ReFACTor v1.0

Reference-Free Adjustment for Cell-Type composition (ReFACTor) is an unsupervised method for the correction of cell-type heterogeneity in epigenome-wide association studies (EWAS), which is based on a variant of principal component analysis (PCA). ReFACTor is described in the following paper (upcoming).

As decribed bellow, ReFACTor is available in R and Python. We recommend using the much faster python version.

Download

1. Press the 'Download ZIP' button on the right side of this page
2. Extract the ZIP file to a folder
3. Make sure the demo provided works (see below)

Dependencies for the Python version are described at the end of this README file.

Input

ReFACTor gets the following arguments as an input:

Required:

- datafile: path to a sites by samples matrix file of tab-delimited beta-normalized methylation levels; the first row should contain the sample IDs and the first column should contain the CpG IDs (see 'demo_files/demo_datafile.txt' for example). Important data preparation instructions are described below under 'Data preparation'.
- k: the number of assumed cell types; guidlines for selecting k are desribed below under 'Parameters selection'.

Optional:

- covarfile: path to a samples by covariates matrix file of tab-delimited covariates; the first column should contain the sample IDs ordered as in the first row of the data file (see 'demo_files/demo_covariates.txt' for example). If provided, the data are adjusted for the covariates before running ReFACTor. For more details see 'Data preparation' below.
- t: the number of sites to use for computing the ReFACTor components (default is 500); guidlines for selecting t are desribed below under 'Parameters selection'.
- numcomp: the number of ReFACTor components to output (default is the same as k)
- stdth: standard deviation (std) threshold for excluding low variance sites; all sites with std < stdth will be excluded before running ReFACTor (default is 0.02). For more details see 'Data preparation' below.
- out: prefix of the output files (default is 'refactor')

Output

The software outputs two files:

1. refactor.out.components.txt - a matrix with the first numcomp ReFACTor components for each individual
2. refactor.out.rankedlist.txt - a ranked list of the methylation sites; from the most informative to the least informative

Note that the default prefix of these files ('refactor') can be changed using the 'out' argument.

R

The refactor.R function in the 'R' folder implements ReFACTor and can be executed directly from R. For example:

```
# <R code>
```

```
source("refactor.R")
k = 5
datafile = "../demo_files/demo_datafile.txt"
results <- refactor(datafile,k)
RC <- results$refactor_components # Extract the ReFACTor components
ranked_list <- results$ranked_list # Extract the list of sites ranked by ReFACTor

# Can also provide one or more of the optional arguments
results <- refactor(datafile,k,covarfile="../demo_files/demo_covariates.txt",t=500,numcomp=10,stdth=0.01,out="demo_results")
```

Demo

The following demo computes the ReFACTor components of a simulated example dataset and performs an EWAS. The demo shows that while a standard PCA cannot adjust the data well, using ReFACTor can adjust the data similarly to using the true cell proportions. From the command line run:

```
Rscript demo.R
```

Python

The refactor.py script in the 'python' folder implements ReFACTor and can be executed from the command line as follows:

```
python refactor.py --datafile <datafile> --k <k>
```

or, if including one or more of the optional arguments:

```
python refactor.py --datafile <datafile> --k <k> --covarfile <covarfile> --t <t> --numcomp <numcomp> --stdth <stdth> --out <out>
```

Demo

The following demo computes the ReFACTor components of a simulated example dataset and performs an EWAS. The demo shows that while a standard PCA cannot adjust the data well, using ReFACTor can adjust the data similarly to using the true cell proportions. From the command line run:

```
python demo.py
```

Data preparation

Preprocessing raw data

ReFACTor is designed to handle Beta normalized methylation levels (although it may perform well on M-value normalized data as well). Prior to running ReFACTor, the data should be adjusted for known technical atrifacts of the technology used for probing the methylation levels as well as adjusted for known technical covariates (such as batches). For a comprehensive comparison between methods for preprocessing raw data collected by the Illumina 27K/450K platforms see Lenhe et al. 2015. In order to best fit to the assumptions of ReFACTor, any normalization applied should keep the data approximately normal.

Preparing data for ReFACTor

For best performance, we suggest to take the following steps when preparing the data for ReFACTor:

- Exclude problematic probes - remove non-autosomal probes, cross-hybridized probes and probes with SNPs. Note that once the ReFACTor components are computed, any of the exluded probes can be rejoined to the data for the rest of the analysis.
- Adjust the data for covariates - adjusting the methylation levels, before running ReFACTor, for known technical covariates such as batchs can be crutial in some cases. In addition, we observe that adjusting the methylation levels for genome-wide affecting factors, such as gender, smoking and global ancestry, improves the performance of ReFACTor. We do not suggest to adjust the data for covariates that are correlated with the cell type composition, such as age, before running ReFACTor (these covariates should be

accounted for after running ReFACTor). The 'covarfile' optional argument allows to adjust the data for covariates before running ReFACTor.

Additional remarks:

- A large number of sites in the Illumina 27K/450K platforms are constant or nearly-constant. We observe that removing sites of very low variance improves the performance of ReFACTor (defined by the 'stdth' argument; the default value should be sufficient in most cases).
- The current version of ReFACTor does not handle missing values. If missing values exist in the data they should be assigned with values before running ReFACTor (e.g. for each site its missings values can be assigned with the mean value of the site - across all smaples with no missing values).

Parameters selection

The manuscript describing ReFACTor demonstrates that the algorithm is robust to the selection of the parameters k and t in simulated and real data. However, sometimes even an approximation of k is not available, and the default value of t (t=500) may not be adequate in some cases. Therefore, we provide the following tools for guiding the parameters selection. These tools are available in the Python version only.

Selecting k (the number of assumed cell types)

The estimate_k.py script (under the 'python' folder) computes a score for each of the first several eigenvalues of the covariance matrix of the input data. The score of the i-th eigenvalue is defined to be -log of the ratio between the i-th eigenvalue to the (i-1)-th eigenvalue, thus a high score of a specific eigenvalue suggests its eigenvector as a substantial variance component in the data (compared with the previous one). The ratio between adjacent eigenvalues, as well as other test statistics of the eigenvalues, is desribed by Peres-Neto et al. (2005) as a method for determining the number of non-trivial axes of variance in data.

For plotting the scores of the first several eigenvalues (starting from the second eigenvalue), run:

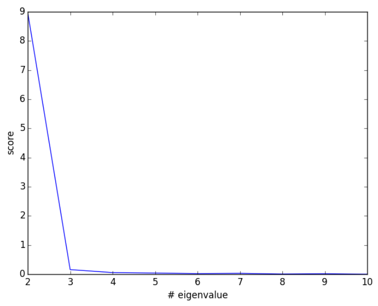
```
python estimate_k.py --datafile <datafile>
```

The maximal number of eigenvalues to plot can be changed:

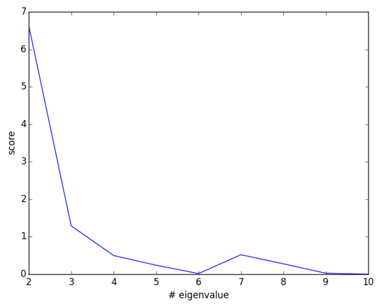
```
python estimate_k.py --datafile <datafile> --max_k <max_k>
```

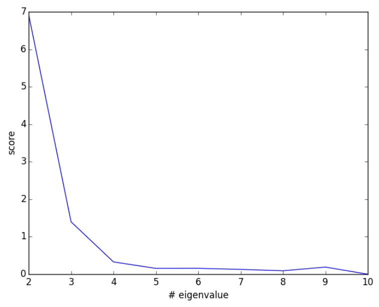
k should be selected to be the number of high score eigenvalues, before reaching a right tail of flat scores (the scores of the last several eigenvalues). Below are examples of plots generated by estimate_k.py:

- estimate_k_results_simulated.png - the result of applying the script on the example dataset provided here under the 'demo_files' folder. The plot suggests 4 or 5 to be a reasonable choice of k. These simulated data were in fact generated using k=5.



- estimate_k_results_ra.png, estimate_k_results_gala.png - the result of applying the script on the Rheumatoid arthritis and GALA II datasets presented in the manuscript describing ReFACTor. Both plots suggest 5 or 6 to be a reasonable choice of k.





Selecting t (the number of sites to use for computing ReFACTor's components)

The estimate_t.py script (under the 'python' folder) provides a qualitative tool for assesing the number of features in the data that are highly informative in terms of the main structure of the data. The script first follows the ReFACTor algorithm in order to find the distance of each site from its low-rank approximation. Then, the sites are sorted by their distance, and a score for site i in the sorted list is defined to be the difference between the distance of the i-th site and the distance of the (i-1)-th site. Finally, the scores of the first several thousands of sites are plotted (using moving average for smoothing the signal), thus providing a qualitative way to get the number of highly informative sites.

Execution:

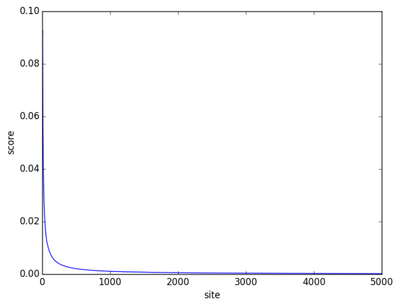
```
python estimate_t.py --datafile <datafile> --k <k>
```

The number of sites to plot can be changed:

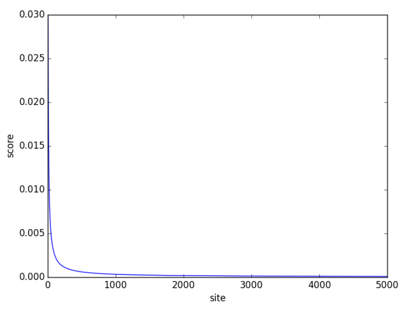
```
python estimate_t.py --datafile <datafile> --k <k> --numsites <num_sites>
```

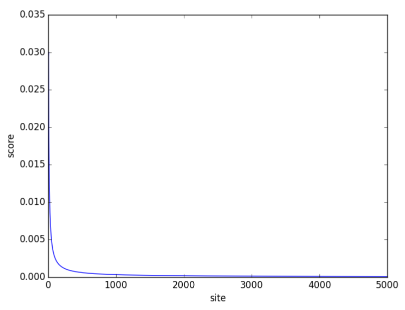
t should be selected to be the number of sites after which the signal dramatically decays. Below are examples of plots generated by estimate_t.py:

- estimate_t_results_simulated.png - the result of applying the script on the example dataset provided here under the 'demo_files' folder. The plot suggests the range between 500 and 1000 to be a reasonable choice of t.



- estimate_t_results_ra.png, estimate_t_results_gala.png - the result of applying the script on the Rheumatoid arthritis and GALA II datasets presented in the manuscript describing ReFACTor. Both plots suggest the range between 500 and 1000 to be a reasonable choice of t.





Dependencies

For the Python version we recommend to use a standard Python distribution such as Anaconda (https://www.continuum.io/downloads). This release of ReFACTor was implemented for Python v2.7 and has the following dependencies:

```
numpy
scipy
sklearn
matplotlib (required only for demo.py)
statsmodels (required only for demo.py)
```

Citing ReFACTor

If you use ReFACTor in any published work, please cite the manuscript describing the method (upcoming).

Authors

This software was developed by Reut Yedidim, Noah Zaitlen and Elior Rahmani.

For any question and for reporting bugs please send an email to Elior Rahmani at: elior.rahmani@gmail.com