AutoClassWrapper

Release 1.5.1

Pierre Poulain

Jul 25, 2019

CONTENTS

Version 1.5.1

> AutoClass is an unsupervised Bayesian classification system. AutoClass C is an implementation of the AutoClass algorithm developed by the NASA in 1995.

AutoClassWrapper is a Python wrapper to ease the use of Autoclass C.

Install AutoClassWrapper with *pip*:

```
$ python3 -m pip install autoclasswrapper
```

you will also need AutoClass C:

```
$ wget https://ti.arc.nasa.gov/m/project/autoclass/autoclass-c-3-3-6.tar.gz
$ tar zxvf autoclass-c-3-3-6.tar.gz
$ rm -f autoclass-c-3-3-6.tar.gz
$ export PATH=$PATH:$(pwd)/autoclass-c

# if you use a 64-bit operating system,
# you also need to install the standard 32-bit C libraries:
$ sudo apt-get install -y libc6-i386
```

CHAPTER

ONE

USER MANUAL

1.1 AutoClass algorithm and context

AutoClass is an unsupervised Bayesian classification system developed at the NASA Ames Research Center in 1991 by Hanson, Stutz and Cheeseman. This algorithm has many interesting features:

- The number of classes are determined automatically.
- Missing values are supported.
- Discret and real values can be mixed.
- For all classified objects, the class membership probability is provided.

AutoClass C is the implementation of the AutoClass algorithm in C. It has been developed by Cheeseman and Stutz in 1996. AutoClass C has been successful in classifying data as diverse as infrared spectra of stars, protein structures, introns from human DNA sequences, Landsat satellites images, body pattern in the common cuttlefish, patterns between rich and poor countries, network traffic, or catchments in the Australian landscape. In proteomics and genomics, where thousands of proteins or genes are detected at once, AutoClass C has been proven to produce insightful results.

However, AutoClass C user interface isn't very friendly and requires that data and parameters are input in a very precise way. To help user to prepare input data, perform classification and analyze output clusters, we developed AutoClassWrapper as a Python wrapper around AutoClass C.

1.2 Installation

To install AutoClassWrapper, use `pip`:

```
$ python3 -m pip install autoclasswrapper
```

you will also need AutoClass C:

```
$ wget https://ti.arc.nasa.gov/m/project/autoclass/autoclass-c-3-3-6.tar.gz
$ tar zxvf autoclass-c-3-3-6.tar.gz
$ rm -f autoclass-c-3-3-6.tar.gz
$ export PATH=$PATH:$(pwd)/autoclass-c

# if you use a 64-bit operating system,
# you also need to install the standard 32-bit C libraries:
$ sudo apt-get install -y libc6-i386
```

1.3 Data preparation

AutoClass C can handle 3 different types of data:

- *real scalar*: numerical values bounded by 0. Examples: length, weight, age…
- *real location*: numerical values, positive and negative. Examples: position, microarray log ratio, elevation…
- *discrete*: qualitative data. Examples: color, phenotype, name…

Each data type must be entered in **separate input file** (one for each type).

The usual workflow to prepare data is to instantiate an object from the `Input()` class:

```python
import autoclasswrapper as wrapper
clust = wrapper.Input()
```

then add as many datasets as wanted, usually one per different data types:

```python
clust.add_input_data("example1.tsv", "real scalar")
clust.add_input_data("example2.tsv", "real location")
```

Default input data format is tab-separated values. If data are formated as comma-separated values, use the `input_separator_char=","` parameter.

- The first line must be a header with column names. Avoid accentuated or special characters ($&!/) or space. These characters will be automatically replaced by _. Avoid lengthy column names. Column names must be unique.
- The first column must be gene/protein/object names.
- Missing data are allowed. They must be represented by nothing (no `NA`, `?`, `None`, `NULL`…).

Together with the name of the input file, user must provide the type of data (either `real scalar`, `real location` or `discrete`).

The default error on real values is 0.01. Error is relative for *real scalar* values (0.01 means 1%) but absolute for *real location* values. There is no error for *discrete* values. For *real scalar* and *real location* values, custom error can be defined with the `input_error` parameter of the `.add_input_data()` method.

The next step is to prepare input data and generate input files required by AutoClass C:

```python
clust.prepare_input_data()
clust.create_db2_file()
clust.create_hd2_file()
clust.create_model_file()
clust.create_sparams_file()
clust.create_rparams_file()
```

All this commands are compulsory and will create several parameter files in the current directory.

1.4 Classification / clustering

Once input files are created, one can build Bash run script and actually run AutoClass C:

```python
import autoclasswrapper as wrapper
run = wrapper.Run()
run.create_run_file()
run.run()
```

At this stage, AutoClass C must be installed and available in PATH (see installation section).

The Bash script that run AutoClass C runs it actually twice. The first time to perform the classification (clustering). The second time to build a report from the raw results.

The Bash script that run AutoClass C is loaded itself with the `nohup` command. This means that the only way to stop this script is by killing it!

Depending on the size of the datasets (number of lines and columns), the classification might take some time to run (from few seconds to several hours). By default, the maximum running time is 3600 seconds (1 hour). This setting can be modified with the `max_duration` parameter of the `.create_sparams_file()` method.

1.5 Results analysis

Upon classification, results are ouput in different formats:

- `.cdt`: cluster data (CDT) files can be open with Java Treeview
- `.tsv`: Tab-separated values (TSV) file can be easily open and process with Microsoft Excel, R, Python…
- `_stats.tsv`: basic statistics for all classes
- `_dendrogram.png`: figure with a dendrogram showing relationship between classes

Note that the first class has number **1** (not 0).

```
import autoclasswrapper as wrapper
results = wrapper.Output()
results.extract_results()
results.aggregate_input_data()
results.write_cdt()
results.write_cdt(with_proba=True)
results.write_class_stats()
results.write_dendrogram()
```

The `.tsv` files contains:

- The initial dataset.
- A `main-class` column that gives the class with the highest probability.
- A `main-class-proba` column that contains the actual probability value (between 0.0 and 1.0) of the most probable class.
- `class-x-proba` columns (with `x` being a class number) that provide the probability to belong to the `x` class.

CHAPTER

TWO

TUTORIAL

2.1 Toy dataset

We used here an artificial toy dataset composed of 3 sets of points with 2D random coordinates around an arbitrary center.

To go through this example, you need to install AutoClassWrapper:

```
$ python3 -m pip install autoclasswrapper
```

AutoClass C also needs to be installed locally and available in path.

Here is a quick solution for a Linux Bash shell:

```
wget https://ti.arc.nasa.gov/m/project/autoclass/autoclass-c-3-3-6.tar.gz
tar zxvf autoclass-c-3-3-6.tar.gz
rm -f autoclass-c-3-3-6.tar.gz
export PATH=$PATH:$(pwd)/autoclass-c

# if you use a 64-bit operating system,
# you also need to install the standard 32-bit C libraries:
# sudo apt-get install -y libc6-i386
```

[1]:
```
from pathlib import Path
import sys
import time

import matplotlib
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import numpy as np
import pandas as pd

%matplotlib inline

print("Python:", sys.version)
print("matplotlib:", matplotlib.__version__)
print("numpy:", np.__version__)
print("pandas:", pd.__version__)

import autoclasswrapper as wrapper
print("AutoClassWrapper:", wrapper.__version__)

version = sys.version_info
```

(continues on next page)

(continued from previous page)

```
if not ((version.major >= 3) and (version.minor >= 6)):
    sys.exit("Need Python>=3.6")
```

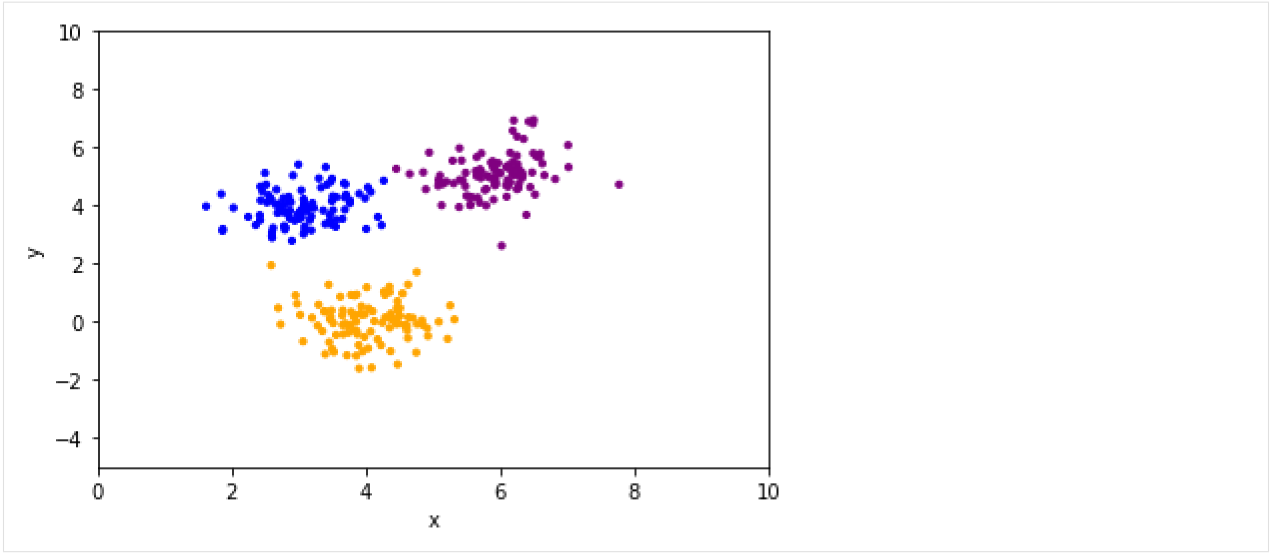
```
Python: 3.7.1 | packaged by conda-forge | (default, Feb 26 2019, 04:48:14)
[GCC 7.3.0]
matplotlib: 3.0.3
numpy: 1.16.2
pandas: 0.24.1
AutoClassWrapper: 1.4.1
```

2.1.1 Dataset generation

[2]:
```
size = 100
sigma = 0.6
x = np.concatenate((np.random.normal(3, sigma, size), np.random.normal(4, sigma,␣
↪size), np.random.normal(6, sigma, size)))
y = np.concatenate((np.random.normal(4, sigma, size), np.random.normal(0, sigma,␣
↪size), np.random.normal(5, sigma, size)))
color = ["blue"]*size+["orange"]*size+["purple"]*size
name = ["id{:03d}".format(id) for id in range(size*3)]
df = pd.DataFrame.from_dict({"x":x, "y":y, "color":color})
df.index = name
df.index.name = "name"
df.head()
```

[2]:
```
              x         y color
name
id000  4.268207  4.857065  blue
id001  3.079047  3.985566  blue
id002  3.704671  4.374195  blue
id003  2.681800  3.745056  blue
id004  3.012738  3.704896  blue
```

[3]:
```
plt.scatter(df["x"], df["y"], color=df["color"], s=10)
plt.xlabel("x")
plt.ylabel("y")
plt.xlim(0, 10)
plt.ylim(-5, 10);
```



[4]:
```
# verify all x are > 0
assert min(df["x"]) > 0
```

Save `x` and `y` in 2 different files (that will be later merged)

[5]:
```
df["x"].to_csv("demo_real_scalar.tsv", sep="\t", header=True)
df["y"].to_csv("demo_real_location.tsv", sep="\t", header=True)
```

2.1.2 Step 1 - prepare input files

AutoClass C can handle different types of data:

- *real scalar*: numerical values bounded by 0. Examples: length, weight, age...
- *real location*: numerical values, positive and negative. Examples: position, microarray log ratio, elevation...
- *discrete*: qualitative data. Examples: color, phenotype, name...

Each data type must be entered in separate input file.

AutoClass C handles very well missing data. Missing data must be represented by nothing (no `NA`, `?`, `None`...)

[6]:
```
# Create object to prepare dataset.
clust = wrapper.Input()

# Load datasets from tsv files.
clust.add_input_data("demo_real_scalar.tsv", "real scalar")
clust.add_input_data("demo_real_location.tsv", "real location")

# Prepare input data:
# - create a final dataframe
# - merge datasets if multiple inputs
clust.prepare_input_data()

# Create files needed by AutoClass.
clust.create_db2_file()
clust.create_hd2_file()
clust.create_model_file()
```

(continues on next page)

(continued from previous page)

```
clust.create_sparams_file()
clust.create_rparams_file()
```

```
2019-07-07 19:07:11 INFO     Reading data file 'demo_real_scalar.tsv' as 'real␣
↪scalar' with error 0.01
2019-07-07 19:07:11 INFO     Detected encoding: ascii
2019-07-07 19:07:11 INFO     Found 300 rows and 2 columns
2019-07-07 19:07:11 DEBUG    Checking column names
2019-07-07 19:07:11 DEBUG    Index name 'name'
2019-07-07 19:07:11 DEBUG    Column name 'x'
2019-07-07 19:07:11 INFO     Checking data format
2019-07-07 19:07:11 INFO     Column 'x'
2019-07-07 19:07:11 INFO     count    300.000000
2019-07-07 19:07:11 INFO     mean       4.331423
2019-07-07 19:07:11 INFO     std        1.316879
2019-07-07 19:07:11 INFO     min        1.616267
2019-07-07 19:07:11 INFO     50%        4.038210
2019-07-07 19:07:11 INFO     max        7.776609
2019-07-07 19:07:11 INFO     --
2019-07-07 19:07:11 INFO     Reading data file 'demo_real_location.tsv' as 'real␣
↪location' with error 0.01
2019-07-07 19:07:11 INFO     Detected encoding: ascii
2019-07-07 19:07:11 INFO     Found 300 rows and 2 columns
2019-07-07 19:07:11 DEBUG    Checking column names
2019-07-07 19:07:11 DEBUG    Index name 'name'
2019-07-07 19:07:11 DEBUG    Column name 'y'
2019-07-07 19:07:11 INFO     Checking data format
2019-07-07 19:07:11 INFO     Column 'y'
2019-07-07 19:07:11 INFO     count    300.000000
2019-07-07 19:07:11 INFO     mean       3.018801
2019-07-07 19:07:11 INFO     std        2.263316
2019-07-07 19:07:11 INFO     min       -1.607160
2019-07-07 19:07:11 INFO     50%        3.882919
2019-07-07 19:07:11 INFO     max        6.949139
2019-07-07 19:07:11 INFO     --
2019-07-07 19:07:11 INFO     Preparing input data
2019-07-07 19:07:11 INFO     Final dataframe has 300 lines and 3 columns
2019-07-07 19:07:11 INFO     Searching for missing values
2019-07-07 19:07:11 INFO     No missing values found
2019-07-07 19:07:11 INFO     Writing autoclass.db2 file
2019-07-07 19:07:11 INFO     If any, missing values will be encoded as '?'
2019-07-07 19:07:11 DEBUG    Writing autoclass.tsv file [for later use]
2019-07-07 19:07:11 INFO     Writing .hd2 file
2019-07-07 19:07:11 INFO     Writing .model file
2019-07-07 19:07:11 INFO     Writing .s-params file
2019-07-07 19:07:11 INFO     Writing .r-params file
```

2.1.3 Step 2 - prepare run script & run autoclass

The file `autoclass-run-success` is created if AutoClass C has run without any issue. Otherwise, the file `autoclass-run-failure` is created.

[7]:
```
# Clean previous status file and results if a classification has already been␣
↪performed.
!rm -f autoclass-run-* *.results-bin
```

(continues on next page)

(continued from previous page)

```
# Search autoclass in path.
wrapper.search_autoclass_in_path()

# Create object to run AutoClass.
run = wrapper.Run()

# Prepare run script.
run.create_run_file()

# Run AutoClass.
run.run()
```

```
2019-07-07 19:07:14 INFO     AutoClass C executable found in /home/pierre/.soft/bin/
↪autoclass
2019-07-07 19:07:14 INFO     Writing run file
2019-07-07 19:07:14 INFO     AutoClass C executable found in /home/pierre/.soft/bin/
↪autoclass
2019-07-07 19:07:14 INFO     AutoClass C version: AUTOCLASS C (version 3.3.6unx)
2019-07-07 19:07:14 INFO     Running clustering...
```

2.1.4 Step 3 - parse and format results

AutoClass C results are parsed and formated for an easier use :

- `.cdt`: cluster data (CDT) files can be open with Java Treeview
- `.tsv`: Tab-separated values (TSV) file can be easily open and process with Microsoft Excel, R, Python…
- `_stats.tsv`: basic statistics for all classes
- `_dendrogram.png`: figure with a dendrogram showing relationship between classes

Note that the n classes are numbered from 1 to n.

Results are analyzed only when classification is completed.

[8]:
```
timer = 0
step = 2
while not Path("autoclass-run-success").exists():
    timer += step
    sys.stdout.write("\r")
    sys.stdout.write(f"Time: {timer} sec.")
    sys.stdout.flush()
    time.sleep(step)

results = wrapper.Output()
results.extract_results()
results.aggregate_input_data()
results.write_cdt()
results.write_cdt(with_proba=True)
results.write_class_stats()
results.write_dendrogram()
```

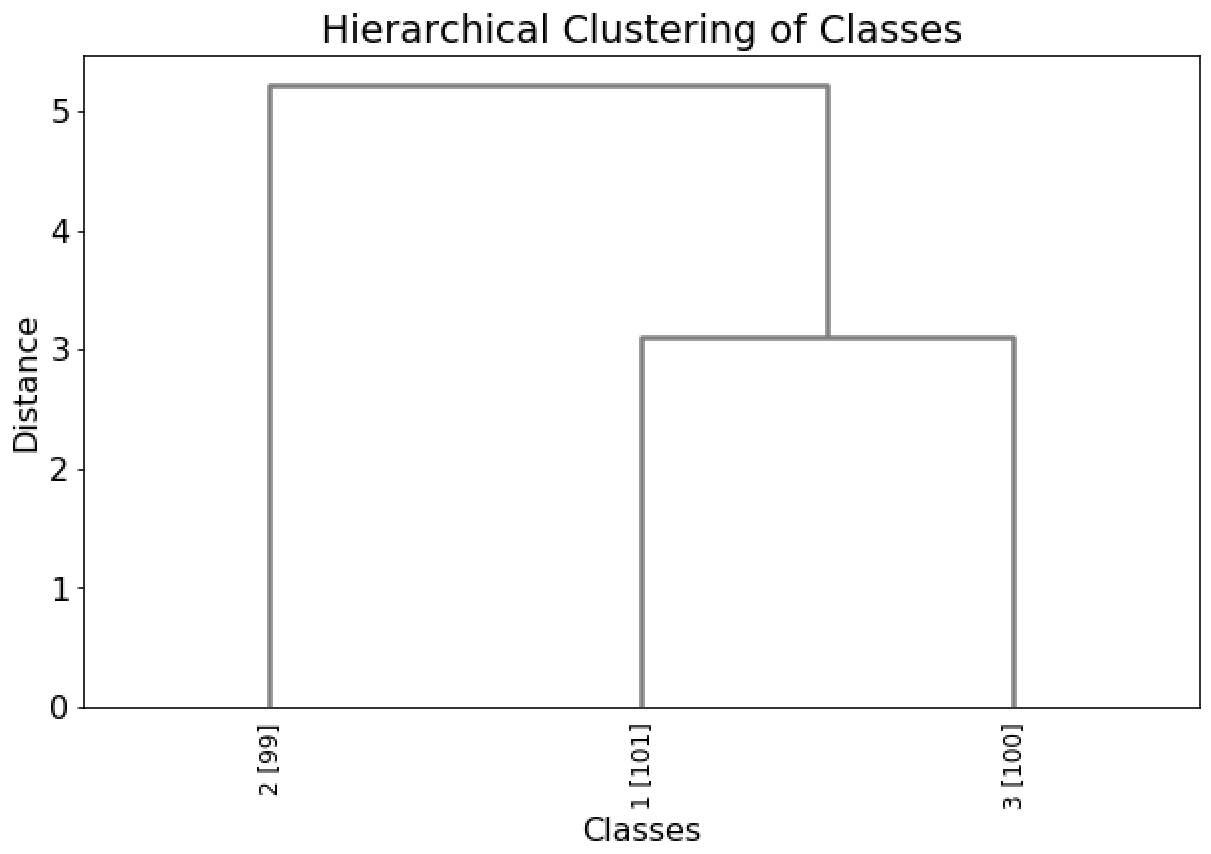
```
Time: 2 sec.
```

```
2019-07-07 19:07:18 INFO     Extracting autoclass results
2019-07-07 19:07:18 INFO     Found 300 cases classified in 3 classes
```

(continues on next page)

(continued from previous page)

```
2019-07-07 19:07:18 INFO     Aggregating input data
2019-07-07 19:07:18 INFO     Writing classes + probabilities .tsv file
2019-07-07 19:07:18 INFO     Writing .cdt file
2019-07-07 19:07:18 INFO     Writing .cdt file (with probabilities)
2019-07-07 19:07:18 INFO     Writing class statistics
2019-07-07 19:07:18 INFO     Writing dendrogram
```



The dendrogram exhibits relationship between classes.

Numbers in brakets are the number of cases (genes, proteins) in a class.

In the above plot, classes 1 and 3 are closer to each other than to class 2. Class 1 has 101 cases, class 3 has 99 and class 2 has 100.

Results exploration

All results are combined in `*_out.tsv` file.

In addition to the original data (columns `name`, `x` and `y`), the class assigned to a particular case (gene, protein) is given (`main-class`) along with its probability (`main-class-proba`). Probability to belong to all classes (`class-x-proba`) are also provided.

[9]:
```
df_res = pd.read_csv("autoclass_out.tsv", sep="\t")
df_res.head()
```

[9]:
```
    name         x         y  main-class  main-class-proba  class-1-proba  \
0  id000  4.268207  4.857065           1             0.964          0.964
```

(continues on next page)

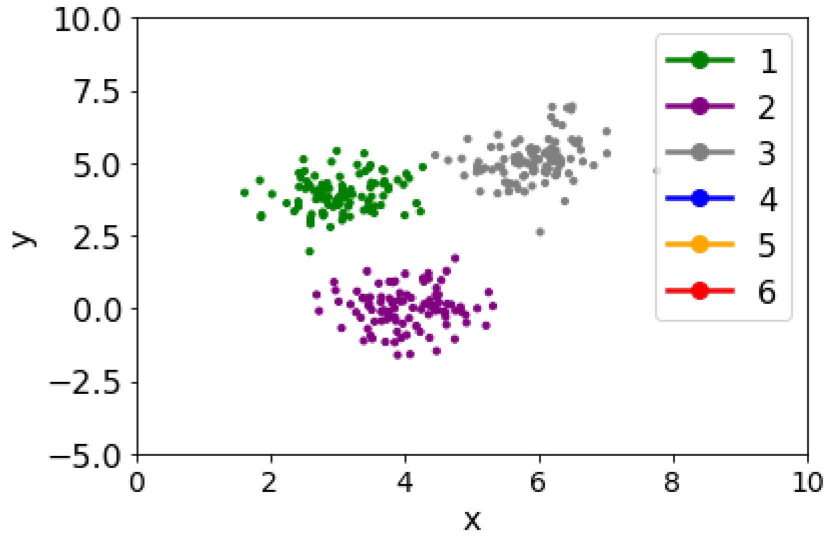
(continued from previous page)

```
1  id001  3.079047  3.985566          1             1.000          1.000
2  id002  3.704671  4.374195          1             1.000          1.000
3  id003  2.681800  3.745056          1             1.000          1.000
4  id004  3.012738  3.704896          1             1.000          1.000

   class-2-proba  class-3-proba
0            0.0          0.036
1            0.0          0.000
2            0.0          0.000
3            0.0          0.000
4            0.0          0.000
```

[10]:
```
class_to_color = {1: "green",
                  2: "purple",
                  3: "gray",
                  4: "blue",
                  5: "orange",
                  6: "red"}
df_res["main-class"] = df_res["main-class"].replace(class_to_color)
```

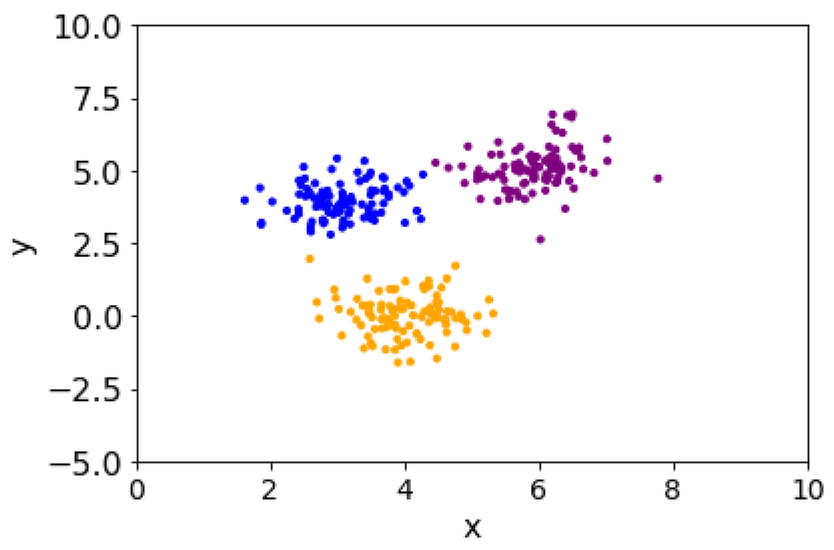
Display classes

[11]:
```
plt.scatter(df_res["x"], df_res["y"], color=df_res["main-class"], s=10)
plt.xlabel("x")
plt.ylabel("y")
plt.xlim(0, 10)
legend_elements = [Line2D([0], [0], marker='o', color=value, label=str(key),␣
↪markersize=8)
                   for key, value in class_to_color.items()]
plt.legend(handles=legend_elements)
plt.ylim(-5, 10);
```



Compare to original distribution

[12]:
```
plt.scatter(df["x"], df["y"], color=df["color"], s=10)
plt.xlabel("x")
plt.ylabel("y")
plt.xlim(0, 10)
plt.ylim(-5, 10);
```



Slight differences could appear at the margin between groups of points but the overall groups are found.

2.2 Iris flower dataset

The iris flower dataset is a common dataset used in machine learning.

It has been created Ronald Fisher in 1936. It contains the petal length, petal width, sepal length and sepal width of 150 iris flowers from 3 different species.

Dataset has been downloaded from Kaggle.

To go through this example, you need to install AutoClassWrapper:

```
$ python3 -m pip install autoclasswrapper
```

AutoClass C also needs to be installed locally and available in path.

Here is a quick solution for a Linux Bash shell:

```
wget https://ti.arc.nasa.gov/m/project/autoclass/autoclass-c-3-3-6.tar.gz
tar zxvf autoclass-c-3-3-6.tar.gz
rm -f autoclass-c-3-3-6.tar.gz
export PATH=$PATH:$(pwd)/autoclass-c

# if you use a 64-bit operating system,
# you also need to install the standard 32-bit C libraries:
# sudo apt-get install -y libc6-i386
```

[1]:
```
from pathlib import Path
import sys
import time

import matplotlib
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import numpy as np
import pandas as pd

%matplotlib inline

print("Python:", sys.version)
print("matplotlib:", matplotlib.__version__)
print("numpy:", np.__version__)
print("pandas:", pd.__version__)

import autoclasswrapper as wrapper
print("AutoClassWrapper:", wrapper.__version__)

version = sys.version_info
if not ((version.major >= 3) and (version.minor >= 6)):
    sys.exit("Need Python>=3.6")
```

```
Python: 3.7.1 | packaged by conda-forge | (default, Feb 26 2019, 04:48:14)
[GCC 7.3.0]
matplotlib: 3.0.3
numpy: 1.16.2
pandas: 0.24.1
AutoClassWrapper: 1.4.1
```

2.2.1 Dataset preparation

[2]:
```
df = pd.read_csv("iris.csv", index_col="Id")
df.head()
```

[2]:
```
    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
Id
1             5.1           3.5            1.4           0.2  Iris-setosa
2             4.9           3.0            1.4           0.2  Iris-setosa
3             4.7           3.2            1.3           0.2  Iris-setosa
4             4.6           3.1            1.5           0.2  Iris-setosa
5             5.0           3.6            1.4           0.2  Iris-setosa
```

[3]:
```
df.describe(include='all').T
```

[3]:
```
              count unique              top freq     mean       std  min  25%  \
SepalLengthCm   150    NaN              NaN  NaN  5.84333  0.828066  4.3  5.1
SepalWidthCm    150    NaN              NaN  NaN    3.054  0.433594    2  2.8
PetalLengthCm   150    NaN              NaN  NaN  3.75867   1.76442    1  1.6
PetalWidthCm    150    NaN              NaN  NaN  1.19867  0.763161  0.1  0.3
Species         150      3  Iris-versicolor   50      NaN       NaN  NaN  NaN

               50%  75%  max
SepalLengthCm  5.8  6.4  7.9
SepalWidthCm     3  3.3  4.4
```

(continues on next page)

(continued from previous page)

```
PetalLengthCm  4.35  5.1  6.9
PetalWidthCm    1.3  1.8  2.5
Species         NaN  NaN  NaN
```

Add discrete values

Apart iris species, data in this dataset are numerical values only.

To demonstrate the ability of AutoClass C to handle discrete values, we will convert `PetalWidthCm` column to discrete categorical values.

[4]:
```
def categorize(value):
    if value <= 0.75:
        return "small"
    elif 0.75 < value <= 1.75:
        return "medium"
    elif 1.75 < value:
        return "large"
```

[5]:
```
df["PetalWidthCat"] = df["PetalWidthCm"].apply(categorize)
df.head()
```

[5]:
```
    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
Id
1             5.1           3.5            1.4           0.2  Iris-setosa
2             4.9           3.0            1.4           0.2  Iris-setosa
3             4.7           3.2            1.3           0.2  Iris-setosa
4             4.6           3.1            1.5           0.2  Iris-setosa
5             5.0           3.6            1.4           0.2  Iris-setosa

   PetalWidthCat
Id
1          small
2          small
3          small
4          small
5          small
```

Add missing values

To demonstrate the ability of AutoClass C to handle missing values, we will delete some values.

[6]:
```
df.loc[1, "SepalLengthCm"] = np.nan
df.loc[2, "SepalWidthCm"] = np.nan
df.loc[3, "PetalLengthCm"] = np.nan
df.head()
```

[6]:
```
    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
Id
1             NaN           3.5            1.4           0.2  Iris-setosa
2             4.9           NaN            1.4           0.2  Iris-setosa
3             4.7           3.2            NaN           0.2  Iris-setosa
4             4.6           3.1            1.5           0.2  Iris-setosa
5             5.0           3.6            1.4           0.2  Iris-setosa
```

(continues on next page)

(continued from previous page)

```
   PetalWidthCat
Id
1           small
2           small
3           small
4           small
5           small
```

Save dataset in two different files. One with real values and the other one with discrete values (column `PetalWidthCat`). Missing values must encoded with nothing.

[7]:
```
df.drop(["Species", "PetalWidthCm", "PetalWidthCat"], axis=1).to_csv("iris_real.tsv",␣
↪sep="\t", header=True)
!head iris_real.tsv
```

```
Id      SepalLengthCm   SepalWidthCm    PetalLengthCm
1               3.5     1.4
2       4.9             1.4
3       4.7     3.2
4       4.6     3.1     1.5
5       5.0     3.6     1.4
6       5.4     3.9     1.7
7       4.6     3.4     1.4
8       5.0     3.4     1.5
9       4.4     2.9     1.4
```

[8]:
```
df["PetalWidthCat"].to_csv("iris_discrete.tsv", sep="\t", header=True)
!head iris_discrete.tsv
```

```
Id      PetalWidthCat
1       small
2       small
3       small
4       small
5       small
6       small
7       small
8       small
9       small
```

2.2.2 Step 1 - prepare input files

[9]:
```
# Create object to prepare dataset.
clust = wrapper.Input()

# Load datasets from tsv files.
clust.add_input_data("iris_real.tsv", "real scalar")
clust.add_input_data("iris_discrete.tsv", "discrete")

# Prepare input data:
# - create a final dataframe
# - merge datasets if multiple inputs
clust.prepare_input_data()
```

(continues on next page)

(continued from previous page)

```
# Create files needed by AutoClass.
clust.create_db2_file()
clust.create_hd2_file()
clust.create_model_file()
# We wanted reproducible results to ease documentation.
# But bear in mind, that this parameter is not advised by authors of AutoClass C in␣
↪production run.
# Use clust.create_sparams_file() instead.
clust.create_sparams_file(reproducible_run=True)
clust.create_rparams_file()
```

```
2019-07-07 19:07:58 INFO     Reading data file 'iris_real.tsv' as 'real scalar' with␣
↪error 0.01
2019-07-07 19:07:58 INFO     Detected encoding: ascii
2019-07-07 19:07:59 INFO     Found 150 rows and 4 columns
2019-07-07 19:07:59 DEBUG    Checking column names
2019-07-07 19:07:59 DEBUG    Index name 'Id'
2019-07-07 19:07:59 DEBUG    Column name 'SepalLengthCm'
2019-07-07 19:07:59 DEBUG    Column name 'SepalWidthCm'
2019-07-07 19:07:59 DEBUG    Column name 'PetalLengthCm'
2019-07-07 19:07:59 INFO     Checking data format
2019-07-07 19:07:59 INFO     Column 'SepalLengthCm'
2019-07-07 19:07:59 INFO     count    149.000000
2019-07-07 19:07:59 INFO     mean       5.848322
2019-07-07 19:07:59 INFO     std        0.828594
2019-07-07 19:07:59 INFO     min        4.300000
2019-07-07 19:07:59 INFO     50%        5.800000
2019-07-07 19:07:59 INFO     max        7.900000
2019-07-07 19:07:59 INFO     --
2019-07-07 19:07:59 INFO     Column 'SepalWidthCm'
2019-07-07 19:07:59 INFO     count    149.000000
2019-07-07 19:07:59 INFO     mean       3.054362
2019-07-07 19:07:59 INFO     std        0.435034
2019-07-07 19:07:59 INFO     min        2.000000
2019-07-07 19:07:59 INFO     50%        3.000000
2019-07-07 19:07:59 INFO     max        4.400000
2019-07-07 19:07:59 INFO     --
2019-07-07 19:07:59 INFO     Column 'PetalLengthCm'
2019-07-07 19:07:59 INFO     count    149.000000
2019-07-07 19:07:59 INFO     mean       3.775168
2019-07-07 19:07:59 INFO     std        1.758720
2019-07-07 19:07:59 INFO     min        1.000000
2019-07-07 19:07:59 INFO     50%        4.400000
2019-07-07 19:07:59 INFO     max        6.900000
2019-07-07 19:07:59 INFO     --
2019-07-07 19:07:59 INFO     Reading data file 'iris_discrete.tsv' as 'discrete'
2019-07-07 19:07:59 INFO     Detected encoding: ascii
2019-07-07 19:07:59 INFO     Found 150 rows and 2 columns
2019-07-07 19:07:59 DEBUG    Checking column names
2019-07-07 19:07:59 DEBUG    Index name 'Id'
2019-07-07 19:07:59 DEBUG    Column name 'PetalWidthCat'
2019-07-07 19:07:59 INFO     Checking data format
2019-07-07 19:07:59 INFO     Column 'PetalWidthCat': 3 different values
2019-07-07 19:07:59 INFO     Preparing input data
2019-07-07 19:07:59 INFO     Final dataframe has 150 lines and 5 columns
2019-07-07 19:07:59 INFO     Searching for missing values
2019-07-07 19:07:59 WARNING  Missing values found in column: SepalLengthCm
```

(continues on next page)

(continued from previous page)

```
2019-07-07 19:07:59 WARNING  Missing values found in column: SepalWidthCm
2019-07-07 19:07:59 WARNING  Missing values found in column: PetalLengthCm
2019-07-07 19:07:59 INFO     Writing autoclass.db2 file
2019-07-07 19:07:59 INFO     If any, missing values will be encoded as '?'
2019-07-07 19:07:59 DEBUG    Writing autoclass.tsv file [for later use]
2019-07-07 19:07:59 INFO     Writing .hd2 file
2019-07-07 19:07:59 INFO     Writing .model file
2019-07-07 19:07:59 INFO     Writing .s-params file
2019-07-07 19:07:59 INFO     Writing .r-params file
```

2.2.3 Step 2 - prepare run script & run autoclass

[10]:
```
# Clean previous status file and results if a classification has already been␣
↪performed.
!rm -f autoclass-run-* *.results-bin

# Search autoclass in path.
wrapper.search_autoclass_in_path()

# Create object to run AutoClass.
run = wrapper.Run()

# Prepare run script.
run.create_run_file()

# Run AutoClass.
run.run()
```

```
2019-07-07 19:08:02 INFO     AutoClass C executable found in /home/pierre/.soft/bin/
↪autoclass
2019-07-07 19:08:02 INFO     Writing run file
2019-07-07 19:08:02 INFO     AutoClass C executable found in /home/pierre/.soft/bin/
↪autoclass
2019-07-07 19:08:02 INFO     AutoClass C version: AUTOCLASS C (version 3.3.6unx)
2019-07-07 19:08:02 INFO     Running clustering...
```

2.2.4 Step 3 - parse and format results

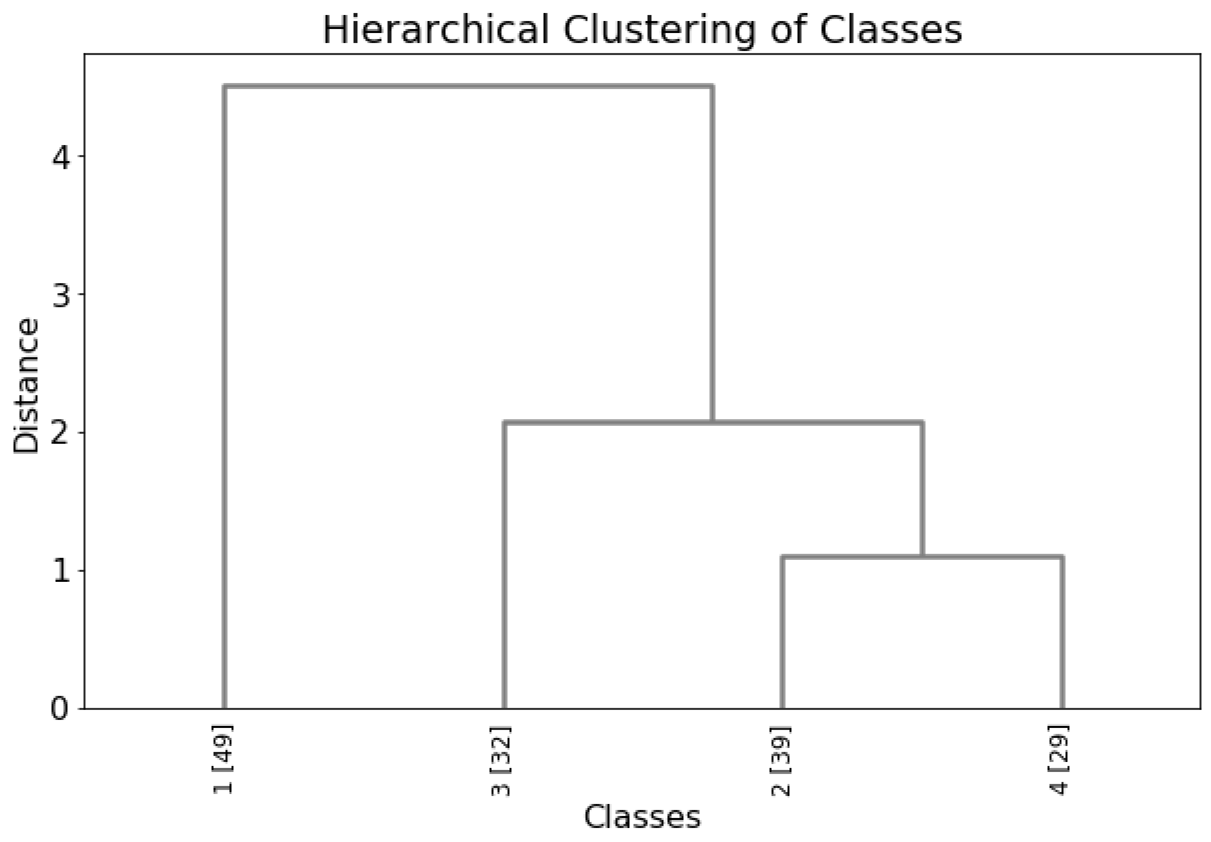
[11]:
```
timer = 0
step = 2
while not Path("autoclass-run-success").exists():
    timer += step
    sys.stdout.write("\r")
    sys.stdout.write(f"Time: {timer} sec.")
    sys.stdout.flush()
    time.sleep(step)

results = wrapper.Output()
results.extract_results()
results.aggregate_input_data()
results.write_cdt()
results.write_cdt(with_proba=True)
```

(continues on next page)

(continued from previous page)

```
results.write_class_stats()
results.write_dendrogram()
```

```
2019-07-07 19:08:05 INFO     Extracting autoclass results
2019-07-07 19:08:05 INFO     Found 150 cases classified in 4 classes
2019-07-07 19:08:05 INFO     Aggregating input data
2019-07-07 19:08:05 INFO     Writing classes + probabilities .tsv file
2019-07-07 19:08:05 INFO     Writing .cdt file
2019-07-07 19:08:05 INFO     Writing .cdt file (with probabilities)
2019-07-07 19:08:05 INFO     Writing class statistics
2019-07-07 19:08:05 INFO     Writing dendrogram
```



For comparison, add class number to original dataset.

```
[12]: df_class = pd.read_csv("autoclass_out.tsv", sep="\t", index_col="Id")
df = pd.concat([df, df_class["main-class"]], axis=1, join="outer")
df.head()
```

```
[12]:    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
Id
1              NaN           3.5            1.4           0.2  Iris-setosa
2              4.9           NaN            1.4           0.2  Iris-setosa
3              4.7           3.2            NaN           0.2  Iris-setosa
4              4.6           3.1            1.5           0.2  Iris-setosa
5              5.0           3.6            1.4           0.2  Iris-setosa

   PetalWidthCat  main-class
Id
```

(continues on next page)

(continued from previous page)

```
1          small          1
2          small          1
3          small          1
4          small          1
5          small          1
```

2.2.5 Compute class distribution for iris species

[13]:
```
pd.pivot_table(df, index=["Species"], columns=["main-class"], values=[], aggfunc=len,␣
↪fill_value=0)
```

[13]:
```
main-class        1   2   3   4
Species
Iris-setosa      50   0   0   0
Iris-versicolor   0  23   0  27
Iris-virginica    0  16  32   2
```

The setosa species is found only in cluster 1. Note that missing values did not interfere in the classification of the 3 first flowers as setosa.

The versicolor species is found in cluster 2 and 4.

The virginica species is found mainly in cluster 3 but also in cluster 2 and 4.

AutoClass-C determines automatically what is the optimal number of classes. It's always a good idea to analyse the final results to check if some cluster can be merged (for instance cluster 2 and 4).

CHAPTER

THREE

REFERENCE MANUAL

3.1 API reference for Input() class

class autoclasswrapper.**Input**(*root_name='autoclass'*, *db2_separator_char='t'*, *db2_missing_char='?'*, *tolerate_error=False*)

AutoClass C input files and parameters.

Parameters

- **root_name** (*string, optional (default "autoclass")*) – Root name to generate input files for AutoClass C. Example: "autoclass" will lead to "autoclass.db2", "autoclass.model", "autoclass.s-params"...
- **db2_separator_char** (*string, optional (default: "t")*) – Character used to separate columns of data in AutoClass C db2 file.
- **db2_missing_char** (*string, optional (default: "?")*) – Character used to encode missing data in AutoClass C db2 file.
- **tolerate_error** (*bool, optional (default: False)*) – If True, countinue generation of AutoClass C input files even if an error is encounter. If False, stop at first error.

had_error

Set to True if an error has been found in the generation of AutoClass C input files.

Type bool (defaut False)

input_datasets

List of all input Datasets.

Type list of Dataset() objects

full_dataset

Final Dataset used by AutoClass C.

Type *Dataset*() object

add_input_data(**args*, ***kwargs*)

Read input data file and append to list of datasets.

Parameters

- **input_file** (*string*) – Name of the data file to read.
- **input_type** (*string*) – Type of data contained in input file. Either "real scalar", "real location" or "discrete"
- **input_error** (*float, optional (default: 0.01)*) – Input error value.

- **`input_separator_char`** (*`string, optional (default: "t")`*) – Character used to separate columns of data in input file.
- **`input_missing_char`** (*`string, optional (default: "")`*) – Character used to encode missing data in input file.

`create_db2_file`(**args*, ***kwargs*)
Create .db2 file (AutoClass C data).

Also save all data into a .tsv file for later user.

`create_hd2_file`(**args*, ***kwargs*)
Create .hd2 file (AutoClass C data descriptions).

`create_model_file`(**args*, ***kwargs*)
Create .model file (AutoClass C data models).

Choice of model based on data type and missing values

`create_rparams_file`(**args*, ***kwargs*)
Create .r-params file (AutoClass C report parameters).

`create_sparams_file`(**args*, ***kwargs*)
Create .s-params file (AutoClass C search parameters).

Parameters

- **`max_duration`** (*`int, optional (default: 3600)`*) – Maximum time (in seconds) for the AutoClass C simulation. If set max_duration = 0, simulation will run with NO time limit For more details, see AutoClass C documentation: file search-c.text, lines 493-495
- **`max_n_tries`** (*`int, optional (default: 200)`*) – Number of trials to run. For more details, see AutoClass C documentation: file search-c.text, lines 403-404
- **`max_cycles`** (*`int, optional (default: 1000)`*) – Max number of cycles per trial. This is maximum that may not be reached. For more details, see AutoClass C documentation: file search-c.text, lines 316-317
- **`start_j_list`** (*`list of int, optional (default: [2, 3, 5, 7, 10, 15, 25, 35,`*) – 45, 55, 65, 75, 85, 95, 105]) Initial guesses of the number of clusters Autoclass default: 2, 3, 5, 7, 10, 15, 25 For more details, see AutoClass C documentation: file search-c.text, line 332
- **`reproducible_run`** (*`boolean, optional (default: False)`*) – If set to True, define parameters to obtain reproducible run. According to AutoClass C developers: "These parameter settings are for testing *only* – they should not be utilized for normal AutoClass runs."
  - **randomize_random_p = false** Random seed is set to 1 (instead of the usual current time)
  - **start_fn_type = "block"** Instead of "random"
  - min_report_period = value greater than duration of run

  For more details, see AutoClass C documentation:
  - file search-c.text, line 678
  - file search-c.text, line 565
  - file search-c.text, line 525

handle_error()

Handle error during data parsing and formating.

Function decorator.

Parameters **f** (*function*) –

Returns **try_function**

Return type function wrapped into error handler

prepare_input_data(**args*, ***kwargs*)

Prepare input data.

- Create a final dataframe.
- Merge datasets if multiple inputs.

Notes

Dataframes are merged based on an 'outer' join https://pandas.pydata.org/pandas-docs/stable/merging.html - all lines are kept - missing data might appear

print_files(**args*, ***kwargs*)

Print generated files.

Debug usage.

Returns **content** – Contain all AutoClass C parameter files concatenated.

Return type string

3.2 API reference for Dataset() class

class autoclasswrapper.**Dataset**(*input_file=''*, *data_type=''*, *error=None*, *separator_char='t'*, *missing_char=''*)

Handle input data.

Parameters

- **input_file** (*string (defaut: "")*) – Name of the file to read data from.
- **data_type** (*string (dafault: "")*) – Type of data contained in input file. Either "real scalar", "real location", "discrete" or "merged" "merged" is a special case corresponding to merged datasets.
- **error** (*float, optional (default: 0.01)*) – Value of error on data.
- **separator_char** (*string, optional (defaut: "t")*) – Character used to separate columns of data in input file.
- **missing_char** (*string, optional (default: "")*) – Character used to encode missing data in input file.

input_file

Name of the file to read data from.

Type string (defaut: "")

separator_char

Character used to separate columns of data in input file.

Type string (defaut: “t”)

df

Pandas dataframe that contains all data.

Type Pandas dataframe (default: None)

column_meta

Dictionnary that contains metadata for each column. Keys are column names. Values are another dictionnary: {“type”: data_type, “error”: error, “missing”: False}

Type dict (default: {})

check_data_type()

Check data type.

Cast ‘real scalar’ and ‘real location’ to float64

check_duplicate_col_names()

Check duplicate column names.

clean_column_names()

Clean column names.

Allowed characters are:

- *ABCDEFGHIJKLMNOPQRSTUVWXYZ*
- *abcdefghijklmnopqrstuvwxyz*
- *0123456789*
- . (dot)
- + (plus signe)
- - (minus signe)
- _ (underscore)

Unauthorized characters are replaced by ‘_’

guess_encoding()

Guess input file encoding.

Returns Type of encoding.

Return type string

read_datafile()

Read data file as pandas dataframe.

Header must be on the first row (header=0) Gene/protein/orf names must be on the first column (index_col=0)

search_missing_values()

Search for missing values.

3.3 API reference for Run() class

class autoclasswrapper.**Run**(*root_name='autoclass'*, *tolerate_error=False*)

Autoclass running script.

Parameters

- **`root_name`** (*`string, optional (default: "autoclass")`*) – Root name for input files and running script. Example: "autoclass" will lead to "autoclass.db2", "autoclass.model", "autoclass.sh"…
- **`tolerate_error`** (*`bool, optional (default: False)`*) – If True, countinue generation of autoclass input files even if an error is encounter. If False, stop at first error.

`had_error`

Set to True if an error has been found in the generation of autoclass input files.

Type bool (defaut False)

`create_run_file`(**args*, ***kwargs*)

Create bash script that runs AutoClass C.

`create_run_file_test`(**args*, ***kwargs*)

Create dummy script.

Scrit will wait for xx seconds while touching .log file every second.

Parameters **`time`** (*`int (default: 60), optional`*) – Time in seconds to wait.

`handle_error`()

Handle error during data parsing and formating.

Function decorator.

Parameters **`f`** (*`function`*) –

Returns **try_function**

Return type function wrapped into error handler

`run`(**args*, ***kwargs*)

Run AutoClass C classification.

autoclass-c executable must be in PATH!

Parameters **`tag`** (*`string (default: ""), optional`*) – Tag to identify the autoclass run among other processes

3.4 API reference for Output() class

`class` `autoclasswrapper.`**`Output`**(*root_in_name='autoclass'*, *root_out_name='autoclass_out'*, *tolerate_error=False*)

AutoClass output files and results.

Parameters

- **`root_in_name`** (*`string, optional (default: "autoclass")`*) – Root name to read input files generated by autoclass. Example: "autoclass" will lead to "autoclass.db2", "autoclass.model", "autoclass.s-params"…
- **`root_out_name`** (*`string, optional (default: "autoclass_out")`*) – Root name to write output files Ex.: "autoclass_out" will lead to "autoclass_out.cdt", "autoclass_out_stats.tsv"
- **`tolerate_error`** (*`bool, optional (default: False)`*) – If True, countinue generation of autoclass input files even if an error is encounter. If False, stop at first error.

had_error
Set to True if an error has been found in the generation of autoclass input files.

Type bool (defaut False)

case_number
Number of cases (i.e. of genes/proteins).

Type int (default 0)

class_number
Number of classes (i.e. clusters).

Type int (default 0)

stats
Dataframe that contains, for all cases, main class and probability for all classes.

Type Pandas dataframe (default None)

df
Dataframe that contains initial input data and associated clusters.

Type Pandas dataframe (default None)

experiment_names
List of experiment (conditions) names. Corresponds to columns in the input data.

Type list of string (defaut [])

aggregate_input_data(**args*, ***kwargs*)
Aggregate autoclass classes with input data.

extract_results(**args*, ***kwargs*)
Extract results from autoclass.

Results extracted are: - Number of cases (i.e. genes/proteins) - Number of classes (i.e. clusters) - For each case X, most probable class - For each case X, probability to belong to class Y

handle_error()
Handle error during data parsing and formating.

Function decorator.

Parameters **f** (*function*) –

Returns **try_function**

Return type function wrapped into error handler

write_cdt(**args*, ***kwargs*)
Write .cdt file for visualisation.

Parameters **with_proba** (*bool (default False), optional*) – If True, also writes probability of case to belong to each class.

write_class_stats(**args*, ***kwargs*)
Write class stat file.

Number of elements per class. Mean and standard deviation values per experiment. Missing values are skipped.

write_dendrogram(**args*, ***kwargs*)
Write dendrogram of hierarchical clustering of classes to file.

3.5 API reference for the tools module

`autoclasswrapper.`**`search_autoclass_in_path`**`()`

Search if AutoClass C executable is in PATH.

Returns Path to Autoclass C binary.

Return type str

`autoclasswrapper.`**`get_autoclass_version`**`()`

Output AutoClass C version.

Returns **version** – Autoclass version

Return type str

CHAPTER

FOUR

INDEX

- genindex

CHAPTER

FIVE

RECENT CHANGES

- **Add JOSS badge** by *Pierre Poulain* at *2019-07-25 22:23:54*
- **Update metadata for JMC** by *Pierre Poulain* at *2019-07-25 21:48:20*
- **Merge pull request #17 from kyleniemeyer/patch-1** by *Pierre Poulain* at *2019-07-25 21:32:23* Update paper.md
- **Update paper.md** by *Kyle Niemeyer* at *2019-07-25 18:59:57* Fixes to citations to multiple papers in one command (should use semicolon rather than comma)
- **Merge pull request #16 from trallard/patch-1** by *Pierre Poulain* at *2019-07-19 13:31:08* Add missing DOI for reference
- **Add missing DOI for reference** by *Tania Allard* at *2019-07-19 13:26:53*
- **update Zenodo DOI (generic one)** by *Pierre Poulain* at *2019-07-17 11:50:33*
- **Bump version: 1.5.0 → 1.5.1** by *Pierre Poulain* at *2019-07-17 11:46:04*
- **Create new release of the Journal of Open Source Software** by *Pierre Poulain* at *2019-07-17 11:45:33*
- **update Software Heritage link** by *Pierre Poulain* at *2019-07-11 15:28:31*

INDEX