EPCY

Release 0.1.0

Eric Olivier Audemard

Mar 08, 2023

DOCUMENTATION INDEX

EPCY is a method used to rank genes (features) according to their potential as predictive (bio)markers, using quantitative data (like gene expression).

Installation

Using PyPI

```
pip install epcy
epcy -h
```

Overview

Similarly to Differential Expression analyses, EPCY take as input two tabulated files:

- a table which describes the comparative design of the experience
- a matrix which contains quantitative data for each gene (feature) and sample

Using these two input files, EPCY will evaluate the predictive capacity of each gene individually and return predictive scores, along with their confidence intervals.

To guarantee the reliability of predictive scores, EPCY uses a leave-one-out cross validation to train multiple Kernel Density Estimation (KDE) classifiers and evaluate their performances on unseen samples (see method for more details).

Background

EPCY is a product of the Leucegene project and has been developed and tested specifically to analyse RNA-seq data of acute myeloid leukemia (AML) patients. However, the method implemented in EPCY is generic and should work on different quantitative data.

Citing

While we are finalizing work on the official paper, more details can be found in a poster presented at ISMB ECCB 2019:

Audemard E, Sauvé L and Lemieux S. EPCY: Evaluation of Predictive CapabilitY for ranking biomarker gene candidates [version 1; not peer reviewed]. *F1000Research 2019*, 8(ISCB Comm J):1349(poster)

CHAPTER

ONE

FIRST STEPS WITH EPCY

EPCY is implemented in python3 and allows to perform predictive analyses using bash command line.

If it's not already done, you need to install EPCY

```
pip install epcy
epcy -h
```

1.1 Input files

EPCY is designed to work on several quantitative data (like genes expression), provided that this data is normalized, that is quantitative values for each genes (features) need to be comparable between each samples. For RNA-seq data, TPM, FPKM or even counts per million reads (CPM) values would be appropriate (normalization per transcript length is not critical since we will be comparing quantities between samples, not within samples).

To run EPCY, you need two **tabulated** files, as input:

- A **matrix of quantitative data**, normalized for each samples (in columns) with an *ID* column to identify each genes (features), in first.

Table 1: Example of a quantitative matrix

id	Sample1	...	SampleX
Gene1	10	...	20
...	...	...	...
GeneY	12	...	16

- A **design table** which describes the parameters or conditions on which to perform the analyses. This table is composed of a first column *Sample*, followed by at least one column which describes each sample for each parameter. A new column is needed for each parameter. For a gene knock-out experiment for instance, a column would indicate which samples are wild-type and which is knock-out.

Table 2: Example of a design table

sample	Condition	Knockout_exp	Gender
Sample1	Query	KO	F
...	...	...	...
SampleX	Ref	WT	M

1.2 Download input files

For this tutorial, we propose to download a part of the Leucegene cohort composed by 88 Acute Myeloid Leukemia (AML) individual samples. To reduce the execution time, we are going to only analyze coding genes (19,892 genes). These input files are available in a specific git repo named *epcy_tuto*, and can be downloaded using *git*:

```
git clone git@github.com:iric-soft/epcy_tuto.git
cd epcy_tuto/data/leucegene
```

If you examine the *design.txt* file, you can see that an *AML* column is used to classify each sample into one of these 3 subtypes of AML: *t15_17*, *inv16* and *other*. These refer to samples that show either a chromosome 15-17 translocation, a chromosome 16 inversion or other defect respectively.

Table 3: design.txt

Sample	AML
01H001	Other
01H002	t15_17
...	...
13H120	inv16
...	...
14H133	t15_17

We will start by comparing *t15_17* samples versus all other samples (*inv16* and *other*). On a macbook pro 2 GHz Dual-Core Intel Core i5, this analysis takes 10 min, using 4 thread.

1.3 Run your first EPCY analysis

EPCY is divided into severals tools, which can be listed using:

```
epcy -h
```

Among all these tools, *epcy pred* is the one which allows to run a default comparative predictive analysis. In our current case, we would write:

```
epcy pred --log -t 4 -m cpm.tsv  -d design.txt --condition AML --query t15_17 -o ./29_
↪t15_17_vs_59/
# Or if you only want compare versus inv16 subgroup
epcy pred --log -t 4 -m cpm.tsv  -d design.txt --condition AML --query t15_17 --ref␣
↪inv16 -o ./29_t15_17_vs_27_inv16/
```

where:

- **--log**: specifies that quantitative data needs to to be log transformed before being analyzed.
- **-t 4**: allows to use 4 threads for the analysis.
- **-m cpm.tsv**: specifies the quantitative matrix file.
- **-d design.txt**: specifies the design table.
- **--condition AML**: determines the condition column we want use.
- **--query t15_17**: specifies which subgroup of AML samples we want to compare to all the other.
- **-o ./29_t15_17_vs_59/**: specifies the output directory.

More information can be found, using *epcy pred -h*.

If everything is correct, the analysis will complete by displaying the following output:

```
15:43:02: Read design and matrix features
15:43:04: 181 features with sum==0 have been removed.
15:43:04: Start epcy analysis of 19766 features
15:52:22: Save epcy results
15:52:22: End
```

1.4 Results

predictive_capability.tsv is the main output of an EPCY analysis. It is a tabulated file which contains the evaluation of each genes (features) for its predictive value, using 9 columns:

- **id**: the id of each gene (feature).
- **l2fc**: log2 fold change.
- **kernel_mcc**: Matthews Correlation Coefficient (MCC) compute by a predictor using KDE.
- **kernel_mcc_low**: lower bound of the confidence interval (90%).
- **kernel_mcc_high**: upper bound of the confidence interval (90%).
- **mean_log2_query**: average of log transformed values of this feature for samples in the subgroup of interest defined using the –query parameter.
- **mean_log2_ref**: average of log transformed values of this feature for samples in the reference group.
- **bw_query**: estimated bandwidth used by KDE, to calculate the density of query samples.
- **bw_ref**: estimated bandwidth used by KDE, to calculate the density of ref samples.

Genes (features) with the highest *kernel_mcc* values correspond to the most prodictive ones. The file may then be sorted on that column to obtain the following:

Table 4: ./29_t15_17_vs_59/predictive_capability.tsv ordered on kernel_mcc

id	l2fc	kernel_mcc	kernel_mcc_low	kernel_mcc_high	mean_query	mean_ref	bw_query	bw_ref
ENSG00000183570.16	3.97	0.98	0.95	1	4.84	0.87	0.24	0.42
ENSG00000168004.9	3.75	0.97	0.97	0.97	4.00	0.24	0.28	0.10
ENSG00000089820.15	-4.25	0.97	0.62	0.97	4.21	8.47	0.34	0.23
…	…	…	…	…	…	…	…	…

Note: Since EPCY uses some random steps in its implementation, you may observe small variations in your results. The argument -- *randomseed 42* can be used to obtain the exact same results (see Reproductibility section).

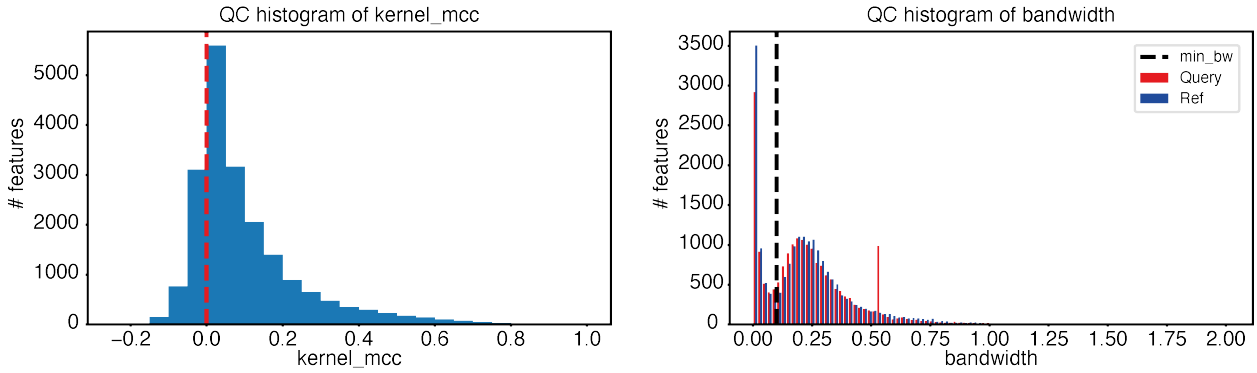
1.5 Quality control

EPCY needs to have enough data to train the KDE classifier and evaluate the predictive capability of each gene (feature) accurately. Without enough samples, EPCY will overfit and return a large number of negative MCC.

Unfortunately, it is *a priori* difficult to detemine a lower bound for the number samples needed, as this number will depend on the dataset analyzed. However, EPCY provides some quality control tools (*epcy qc*), to verify if there is overfitting or not, by checking the distribution of MCC and bandwidth.

Using *epcy qc*, we can plot two quality control figures, as follow:

```
epcy qc -p ./29_t15_17_vs_59/predictive_capability.tsv -o ./29_t15_17_vs_59/qc
```

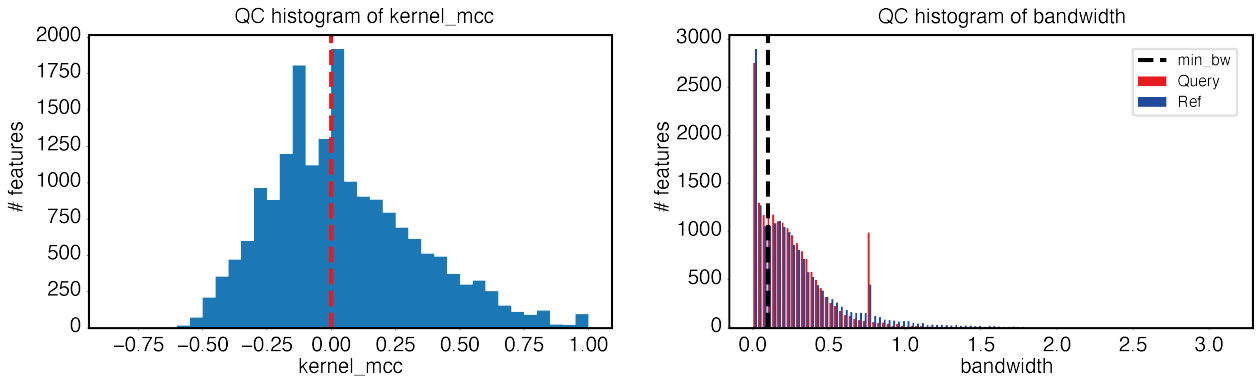


We can see in these graphs that quality is good, since:

- Most negative MCC, are close to 0.
- The minimum bandwidth (default 0.1), avoids learning from variations represented by the first mode of the distribution.

An example of bad quality control results can be made by simulating a dataset that is too small, as follows:

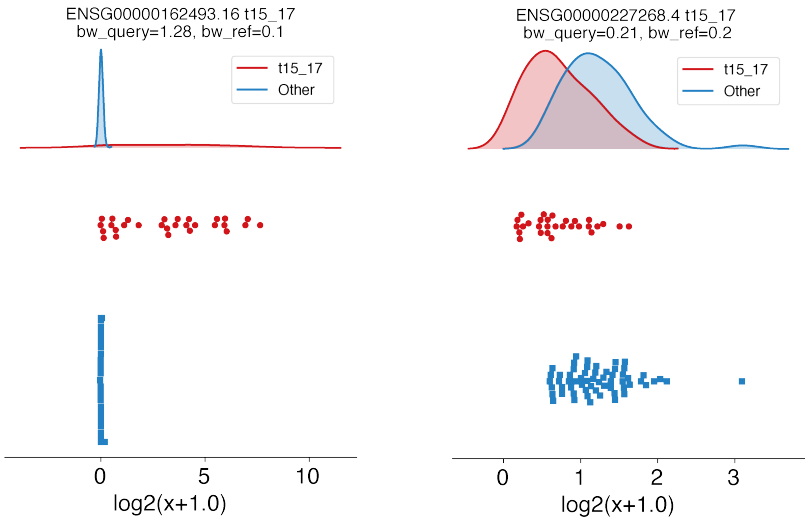
```
epcy pred --log -t 4 -m cpm.tsv  -d design_10_samples.txt --condition AML --query t15_17␣
↪-o ./5_t15_17_vs_5/
epcy qc -p ./5_t15_17_vs_5/predictive_capability.tsv -o ./5_t15_17_vs_5/qc
```



1.6 Plot a KDE trained on gene expression

EPCY also provides some visual tools, which can help with the exploration of your dataset. Using *epcy profile*, we can plot the gene expression distribution, along with the trained KDE classifier that represents each condition.

```
# ENSG00000162493.16 (PDPN, MCC=0.87), ENSG00000227268.4 (KLLN, MCC=0.33)
epcy profile --log -m cpm.tsv -d design.txt --condition AML --query t15_17 -o ./29_t15_
↪17_vs_59/figures/ --ids ENSG00000162493.16 ENSG00000227268.4
```



1.7 Reproducibility

EPCY draws a random value to assign a class according to probabilities learned by the KDE classifier, to fill a contingency table (see algorithm section). This means that different runs of EPCY can produce different results.

However, the output of EPCY is relatively stable, since each predictive score returned is already a mean of several predictive score calculations (by default 100), which are performed to minimize variance between runs. Nevertheless, different runs might show small variations. To ensure reproducibility, we add a parameter to specify the seed of the random number generator, using **--randomseed**.

Here is an example on the dataset used for the tutorial (see, How to use EPCY).

```
epcy pred --randomseed 42 --log -t 4 -m cpm.tsv  -d design.txt --condition AML --query␣
↪inv16 -o ./27_inv16_vs_61/
```

1.8 Some details on the design table

As mentioned before, the *design.txt* file classifies samples in 3 different subtypes (*t15_17*, *inv16* and *other*). Similarly as we did for *t15_17*, we can analyse *inv16* samples vs all others samples (*t15_17* and *other*), using the command below:

```
epcy pred --log -t 4 -m cpm.tsv  -d design.txt --condition AML --query inv16 -o ./27_
↪inv16_vs_61/
```

Moreover, it is possible to add more columns in **design.txt**, each one representing conditions you want to compare. Indeed, with the design table given as example (in introduction), we could perform an analysis on **Gender**, using *--condition Gender --query M -o ./gender*.

Also, if some annotations are unknown for some samples, we can remove these samples from the analysis by using **None** in the corresponding cell.

Table 5: Example where the AML subtype of sampleX is unknown and needs to be removed from the analysis.

Sample	AML	Gender
Sample1	t15_17	M
…	…	…
SampleX	None	F

With all these variations, you should be able to perform any number of comparisons using a unique design file, or by creating a different design file for each comparison.

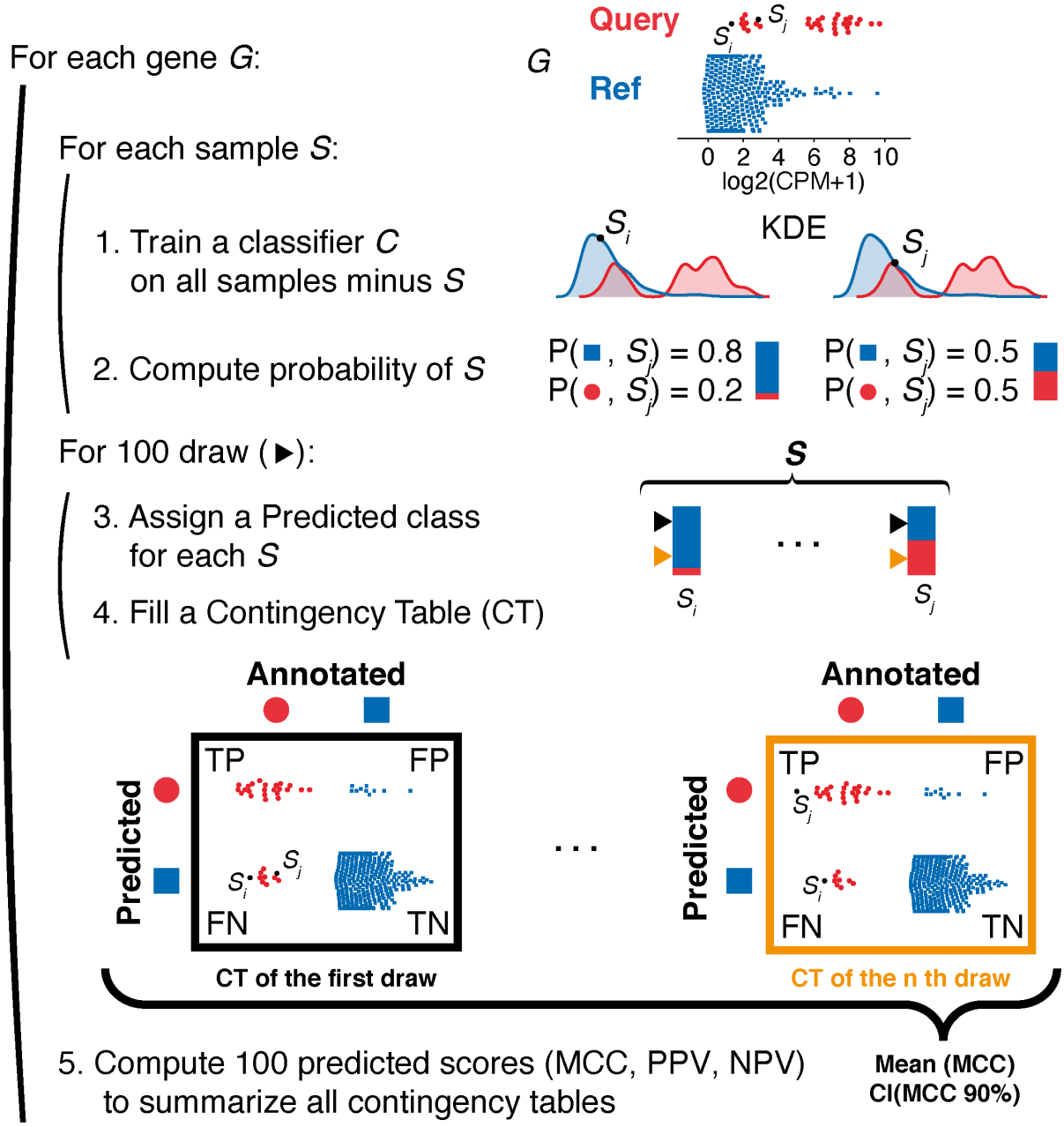
CHAPTER

TWO

HOW EPCY WORKS

EPCY is a general method to rank genes (or features) according to their potential as predictive (bio)markers, using quantitative data (like gene expression).

2.1 Visual description of the method implemented in EPCY



EPCY evaluates each gene's predictive capacity through a leave-one-out cross-validation protocol. Subgroup densities are first modeled by a Kernel Density Estimation (KDE) and used as part of a classifier Ckde on all samples minus one. Then Ckde is used to compute the probability of each class of the removed sample. This procedure is repeated for each sample. Next, we fill a contingency table (CT) by randomly drawing, for each sample, a predicted class according to the classifier's probability. This procedure is repeated m times to create m CTs (m = 100 by default). Finally, a Matthew's correlation coefficient (MCC) is computed for each CT and these values are summarized as a mean MCC with a confidence interval (CI).

EPCY can also reports other predictive scores.

CHAPTER

THREE

DETAILS OF PREDICTIVE CAPABILITY COLUMNS

By default, *epcy pred* and *epcy pred_rna* will return an output file *predictive_capability.tsv* of 9 columns:

- Default columns:
  - **id**: the id of each feature.
  - **l2fc**: log2 fold change.
  - **kernel_mcc**: Matthews Correlation Coefficient (MCC) compute by a predictor using KDE.
  - **kernel_mcc_low**, **kernel_mcc_high**: lower and upper bounds of confidence interval (90%).
  - **mean_query**: average values of this feature for samples in the subgroup of interest defined using the –query parameter.
  - **mean_ref**: average values of this feature for samples in the reference group (not in the query subset).
  - **bw_query**: estimated bandwidth used by KDE, to calculate the density of query samples.
  - **bw_ref**: estimated bandwidth used by KDE, to calculate the density of ref samples.

However, epcy can expand or modify this default output using several options, see:

```
epcy pred -h
```

3.1 Predictive scores

By default we decide to return MCC scores. However, it's possible to compute other predictive scores, in case they are more suitable for your needs. Using the following parameters will add new columns to the default output, as:

- --ppv:
  - **kernel_ppv**: Positive Predictive value (PPV, precision) compute by a predictor using KDE.
  - **kernel_ppv_low**, **kernel_ppv_high**: boundaries of confidence interval (90%).
- --npv:
  - **kernel_npv**: Negative Predictive value (NPV) compute by a predictor using KDE.
  - **kernel_npv_low**, **kernel_ppv_high**: boundaries of confidence interval (90%).
- --tpr:
  - **kernel_tpr**: True Positive Rate value (TPR, sensitivity) compute by a predictor using KDE.
  - **kernel_tpr_low**, **kernel_tpr_high**: boundaries of confidence interval (90%).
- --tnr:

  - **kernel_tnr**: True Negative Rate value (TNR, specificity) compute by a predictor using KDE.
  - **kernel_tnr_low**, **kernel_tnr_high**: boundaries of confidence interval (90%).
- --fnr:
  - **kernel_fnr**: False Negative Rate value (FNR, miss rate) compute by a predictor using KDE.
  - **kernel_fnr_low**, **kernel_fnr_high**: boundaries of confidence interval (90%).
- --fpr:
  - **kernel_fpr**: False Positive Rate Rate value (FPR, fall-out) compute by a predictor using KDE.
  - **kernel_fpr_low**, **kernel_fpr_high**: boundaries of confidence interval (90%).
- --fdr:
  - **kernel_fdr**: False Discovery Rate value (FDR) compute by a predictor using KDE.
  - **kernel_fdr_low**, **kernel_fdr_high**: boundaries of confidence interval (90%).
- --for:
  - **kernel_for**: False Omission Rate value (FOR) compute by a predictor using KDE.
  - **kernel_for_low**, **kernel_for_high**: boundaries of confidence interval (90%).
- --ts:
  - **kernel_ts**: Threat Score value (TS, critical sucess index) compute by a predictor using KDE.
  - **kernel_ts_low**, **kernel_ts_high**: boundaries of confidence interval (90%).
- --acc:
  - **kernel_acc**: Accuracy value (ACC) compute by a predictor using KDE.
  - **kernel_acc_low**, **kernel_acc_high**: boundaries of confidence interval (90%).
- --f1:
  - **kernel_f1**: F1 score value (F1) compute by a predictor using KDE.
  - **kernel_f1_low**, **kernel_f1_high**: boundaries of confidence interval (90%).
- --auc:
  - **auc**: Area Under the Curve

3.2 Statistical test

EPCY is able to perform statistical tests, using:

- --auc --utest:
  - **u_pv**: pvalue compute by a MannWhitney rank test
- --ttest:
  - **t_pv**: pvalue compute by ttest_ind

3.3 Normal distribution

The type of classifier used to evaluate the predictive score of each gene (feature), is a parameter of EPCY. By default, EPCY will use a KDE classifier. However, it is possible to replace the KDE classifier by a normal classifier, using --normal.

Using the normal classifier, all predictive scores (listed above) remain available. However, the column name of each predictive score, will be changed to start with *normal* instead of *kernel* (**normal_mcc** vs **kernel_mcc**), to be consistant.

3.4 Missing values

On some dataset (as in proteomics or single-cell), quantitative matrix can have some missing values (*nan*). In that case there are different alternatives to manage these missing values within EPCY: * Impute missing values before running EPCY. * Replace missing values by a constant, using --replacena. * For each gene (or feature), remove samples with missing values.

If you choose to remove samples with missing values, EPCY will return a *predictive_capability.tsv* with two new columns, **sample_query** and **sample_ref**, to report for each gene (feature), the number of query and reference samples used (without missing values).

If you have downloaded the source code or data on git, you can test these procedures using:

```
epcy pred --norm --log -d ./data/small_for_test/design.tsv -m ./data/small_for_
↪test/matrix.tsv -o ./data/small_for_test/using_na
epcy pred --replacena 0 --norm --log -d ./data/small_for_test/design.tsv -m ./
↪data/small_for_test/matrix.tsv -o ./data/small_for_test/replace_na
```

CHAPTER

FOUR

WORKING ON RNA QUANTIFICATION

In contrast to most Differential Expression methods specifically designed to analyse RNA quantification data, EPCY is a generic method that can be used to analyse several types of quantification data, similarly to statistical tests. However, EPCY provides specific tools, *pred_rna* and *profile_rna*, to streamline the analysis of RNA data.

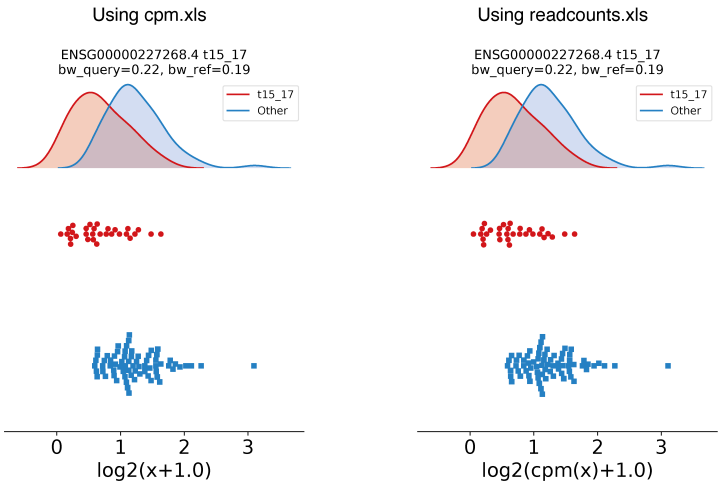
4.1 bulk RNA

In the First steps with EPCY, we saw how to run EPCY on normalized quantitative data. However, EPCY can work directly on read counts, using *pred_rna* tool:

```
# The commande seen in first steps sections
epcy pred --log -t 4 -m cpm.tsv  -d design.txt --subgroup AML --query t15_17 -o ./30_t15_
↪17_vs_70/ --randomseed 42
# Equivalent analysis using read counts quantification and pred_rna
epcy pred_rna --cpm --log -t 4 -m readcounts.tsv  -d design.txt --subgroup AML --query␣
↪t15_17 -o ./30_t15_17_vs_70_readcounts/ --randomseed 42
```

Similarly, you can use *profile_rna* to plot a trained KDE with its gene expression distribution, directly from read counts:

```
# The commande seen in first steps sections
epcy profile --log -m cpm.tsv -d design.txt --subgroup AML --query t15_17 -o ./30_t15_17_
↪vs_70/figures/ --ids ENSG00000162493.16 ENSG00000227268.4

# Equivalent commande using read counts and profile_rna
epcy profile_rna --cpm --log -m readcounts.tsv -d design.txt --subgroup AML --query t15_
↪17 -o ./30_t15_17_vs_70_readcounts/figures/ --ids ENSG00000162493.16 ENSG00000227268.4
```



More help can be found using:

```
epcy pred_rna -h
epcy profile_rna -h
```

4.2 Kallisto quantification

EPCY allows to work directly on kallisto[1] transcript quantifications using the HDF5 files to take into consideration the expression values of bootsrapped samples computed by this software. To do so, a *kallisto* column needs to be added to the design file (which specifies the directory path of the *abundant.h5* file for each sample).

Using data available on git and **epcy pred_rna**, you can run EPCY as follow:

```
# To run on kallisto quantification, add --kall --cpm --log
epcy pred_rna --kal --cpm --log -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./
↪data/small_leucegene/5_inv16_vs_5/trans/
# Note that:
# - kallisto quantification is on transcript, not on gene
# - On real (complet) dataset, it is recommended to add some threads (-t)
```

To analyse gene level expression, a gff3 file of the genome annotation needs to be specified, to provide the correspondence between transcript and gene ids. For the Leucegene toy dataset, which was quantified using Ensembl annotatiosn, this file can be downloaded from ensembl and added in the command, as follow:

```
# To run on kallisto quantification and gene level, add --gene --anno [file.gff]
epcy pred_rna --kal --cpm --log --gene --anno ./data/small_genome/Homo_sapiens.GRCh38.84.
↪reduce.gff3 -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/small_
↪leucegene/5_inv16_vs_5/gene/ --randomseed 42
epcy profile_rna --kal --cpm --log --gene --anno ./data/small_genome/Homo_sapiens.GRCh38.
↪84.reduce.gff3 -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/small_
↪leucegene/5_inv16_vs_5/gene/figures --ids ENSG00000100345
# If you prefer analyse your data on tpm, replace --cpm by --tpm
```

[1] Nicolas L Bray, Harold Pimentel, Páll Melsted and Lior Pachter, Near-optimal probabilistic RNA-seq quantification, Nature Biotechnology **34**, 525–527 (2016), doi:10.1038/nbt.3519

To take account the inferential variance (introduced by sleuth[2]), EPCY can use bootstrapped samples, using --bs:

```
epcy pred_rna --kal --cpm --log --gene --bs 10 --anno ./data/small_genome/Homo_sapiens.
↪GRCh38.84.reduce.gff3 -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/
↪small_leucegene/5_inv16_vs_5_bs/gene/ --randomseed 42
epcy profile_rna --kal --cpm --log --gene --bs 10 --anno ./data/small_genome/Homo_
↪sapiens.GRCh38.84.reduce.gff3 -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./
↪data/small_leucegene/5_inv16_vs_5_bs/gene/figures --ids ENSG00000100345
```

When reading all kallisto files is time consuming, you can use *epcy kal2mat* tool, to create a quantification matrix file and use EPCY, as before:

```
# Without bootstrapped samples
epcy kal2mat --gene --anno ./data/small_genome/Homo_sapiens.GRCh38.84.reduce.gff3 -d ./
↪data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/small_leucegene/5_inv16_vs_5_
↪mat/gene/
epcy pred_rna --cpm --log -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -m ./data/
↪small_leucegene/5_inv16_vs_5_mat/gene/readcounts.tsv -o ./data/small_leucegene/5_inv16_
↪vs_5_mat/gene/ --randomseed 42
epcy profile_rna --cpm --log -m ./data/small_leucegene/5_inv16_vs_5_mat/gene/readcounts.
↪tsv -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/small_leucegene/5_
↪inv16_vs_5_mat/gene/figures --ids ENSG00000100345

# With bootstrapped samples
epcy kal2mat --gene --bs 10 --anno ./data/small_genome/Homo_sapiens.GRCh38.84.reduce.
↪gff3 -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/small_leucegene/5_
↪inv16_vs_5_mat_bs/gene/
epcy pred_rna --bs 10 --cpm --log -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -m ./
↪data/small_leucegene/5_inv16_vs_5_mat_bs/gene/readcounts.tsv -o ./data/small_leucegene/
↪5_inv16_vs_5_mat_bs/gene/ --randomseed 42
epcy profile_rna --bs 10 --cpm --log -m ./data/small_leucegene/5_inv16_vs_5_mat_bs/gene/
↪readcounts.tsv -d ./data/small_leucegene/5_inv16_vs_5/design.tsv -o ./data/small_
↪leucegene/5_inv16_vs_5_mat_bs/gene/figures --ids ENSG00000100345
```

4.3 Single-cell

Several developments are planned in order to facilitate the use of EPCY for single-cell data (to manage sparse matrix and run on GPU for instance). In the meantime, you can analyse your single-cell data with *epcy pred* and *epcy profile* using the RNA-seq pipeline described in First steps with EPCY on normalized expression data.

On read counts (not normalized), you can use *epcy pred_rna* and *epcy profile_rna* with --cpmed (in place of --cpm) to normalized read counts according to median depth of the dataset.

```
epcy pred_rna --cpmed --log ...
```

[2] Harold J. Pimentel, Nicolas Bray, Suzette Puente, Páll Melsted and Lior Pachter, Differential analysis of RNA-Seq incorporating quantification uncertainty, Nature Methods (2017), advanced access http://dx.doi.org/10.1038/nmeth.4324

CHAPTER

FIVE

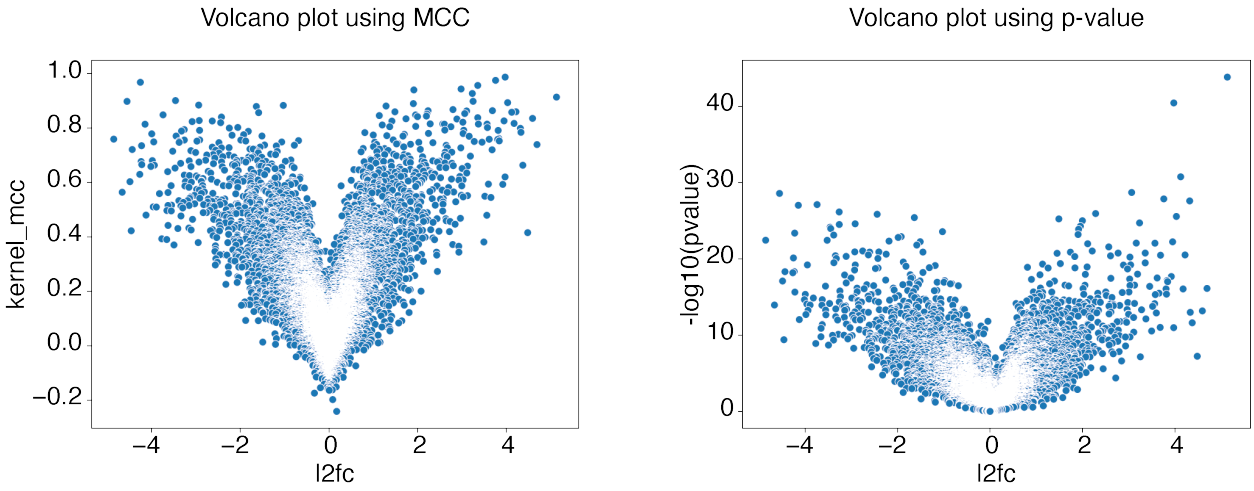
HOW TO EXPLORE EPCY OUTPUT TO SELECT BEST CANDIDATES

EPCY output is comparable to statistical (or differential) analysis, except that p-values are replaced by a predictive score (MCC by default, see predictive scores). Consequently most tools already developed to explore statistical output can be transpose to explore EPCY output, starting by the volcano plot.

5.1 Volcano plot

Using same data and analysis made in First steps with EPCY, we can create a volcano plot like this:

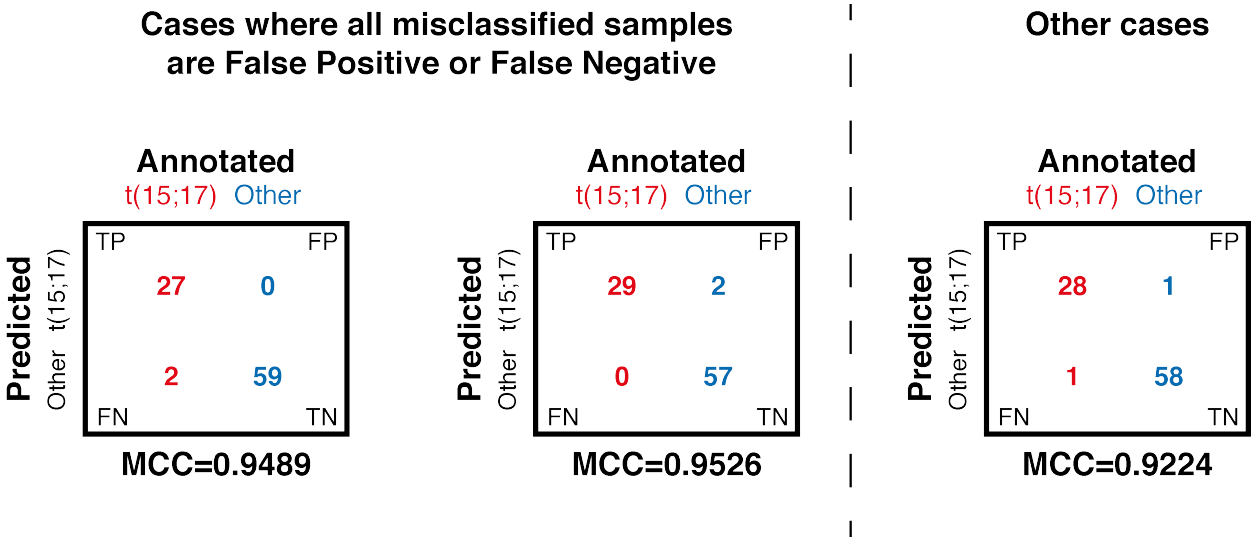
```
# If not done, start by downloading data and scripts from epcy_tuto
git clone git@github.com:iric-soft/epcy_tuto.git
cd epcy_tuto/data/leucegene

# Run EPCY analysis with --ttest to add a pvalue column
# (see Details in predictive capability columns section)
epcy pred_rna --log --cpm -t 4 -m readcounts.tsv --ttest -d design.txt --condition AML --
↪query t15_17 -o ./29_t15_17_vs_59/ --randomseed 42

# Display volcano plot using MCC
python ../../script/volcano.py -i ./29_t15_17_vs_59/predictive_capability.tsv -o ./29_
↪t15_17_vs_59/

# Display volcano plot using pvalue
python ../../script/volcano.py -i ./29_t15_17_vs_59/predictive_capability.tsv -o ./29_
↪t15_17_vs_59/ --pvalue
```



5.2 Identify a threshold

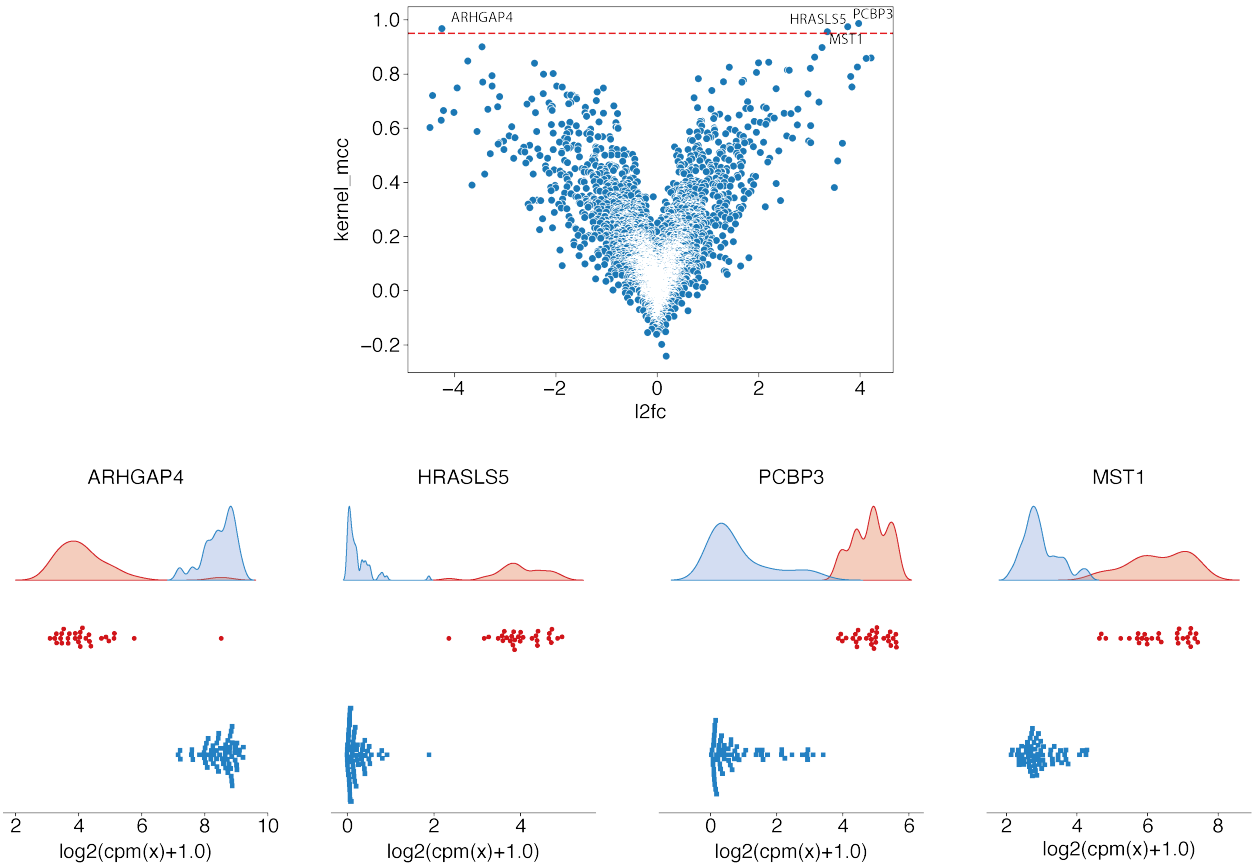
Generally, the next step is to identify a MCC threshold to select best candidates.

In case the expected predicted performance to reach is known, we can use it directly as a threshold. For example, if we can accept a maximum of 3% of misclassified samples (or 2 samples in this case), summarized by these three contingency tables (CT):



Using these three theoretical cases, we can identify that a MCC threshold of more than 0.95 is needed and identify the 4 genes which satisfy the objective previously defined:

```
# Display volcano plot using MCC
python ../../script/volcano.py -t 0.95 -i ./29_t15_17_vs_59/predictive_capability.tsv -o␣
↪./29_t15_17_vs_59/ --anno ./ensembl_anno_GRCh38_94.tsv
epcy profile_rna --log --cpm -m readcounts.tsv -d design.txt --condition AML --query t15_
↪17 -o ./29_t15_17_vs_59/profile_cutoff/ --ids ENSG00000173531.15 ENSG00000168004.9␣
↪ENSG00000089820.15 ENSG00000183570.16
```

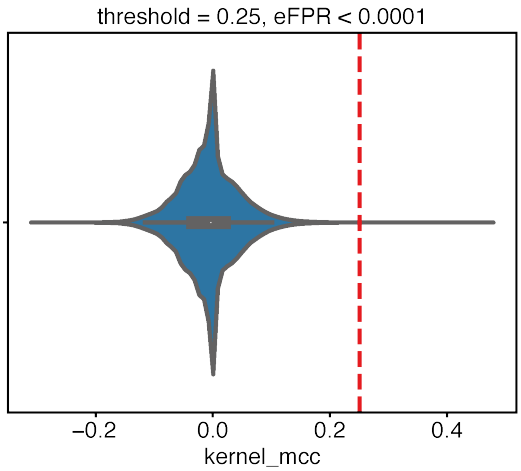


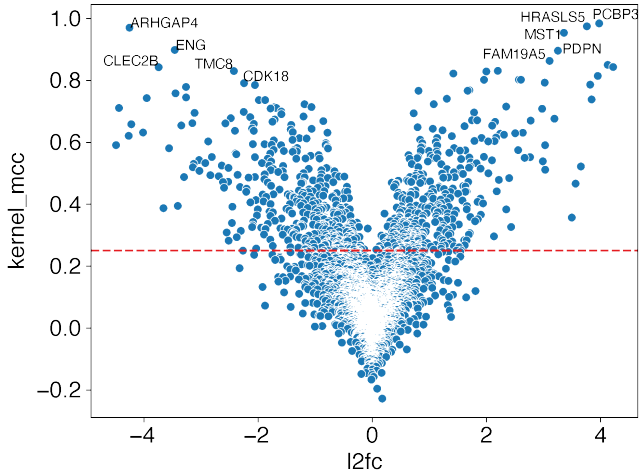
In case the expected performance is directly formulate using predictive scores (as accuracy, sensibility, specificity or other), this is even simpler. We can add these scores to the *epcy pred* command line (see predictive scores) to be able to filter EPCY's output on each of them.

5.3 Using empirical False Positive Rate

Now, when we have no expectation and want select all genes (features) with a "significant" predictive score, we can use the **--shuffle** option of *epcy pred* to compute predictive scores on random designs similar to our initial experiment. Using several **shuffled analyses**, we can estimate a null distribution and use it to identify a threshold, according to a percentage of accepted False Positive Rate (FPR):

```
# Take around 80 min using a macbook pro 2 GHz Dual-Core Intel Core i5.
for n in `seq 1 10`; do epcy pred_rna --log --cpm -t 4 -m readcounts.tsv  -d design.txt -
↪-condition AML --query t15_17 --shuffle -o ./29_t15_17_vs_59/shuffled/$n; done

# Display:
#  - the MCC distribution computed on shuffled analyses
#  - the cutoff for eFPR < 0.0001
python ../../script/eFPR.py -d ./29_t15_17_vs_59/shuffled/ -o ./29_t15_17_vs_59/ -p 0.
↪0001

# Display volcano plot with a threshold = 0.25
python ../../script/volcano.py -t 0.25 -i ./29_t15_17_vs_59/predictive_capability.tsv -o␣
↪./29_t15_17_vs_59/ --anno ./ensembl_anno_GRCh38_94.tsv
```


threshold = 0.25, eFPR < 0.0001
−0.2
0.0
0.2
0.4
kernel_mcc


ARHGAP4
ENG
CLEC2B
TMC8
CDK18
HRASLS5
PCBP3
MST1
PDPN
FAM19A5
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
kernel_mcc
−4
−2
0
2
4
l2fc

CHAPTER

SIX

WORKING ON SMALL DATASET

Coming soon.