**ReadMe: Supervised classification of cell phenotypes from experimental high-throughput screening data**

Célia Benquet, Neil Scheidwasser, Kamil Seghrouchni

This project is based on a data set consisted of 1,597,594 rows and 211 features obtained by high-throughput screening in the Biological Screening Facility. To perform the classification, we wrote 2 ‘.ipynb’ files, one to pre-process the raw data and the other one for the machine learning implementation.

The data is available upon request at [neil.scheidwasser-clow@epfl.ch](mailto:neil.scheidwasser-clow@epfl.ch).

The files were implemented using Google Colab, and should thus be run using Colab. Indeed, some cells might not compile/execute under other circumstances, as calling a mounting of the user’s local Drive with Colab to access the data files.

To run the project:

* Add the folder project to the section “MyDrive”
* Run all cells of clean\_data.ipynb to preprocess raw data
* Run all cells of run.ipynb to see the performance of the best model

1. clean\_data.ipynb

A first file, clean\_data.ipynb, consists in our pre-processing step. The following steps are implemented:

* Data loading, using *load\_data()*
* Removing unnecessary features, using *clean()*
* Splitting data into unlabeled and labeled data and collecting corresponding metadata, using *split\_data()*
* Grouping all these subdata for each timepoint using *group()*
* Removing outliers in each well for each timepoint, using *remove\_outliers\_per\_well()*, which uses *remove\_outliers()*
* Saving processed data, using *save\_data()*

1.1. load\_data()

* Two sets of data are loaded: ‘SoftC\_perObject.csv’ corresponds to the data for the 4 initially chosen time points (t= 1h, 12h, 24h and 48h) and ‘SoftC\_perObject\_Supplement1\_more\_data.csv’ is a complement to the initial data with data around the chosen 4 time points (more and less 15 minutes around each main timepoint).
* Returns: DataFrame concatenation of these two raw datasets.

1.2 clean(data)

* A function to remove unncessary features for our classification project. Unncessary features include positional data about each cell (e.g., X/Y/Z positions) and other negligible metadata.
* *data*: the raw data to be cleaned
* returns: the raw data without unnecessary features

1.3. split\_data(data)

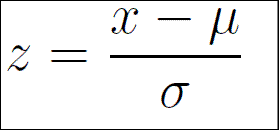
* A function to separate our data into 5 categories: labeled input, the labels and the metadata corresponding to that input, and finally the unlabeled data with its metadata.
* *data:* the data after removing unnecessary features
* returns: labeled data (*labeled*), its labels (*labels*), its metadata (*meta\_labeled*), then unlabeled data (*unlabeled*), and its metadata (*meta\_unlabeled*)

1.4. group(data, m\_data, labels=None)

* A function to group data along each timepoint
* *data*: data to be grouped
* *m\_data*: corresponding metadata
* labels: corresponding labels (only if provided)
* Returns: a list of data (*dfs*), metadata (*mdatas*), and labels (*labels*) (if given) for each timepoint

1.5.1. remove\_outliers(data)

* A function to remove outliers, using a Z-score method. Z-score is calculated by:



Where *µ* the mean value of a feature, and *σ* its standard deviation. An absolute *z* above 3 means that the point is at the tail of a Gaussian curve (assuming the distribution of each feature is Gaussian). Thus, the data point can be considered as an outlier.

* *data:* data before removal
* Returns: indices for which the rows do not have an absolute z-score above 3 for each feature

1.5.2. remove\_outliers\_per\_well(new\_l\_groups, label\_groups)

* Removes object outliers for each well in each plate, using *remove\_outliers*
* *new\_l\_groups*: labeled data, grouped for each timepoint
* *label\_groups*: corresponding labels
* *new\_meta\_lgroups*: corresponding metadata **(not used)**
* Returns: same inputs, but without outliers

. 1.6. save\_data(new\_l\_groups, label\_groups, u\_groups, meta\_u\_groups)

* Saves all transformed data as .csv files
* *new\_l\_groups*: labeled data, grouped for each timepoint
* *label\_groups*: corresponding labels
* *new\_meta\_lgroups*: corresponding metadata **(not used)**
* *u\_groups*: unlabeled data, grouped for each timepoint
* *meta\_u\_groups*: corresponding metadata (for each unlabeled group of data) **(not used)**

2. run.ipynb

Core of the project. The following steps are implemented:

* Loading preprocessed data using *load\_data()*
* Performing grid search using 5-fold cross-validation for each timepoint, using *grid\_search()* (Commented now, the best parameters of the best model (‘lgbm’, for light GBM) are given at a cell under *best\_lgbm\_params*
* The average computation time of *grid\_search()* is 1h-1h30
* Training the model for each timepoint, and predicting corresponding unlabeled data, using *train()*.

2.1. load\_data()

* This function loads all pre-processed data: labeled data, its labels and unlabeled data.
* *n\_groups*: number of timepoints. It is set to 4, although the first time point was not used.
* *x\_groups:* labeled input
* *y\_groups*: labels
* *t\_groups*: unlabeled data

2.2. grid\_search()

* Grid search combined with 5-fold cross-validation, using sklearn’s *GridSearchCV()*.
* *i*: time point index (1, 2 or 3)
* *n\_folds*=5: number of folds used for cross-validation
* *method*=’lgbm’ (string): machine learning method to test. Default value: Light GBM
* *smote*=False: perform grid search using SMOTE or not.
* *rus*=False: perform grid search using random undersampling or not.
* Returns: default parameters merged with optimized hyperparameters for the studied model.

2.3. train()

* Training function. Trains the model, evaluates the performance of the classifier and predicts labels for the unlabeled data
* *i*: group index (0, 1, or 2) (meaning that time points considered are 1, 2, or 3)
* *method:* ML method implemented (string)
* *model\_params*: parameters of the model (dict)
* *smote=False*: perform training using SMOTE or not.
* *rus*=False: perform training using random undersampling or not.
* Returns: predictions for the unlabeled data, as a Counter. Counter is a dictionary that stores elements as keys (i.e., the predicted label in this case), and number of occurrences of the key as values.