectsReturned attribute), 179
args (CytoPy.data.supervised_classifier.CellClassifier.DoesNotExist attribute), 191
args (CytoPy.data.supervised_classifier.CellClassifier.MultipleObjectsReturned attribute), 191

267
attribute), 191
args (CytoPy.data.supervised_classifier.KerasCellClassifier.DoesNotExist), 201
calculate_optimal_neighbours() (in module CytoPy.flow.supervised), 234
calculate_ref_sample() (in module CytoPy.data.experiment), 231
cascade_save() (CytoPy.data.experiment.Experiment method), 214

args (CytoPy.data.supervised_classifier.KerasCellClassifier.MultipleObjectsReturned), 201
assert_population_labels() (in module CytoPy.flow.supervised), 249
auto_class_weights() (CytoPy.data.supervised_classifier.CellClassifier method), 191
auto_class_weights() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 201
auto_class_weights() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 214
auto_class_weights() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 214
auto_class_weights() (in module CytoPy.data.experiment.Panel method), 81
cascade_save() (CytoPy.data.fcs.FileGroup method), 61
cascade_save() (CytoPy.data.gate.EllipseGate method), 96
cascade_save() (CytoPy.data.gate.Gate method), 104
cascade_save() (CytoPy.data.gate.PolygonGate method), 111
cascade_save() (CytoPy.data.gate.ThresholdGate method), 119
cascade_save() (CytoPy.data.gating_strategy.GatingStrategy method), 135
cascade_save() (CytoPy.data.project.Project method), 161
cascade_save() (CytoPy.data.subject.MetaDataDictionary method), 173
cascade_save() (CytoPy.data.subject.Subject method), 179
cascade_save() (CytoPy.data.supervised_classifier.CellClassifier method), 191
cascade_save() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 202
cascade_save() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 214
case_sensitive (CytoPy.data.experiment.NormalisedName attribute), 78
CellClassifier (class in CytoPy.data.experiment.Panel attribute), 79
CellClassifier.DoesNotExist, 191
CellClassifier.MultipleObjectsReturned, 191
channel (CytoPy.data.mapping.ChannelMap attribute), 150
ChannelMap (class in CytoPy.data.mapping), 149
check_downstream_populations() (in module CytoPy.flow.supervised), 250
check_excel_template() (in module CytoPy.data.experiment), 85

B

backgate() (CytoPy.flow.plotting.CreatePlot method), 237
bins (CytoPy.flow.plotting.CreatePlot attribute), 236
Biology (class in CytoPy.data.subject), 166
biology() (in module CytoPy.data.subject), 183
Bugs (class in CytoPy.data.subject), 168
bugs() (in module CytoPy.data.subject), 183
build_keras_model() (in module CytoPy.flow.supervised), 249
build_model() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 202
build_model() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 214
build_sklearn_model() (in module CytoPy.flow.supervised), 249
bw (CytoPy.flow.plotting.CreatePlot attribute), 237
bw_optimisation() (in module CytoPy.flow.variance), 230

C
calc_metrics() (in module CytoPy.flow.supervised), 249
check_matched_pair() (CytoPy.data.mapping.ChannelMap method), 191

Child ((class in CytoPy.data.gate), 89)
ChildPolygon ((class in CytoPy.data.gate), 90)
ChildThreshold ((class in CytoPy.data.gate), 92)

chunks() (in module CytoPy.data.read_write), 185

class_weights (CytoPy.data.supervised_classifier.CellClassifier attribute), 189

class_weights (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 199

class_weights (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 212

clean() (CytoPy.data.experiment.Experiment method), 72

clean() (CytoPy.data.experiment.NormalisedName method), 78

clean() (CytoPy.data.experiment.Panel method), 81

clean() (CytoPy.data.fcs.FileGroup method), 61

clean() (CytoPy.data.gate.Child method), 89

clean() (CytoPy.data.gate.ChildPolygon method), 91

clean() (CytoPy.data.gate.ChildThreshold method), 92

clean() (CytoPy.data.gate.EllipseGate method), 96

clean() (CytoPy.data.gate.Gate method), 104

clean() (CytoPy.data.gate.PolygonGate method), 111

clean() (CytoPy.data.gate.ThresholdGate method), 119

clean() (CytoPy.data.gating_strategy.Action method), 129

clean() (CytoPy.data.gating_strategy.GatingStrategy method), 135

clean() (CytoPy.data.geometry.PolygonGeom method), 144

clean() (CytoPy.data.geometry.PopulationGeometry method), 145

clean() (CytoPy.data.geometry.ThresholdGeom method), 146

clean() (CytoPy.data.mapping.ChannelMap method), 150

clean() (CytoPy.data.population.Cluster method), 153

clean() (CytoPy.data.population.Population method), 155

clean() (CytoPy.data.project.Project method), 161

clean() (CytoPy.data.subject.Biology method), 167

clean() (CytoPy.data.subject.Bug method), 169

clean() (CytoPy.data.subject.Drug method), 170

clean() (CytoPy.data.subject.MetaDataDictionary method), 173

clean() (CytoPy.data.subject.Subject method), 179

clean() (CytoPy.data.supervised_classifier.CellClassifier method), 191

clean() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 202

clean() (CytoPy.data.supervised_classifier.Layer method), 206

clean() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 214

clean_cache() (CytoPy.flow.variance.SimilarityMatrix method), 230

cmap (CytoPy.flow.variance.CreatePlot attribute), 237

cmap (CytoPy.flow.variance.CreatePlot attribute), 237

cmpare_indexes() (CytoPy.data.experiment.Experiment method), 72

compare_indexes() (CytoPy.data.experiment.NormalisedName method), 78

compare_indexes() (CytoPy.data.experiment.Panel method), 81

compare_indexes() (CytoPy.data.fcs.FileGroup method), 61

compare_indexes() (CytoPy.data.gate.Child method), 89

compare_indexes() (CytoPy.data.gate.ChildPolygon method), 91

compare_indexes() (CytoPy.data.gate.ChildThreshold method), 92

compare_indexes() (CytoPy.data.gate.EllipseGate method), 96

compare_indexes() (CytoPy.data.gate.Gate method), 104

compare_indexes() (CytoPy.data.gate.PolygonGate method), 111

compare_indexes() (CytoPy.data.gate.ThresholdGate method), 119

compare_indexes() (CytoPy.data.gating_strategy.Action method), 129

compare_indexes() (CytoPy.data.gating_strategy.GatingStrategy method), 135

compare_indexes() (CytoPy.data.geometry.PolygonGeom method), 144

compare_indexes() (CytoPy.data.geometry.PopulationGeometry method), 145

compare_indexes() (CytoPy.data.geometry.ThresholdGeom method), 146

compare_indexes() (CytoPy.data.mapping.ChannelMap method), 150

compare_indexes() (CytoPy.data.population.Cluster method), 153

compare_indexes() (CytoPy.data.population.Population method), 155

compare_indexes() (CytoPy.data.project.Project method), 161

compare_indexes() (CytoPy.data.subject.Biology method), 167

compare_indexes() (CytoPy.data.subject.Bug method), 169

compare_indexes() (CytoPy.data.subject.Drug method), 170

compare_indexes() (CytoPy.data.subject.MetaDataDictionary method), 173

compare_indexes() (CytoPy.data.subject.Subject method), 179

compensate() (CytoPy.data.read_write.FCSFile method), 185
compile_kwargs (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 198

confusion_matrix_plots() (in module CytoPy.flow.supervised), 250

ConsensusCluster (class in CytoPy.flow.clustering.consensus), 226

construct_tree() (in module CytoPy.tree), 259

create_convex_hull() (in module CytoPy.data.geometry), 147

create_from_dict() (CytoPy.data.experiment.Panel method), 81

create_from_excel() (CytoPy.data.experiment.Panel method), 81

create_index() (CytoPy.data.experiment.Experiment class method), 72

create_index() (CytoPy.data.experiment.Panel class method), 81

create_index() (CytoPy.data.fcs.FileGroup class method), 61

create_index() (CytoPy.data.gate.EllipseGate class method), 96

create_index() (CytoPy.data.gate.Gate class method), 104

create_index() (CytoPy.data.gate.PolygonGate class method), 111

create_index() (CytoPy.data.gate.ThresholdGate class method), 119

create_index() (CytoPy.data.gating_strategy.GatingStrategy class method), 135

create_index() (CytoPy.data.project.Project class method), 161

create_index() (CytoPy.data.subject.MetaDataDictionary class method), 173

create_index() (CytoPy.data.subject.Subject class method), 179

create_index() (CytoPy.data.supervised_classifier.CellClassifier class method), 191

create_index() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

create_index() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 214

create_polygon() (in module CytoPy.data.geometry), 147

create_ref_sample() (in module CytoPy.flow.ref), 241

create_signature() (in module CytoPy.data.population), 156

cREATEPLOT (class in CytoPy.flow.plotting), 236

creation_date (CytoPy.data.gating_strategy.GatingStrategy attribute), 130

culture_source (CytoPy.data.subject.Bug attribute), 168

CytoPy.data.experiment module, 68

CytoPy.data.fcs module, 57

CytoPy.data.gate module, 86

CytoPy.data.geometry module, 142

CytoPy.data.mapping module, 149

CytoPy.data.population module, 151

CytoPy.data.project module, 158

CytoPy.data.read_write module, 184

CytoPy.data.setup module, 141

CytoPy.data.subject module, 166

CytoPy.data.supervised_classifier module, 187

CytoPy.feedback module, 260

CytoPy.flow.clustering.consensus module, 226

CytoPy.flow.clustering.flowsom module, 223

CytoPy.flow.dim_reduction module, 251

CytoPy.flow.feature_extraction module, 253

CytoPy.flow.neighbours module, 233

CytoPy.flow.plotting module, 235

CytoPy.flow.ref module, 240

CytoPy.flow.sampling module, 242

CytoPy.flow.supervised module, 247
<table>
<thead>
<tr>
<th>CytoPy.flow.transforms module</th>
<th>257</th>
</tr>
</thead>
<tbody>
<tr>
<td>CytoPy.flow.tree module</td>
<td>259</td>
</tr>
<tr>
<td>CytoPy.flow.variance module</td>
<td>227</td>
</tr>
</tbody>
</table>

**D**

- `data` (CytoPy.flow.variance.SimilarityMatrix attribute), 229
- `data()` (CytoPy.data.fcs.FileGroup method), 61
- `definition` (CytoPy.data.gate.ChildThreshold attribute), 92
- `delete()` (CytoPy.data.experiment.Experiment method), 72
- `delete()` (CytoPy.data.experiment.Panel method), 81
- `delete()` (CytoPy.data.fcs.FileGroup method), 61
- `delete()` (CytoPy.data.gate.EllipseGate method), 97
- `delete()` (CytoPy.data.gate.Gate method), 104
- `delete()` (CytoPy.data.gate.PolygonGate method), 111
- `delete()` (CytoPy.data.gate.ThresholdGate method), 119
- `delete()` (CytoPy.data.gating_strategy.GatingStrategy method), 135
- `delete()` (CytoPy.data.project.Project method), 161
- `delete()` (CytoPy.data.subject.MetaDataDictionary method), 173
- `delete()` (CytoPy.data.subject.Subject method), 179
- `delete()` (CytoPy.data.supervised_classifier.CellClassifier method), 191
- `delete()` (CytoPy.data.supervised_classifier.KerasCellClassifier method), 202
- `delete()` (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 215
- `delete_actions()` (CytoPy.data.gating_strategy.GatingStrategy method), 135
- `delete_clusters()` (CytoPy.data.population.Population method), 155
- `delete_all_clusters()` (CytoPy.data.experiment.Experiment method), 155
- `delete_all_populations()` (CytoPy.data.population.Population method), 72
- `delete_cluster()` (CytoPy.data.population.Population method), 155
- `delete_clusters()` (CytoPy.data.fcs.FileGroup method), 62
- `delete_gate()` (CytoPy.data.gating_strategy.GatingStrategy method), 135
- `delete_populations()` (CytoPy.data.fcs.FileGroup method), 62
- `delete_populations()` (CytoPy.data.gating_strategy.GatingStrategy method), 135
- `density_dependent_downsampling()` (in module CytoPy.flow.sampling), 243
- `density_probability_assignment()` (in module CytoPy.flow.sampling), 244
- `desc` (CytoPy.data.subject.MetaDataDictionary attribute), 171
- `dim_reduction` (CytoPy.data.gate.EllipseGate attribute), 94
- `dim_reduction` (CytoPy.data.gate.Gate attribute), 102
- `dim_reduction` (CytoPy.data.gate.PolygonGate attribute), 109
- `dim_reduction` (CytoPy.data.gate.ThresholdGate attribute), 117
- `dim_reduction_grid()` (in module CytoPy.flow.variance), 231
- `dimensionality_reduction()` (in module CytoPy.flow.dim_reduction), 252
- `downsample` (CytoPy.data.supervised_classifier.CellClassifier attribute), 188
- `downsample` (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 199
- `downsample` (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 212
- `downsample_kwargs` (CytoPy.data.supervised_classifier.CellClassifier attribute), 189
- `downsample_kwargs` (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 199
- `drop_collection()` (CytoPy.data.experiment.Experiment class method), 73
- `drop_collection()` (CytoPy.data.experiment.Panel class method), 82
- `drop_collection()` (CytoPy.data.fcs.FileGroup class method), 62
- `drop_collection()` (CytoPy.data.gate.EllipseGate class method), 97
- `drop_collection()` (CytoPy.data.gate.Gate class method), 104
- `drop_collection()` (CytoPy.data.gate.PolygonGate class method), 111
- `drop_collection()` (CytoPy.data.gate.ThresholdGate class method), 120
- `drop_collection()` (CytoPy.data.gating_strategy.GatingStrategy method), 135
drop_collection() (CytoPy.data.project.Project class method), 161

drop_collection() (CytoPy.data.subject.MetaDataDictionary class method), 173

drop_collection() (CytoPy.data.subject.Subject class method), 180

drop_collection() (CytoPy.data.supervised_classifier.CellClassifier class method), 191

drop_collection() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

drop_collection() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

drop_collection() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

drop_collection() (CytoPy.data.gate.EllipseGate class), 93

Drug (class in CytoPy.data.subject), 170
drug_data (CytoPy.data.subject.Subject attribute), 177

delete() (CytoPy.data.project.Project class method), 161

drop() (CytoPy.data.subject.MetaDataDictionary class method), 173

drop() (CytoPy.data.subject.Subject class method), 180

drop() (CytoPy.data.supervised_classifier.CellClassifier class method), 191

drop() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

drop() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

drop() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

drop() (CytoPy.data.gate.EllipseGate class), 93

drop() (CytoPy.data.gate.Gate class), 104

drop() (CytoPy.data.gate.PolygonGate class), 112

drop() (CytoPy.data.gate.ThresholdGate class), 120

drop() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

drop() (CytoPy.data.project.Project class method), 161

drop() (CytoPy.data.subject.MetaDataDictionary class method), 173

drop() (CytoPy.data.subject.Subject class method), 180

drop() (CytoPy.data.supervised_classifier.CellClassifier class method), 191

drop() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

drop() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

drop() (CytoPy.data.experiment.Experiment class method), 73

drop() (CytoPy.data.experiment.Panel class method), 82

drop() (CytoPy.data.fcs.FileGroup class method), 62

drop() (CytoPy.data.gate.EllipseGate class method), 97

drop() (CytoPy.data.gate.Gate class method), 104

drop() (CytoPy.data.gate.PolygonGate class method), 112

drop() (CytoPy.data.gate.ThresholdGate class method), 120

drop() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

drop() (CytoPy.data.project.Project class method), 162

drop() (CytoPy.data.subject.MetaDataDictionary class method), 174

drop() (CytoPy.data.subject.Subject class method), 180

drop() (CytoPy.data.supervised_classifier.CellClassifier class method), 192

drop() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

drop() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

drop() (CytoPy.data.experiment.Experiment class method), 73

drop() (CytoPy.data.experiment.Panel class method), 82

drop() (CytoPy.data.fcs.FileGroup class method), 62

drop() (CytoPy.data.gate.EllipseGate class method), 97

drop() (CytoPy.data.gate.Gate class method), 104

drop() (CytoPy.data.gate.PolygonGate class method), 112

drop() (CytoPy.data.gate.ThresholdGate class method), 120

drop() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

drop() (CytoPy.data.project.Project class method), 162

drop() (CytoPy.data.subject.MetaDataDictionary class method), 174

E
eat_gate() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

ellipse_to_polygon() (in module CytoPy.data.geometry), 148

EllipseGate (class in CytoPy.data.gate), 93

EllipseGate.DoesNotExist, 96

EllipseGate.MultipleObjectsReturned, 96

drop() (CytoPy.data.project.Project class method), 161

drop() (CytoPy.data.subject.MetaDataDictionary class method), 173

drop() (CytoPy.data.subject.Subject class method), 180

drop() (CytoPy.data.supervised_classifier.CellClassifier class method), 191

drop() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

drop() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

drop() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

drop() (CytoPy.data.project.Project class method), 162

drop() (CytoPy.data.subject.MetaDataDictionary class method), 174

drop() (CytoPy.data.subject.Subject class method), 180

drop() (CytoPy.data.supervised_classifier.CellClassifier class method), 192

drop() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

drop() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

ease_indexes() (CytoPy.data.experiment.Experiment class method), 73

ensure_index() (CytoPy.data.subject.Subject class method), 180

ensure_index() (CytoPy.data.subject.MetaDataDictionary class method), 174

ensure_index() (CytoPy.data.subject.Subject class method), 180

ensure_index() (CytoPy.data.supervised_classifier.CellClassifier class method), 192

ensure_index() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

ensure_index() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

ensure_indexes() (CytoPy.data.experiment.Experiment class method), 73

ensure_indexes() (CytoPy.data.experiment.Panel class method), 82

ensure_indexes() (CytoPy.data.fcs.FileGroup class method), 62

ensure_indexes() (CytoPy.data.gate.EllipseGate class method), 97

ensure_indexes() (CytoPy.data.gate.Gate class method), 104

ensure_indexes() (CytoPy.data.gate.PolygonGate class method), 112

ensure_indexes() (CytoPy.data.gate.ThresholdGate class method), 120

ensure_indexes() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

ensure_indexes() (CytoPy.data.project.Project class method), 162

ensure_indexes() (CytoPy.data.subject.MetaDataDictionary class method), 174

ensure_indexes() (CytoPy.data.subject.Subject class method), 180

ensure_indexes() (CytoPy.data.supervised_classifier.CellClassifier class method), 192

ensure_indexes() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

ensure_indexes() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

ensure_indexes() (CytoPy.data.experiment.Experiment class method), 73

ensure_indexes() (CytoPy.data.experiment.Panel class method), 82

ensure_indexes() (CytoPy.data.fcs.FileGroup class method), 62

ensure_indexes() (CytoPy.data.gate.EllipseGate class method), 97

ensure_indexes() (CytoPy.data.gate.Gate class method), 104

ensure_indexes() (CytoPy.data.gate.PolygonGate class method), 112

ensure_indexes() (CytoPy.data.gate.ThresholdGate class method), 120

ensure_indexes() (CytoPy.data.gating_strategy.GatingStrategy class method), 136

ensure_indexes() (CytoPy.data.project.Project class method), 162

ensure_indexes() (CytoPy.data.subject.MetaDataDictionary class method), 174

ensure_indexes() (CytoPy.data.subject.Subject class method), 180

ensure_indexes() (CytoPy.data.supervised_classifier.CellClassifier class method), 192

ensure_indexes() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

ensure_indexes() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

estimate_ctrl_population() (CytoPy.data.fcs.FileGroup method), 62

Experiment (class in CytoPy.data.experiment), 69

Experiment.DoesNotExist, 71

Experiment.MultipleObjectsReturned, 71

experiment_id (CytoPy.data.experiment.Experiment attribute), 69

experiment_statistics() (in module CytoPy.data.fcs.FileGroup method), 62

experiment_statistics() (in module CytoPy.data.gate.EllipseGate class method), 97

experiment_statistics() (in module CytoPy.data.gate.Gate class method), 104

experiment_statistics() (in module CytoPy.data.gate.PolygonGate class method), 112

experiment_statistics() (in module CytoPy.data.gate.ThresholdGate class method), 120

experiment_statistics() (in module CytoPy.data.gating_strategy.GatingStrategy class method), 136

experiment_statistics() (in module CytoPy.data.project.Project class method), 162

experiment_statistics() (in module CytoPy.data.subject.MetaDataDictionary class method), 174

experiment_statistics() (in module CytoPy.data.subject.Subject class method), 180

experiment_statistics() (in module CytoPy.data.supervised_classifier.CellClassifier class method), 192

experiment_statistics() (in module CytoPy.data.supervised_classifier.KerasCellClassifier class method), 202

experiment_statistics() (in module CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 215

Index
toPy.flow.feature_extraction), 255
experiments (CytoPy.data.project.Project attribute), 158
explore_channel_mappings() (in module CytoPy.data.read_write), 185

faithful_downsampling() (in module CytoPy.flow.sampling), 244
fcs_files (CytoPy.data.experiment.Experiment attribute), 69
fcs_mappings() (in module CytoPy.data.read_write), 186
FCSFile (class in CytoPy.data.read_write), 185
feature (CytoPy.data.supervised_classifier.CellClassifier attribute), 188
feature (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 198
feature (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 211
figsize (CytoPy.flow.plotting.CreatePlot attribute), 236
FileGroup (class in CytoPy.data.fcs), 58
FileGroup.DoesNotExist (class in CytoPy.data.subject), 60
files (CytoPy.data.subject.Subject attribute), 177
filter_fcs_files() (in module CytoPy.data.read_write), 186
find_inflection_point() (in module CytoPy.data.gate), 124
find_local_minima() (in module CytoPy.data.gate), 125
find_peaks() (in module CytoPy.data.gate), 125
fit() (CytoPy.data.gate.EllipseGate method), 97
fit() (CytoPy.data.gate.PolygonGate method), 112
fit() (CytoPy.data.gate.ThresholdGate method), 120
fit() (CytoPy.data.supervised_classifier.CellClassifier method), 192
fit() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 203
fit() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 215
fit() (CytoPy.flow.clustering.consensus.ConsensusCluster method), 227
fit_cv() (CytoPy.data.supervised_classifier.CellClassifier method), 192
fit_cv() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 203
fit_cv() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 215
fit_predict() (CytoPy.data.gate.EllipseGate method), 97
fit_predict() (CytoPy.data.gate.PolygonGate method), 112
fit_train_test_split() (CytoPy.data.supervised_classifier.CellClassifier method), 120
fit_train_test_split() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 193
fit_train_test_split() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 204
fit_train_test_split() (CytoPy.data.gating_strategy.GatingStrategy attribute), 130
FlowSOM (class in CytoPy.flow.clustering.flowsom), 224
 FlorianScale (CytoPy.flow.plotting.CreatePlot attribute), 237
from_json() (CytoPy.data.experiment.NormalisedName class method), 78
from_json() (CytoPy.data.experiment.Panel class method), 82
from_json() (CytoPy.data.fcs.FileGroup class method), 63
from_json() (CytoPy.data.gate.Child class method), 90
from_json() (CytoPy.data.gate.ChildPolygon class method), 91
from_json() (CytoPy.data.gate.ChildThreshold class method), 93
from_json() (CytoPy.data.gate.EllipseGate class method), 98
from_json() (CytoPy.data.gate.Gate class method), 105
from_json() (CytoPy.data.gate.PolygonGate class method), 112
from_json() (CytoPy.data.gate.ThresholdGate class method), 120
from_json() (CytoPy.data.gating_strategy.Action class method), 129
from_json() (CytoPy.data.gating_strategy.GatingStrategy class method), 136
from_json() (CytoPy.data.geometry.PolygonGeom class method), 144
from_json() (CytoPy.data.geometry.PopulationGeometry class method), 145
from_json() (CytoPy.data.geometry.ThresholdGeom class method), 147
from_json() (CytoPy.data.mapping.ChannelMap class method), 150
from_json() (CytoPy.data.population.Cluster class method), 153
from_json()
(CytoPy.data.population.Population
class method), 155
from_json()
(CytoPy.data.project.Project
class method), 162
from_json()
(CytoPy.data.subject.Biology
class method), 167
from_json()
(CytoPy.data.subject.Bug
class method), 169
from_json()
(CytoPy.data.subject.Drug
class method), 171
from_json()
(CytoPy.data.subject.MetaDataDictionary
class method), 174
from_json()
(CytoPy.data.subject.Subject
class method), 180
from_json()
(CytoPy.data.supervised_classifier.CellClassifier
class method), 193
from_json()
(CytoPy.data.supervised_classifier.KerasCellClassifier
class method), 204
from_json()
(CytoPy.data.supervised_classifier.Layer
class method), 210
from_json()
(CytoPy.data.supervised_classifier.SklearnCellClassifier
class method), 217

G
Gate (class in CytoPy.data.gate), 101
Gate.DoesNotExist, 103
Gate.MultipleObjectsReturned, 104
gate_name (CytoPy.data.gate.EllipseGate attribute), 94
gate_name (CytoPy.data.gate.Gate attribute), 101
gate_name (CytoPy.data.gate.PolygonGate attribute), 108
gate_name (CytoPy.data.gate.ThresholdGate attribute), 116
gate_stats()
(in module CytoPy.data.gating_strategy), 141
get_channels()
(CytoPy.data.experiment.Panel
method), 82
get_clusters()
(CytoPy.data.population.Population
method), 156
get_fcs_file_paths()
(in module CytoPy.data.read_write), 186
get_gate()
(CytoPy.data.gating_strategy.GatingStrategy
method), 137
get_markers()
(CytoPy.data.experiment.Panel
method), 83
get_population()
(CytoPy.data.fcs.FileGroup
method), 63
get_population_by_parent()
(CytoPy.data.fcs.FileGroup
method), 63
get_sample()
(CytoPy.data.experiment.Experiment
method), 74
get_sample_mid()
(CytoPy.data.experiment.Experiment
method), 74
get_text_score()
(CytoPy.data.experiment.Experiment
method), 74
get_text_score()
(CytoPy.data.experiment.NormalisedName
method), 79
get_text_score()
(CytoPy.data.experiment.Panel
method), 83
get_text_score()
(CytoPy.data.fcs.FileGroup
method), 83
get_text_score()
(CytoPy.data.gate.Child
method), 90
get_text_score()
(CytoPy.data.gate.ChildPolygon
method), 91
get_text_score()
(CytoPy.data.gate.Threshold
method), 93
get_text_score()
(CytoPy.data.gate.EllipseGate
method), 98
get_text_score()
(CytoPy.data.gate.Gate
method), 105
get_text_score()
(CytoPy.data.gate.PolygonGate
method), 112
get_text_score()
(CytoPy.data.gate.ThresholdGate
method), 121
get_text_score()
(CytoPy.data.gating_strategy.Action
method), 129
get_text_score()
(CytoPy.data.gating_strategy.GatingStrategy
method), 137
get_text_score()
(CytoPy.data.project.Project
method), 162

gate_stats()
(in module CytoPy.data.gating_strategy), 141
get_channels()
(CytoPy.data.experiment.Panel
method), 82
get_clusters()
(CytoPy.data.population.Population
method), 156
get_fcs_file_paths()
(in module CytoPy.data.read_write), 186
get_gate()
(CytoPy.data.gating_strategy.GatingStrategy
method), 137
get_markers()
(CytoPy.data.experiment.Panel
method), 83
get_population()
(CytoPy.data.fcs.FileGroup
method), 63
get_population_by_parent()
(CytoPy.data.fcs.FileGroup
method), 63
get_sample()
(CytoPy.data.experiment.Experiment
method), 74
get_sample_mid()
(CytoPy.data.experiment.Experiment
method), 74
get_text_score()
(CytoPy.data.experiment.Experiment
method), 74
get_text_score()
(CytoPy.data.experiment.NormalisedName
method), 79
get_text_score()
(CytoPy.data.experiment.Panel
method), 83
get_text_score()
(CytoPy.data.fcs.FileGroup
method), 83
get_text_score()
(CytoPy.data.gate.Child
method), 90
get_text_score()
(CytoPy.data.gate.ChildPolygon
method), 91
get_text_score()
(CytoPy.data.gate.Threshold
method), 93
get_text_score()
(CytoPy.data.gate.EllipseGate
method), 98
get_text_score()
(CytoPy.data.gate.Gate
method), 105
get_text_score()
(CytoPy.data.gate.PolygonGate
method), 112
get_text_score()
(CytoPy.data.gate.ThresholdGate
method), 121
get_text_score()
(CytoPy.data.gating_strategy.Action
method), 129
get_text_score()
(CytoPy.data.gating_strategy.GatingStrategy
method), 137
get_text_score()
toPy.data.geometry.PolygonGeom method), 144
get_text_score() (CytoPy.data.geometry.PopulationGeometry method), 146
global_init() (in module CytoPy.data.setup), 142
gram_status() (CytoPy.data.subject.Bug attribute), 168
gram_status() (in module CytoPy.data.patient), 142
hmbpp_ribo() (in module CytoPy.data.subject), 184
hmbpp_status (CytoPy.data.subject.Bug attribute), 168
hyperparameter_tuning() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 217
id_method (CytoPy.data.subject.Bug attribute), 168
index (CytoPy.data.population.Cluster attribute), 152
individual_transforms() (in module CytoPy.flow.transforms), 258
infection_data (CytoPy.data.subject.Subject attribute), 177
init_date (CytoPy.data.subject.Drug attribute), 170
initiation_date (CytoPy.data.experiment.Panel attribute), 79
inside_ellipse() (in module CytoPy.data.geometry), 148
inside_polygon() (in module CytoPy.data.geometry), 148
K
kde_bw (CytoPy.flow.variance.SimilarityMatrix attribute), 229
defailure_kernel (CytoPy.flow.variance.SimilarityMatrix attribute), 229
defailure_norm (CytoPy.flow.variance.SimilarityMatrix attribute), 229
KerasCellClassifier (class in CytoPy.data.supervised_classifier), 198
KerasCellClassifier.DoesNotExist, 201
KerasCellClassifier.MultipleObjectsReturned, 201
KerasCellClassifier.MultipleObjectsReturned, 201
key (CytoPy.data.subject.MetaDataDictionary attribute), 171
klass (CytoPy.data.supervised_classifier.Layer attribute), 209
klass (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 209
knn() (in module CytoPy.flow.neighbours), 234
kwags (CytoPy.data.supervised_classifier.Layer attribute), 209
L
all_feature_selection() (in module CytoPy.flow.feature_extraction), 255
label_children() (CytoPy.data.gate.EllipseGate method), 98
label_children() (CytoPy.data.gate.PolygonGate method), 113
label_children() (CytoPy.data.gate.ThresholdGate method), 121
last_edit (CytoPy.data.gating_strategy.GatingStrategy attribute), 130
Layer (class in CytoPy.data.supervised_classifier), 209
layers (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 198
left (CytoPy.data.gating_strategy.Action attribute), 129
CytoPy, Release 1.0.0

list_clusters() (CytoPy.data.population.Population method), 156
list_downstream_populations() (CytoPy.data.fcs.FileGroup method), 63
list_experiments() (CytoPy.data.project.Project method), 162
list_gated_controls() (CytoPy.data.fcs.FileGroup method), 64
list_gates() (CytoPy.data.gating_strategy.GatingStrategy method), 137
list_indexes() (CytoPy.data.experiment.Experiment method), 74
list_indexes() (CytoPy.data.experiment.Panel class method), 83
list_indexes() (CytoPy.data.gate.EllipseGate class method), 105
list_indexes() (CytoPy.data.gate.PolygonGate class method), 113
list_indexes() (CytoPy.data.gate.ThresholdGate class method), 121
list_indexes() (CytoPy.data.gating_strategy.GatingStrategy class method), 137
list_indexes() (CytoPy.data.project.Project class method), 162
list_indexes() (CytoPy.data.subject.MetaDataDictionary class method), 174
list_indexes() (CytoPy.data.subject.Subject class method), 180
list_indexes() (CytoPy.data.supervised_classifier.CellClassifier class method), 194
list_indexes() (CytoPy.data.supervised_classifier.KerasCellClassifier class method), 204
listIndexes() (CytoPy.data.supervised_classifier.SklearnCellClassifier class method), 217
list_invalid() (CytoPy.data.experiment.Experiment method), 74
load_and_sample() (in module CytoPy.flow.variance), 232
load_model() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 217
load_population_df() (CytoPy.data.fcs.FileGroup method), 64
load_training_data() (CytoPy.data.supervised_classifier.CellClassifier method), 194
load_training_data() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 205
load_validation() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 217
load_validation() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 205
load_validation() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 218
loss (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 198
mappings (CytoPy.data.experiment.Panel attribute), 79
marker (CytoPy.data.mapping.ChannelMap attribute), 150
marker_variance() (in module CytoPy.flow.variance), 232
match_definition() (CytoPy.data.gate.ChildThreshold method), 93
matrix() (CytoPy.flow.variance.SimilarityMatrix method), 230
merge_children() (in module CytoPy.data.gate), 125
merge_multiple_populations() (in module CytoPy.data.population), 157
merge_populations() (CytoPy.data.fcs.FileGroup method), 64
merge_populations() (in module CytoPy.data.population), 157
meta_cluster() (CytoPy.flow.clustering.flowsom.FlowSOM method), 225
meta_label (CytoPy.data.population.Cluster attribute), 152
meta_labelling() (in module CytoPy.flow.feature_extraction), 255
MetaDataDictionary (class in CytoPy.data.subject), 171
MetaDataDictionary.DoesNotExist, 173
MetaDataDictionary.MultipleObjectsReturned, 173
method (CytoPy.data.gate.EllipseGate attribute), 94
method (CytoPy.data.gate.Gate attribute), 102
method (CytoPy.data.gate.PolygonGate attribute), 109
method (CytoPy.data.gate.ThresholdGate attribute), 117
method (CytoPy.data.gating_strategy.Action attribute), 128
method_kwargs (CytoPy.data.gate.EllipseGate attribute), 94
method_kwargs (CytoPy.data.gate.Gate attribute), 102
method_kwargs (CytoPy.data.gate.PolygonGate attribute), 109
method_kwargs (CytoPy.data.gate.ThresholdGate attribute), 117
metrics (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 198
model (CytoPy.data.supervised_classifier.CellClassifier attribute), 189
model (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 199
model (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 212
modify() (CytoPy.data.experiment.Experiment method), 74
modify() (CytoPy.data.experiment.Panel method), 83
modify() (CytoPy.data.fcs.FileGroup method), 65
modify() (CytoPy.data.gate.EllipseGate method), 98
modify() (CytoPy.data.gate.Gate method), 105
modify() (CytoPy.data.gate.PolygonGate method), 113
modify() (CytoPy.data.gate.ThresholdGate method), 121
modify() (CytoPy.data.gating_strategy.GatingStrategy method), 137
modify() (CytoPy.data.project.Project method), 163
modify() (CytoPy.data.subject.MetaDataDictionary method), 174
modify() (CytoPy.data.subject.Subject method), 180
modify() (CytoPy.data.supervised_classifier.CellClassifier method), 194
modify() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 205
modify() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 218
module
CytoPy.data.experiment, 68
CytoPy.data.fcs, 57
CytoPy.data.gate, 86
CytoPy.data.gating_strategy, 127
CytoPy.data.geometry, 142
CytoPy.data.mapping, 149
CytoPy.data.population, 151
CytoPy.data.project, 158
CytoPy.data.read_write, 184
CytoPy.data.setup, 141
CytoPy.data.subject, 166
CytoPy.data.supervised_classifier, 187
CytoPy.data.experiment, 260
CytoPy.data.gating_strategy, 226
CytoPy.data.gate, 223
CytoPy.data.gate, 215
CytoPy.data.gate, 251
CytoPy.data.gate, 253
CytoPy.data.gate, 233
CytoPy.data.gate, 235
CytoPy.data.gate, 240
CytoPy.data.gate, 242
CytoPy.data.gate, 247
CytoPy.data.gate, 257
CytoPy.data.gate, 259
CytoPy.data.gate, 227

multi_class (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 188
multi_class (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 199
multi_class (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 211
multilabel() (in module CytoPy.flow.supervised), 250
my_metaclass (CytoPy.data.experiment.Experiment attribute), 74
my_metaclass (CytoPy.data.experiment.NormalisedName attribute), 79
my_metaclass (CytoPy.data.experiment.Panel attribute), 83
my_metaclass (CytoPy.data.fcs.FileGroup attribute), 65
my_metaclass (CytoPy.data.gate.Child attribute), 90
my_metaclass (CytoPy.data.gate.ChildPolygon attribute), 91
CytoPy, Release 1.0.0

my_metaclass (CytoPy.data.gate.ChildThreshold attribute), 93
my_metaclass (CytoPy.data.gate.EllipseGate attribute), 99
my_metaclass (CytoPy.data.gate.Gate attribute), 105
my_metaclass (CytoPy.data.gate.PolygonGate attribute), 113
my_metaclass (CytoPy.data.gate.ThresholdGate attribute), 121
my_metaclass (CytoPy.data.gating_strategy.Action attribute), 130
my_metaclass (CytoPy.data.gating_strategy.GatingStrategy attribute), 138
my_metaclass (CytoPy.data.geometry.PolygonGeom attribute), 144
my_metaclass (CytoPy.data.geometry.PopulationGeometry attribute), 146
my_metaclass (CytoPy.data.geometry.ThresholdGeom attribute), 147
my_metaclass (CytoPy.data.mapping.ChannelMap attribute), 151
my_metaclass (CytoPy.data.population.Cluster attribute), 153
my_metaclass (CytoPy.data.population.Population attribute), 156
my_metaclass (CytoPy.data.project.Project attribute), 163
my_metaclass (CytoPy.data.subject.Biology attribute), 168
my_metaclass (CytoPy.data.subject.Bug attribute), 170
my_metaclass (CytoPy.data.subject.Drug attribute), 171
my_metaclass (CytoPy.data.subject.MetaDataDictionary attribute), 175
my_metaclass (CytoPy.data.subject.Subject attribute), 181
my_metaclass (CytoPy.data.supervised_classifier.CellClassifier attribute), 194
my_metaclass (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 205
my_metaclass (CytoPy.data.supervised_classifier.Layer attribute), 210
my_metaclass (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 218

n (CytoPy.data.population.Cluster attribute), 152
name (CytoPy.data.gate.ChildPolygon attribute), 90
name (CytoPy.data.gate.ChildThreshold attribute), 92
name (CytoPy.data.subject.Drug attribute), 170
name (CytoPy.data.supervised_classifier.CellClassifier attribute), 188

n (CytoPy.data.supervised_classifier.CellClassifier attribute), 198
name (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 211
new_population_name (CytoPy.data.gating_strategy.Action attribute), 129
n_jobs (CytoPy.flow.variance.SimilarityMatrix attribute), 229
NormalisedName (class in CytoPy.data.experiment), 77
notes (CytoPy.data.gating_strategy.GatingStrategy attribute), 130
O
optimizer (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 198
org_name (CytoPy.data.subject.Bug attribute), 168
org_type () (in module CytoPy.data.subject), 184
organism_type (CytoPy.data.subject.Bug attribute), 169
overlay_plot() (CytoPy.flow.plotting.CreatePlot method), 238
owner (CytoPy.data.project.Project attribute), 158
P
Panel (class in CytoPy.data.experiment), 79
panel (CytoPy.data.experiment.Experiment attribute), 69
Panel.DoesNotExist, 81
Panel.MultipleObjectsReturned, 81
panel_name (CytoPy.data.experiment.Panel attribute), 79
params (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 211
parent (CytoPy.data.gate.EllipseGate attribute), 94
parent (CytoPy.data.gate.Gate attribute), 101
parent (CytoPy.data.gate.PolygonGate attribute), 108
parent (CytoPy.data.gate.ThresholdGate attribute), 116
patient_biology (CytoPy.data.subject.Subject attribute), 177
percentile_rank_transform() (in module CytoPy.flow.transforms), 258
permutations (CytoPy.data.experiment.NormalisedName attribute), 78
pk() (CytoPy.data.experiment.Experiment property), 75
pk() (CytoPy.data.experiment.Panel property), 83
pk() (CytoPy.data.fcs.FileGroup property), 65
pk() (CytoPy.data.gate.EllipseGate property), 99
pk() (CytoPy.data.gate.Gate property), 106
pk() (CytoPy.data.gate.PolygonGate property), 113
pk() (CytoPy.data.gate.ThresholdGate property), 122
pk() (CytoPy.data.gating_strategy.GatingStrategy property), 138
pk() (CytoPy.data.project.Project property), 163
pk() (CytoPy.data.subject.MetaDataDictionary property), 175
pk() (CytoPy.data.subject.Subject property), 181
pk() (CytoPy.data.supervised_classifier.CellClassifier property), 195
pk() (CytoPy.data.supervised_classifier.KerasCellClassifier property), 206
pk() (CytoPy.data.supervised_classifier.SklearnCellClassifier property), 218
plot() (CytoPy.flow.plotting.CreatePlot method), 239
plot_backgate() (CytoPy.data.gating_strategy.GatingStrategy method), 138
plot_confusion_matrix() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 218
plot_gate() (CytoPy.data.gating_strategy.GatingStrategy method), 138
plot_gate_children() (CytoPy.flow.plotting.CreatePlot method), 239
plot_learning_curve() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 206
plot_learning_curve() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 218
plot_population() (CytoPy.data.gating_strategy.GatingStrategy method), 138
plot_population_geoms() (CytoPy.flow.plotting.CreatePlot method), 239
polygon_overlap() (in module CytoPy.data.geometry), 148
Population (class in CytoPy.data.population), 154
population_prefix (CytoPy.data.supervised_classifier.CellClassifier attribute), 189
population_prefix (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 199
population_prefix (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 212
population_stats() (CytoPy.data.fcs.FileGroup method), 65
population_stats() (in module CytoPy.data.fcs), 68
PopulationGeometry (class in CytoPy.data.geometry), 144
predict() (CytoPy.data.gate.EllipseGate method), 99
predict() (CytoPy.data.gate.PolygonGate method), 113
predict() (CytoPy.data.gate.ThresholdGate method), 122
predict() (CytoPy.data.supervised_classifier.CellClassifier method), 195
predict() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 206
predict() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 218
predict() (CytoPy.flow.clustering.consensus.ConsensusCluster method), 227
predict() (CytoPy.flow.clustering.flowsom.FlowSOM method), 225
predict_data() (CytoPy.flow.clustering.consensus.ConsensusCluster method), 227
preview_gate() (CytoPy.data.gating_strategy.GatingStrategy method), 139
print_population_tree() (CytoPy.data.fcs.FileGroup method), 65
print_population_tree() (CytoPy.data.gating_strategy.GatingStrategy method), 139
prob_downsample() (in module CytoPy.flow.sampling), 245
probabilistic_ellipse() (in module CytoPy.data.geometry), 149
progress_bar() (in module CytoPy.feedback), 260
Project (class in CytoPy.data.project), 158
Project.DoesNotExist, 160
Project.MultipleObjectsReturned, 160
project_id (CytoPy.data.project.Project attribute), 158
prop_of_events (CytoPy.data.population.Cluster attribute), 152
Q query() (CytoPy.data.experiment.NormalisedName method), 79
R radar_plot() (in module CytoPy.flow.feature_extraction), 256
ref_range (CytoPy.data.subject.Biology attribute), 167
CytoPy, Release 1.0.0

reference (CytoPy.flow.variance.SimilarityMatrix attribute), 229

reload() (CytoPy.data.gating_strategy.GatingStrategy method), 139
reload() (CytoPy.data.project.Project method), 163
reload() (CytoPy.data.subject.MetaDataDictionary method), 175
reload() (CytoPy.data.subject.Subject method), 181
reload() (CytoPy.data.supervised_classifier.CellClassifier method), 195
reload() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 207
reload() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 220

remove_sample() (CytoPy.data.gate.Gate method), 106
reset_gate() (CytoPy.data.gate.EllipseGate method), 99
reset_gate() (CytoPy.data.gate.PolygonGate method), 106
reset_gate() (CytoPy.data.gate.ThresholdGate method), 114
reset_gate() (CytoPy.data.project.Project method), 122
reset_gate() (CytoPy.data.subject.Bug method), 167
reset_gate() (CytoPy.data.subject.Biology method), 168
reset_gate() (CytoPy.data.subject.MetaDataDictionary method), 169
reset_gate() (CytoPy.data.subject.Subject method), 175
reset_gate() (CytoPy.data.supervised_classifier.CellClassifier method), 181
reset_gate() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 195
reset_gate() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 207

report_date (CytoPy.data.subject.Bug method), 175
report_date (CytoPy.data.subject.Biology method), 181
report_date (CytoPy.data.subject.MetaDataDictionary method), 185
report_date (CytoPy.data.subject.Subject method), 195
report_date (CytoPy.data.supervised_classifier.CellClassifier method), 207
report_date (CytoPy.data.supervised_classifier.KerasCellClassifier method), 219
report_date (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 231

result (CytoPy.data.gate.Gate method), 106
result (CytoPy.data.gate.EllipseGate method), 109
result (CytoPy.data.gate.PolygonGate method), 117
result (CytoPy.data.gate.ThresholdGate method), 125
result (CytoPy.data.project.Project method), 133
result (CytoPy.data.subject.Bug method), 141
result (CytoPy.data.subject.Biology method), 148
result (CytoPy.data.subject.MetaDataDictionary method), 155
result (CytoPy.data.subject.Subject method), 163
result (CytoPy.data.supervised_classifier.CellClassifier method), 171
result (CytoPy.data.supervised_classifier.KerasCellClassifier method), 183
result (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 195

ribos_status (CytoPy.data.subject.Bug method), 151
ribos_status (CytoPy.data.subject.Biology method), 158
ribos_status (CytoPy.data.subject.MetaDataDictionary method), 165
ribos_status (CytoPy.data.subject.Subject method), 173
ribos_status (CytoPy.data.supervised_classifier.CellClassifier method), 185
ribos_status (CytoPy.data.supervised_classifier.KerasCellClassifier method), 197
ribos_status (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 209

right (CytoPy.data.gating_strategy.Action attribute), 102
right (CytoPy.data.gating_strategy.GatingStrategy attribute), 109
right (CytoPy.data.subject.Bug attribute), 117
right (CytoPy.data.subject.Biology attribute), 125
right (CytoPy.data.subject.MetaDataDictionary attribute), 133
right (CytoPy.data.subject.Subject attribute), 141
right (CytoPy.data.supervised_classifier.CellClassifier attribute), 153
right (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 165
right (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 177

save() (CytoPy.data.experiment.Panel method), 83
save() (CytoPy.data.gate.EllipseGate method), 99
save() (CytoPy.data.gate.Gate method), 106
save() (CytoPy.data.gate.PolygonGate method), 114
save() (CytoPy.data.gate.ThresholdGate method), 122
save() (CytoPy.data.gating_strategy.GatingStrategy method), 139
save() (CytoPy.data.gate.EllipseGate method), 99
save() (CytoPy.data.gate.Gate method), 106
save() (CytoPy.data.gate.PolygonGate method), 114
save() (CytoPy.data.gate.ThresholdGate method), 122
save() (CytoPy.data.gating_strategy.GatingStrategy method), 139
save() (CytoPy.data.project.Project method), 163
save() (CytoPy.data.subject.Bug method), 167
save() (CytoPy.data.subject.Biology method), 168
save() (CytoPy.data.subject.MetaDataDictionary method), 175
save() (CytoPy.data.subject.Subject method), 181
save() (CytoPy.data.supervised_classifier.CellClassifier method), 195

s

sample_exists() (CytoPy.data.gate.Gate attribute), 106
sample_exists() (CytoPy.data.gate.EllipseGate attribute), 114
sampling (CytoPy.data.gate.Gate attribute), 122
sampling (CytoPy.data.gate.EllipseGate attribute), 130
sampling (CytoPy.data.gate.PolygonGate attribute), 138
sampling (CytoPy.data.gate.ThresholdGate attribute), 146
sampling (CytoPy.data.project.Project attribute), 154
sampling (CytoPy.data.subject.Bug attribute), 162
sampling (CytoPy.data.subject.Biology attribute), 170
sampling (CytoPy.data.subject.MetaDataDictionary attribute), 178
sampling (CytoPy.data.subject.Subject attribute), 186
sampling (CytoPy.data.supervised_classifier.CellClassifier attribute), 194
sampling (CytoPy.data.supervised_classifier.KerasCellClassifier attribute), 206
sampling (CytoPy.data.supervised_classifier.SklearnCellClassifier attribute), 218

s
transformations (CytoPy.data.gate.ThresholdGate attribute), 117

uniform_downsampling() (in module CytoPy.flow.sampling), 245

update() (CytoPy.data.experiment.Experiment method), 77
update() (CytoPy.data.experiment.Panel method), 85
update() (CytoPy.data.fcs.FileGroup method), 68
update() (CytoPy.data.gate.EllipseGate method), 101
update() (CytoPy.data.gate.Gate method), 108
update() (CytoPy.data.gate.PolygonGate method), 116
update() (CytoPy.data.gate.ThresholdGate method), 124
update() (CytoPy.data.gating_strategy.GatingStrategy method), 141
update() (CytoPy.data.geometry.PopulationGeometry method), 146
validate() (CytoPy.data.population.Cluster method), 154
validate() (CytoPy.data.population.Population method), 156
validate() (CytoPy.data.project.Project method), 165
validate() (CytoPy.data.subject.Biology method), 168
validate() (CytoPy.data.subject.Bug method), 170
validate() (CytoPy.data.subject.Drug method), 171
validate() (CytoPy.data.subject.MetaDataDictionary method), 177
validate() (CytoPy.data.subject.Subject method), 183
validate() (CytoPy.data.supervised_classifier.CellClassifier method), 197
validate() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 208
validate() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 222
validate_data_directory() (CytoPy.data.experiment.Experiment method), 77
update() (CytoPy.data.experiment.NormalisedName method), 79
update() (CytoPy.data.experiment.Panel method), 85
update() (CytoPy.data.fcs.FileGroup method), 68
update() (CytoPy.data.gate.Child method), 90
update() (CytoPy.data.gate.ChildPolygon method), 92
validate() (CytoPy.data.gate.ChildThresholdGate method), 93
validate() (CytoPy.data.gate.EllipseGate method), 101
validate() (CytoPy.data.gate.Gate method), 108
validate() (CytoPy.data.gate.PolygonGate method), 116
validate() (CytoPy.data.gate.ThresholdGate method), 124
validate() (CytoPy.data.gating_strategy.Action method), 130
validate() (CytoPy.data.gating_strategy.GatingStrategy method), 141
validate() (CytoPy.data.geometry.PopulationGeometry method), 146
validate() (CytoPy.data.geometry.ThresholdGeom method), 147
validate() (CytoPy.data.mapping.ChannelMap method), 151
validate() (CytoPy.data.population.Cluster method), 154
validate() (CytoPy.data.population.Population method), 156
validate() (CytoPy.data.project.Project method), 165
validate() (CytoPy.data.subject.Biology method), 168
validate() (CytoPy.data.subject.Bug method), 170
validate() (CytoPy.data.subject.Drug method), 171
validate() (CytoPy.data.subject.MetaDataDictionary method), 177
validate() (CytoPy.data.subject.Subject method), 183
validate() (CytoPy.data.supervised_classifier.CellClassifier method), 197
validate() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 208
validate() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 222
validate_classifier() (CytoPy.data.supervised_classifier.CellClassifier method), 197
validate_classifier() (CytoPy.data.supervised_classifier.KerasCellClassifier method), 209
validate_classifier() (CytoPy.data.supervised_classifier.SklearnCellClassifier method), 222
verbose (CytoPy.data.supervised_classifier.CellClassifier attribute), 189
x (CytoPy.data.supervised_classifier.SklearnCellClassifier
attribute), 212
x_threshold (CytoPy.data.geometry.ThresholdGeom
attribute), 146
x_values (CytoPy.data.geometry.PolygonGeom
attribute), 143
xlabel (CytoPy.flow.plotting.CreatePlot attribute), 236
xlim (CytoPy.flow.plotting.CreatePlot attribute), 236

Y
y (CytoPy.data.gate.EllipseGate attribute), 94
y (CytoPy.data.gate.Gate attribute), 101
y (CytoPy.data.gate.PolygonGate attribute), 108
y (CytoPy.data.gate.ThresholdGate attribute), 117
y (CytoPy.data.geometry.PopulationGeometry attribute), 145
y (CytoPy.data.supervised_classifier.CellClassifier at-
tribute), 189
y (CytoPy.data.supervised_classifier.KerasCellClassifier
attribute), 200
y (CytoPy.data.supervised_classifier.SklearnCellClassifier
attribute), 212
y_threshold (CytoPy.data.geometry.ThresholdGeom
attribute), 146
y_values (CytoPy.data.geometry.PolygonGeom
attribute), 143
ylabel (CytoPy.flow.plotting.CreatePlot attribute), 236
ylim (CytoPy.flow.plotting.CreatePlot attribute), 236