L1L2Signature Documentation

Release 0.2.2

Salvatore Masecchia, Annalisa Barla

March 19, 2012

CONTENTS

Release
0.2.2

Homepage
http://slipguru.disi.unige.it/Software/L1L2Signature

Repository
https://bitbucket.org/slipguru/l1l2signature

L1L2Signature is an implementation of an unbiased framework originally thought for gene expression analysis. The need of such a framework may be found in the *Framework overview* section illustrating the dramatic effect of a biased approach especially when the sample size is small.

The framework, here implemented, was used in many real applications and a collection of them is referenced in the *Framework Applications* section.

This library is composed by a set of Python scripts (described in the *Quick start tutorial*) and a set of useful classes and functions (described in the *Public API* section) that could be used to manually read and/or analyze high-throughput data extending/integrating the proposed pipeline.

L1L2Signature relies on two libraries also implemented by our research group and already released as open source libraries:

- L1L2Py: implements the gene selection core, based on *elastic net* regularization.
- PPlus: used to parallelize cross validation splits in an easy and effective way across a set of desktop personal computer distributed in our labs.

CHAPTER

ONE

USER DOCUMENTATION

1.1 Framework overview

The main aim of the framework is proposing an unbiased framework for gene expression analysis. Originally this framework was designed by combining a gene selection core with a significance assessment step [Barla08]. Even if we currently use both step in our experiments (see *Framework Applications*), actually **L1L2Signature** implements the latter only partially (permutation tests still missing).

In the context of detecting significant molecular alterations by gene expression profiling a main goal, besides classification, is finding a gene signature, that is a panel of genes able to discriminate between two given classes e.g. patients and controls.

Such an analysis encompasses at least two steps, gene selection and model assessment. The gene selection step, is based on elastic net regularization where we explicitly take into account regularization parameter tuning. That core, described and presented in [DeMol09], is nested into a general architecture to assess the statistical significance of the model via cross validation.

When dealing with high-throughput data the choice of a consistent selection algorithm is not sufficient to guarantee good results. It is therefore essential to introduce a robust methodology to select the significant variables not susceptible of selection bias [Ambroise02].

L1L2Signature is a a framework based on two nested loops:

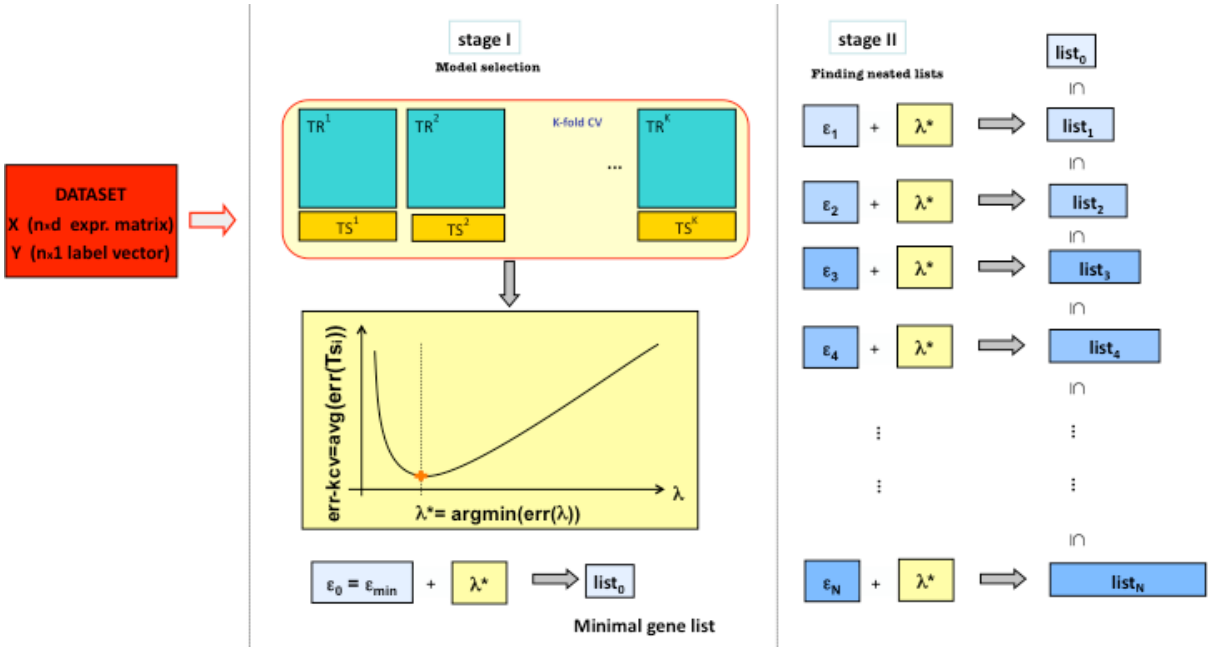
- the internal one is responsible for **model selection** and is based on a cross validation strategy;
- the external loop is for **model assessment**.

1.1.1 Model (gene) selection

A good algorithm should take into account at least linear interaction of multiple genes. Standard statistical (univariate) approaches take into consideration one gene at the time and then rank them according to their fold-change or to their prediction power. Indeed in most cases a multivariate model is preferable.

Another drawback of many variable selection algorithms is the rejection of part of the relevant genes due to redundancy. In many biological studies some of the input variables may be highly correlated with each other. As a consequence, when one variable is considered relevant to the problem, its correlated variables should be considered relevant as well.

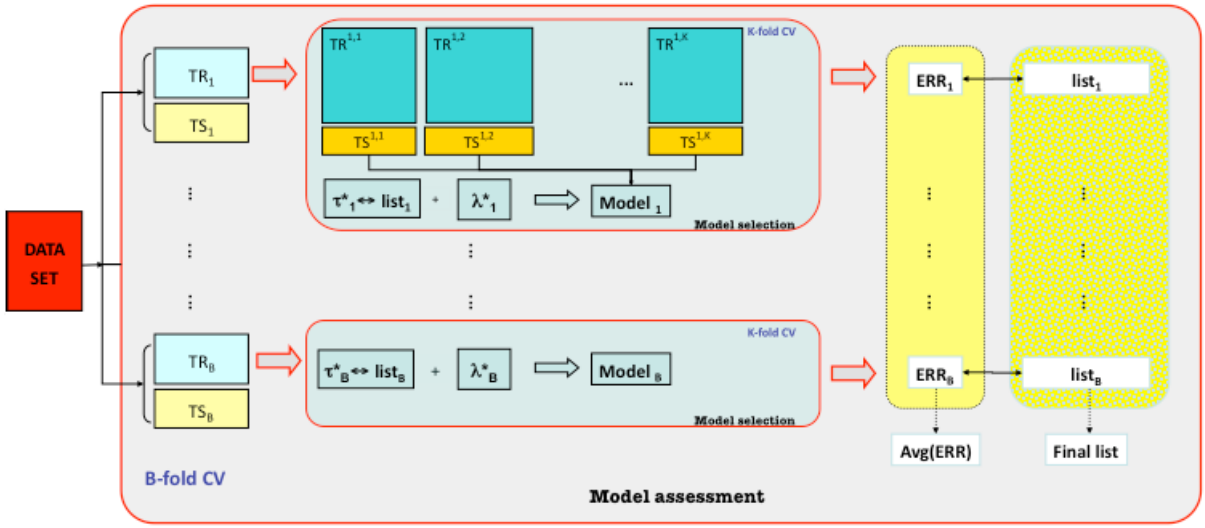
Given the above premises we focused on the elastic net selection method as presented in [DeMol09].



1.1.2 Model assessment

The gene selection step is nested in an external cross validation loop, needed to verify the goodness of the estimated model both in terms of performance stability and significance.

In order to obtain an unbiased estimate of the classification performance [Ambroise02], this step must be carefully designed by holding out a blind test set. Since the available samples are very few compared to the number of variables, this step has to be performed on different subsamplings and its results averaged.



References

1.2 Framework Applications

Such a framework (in a previous Matlab implementation or in the implementation here described), was used in many biological application:

- Barla A., Jurman G., Visintainer R., Squillario M., Filosi M., Riccadonna S. and Furlanello C., **A machine learning pipeline for discriminant pathways identification**. Proceedings CIBB 2011.

- Zycinski G., Barla A. and Verri A., **SVS: Data and knowledge integration in computational biology**. Proceedings of IEEE EMBC 2011, 2011.
- Squillario M. and Barla A., **A computational procedure for functional characterization of potential marker genes from molecular data: Alzheimer's as a case study**. BMC Medical Genomics, 4 (55) , 2011.
- Fardin P., Cornero A., Barla A., Mosci S., Acquaviva M., Rosasco L., Gambini C., Verri A. and Varesio L., **Identification of multiple hypoxia signatures in neuroblastoma cell lines by l1-l2 regularization and data reduction**. Journal of Biomedicine and Biotechnology, 2010.
- Fardin P., Barla A., Mosci S., Rosasco L., Verri A., Versteeg R., Caron H., Molenaar J., Ora I., Eva A., Puppo M. and Varesio L., **A biology-driven approach identifies the hypoxia gene signature as a predictor of the outcome of neuroblastoma patients**. Molecular Cancer, 2010.
- Fardin P., Barla A., Mosci S., Rosasco L., Verri A. and Varesio L., **The l1-l2 regularization framework unmasks the hypoxia signature hidden in the transcriptome of a set of heterogeneous neuroblastoma cell lines**. BMC Genomics, 10 , pp.474, 2009.
- Mosci S., Barla A., Verri A. and Rosasco L., **Finding structured gene signatures**. IEEE Proceedings BIBM 2008, pp.8, 2008

1.3 Quick start tutorial

Assuming L1L2Py and PPlus already installed, **L1L2Signature** may be installed using standard Python tools (with administrative or sudo permissions on GNU-Linux platforms):

```
$ pip install L1L2Signature

or

$ easy_install L1L2Signature
```

1.3.1 Installation from sources

If you like to manually install **L1L2Signature**, download the source tar.gz from our BitBucket repository. Then extract it and move into the root directory:

```
$ tar xvf L1L2Signature-|release|.tar.gz
$ cd L1L2Signature-|release|/
```

From here, you may use the standard Python installation step:

```
$ python setup.py install
```

Note: If you would like to use **L1L2Signature** in a distributed environment you need to install `L1L2Py` and `PPlus` on each node, as described into the documentation of both libraries. You can also run **L1L2Signature** script on a single machine (installing dependencies only there) using `PPlus` debug mode, configurable from **L1L2Signature** configuration file, as described below.

After **L1L2Signature** installation, you should have access to three scripts, named with a common `l1l2_` prefix:

```
$ l1l2_<TAB>
l1l2_analysis.py    l1l2_run.py    l1l2_tau_choice.py
```

This tutorial assumes that you downloaded and extracted **L1L2Signature** source package which contains a `Golub99_Leukemia` directory with Test data, which will be used to show **L1L2Signature**'s tools functionalities.

L1L2Signature needs only 3 ingredients:

- A `gene expressions` matrix
- A set of `labels` for each sample (e.g. phenotype)
- A `configuration` file

1.3.2 Input data format

Input data (gene expression matrix and labels) are assumed to be textual ad separated by a char (delimiter). For example, the given data matrix (of Leukemia gene expressions) is a text file where samples are organized by columns and microarray probes by row and gene expressions values are separated by a comma (`','`).

```
Gene Accession Number,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,34,
AF009426_at,36,58,63,38,120,92,16,169,43,-18,26,-81,123,89,268,171,53,-5,86,267,206,-23,-87,135,-
AF009674_at,-66,-239,-328,307,212,314,-653,332,221,-106,-169,-446,-93,66,123,-60,-404,-146,143,33
AF010193_at,87,548,93,-31,159,101,119,589,245,451,91,484,458,158,128,225,555,288,97,1757,363,598,
AF012270_at,-12,139,237,104,29,-8,223,416,10,59,29,91,-14,105,37,106,485,134,109,104,144,158,106,
AF014958_at,318,243,19,108,40,112,329,166,110,159,96,-71,139,112,-36,345,1057,176,268,180,382,15,
AF015910_at,1219,1055,1820,1074,822,394,1154,1763,1157,902,837,178,1077,171,316,1317,2041,1140,10
AF015913_at,476,611,850,691,1304,346,653,550,826,472,439,239,1558,705,942,560,1366,369,584,2460,0
AF015950_at,-466,-49,-277,-670,-180,-453,-700,-191,-163,-183,-329,-431,-43,-296,-73,334,-436,105,
AJ000480_at,895,749,1049,1016,634,752,920,254,934,536,607,456,617,592,624,608,925,833,473,801,348
AJ001047_at,-451,-108,-265,-378,-171,-257,-514,-250,-291,-218,-129,-137,-146,-128,-159,-223,-311,
AJ001421_at,687,829,914,982,1294,743,680,527,1352,675,383,616,1071,1004,1106,547,1790,574,660,182
...
```

Labels contains information about the given samples, indicated if they belong to the ALL (Acute Lymphoblastic Leukemia) or AML (Acute Myeloid Leukemia) group:

```
sample,status
1,ALL
2,ALL
3,ALL
4,ALL
5,ALL
28,AML
29,AML
30,AML
31,AML
32,AML
33,AML
...
```

See also `l1l2signature.utils.BioDataReader` API and next section for more information.

1.3.3 Configuration File

L1L2Signature configuration file is a standard Python script. It is imported as a module, then all the code is executed. Actually all configuration option are mandatory and it is possible to generate a default configuration file, as the one that follows, with the `l1l2_run.py` script (see *Experiment runner* section):

```
# Configuration file example for L1L2Signature
# version: '0.2.2'

import l1l2py

#~~ Data Input/Output ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# * Data assumed csv with samples and features labels
# * All the path are w.r.t. config file path
data_matrix = 'data/gedm.csv'
labels = 'data/labels.csv'
delimiter = ','
```

```
samples_on = 'col' # or 'row': samples on cols or rows
result_path = '.'

#~~ Data filtering/normalization ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sample_remover = None # removes samples with this label value
variable_remover = 'affx' # remove vars where name starts with (not case-sens.)
data_normalizer = l1l2py.tools.center
labels_normalizer = None

#~~ Cross validation options ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# * See l1l2py.tools.{kfold_splits, stratified_kfold_splits}
external_k = 4    # (None means Leave One Out)
internal_k = 3
cv_splitting = l1l2py.tools.stratified_kfold_splits

#~~ Errors functions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# * See l1l2py.tools.{regression_error, classification_error,
#                     balanced_classification_error}
cv_error = l1l2py.tools.regression_error
error = l1l2py.tools.balanced_classification_error
positive_label = None # Indicates the positive class in case of 2-class task

#~~ L1l2 Parameters ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# * Ranges will be sorted from smaller to bigger value!
# * See l1l2py.tools.{geometric_range, linear_range}
tau_range = l1l2py.tools.geometric_range(1e-3, 0.5, 20) # * MAX_TAU
mu_range = l1l2py.tools.geometric_range(1e-3, 1.0, 3)    # * CORRELATION_FACTOR
lambda_range = l1l2py.tools.geometric_range(1e0, 1e4, 10)

#~~ Signature Parameters ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
frequency_threshold = 0.5

#~~ PPlus options ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
debug = True # If True, the experiment runs only on the local pc cores
```

Configuration file is fully documented and it imports `L1L2Py` in order to use some useful tools. User is free to use personalized functions if they follow the same API. For example, if the user would like to use a different error function, this must be written as:

```
def my_error(true_labels, predicted_labels):
    something(...)
    return error_as_float
```

After the user defines all the option needed to read the data and to perform the model assessment, the crucial phase is to properly define a set of ranges of parameter involved, namely τ, μ and λ (see `L1L2Py` for a complete explanation of them). Usually, a good option for λ is to use a wide range of values in geometric series.

Instead, for τ we automatically scale the range respect to the maximum value (`MAX_TAU`) it can assume. Values greater than `MAX_TAU` produce empty models (no genes are selected). Actually, the `tau_range` option is considered as relative with respect to `MAX_TAU`.

About μ, we implemented an heuristic which follows the same criteria implemented for τ. Because μ is related to the amount of correlation we would like to introduce into the generated signatures, we would like to use relative values with respect to each split (sub)matrix. So, we evaluate a `CORRELATION_FACTOR`, used as scaling factor, which is based on the spectral properties of the correlation (sub)matrix.

1.3.4 Tau choice helper

In order to help users choosing a good *relative* τ range, they can use the `l1l2_tau_choice.py` script which has the following prototype:

```
$ l1l2_tau_choice.py --help
Usage: l1l2_tau_choice.py [-s] configuration-file.py

Options:
  --version   show program's version number and exit
  -h, --help  show this help message and exit
  -s, --show  show interactive plot
```

It needs a properly configured configuration file, with an *initial* tau range and performs the following steps:

- Reads data
- Checks `MAX_TAU` value (it must select at least 1 variable)
- Calculates a full path of $\ell_1\ell_2$ solutions using the given τ range (and a μ very close to zero). User may check minimum and maximum number of selected variables, estimating minimal (without correlation) gene signatures lengths.
- Performs a 5-fold cross validation saving a `tau_choice_plot.png` (as the one in the next Figure) into the configuration file directory. User can estimate a basic (and maybe biased) performance with the given ranges of values.

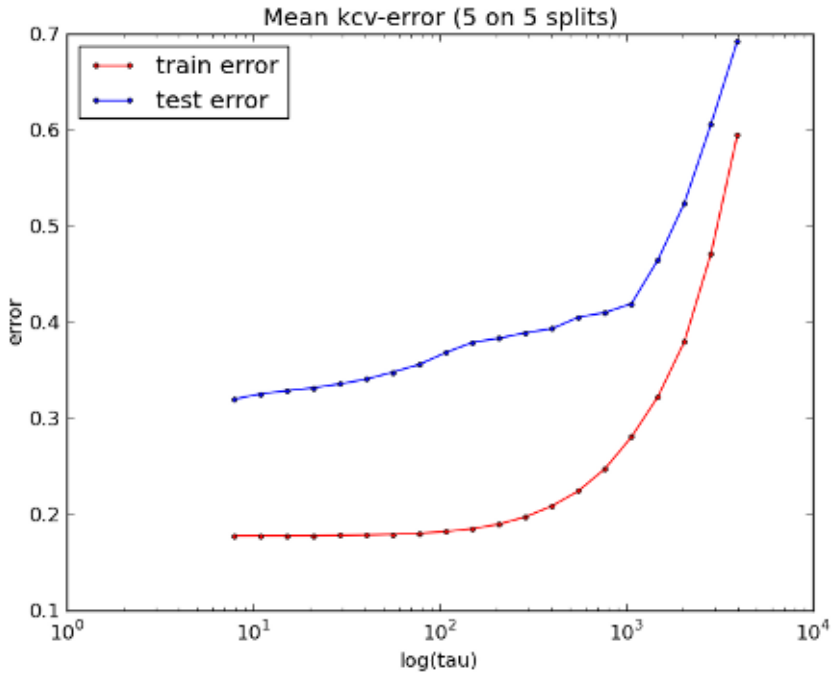


Figure 1.1: **Tau choice plot for Golub dataset with a default configuration file**

Usually we perform this step many times in order to drive our parameters ranges choice, before launching the real experiment (which usually requires many hours).

1.3.5 Experiment runner

The `l1l2_run.py` script, executes the full framework described in the *Framework overview* section. The prototype is the following:

```
$ l1l2_run.py --help
Usage: l1l2_run.py [-c] configuration-file.py

Options:
  --version     show program's version number and exit
  -h, --help    show this help message and exit
  -c, --create  create config file
```

As described before, this script may be also used to generate a valid default configuration file (`-c` option).

When launched, the script reads and splits the data, then it runs `L1L2Py` on each external split and collects the results in a new sub-directory of the `result_path` directory (see *Configuration File*). Such a directory is named as:

```
l1l2_result_<TIMESTAMP>
```

and it contains all information needed for the following analysis step.

Note: Note that data and configuration file are *hard-linked* inside the result directory which, in that way, becomes completely portable and self contained.

1.3.6 Results analysis

This is the last step, needed to be performed in order to get some useful summaries and plots from an already executed experiment. The `l1l2_analysis.py` script accepts as only parameter a result directory already created:

```
$ l1l2_analysis.py --help
Usage: l1l2_analysis.py result-dir

Options:
  --version   show program's version number and exit
  -h, --help  show this help message and exit
  -d DPI, --dpi=DPI  figures dpi resolution (default 300)
  -s, --show         show interactive plots
```

The script prints some results and produces a set of textual and graphical results.

Cross Validation Errors

The script generates a list of `kcv_err_split_*.png`, one for each external split (as averaged error across internal splits). Moreover, it generates an averaged plot: `avg_kcv_err.png`. On each plot, a blue dot indicates the minimum.

See also: `l1l2signature.plots.kfold_errors()`.

Prediction Errors

The script generates a box plot for both test and training errors, respectively `prediction_error_ts.png` and `prediction_error_tr.png`. They show the averaged prediction error over external splits in order to asses performance and stability of the signatures (for each level of considered correlation, μ values).

See also: `l1l2signature.plots.errors_boxplot()`.

Frequencies Threshold

In order to help the user defining a good stability threshold (see `frequency_threshold` in *Configuration File*) the script also plots (and actually print and save as `selected_over_threshold.png`), an overall summary of the number of genes selected for different thresholds and for each correlation level.

See also: `l1l2signature.plots.selected_over_threshold()` and `l1l2signature.utils.selection_summary()`.

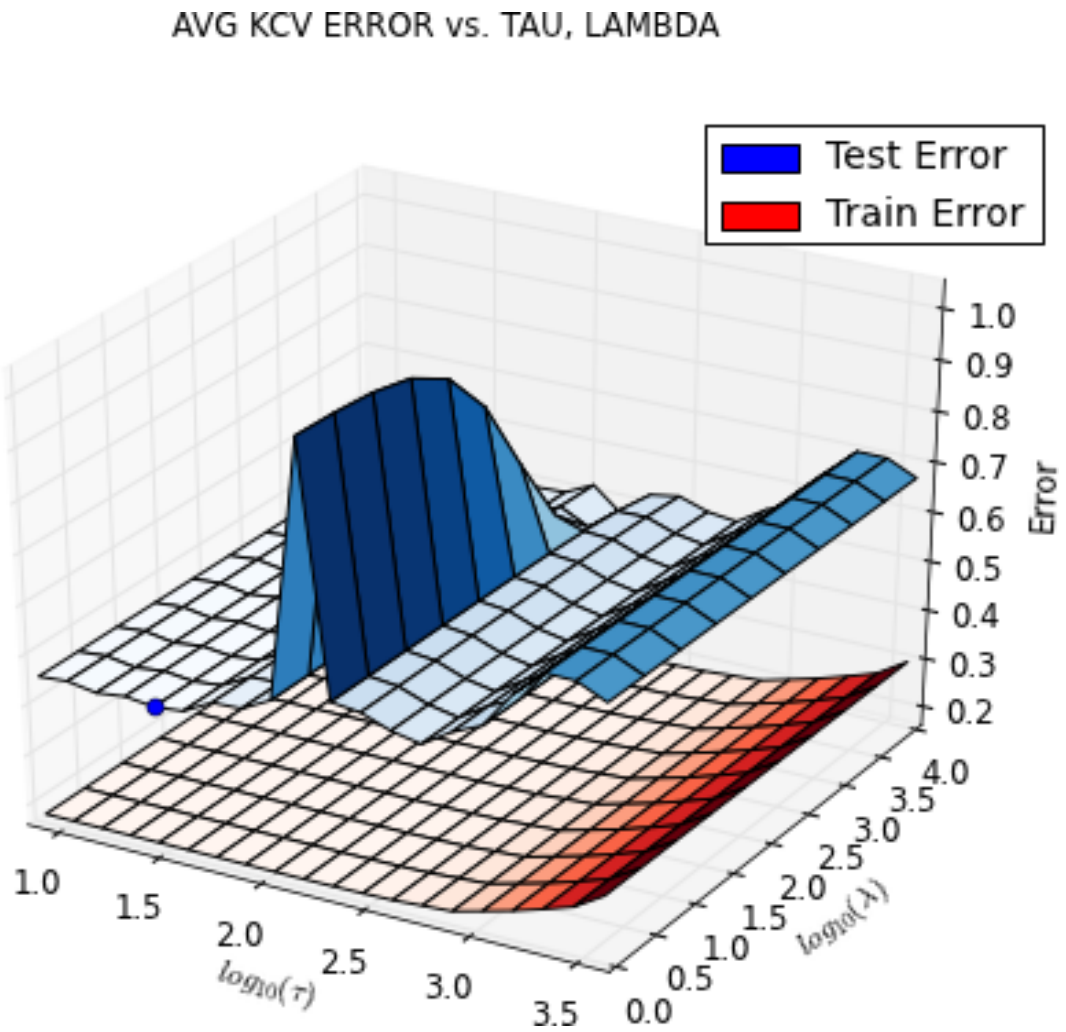


Figure 1.2: **Averaged Cross Validation Error for Golub dataset with a default configuration file**

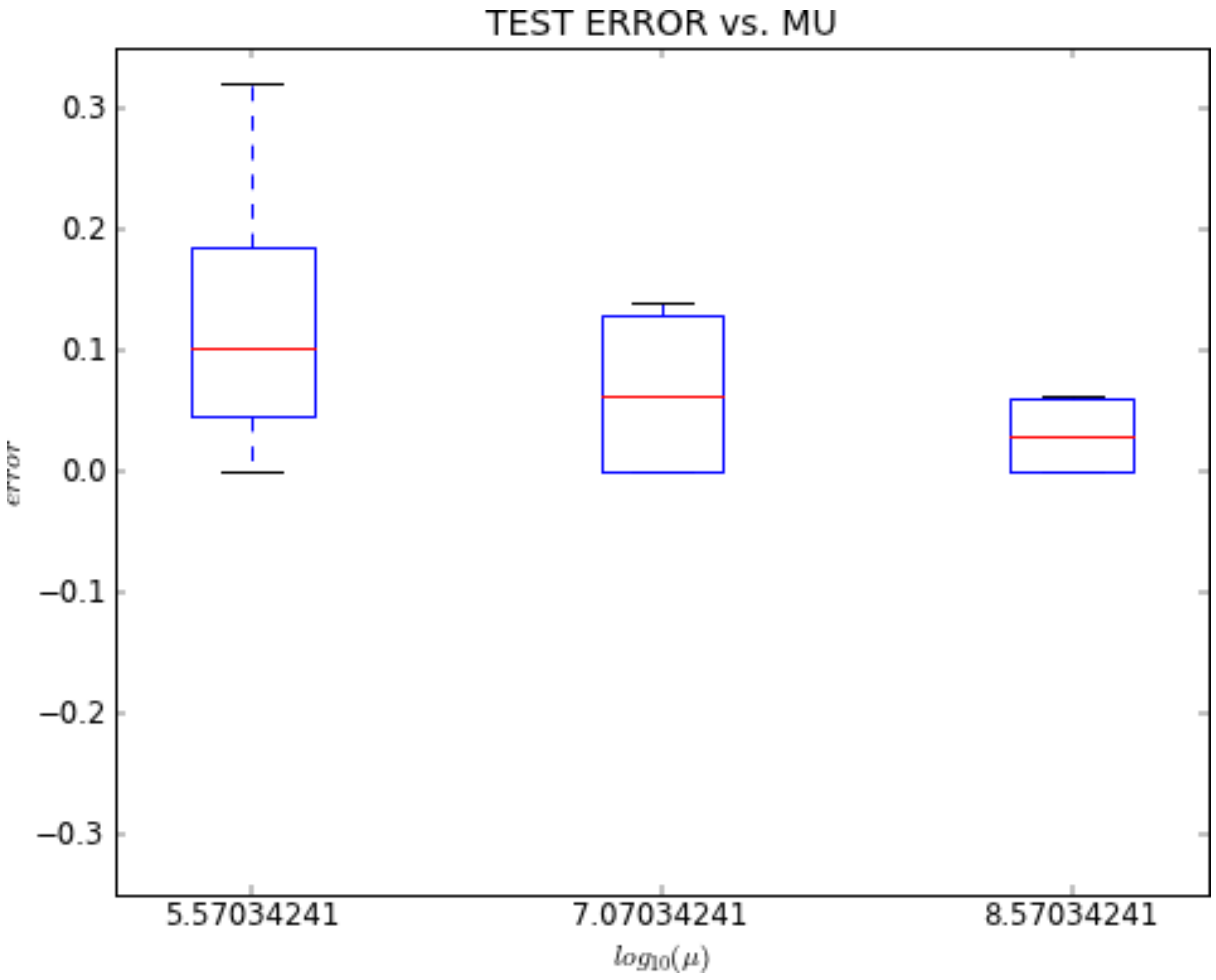


Figure 1.3: **Prediction Error Box Plot for Golub dataset with a default configuration file**

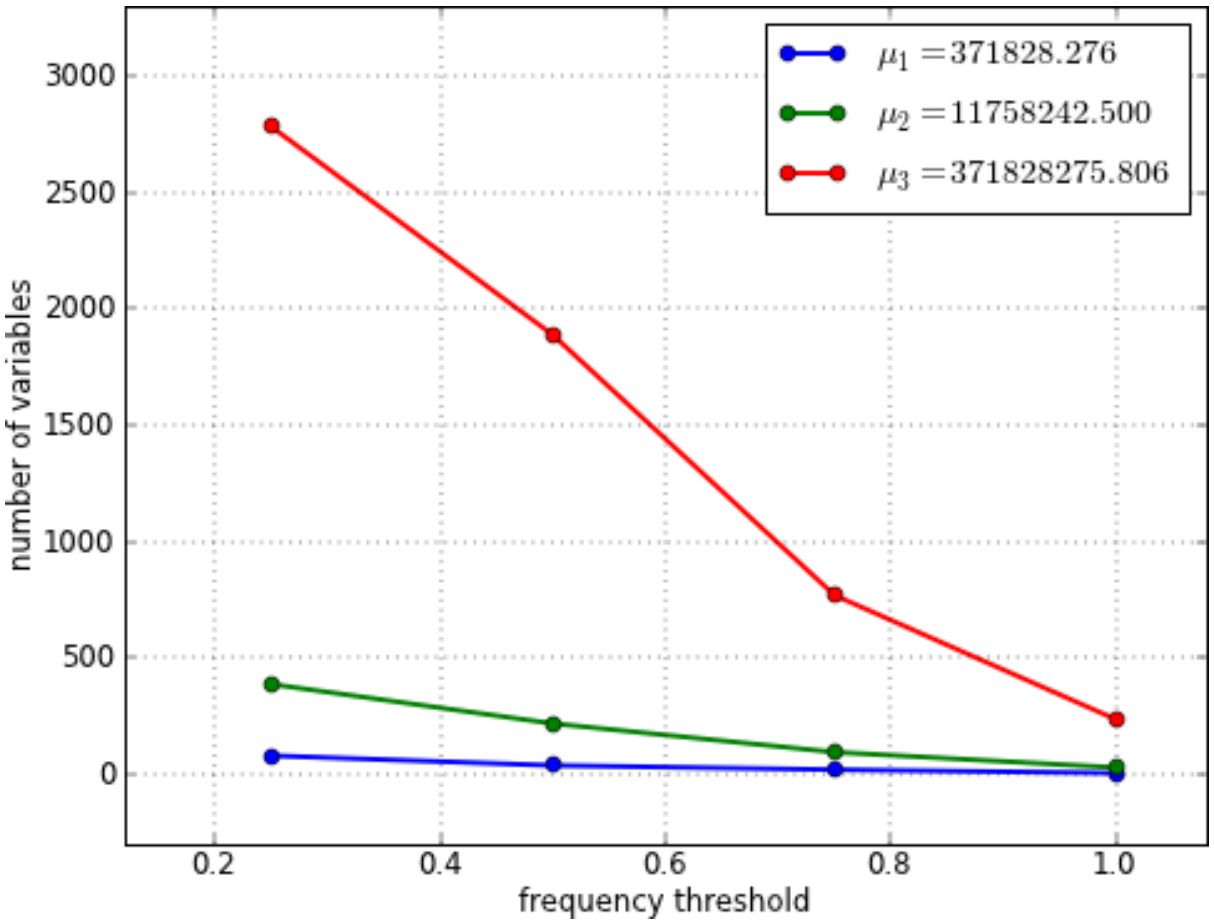


Figure 1.4: **Number of selected variables wrt a frequency (stability) threshold over external split signatures (Golub dataset)**

Signatures Heatmaps

In case of classification (automatically identified when labels assume only two values), the script creates a heatmap plot for each final signature (then they also depend by `frequency_threshold` option value). Images are saved as `heatmap_mu*.png` files where samples and variables are **properly clustered** in order to improve the visualization.

Note: Using a very small `frequency_threshold` (e.g. 0.0), signature contains the full list of variables. In that case, variables are not clustered but only ordered by frequency across splits. In order to generate such a plot, the `l1l2signature.plots.heatmap()` can be used.

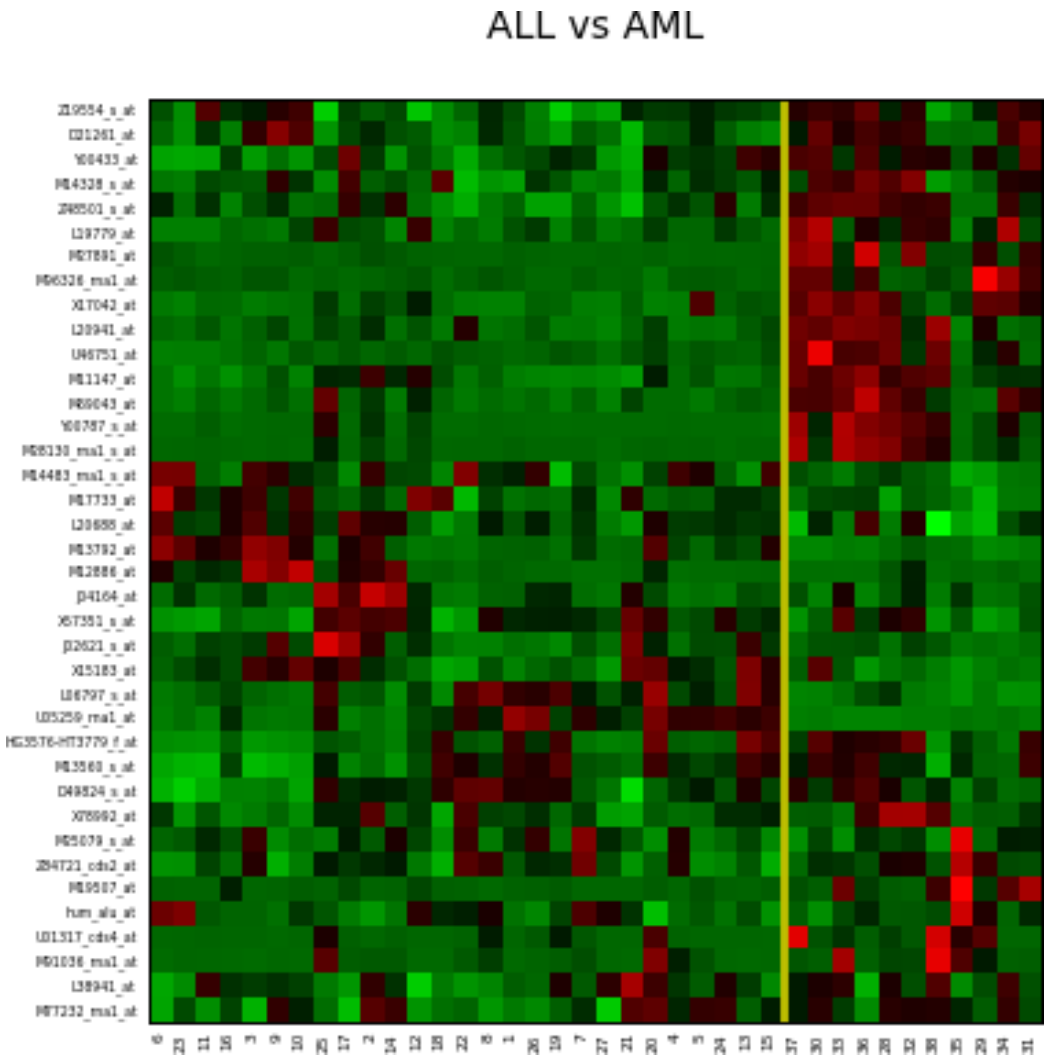


Figure 1.5: **Heatmap for Golub dataset with a default configuration file**

See also: `l1l2signature.plots.heatmap()` and `l1l2signature.utils.ordered_submatrices()`.

Samples in PCA space

In case of classification (automatically identified when labels assume only two values), the script plots samples in a 3D space, using Principal Component Analysis (PCA), for each final signature. Images are saved as `pca_mu*.png` files.

See also: `l1l2signature.plots.pca()`.

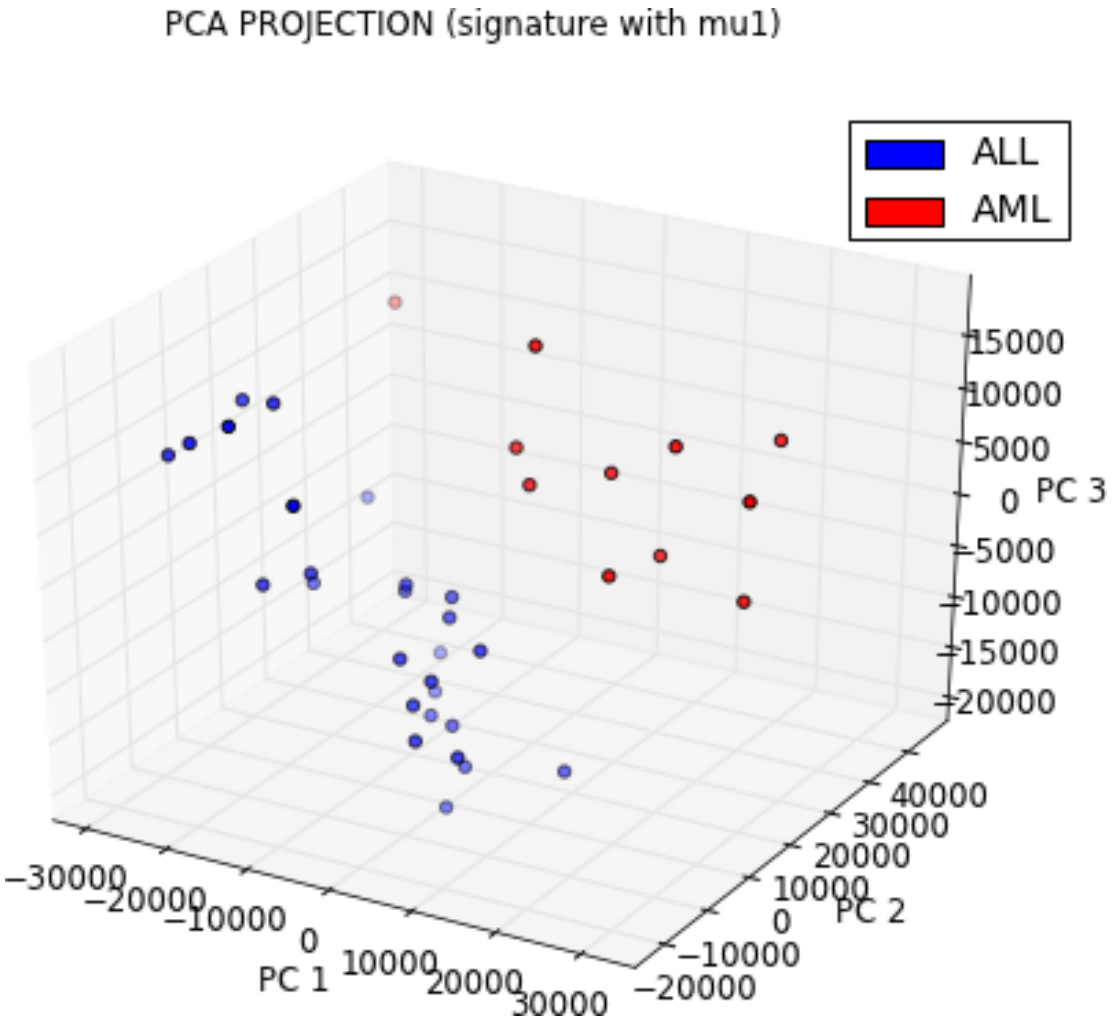


Figure 1.6: **PCA plot for Golub dataset with a default configuration file**

Performance Statistics

The analysis script also produces some textual results, saved into a `stats.txt` file. That file is divided into some sections, each one containing at least a short table.

Optimal Parameters

```
Optimal Parameters
=================================================================================
Split  #  |    lambda*    |     tau*      |   log(lam*)   |   log(tau*)
----------+---------------+---------------+---------------+---------------
Split   1 |         1.000 |        40.550 |         0.000 |         1.608
Split   2 |         1.000 |       555.132 |         0.000 |         2.744
Split   3 |         1.000 |       150.035 |         0.000 |         2.176
Split   4 |         1.000 |        56.239 |         0.000 |         1.750
```

This table describes the best parameter pairs (λ^*, τ^*) found in each cross validation split. They correspond to blue points in generated *Cross Validation Errors* images.

Prediction errors

```
Test set Prediction
=========================================================
Stat type |      mu1      |      mu2      |      mu3
----------+---------------+---------------+---------------
mean      |         0.131 |         0.066 |         0.031
std       |         0.139 |         0.077 |         0.035
median    |         0.101 |         0.062 |         0.030
----------+---------------+---------------+---------------
sqrt(mean)|         0.361 |         0.257 |         0.175
sqrt(std) |         0.372 |         0.277 |         0.188
sqrt(med) |         0.318 |         0.250 |         0.173


Training set Prediction
=========================================================
Stat type |      mu1      |      mu2      |      mu3
----------+---------------+---------------+---------------
mean      |         0.000 |         0.000 |         0.000
std       |         0.000 |         0.000 |         0.000
median    |         0.000 |         0.000 |         0.000
----------+---------------+---------------+---------------
sqrt(mean)|         0.000 |         0.000 |         0.000
sqrt(std) |         0.000 |         0.000 |         0.000
sqrt(med) |         0.000 |         0.000 |         0.000
```

These tables show the averaged prediction error of the signatures, before frequency/stability thresholding, for each value of correlation μ. They correspond to the generated *Prediction Errors* box plots.

Classification Performances

```
Confusion matrix and classification performance with mu3
=================================================================================
  Real vs Pred. |        AML         |        ALL         |   Predictive Values
----------------+--------------------+--------------------+--------------------
            AML |          11        |           2        |             84.62 %
----------------+--------------------+--------------------+--------------------
            ALL |           0        |          25        |            100.00 %
----------------+--------------------+--------------------+--------------------
     True rates |      100.00 %      |      92.59 %       |
```

This table, generated only in classification for each μ, summarizes classification performances of the signature. It is a classical *Confusion Matrix* where, e.g. the entry AML vs ALL indicates the number of real AML sam-

ples classified as belonging to ALL class (results generate on given Golub "Leukemia" dataset). On the bottom we have *True rates*: `#(Correctly predicted as C) / #(Predicted as C)` while on the right we have *Predictive values*: `#(Correctly predicted as C) / #(Samples in C)`

Moreover, this section contains some other performance measures:

```
Classification performance measures:
  * Accuracy:          94.74 %
  * Balanced Accuracy: 96.30 %
  * MCC:               0.8851
```

where, MCC is the *Matthews correlation coefficient*.

At last, if the `positive_label` parameter is given, into the *Configuration File*, the script is able to calculate some other measures that assume the presence of a *positive class* as in the case of patients vs. controls. For example, the following table is generated if we assume that for the "Leukemia" dataset the **AML** class has to be considered as positive:

```
Considering AML as the positive class:
  * Sensitivity:   100.00 %
  * Specificity:   92.59 %
  * Precision:     84.62 %
  * Recall:        100.00 %
  * F-measure:     0.9167
```

Note that some of this measures are pure aliases of the ones already calculated on the last row or column of the confusion matrix.

See also: `l1l2signature.utils.confusion_matrix()` and `l1l2signature.utils.classification_measu`

Signatures

Obviously, the script generates a set of signatures, each one written in a separated text file `signature_mu*.txt`, in order to eventually simplify the parsing. Each file contains the ordered list of probes belonging to the signature:

```
index        probe        freq        num_sel
1823        M27891_at        1.000        4/4
4877        Y00433_at        1.000        4/4
1720        M19507_at        1.000        4/4
6142        Y00787_s_at        1.000        4/4
2343        M96326_rna1_at        1.000        4/4
6150        Z19554_s_at        0.750        3/4
1317        L19779_at        0.750        3/4
5589        D49824_s_at        0.750        3/4
6730        HG3576-HT3779_f_at        0.750        3/4
6120        M14328_s_at        0.750        3/4
6825        U01317_cds4_at        0.750        3/4
2286        M91036_rna1_at        0.750        3/4
5249        L20688_at        0.750        3/4
5493        L06797_s_at        0.750        3/4
4560        X78992_at        0.750        3/4
1050        J04164_at        0.750        3/4
6554        Z48501_s_at        0.750        3/4
1615        M11147_at        0.750        3/4
1335        L20941_at        0.750        3/4
0        hum_alu_at        0.750        3/4
1704        M17733_at        0.500        2/4
3199        U46751_at        0.500        2/4
1645        M13792_at        0.500        2/4
1635        M12886_at        0.500        2/4
224        D21261_at        0.500        2/4
6109        M13560_s_at        0.500        2/4
```

```
2583        U05259_rna1_at        0.500        2/4
1491        L38941_at        0.500        2/4
4137        X17042_at        0.500        2/4
5880        J02621_s_at        0.500        2/4
6122        M14483_rna1_s_at        0.500        2/4
5657        X57351_s_at        0.500        2/4
2127        M69043_at        0.500        2/4
2175        M77232_rna1_at        0.500        2/4
6141        M28130_rna1_s_at        0.500        2/4
5651        M25079_s_at        0.500        2/4
5170        Z84721_cds2_at        0.500        2/4
4099        X15183_at        0.500        2/4
```

The file is tab-delimited, the signatures are thresholded with respect to the `frequency_threshold` option, and they correspond to the signatures used to generate *Heatmaps plots*.

See also: `l1l2signature.utils.signatures()` and `l1l2signature.utils.selection_summary()`.

CHAPTER

TWO

PUBLIC API

2.1 Utility functions and classes (`utils`)

This module contains functions and classes useful to manipulate input data (e.g. gene expressions, labels), create outputs and collect results.

2.1.1 Data and parameters

exception `l1l2signature.utils.`**`L1L2SignatureException`**
Exception raised by `L1L2Signature` classes and functions.

class `l1l2signature.utils.`**`BioDataReader`**(*data_file*, *labels_file*, *sample_remover=None*, *variable_remover=None*, *delimiter=','*, *samples_on='col'*, *positive_label=None*)
Biological Data reader.

This class reads a pair of CSV files containing respectively a data matrix and a list of labels.

The reader can discard some samples and some variables according to give arguments (as described below) and assumes the presence of an header line in both files.

If labels file contains exactly 2 labels, *BioDataReader* automatically maps two classes in -1 and +1. Then, the *labels_reverse* attribute will contain a dictionary mapping from numeric to string labels (otherwise it is None). Otherwise *BioDataReader* assumes to find numeric values (regression task).

Parameters

data_file : file or str

File, filename, or generator to read (see also `numpy.loadtxt()`)

labels_file : file or str

File, filename, or generator to read (see also `numpy.loadtxt()`)

variable_remover : int or None, optional (default None)

Variable names prefix used to discard samples (e.g. `AFFX` for Affymetrix Gene Expression MicroArray)

sample_remover : str or None, optional (default None)

Label value used to discard samples prefix. This value must refer to the original labels into the labels file.

delimiter : str, optional (default ',')

CSV char delimiter (if it is not a comma)

samples_on : 'col' or 'row', optional (default 'col')

Indicates if the samples are arranged on rows or columns into the data file

positive_label : str, optional (default None)

Indicates what label has to be considered as the positive class (mapped to +1 value). If None, mapping follows a lexicographic order.

Examples

```
>>> from l1l2signature.utils import BioDataReader
>>> from cStringIO import StringIO
>>> STD_DATA = '\n'.join(['probe, A,   B,   C,   D',
...                       'p1,    0.0, 0.1, 0.2, 0.3',
...                       'p2,    0.0, 0.1, 0.2, 0.3',
...                       'p3,    0.0, 0.1, 0.2, 0.3',
...                       'p4,    0.0, 0.1, 0.2, 0.3',
...                       'p5,    0.0, 0.1, 0.2, 0.3'])
>>> STD_LABELS = '\n'.join(['name, value',
...                         'A,    1',
...                         'B,    0',
...                         'C,    1',
...                         'D,    1'])
>>> br = BioDataReader(StringIO(STD_DATA), StringIO(STD_LABELS))
>>> print br.samples
['A' 'B' 'C' 'D']
>>> print br.variables
['p1' 'p2' 'p3' 'p4' 'p5']
>>> print br.data
[[ 0.   0.   0.   0.   0. ]
 [ 0.1  0.1  0.1  0.1  0.1]
 [ 0.2  0.2  0.2  0.2  0.2]
 [ 0.3  0.3  0.3  0.3  0.3]]
>>> print br.labels
[ 1. -1.  1.  1.]
>>>
```

Attributes

data	numpy.ndarray	Data matrix (float) of dimensions samples X variables
labels	numpy.ndarray	Labels array (float)
samples	numpy.ndarray	Samples names (str)
variables	numpy.ndarray	Variables names (str)
labels_reverse	dict	Mapping from -1 and +1 classes to original string labels. The attribute is None if automatic mapping is not performed.

class l1l2signature.utils.**RangesScaler**(*data*, *labels*, *data_normalizer=None*, *labels_normalizer=None*)

Given data and labels helps to scale L1L2 parameters ranges properly.

This class works on tau and mu ranges passed to the l1l2 selection framework (see also l1l2py.model_selection() and related function for details).

Scaling ranges permits to use relative (and not absolute) ranges of parameters.

Attributes

norm_data	numpy.ndarray	Normalized data matrix.
norm_labels	numpy.ndarray	Normalized labels vector.

tau_range(*trange*)

Returns a scaled tau range.

Tau scaling factor is the maximum tau value to avoid and empty solution (where all variables are discarded). The value is estimated on the maximum correlation between data and labels.

Parameters

trange : numpy.ndarray

Tau range containing relative values (expected maximum is lesser than 1.0 and minimum greater than 0.0).

Returns

tau_range : numpy.ndarray

Scaled tau range.

Raises

L1L2SignatureException :

If trange values are not in the [0, 1) interval (right extreme excluded).

mu_range(*mrange*)

Returns a scaled mu range.

Mu scaling factor is estimated on the maximum eigenvalue of the correlation matrix and is used to simplify the parameters choice.

Parameters

mrange : numpy.ndarray

Mu range containing relative values (expected maximum is lesser than 1.0 and minimum greater than 0.0).

Returns

mu_range : numpy.ndarray

Scaled mu range.

Raises

L1L2SignatureException :

If mrange values are not all greater than 0.

tau_scaling_factor

Tau scaling factor calculated on given data and labels.

mu_scaling_factor

Mu scaling factor calculated on given data matrix.

2.1.2 Results analysis

l1l2signature.utils.**ordered_submatrices**(*data*, *labels*, *signatures_idxs*)

Returns a list of sorted and filtered submatrices.

The matrices are sorted by `labels` and filtered by `signatures_idxs`.

Parameters

labels : list

Data labels

signatures_idxs : list of numpy.ndarray

Each list item contains a signature in terms of variables boolean mask or indexes.
If indexes are given, submatrices are also properly ordered.

Returns

labels_idxs : numpy.ndarray

Labels ordering used to produce submatrices.

sub_matrices : list of numpy.ndarray

List of ordered and filtered submatrices.

Examples

```
>>> from l1l2signature.utils import ordered_submatrices
>>> data = [[1., 2., 3.],
...         [4., 5., 6.],
...         [7., 8., 9.]]
>>> labels = [1, -1, 1]
>>> signatures_idxs = [[True, False, False],
...                    [0, 1, 2],
...                    [2, 1]]
>>> labels_idxs, sub_matrices = ordered_submatrices(data, labels,
...                                                 signatures_idxs)
>>> print labels_idxs
[1 0 2]
>>> print sub_matrices[0]
[[ 4.]
 [ 1.]
 [ 7.]]
>>> print sub_matrices[1]
[[ 4.  5.  6.]
 [ 1.  2.  3.]
 [ 7.  8.  9.]]
>>> print sub_matrices[2]
[[ 6.  5.]
 [ 3.  2.]
 [ 9.  8.]]
```

l1l2signature.utils.**signatures**(*splits_results*, *frequency_threshold=0.0*)

Returns (almost) nested signatures for each correlation value.

The function returns 3 lists where each item refers to a signature (for increasing value of linear correlation). Each signature is orderer from the most to the least selected variable across KCV splits results.

Parameters

splits_results : iterable

List of results from L1L2Py module, one for each external split.

frequency_threshold : float

Only the variables selected more (or equal) than this threshold are included into the signature.

Returns

sign_totals : list of `numpy.ndarray`.

Counts the number of times each variable in the signature is selected.

sign_freqs : list of `numpy.ndarray`.

Frequencies calculated from `sign_totals`.

sign_idxs : list of `numpy.ndarray`.

Indexes of the signatures variables .

Examples

```
>>> from l1l2signature.utils import signatures
>>> splits_results = [{'selected_list':[[True, False], [True, True]]},
...                   {'selected_list':[[True, False], [False, True]]}]
>>> sign_totals, sign_freqs, sign_idxs = signatures(splits_results)
>>> print sign_totals
[array([ 2.,  0.]), array([ 2.,  1.])]
>>> print sign_freqs
[array([ 1.,  0.]), array([ 1. ,  0.5])]
```

```
>>> print sign_idxs
[array([0, 1]), array([1, 0])]
```

l1l2signature.utils.**selection_summary**(*splits_results*)

Counts how many times each variables was selected.

Parameters

splits_results : iterable

List of results from L1L2Py module, one for each external split.

Returns

summary : `numpy.ndarray`

Selection summary. `# mu_values X # variables` matrix.

l1l2signature.utils.**confusion_matrix**(*labels*, *predictions*)

Calculates a confusion matrix.

From given real and predicted labels, the function calculated a confusion matrix as a double nested dictionary. The external one contains two keys, `'T'` and `'F'`. Both internal dictionaries contain a key for each class label. Then the `['T']['C1']` entry counts the number of correctly predicted `'C1'` labels, while `['F']['C2']` the incorrectly predicted `'C2'` labels.

Note that each external dictionary correspond to a confusion matrix diagonal and the function works only on two-class labels.

Parameters

labels : iterable

Real labels.

predictions : iterable

Predicted labels.

Returns

cm : dict

Dictionary containing the confusion matrix values.

l1l2signature.utils.**classification_measures**(*confusion_matrix*, *positive_label=None*)

Calculates some classification measures.

Measures are calculated from a given confusion matrix (see `confusion_matrix()` for a detailed description of the required structure).

The `positive_label` arguments allows to specify what label has to be considered the positive class. This is needed to calculate some measures like F-measure and set some aliases (e.g. precision and recall are respectively the 'predictive value' and the 'true rate' for the positive class).

If `positive_label` is None, the resulting dictionary will not contain all the measures. Assuming to have to classes 'C1' and 'C2', and to indicate 'C1' as the positive (P) class, the function returns a dictionary with the following structure:

```
{
    'C1': {'predictive_value': --,  # TP / (TP + FP)
           'true_rate':        --}, # TP / (TP + FN)
    'C2': {'predictive_value': --,  # TN / (TN + FN)
           'true_rate':        --}, # TN / (TN + FP)
    'accuracy':          --,        # (TP + TN) / (TP + FP + FN + TN)
    'balanced_accuracy': --,        # 0.5 * ( (TP / (TP + FN)) +
                                    #         (TN / (TN + FP)) )
    'MCC':               --,        # ( (TP * TN) - (FP * FN) ) /
                                    # sqrt( (TP + FP) * (TP + FN) *
                                    #       (TN + FP) * (TN + FN) )
```

```
        # Following, only with positive_labels != None
        'sensitivity':       --,        # P true rate: TP / (TP + FN)
        'specificity':       --,        # N true rate: TN / (TN + FP)
        'precision':         --,        # P predictive value: TP / (TP + FP)
        'recall':            --,        # P true rate: TP / (TP + FN)
        'F_measure':         --         # 2. * ( (Precision * Recall ) /
                                        #        (Precision + Recall) )
    }
```

Parameters

confusion_matrix : dict

Confusion matrix (as the one returned by `confusion_matrix()`).

positive_label : str

Positive class label.

Returns

summary : dict

Dictionary containing calculated measures.

2.2 Plotting functions (`plots`)

This module contains all the utilities used to plot useful results.

`l1l2signature.plots.`**`kfold_errors`**(*xrange*, *yrange*, *labels*, *ts_errors*, *tr_errors=None*, *fig_num=None*)

Returns a matplotlib figure object containing a kfold error plot.

Parameters

xrange : iterable

Range of values on the x-axis

yrange : iterable

Range of values on the y-axis

labels : iterable

Pair of string labels for X and Y axes

ts_errors : `numpy.ndarray`

Test Error matrix (`float`) of dimensions `len(xrange) X len(yrange)`

tr_errors : `numpy.ndarray`, optional

Train Error matrix (`float`) of dimensions `len(xrange) X len(yrange)`

fig_num : int, optional

Figure Number. If not given a new figure is initialized

Returns

fig : `matplotlib.figure.Figure`

Created figure handle

`l1l2signature.plots.`**`errors_boxplot`**(*errors*, *positions*, *label=None*, *title=None*, *fig_num=None*)

Returns a matplotlib figure object containing errors box plots.

Parameters

errors : `numpy.ndarray`

Error matrix (`float`) of dimensions `K X len(positions)`

positions : `numpy.ndarray`

Box plot x axis

label : str, optional

X-Axis label

title : str, optional

Plot title

fig_num : int, optional

Figure Number. If not given a new figure is initialized

Returns

fig : `matplotlib.figure.Figure`

Created figure handle

l1l2signature.plots.**heatmap**(*submatrix*, *labels*, *sample_names=None*, *var_names=None*, *clustering_method='ward'*, *clustering_metric='euclidean'*, *var_preorder=None*, *fig_num=None*)

Returns a matplotlib figure object containing an heatmap plot.

If *scipy* is not installed samples and variables will be shown is given order.

Parameters

submatrix : `numpy.ndarray`

Submatrix obtained from a signature.

labels : `numpy.ndarray`

Samples labels.

sample_names : iterable or str, optional

Sample names. If None, heatmap will be anonymous.

var_names : iterable or str, optional

Variable names. If None, heatmap does not contain variables labels.

clustering_method : str, optional (default 'ward')

Clustering method used to order samples and variables. See `scipy.cluster.hierarchy.linkage()` function.

clustering_metric : str, optional (default 'euclidean')

Clustering metric used to order samples and variables. See `scipy.cluster.hierarchy.linkage()` function.

var_preorder : `numpy.ndarray` like, optional (default None)

If given, variables are not clustered but given indexes are used to order them.

fig_num : int, optional

Figure Number. If not given a new figure is initialized.

Returns

fig : `matplotlib.figure.Figure`

Created figure handle

l1l2signature.plots.**pca**(*submatrix*, *labels*, *fig_num=None*)

Returns a matplotlib figure containing sample points.

Starting from given `submatrix` calculates a PCA projection to plot samples in a 3D-space. If the signatures contains only 2 or 3 variables, PCA is obviously not performed.

Parameters

submatrix : `numpy.ndarray`

Submatrix obtained from a signature.

labels : `numpy.ndarray`

Samples labels.

fig_num : int, optional

Figure Number. If not given a new figure is initialized

Returns

fig : `matplotlib.figure.Figure`

Created figure handle

`l1l2signature.plots.`**`selected_over_threshold`**(*frequencies*, *mu_range*, *fig_num=None*)

Returns a figure containing a plot of selected vars cumulative counting.

For each mu value plots a curve which indicates how many variables are been selected for each frequency threshold.

Parameters

frequencies : `numpy.ndarray`

List of `len(mu_range)` lists containing coordinates to plot.

mu_range : class:*numpy.ndarray*

Range of mu values.

fig_num : int, optional

Figure Number. If not given a new figure is initialized

Returns

fig : `matplotlib.figure.Figure`

Created figure handle

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*

BIBLIOGRAPHY

[Ambroise02] Ambroise C., McLachlan G. J., **Selection bias in gene extraction on the basis of microarray gene-expression data**. Proceedings of the National Academy of Sciences of the United States of America, 99(10), 6562-6, 2002

[Barla08] Barla A., Mosci S., Rosasco L., Verri A., **A method for robust variable selection with significance assessment**. European Symposium on Artificial Neural Networks, 2008

[DeMol09] De Mol C., Mosci S., Traskine M., Verri A., **A Regularized Method for Selecting Nested Group of Genes from Microarray Data**. Journal of Computational Biology, vol. 16, pp. 677-690, 2009.

PYTHON MODULE INDEX

l

PYTHON MODULE INDEX

l

INDEX