BioFeatureFinder

V. 1.1.4 MANUAL

Felipe E. Ciamponi
felipe.ciamponi@gmail.com

January 21, 2019

Index

Installation

Pre-requisites and external dependencies

BioFeatureFinder runs on a python framework (supports either Python 2 or 3) and requires several non-native libraries/packages installed:

- Glob2 (v0.6)
- Matplotlib (v2.2.3)
- Numpy (v1.15.4)
- Pandas (v0.23.4)
- Pybedtools (v0.8.0)
- Pysam (v0.15.0)
- Rpy2 (v2.9.4)
- Scikit-learn (v0.20.0)
- Scipy (v1.1.0)
- Seaborn (v0.9.0)
- Pdfrw (v0.4)

It is highly recommended to use Anaconda python distribution, which already contains most of the necessary packages installed (https://www.anaconda.com/download/). However, you can install any missing packages via the following command:

```
conda install glob2 matplotlib numpy pandas pybedtools pysam rpy2 scikit-learn scipy seaborn pdfrw
```

Additionally, BioFeatureFinder uses other tools to gather information that must be installed and available as executables on PATH. These external dependencies are:

- EMBOSS (v6.6.0.0) - http://emboss.sourceforge.net/download/
- ViennaRNA (v2.4.9) - https://www.tbi.univie.ac.at/RNA/
- QGRS Mapper - https://github.com/freezer333/qgrs-cpp
- BEDTools (v2.27.1) - http://bedtools.readthedocs.io/en/latest/
- SAMTools (v1.9) - http://www.htslib.org/
- HTSlib (v1.9) - http://www.htslib.org/

Alternatively, you can also download the tools above in our project SourceForge directory (https://sourceforge.net/projects/biofeaturefinder/files/external_dependencies/).

Installing the software

BioFeatureFinder latest development release can be obtained from the official GitHub repository (https://github.com/kbmlab/BioFeatureFinder.git) and installed either via “pip” or the “setup.py” script included. In order to install the software, follow the steps below:

- Clone the git repository to a local folder
  - `git clone https://github.com/kbmlab/BioFeatureFinder.git`
- Enter BioFeatureFinder directory
  - `cd BioFeatureFinder`
- (recommended) Development install, local changes are reflected in command-line calls
  - `pip install -e .`
- (alternative) Install the scripts with “setup.py” file
  - `python setup.py install`

If properly installed, the scripts should be called from command line as executables without issues. In order to check, type the commands below to see the help for each of the scripts.

- `build_datamatrix.py --help`
- `analyze_features.py --help`
- `extract_gtf_regions.py --help`
- `analyze_gtf_regions.py --help`

Alternatively, if you are unable to clone from the git repository, or want to download a previous version of BioFeatureFinder, you can download the tarballs from our project SourceForge page (https://sourceforge.net/projects/biofeaturefinder/files/stable_releases/). The installation for these tarballs is the same as the one describe above.

Getting started

Running a full analysis of genomic regions using BioFeatureFinder is divided in two-steps, divided in the two main scripts. The first step is converting biological information in a numeric datamatrix that will be used as a reference (`build_datamatrix.py`). The second step is processing this datamatrix with an algorithm that identifies and classifies distinguishing features that separate your regions of interest from randomized background (`analyze_features.py`). This process is supported by two secondary scripts: one is used to extract specific (or multiple) regions (`extract_gtf_regions.py`) from an annotation file, which can used as a guided background for the first main script, while the second (`analyze_gtf_regions.py`) is used to compare an input file (or a list of files) with a background region (or list of regions), in order to identify which region(s) are more/less associated with the input.

Main scripts

build_datamatrix.py

Required arguments:

> **-i** */* **--input** - *list of bed interval of interest*
> **-gen** */* **--genome** - *Genomic FASTA/multi-FASTA sequence associated with the intervals*
> **-o** */* **--outfile** - *Prefix used for file output*

Optional arguments:

> **-g** */* **--gtf** - *GTF file for guided background generation. Default: False*
>
> **-nuc** */* **--nucleotide_content** - *Defines the ammount of information included from the nucleotide sequence, 3 options available: simple [Length and %GC], intermediate [Length, %GC, %G, %C, %A, %T], full [All data from BedTools nucleotide sequence]. Default: simple.*
>
> **-bw** */* **--bigwig-scores** - *This option takes as input bigWig files with scores and calculates the mean score over each input region. Can take multiple files as input and accepts wildcard characters (*). REQUIRES bigWigAverageOverBed tool to be installed and available on PATH (can be obtained at UCSCs binaries directory). Default: False.*
>
> **-var** */* **--variation** - *This option takes as input BED files containing regions of biological features found in the genome (can be SNPs, strucutural variations, mutations or custom annotations) and counts the number of occurences in each input region. Can take*

multiple files as input and accepts wildcard characters (*). REQUIRES bedtools to be installed and available on PATH. Default: False

-f */* **--fasta** - *Use this option to create fasta files for each entry in the analysis. Required for k-mer search (-k) and structural MFE (-rf and -qg). Default: False*

-k */* **--kmer** - *List of integers (numbers) to create k-mers for counting. Default: False*

-rf */* **--rnafold** - *Run RNAFold (from Vienna RNA package) on each entry and extract MFE values. Requires Vienna RNA Package installed locally and available on PATH. Default: False*

-qg */* **--qgrs** - *Run QGRS Mapper on each entry and extract G-Quadruplex scores. Requires QGRS Mapper installed locally and available on PATH. Default: False*

-custom_seq */* **--custom-sequence** - *Text list containing custom sequences to be counted in the regions. Default: False*

-n */* **--n_rand** - *Number of times to shuffle input for background generator. Default: 3*

--keepBED - *Save the bed files generated for each class. Default: False*

--keepTEMP - *Keep the temporary files generated in each step. Default: False*

-c */* **--ncores** - *Number of CPU cores used. Default:(ALL_CORES)-1*

-d */* **--debug** - *Only searches for the first N entries. Useful for debugging and checking if the code is working properly. Default: False*

Usage example:

```
build_datamatrix.py -i input.bed -gen some_genome.fa -o outfile_prefix
```

or

```
build_datamatrix.py -i input.bed -gen some_genome.fa -o outfile_prefix -gtf ./my_regions/background.gtf -nuc intermediate -var ./var_files/*.bed -bw ./bw_files/*.bw -f -k 4 5 6 -rf -qg -custom_seq my_seqs.txt -n 3 --keepBED --keepTEMP -c 8
```

analyze_features.py

Required arguments:

-i */* **--input** - *list of bed interval of interest*
-m */* **--matrix** - *Datamatrix generated by build_datamatrix.py*
-o */* **--outfile** - *Prefix used for file output*

Optional arguments:

-f */* **--filter_columns** - *Text file containing a comma-separated list with names of the columns to be removed from the datxamatrix in the analysis. Default: False*

-p */* **--p_adjust** - *Type of p-value correction used after Kolmogorov-Smirnov test, available options are: 'holm', ''hochberg', 'hommel', 'bonferroni', 'BH', 'BY', 'fdr' and 'none'. Default:'fdr'*

-pth */* **--p_adjust_threshold** - *Threshold of adjusted p-value for significance. If using --sig-only, only significantly different features are passed down to the classifier for group separation. Default: 0.05*

-s */* **--sig-only** - *Use only the statistically significant features (found by KS test) in the plotting classification step. Useful for filtering large data matrices to reduce computational time. Can use the '-pth' option to select the threshold of significance for feature selection. Default: False*

-c */* **--correlation-filter** - *Remove features which exhibit linear correlation scores above a certain threshold (from 0 to 1). Default: False*

-mi */* **--mi-filter** - *Remove non-informative features based on mutual information (MI) scores below a certain threshold (from 0 to 1). By default, it removes all features with 0 aMI scores and keep all features with positive scores. If set to a negative number (ex. -1) it disables the mi-filtering entirely (but still calculates the results and outputs the tables). Default: False*

-r */* **--runs** - *Number of times (repetitions) to run the classification step. Default:10*

-n */* **--sample_size** - *Relative size of randomly sampled regions in comparisson to input. Default:1 (i.e. 1x the amount of input regions)*

-t */* **--train_size** - *Fraction of sample used for training the classifier model. The remaining sample pool will be used for testing the classifier (from 0 to 1). Default: 0.80*

-pr */* **--parameters** - *Type of parameter selection to be used by the classifier. Available options are: 'optimize' (runs an optimization step with GridSearchCV), 'default' (uses default parameters from GradientBoostClassifier) and 'file' (take in input txt file with a dictionary-like structure with classifier parameters, requires the use of -pf option). Options: 'default', 'optimize' and 'file'. Default:'optimize'*

-scr */* **--scoring_metric** - *Type of scoring metric used to optimize the classifier hyperparameters if the 'optmize' param option is selected. Possible options include: 'roc_auc', 'accuracy', 'balanced_accuracy', 'precision', ''average_precision' and 'recall'. Default: 'roc_auc'*

-pf */* **--param_file** - *Input text with with dictionary-like structure with parameter options for GradientBoostClassifier. Ex. {'n_estimators':300,'loss':'deviance',...}*

--no-plot - *Use this flag if you want to skip plotting graphs for each feature in the matrix. Default: False*

--no-CLF - *Use this flag if you want to skip the classifying with Stochastic GradientBoost. Default: False*

--ncores - *Number of CPU cores used to multiple jobs on the classifier. Default:(ALL_CORES)-1*

Usage example:

```
analyze_features.py -i input.bed -m my_datamatrix.tsv -o my_analysis
```

or

```
analyze_features.py -i input.bed -m my_datamatrix.tsv -o my_analysis -f cols_to_drop.txt -p BH -pth 0.05 -s -c 0.8 -mi 0.01 -r 20 -n 1 -t 0.75 -pr optimize -scr balanced_accuracy --no-plot --ncores 12
```

Secondary scripts

extract_gtf_regions.py

Required arguments:

-g */* **--gtf** - *GTF file downloaded from Ensembl database or similar.*
-o */* **--outfile** - *Prefix used for file output*

Optional arguments:

-r */* **--region** - *List of regions for extraction from GTF. Can be passed as a list (Ex. 'three_prime_utr five_prime_utr CDS'), must be an exact match of the 'feature' column in the GTF file. If -r is not used, it detects all features present in the GTF and extract separate le for each. Default: auto*

--intron - *Use this option to extract intron annotations from the GTF. Requires the presence of genes and exons positions in the annotation. Default: False*

--split-intron - *Use this option to split intron annotations from the GTF in proximal and distal regions. Requires --intron option. Default: False*

--splice-sites - *Use this option to extract 3' and 5' splice site annotations from the GTF. Requires --intron option. Default: False*

-w */* **--window** - *Window size that will span the splice site in bp. Default: 20*

-rt */* **--ratio** - *Ratio of window size that will span inside the intron. Default: 0.5*

--ncores - Number of CPU cores used. Default:(ALL_CORES)-1

Usage example:

```
extract_gtf_regions.py -g my_annotation.gtf -o outfile_prefix
```

or

```
extract_gtf_regions.py -g my_annotation.gtf -o outfile_prefix --intron --splice-sites -w 50 -r 0.75
```

analyze_gtf_regions.py

Required arguments:

> **-i** */* **--input** - *List of BED intervals to be analyzed*
> **-r** */* **--reference** - *List of GTF files for reference*
> **-o** */* **--outfile** - *Prefix used for file output*

Optional arguments:

> **-l** */* **--labels** - *Customized labels for region list. Must match the input order of -r list.*
>
> **-f** */* **--fraction** - *Fraction of the input that must overlap with reference region (must be float from 0 to 1). Default: False*

Usage example:

```
analyze_gtf_regions.py -i my_inputs/input.*.bed -r my_regions/reference.*.gtf -o outfile_prefix
```

or

```
analyze_gtf_regions.py -i my_inputs/input.*.bed -r CDS.gtf three_prime_utr.gtf introns.gtf -l CDS 3UTR INTRONS -o outfile_prefix
```

Quick start with test dataset (a.k.a. TL;DR)

For the example below, we will use the files located in the BioFeatureFinder “test_data” folder

Enter the "hg19_data" inside the "test_data" folder

```
cd /full/path/to/BioFeatureFinder/test_data/hg19_data/
```

Download and unzip the hg19 (GRCh37.p13) from GENCODE ftp website

```
wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_19/GRCh37.p13.genome.fa.gz && gunzip GRCh37.p13.genome.fa.gz
```

Download the 100way phastCons scores from UCSC goldenPath

```
wget
http://hgdownload.soe.ucsc.edu/goldenPath/hg19/phastCons100way/hg19.100way.phas
tCons.bw
```

(OPTIONAL) Enter the "hg19_annotations" folder and extract the GTF regions from the Homo sapiens GRCh37.p13 annotation from Ensembl (with chameleon for UCSC-style)

```
cd ./hg19_annotations
extract_gtf_regions.py -g ./Homo_sapiens.GRCh37.p13.chameleonUCSC.gtf.gz -o
grch37 --intron
```

Go back to "test_data" folder

```
cd /full/path/to/BioFeatureFinder/test_data/
```

Run region analysis to identify preferential occurrence region for input dataset

```
analyze_gtf_regions.py \
-i ./rbfox2_sample.bed \
-r ./hg19_data/hg19_annotations/grch37.* \
-l 3pss 3utr 5pss 5utr cds intron \
-o rbfox2_test_regions
```

Run the "build_datamatrix" script to create extract biological features from the files

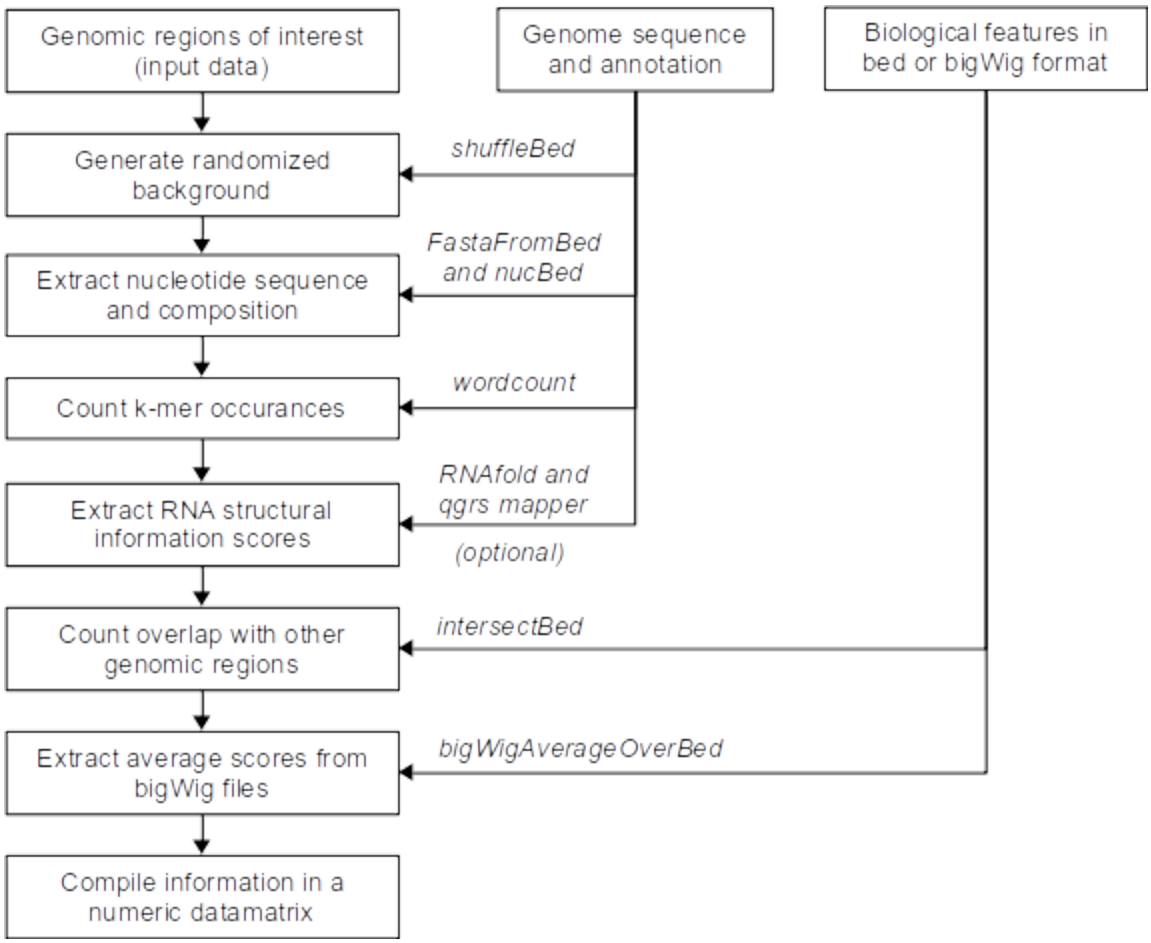
```
build_datamatrix.py \
-i ./rbfox2_sample.bed \
-gen ./hg19_data/GRCh37.p13.genome.fa \
-g ./hg19_data/hg19_annotations/grch37.intron.gtf.gz \
-bw ./hg19_data/hg19.100way.phastCons.bw \
-var ./hg19_data/hg19_var/* \
--fasta \
-k 4 5 6 \
--rnafold \
--qgrs \
--keepBED \
--keepTEMP \
-o rbfox2.test
```

Run the "analyze_features.py" script to identify significant features

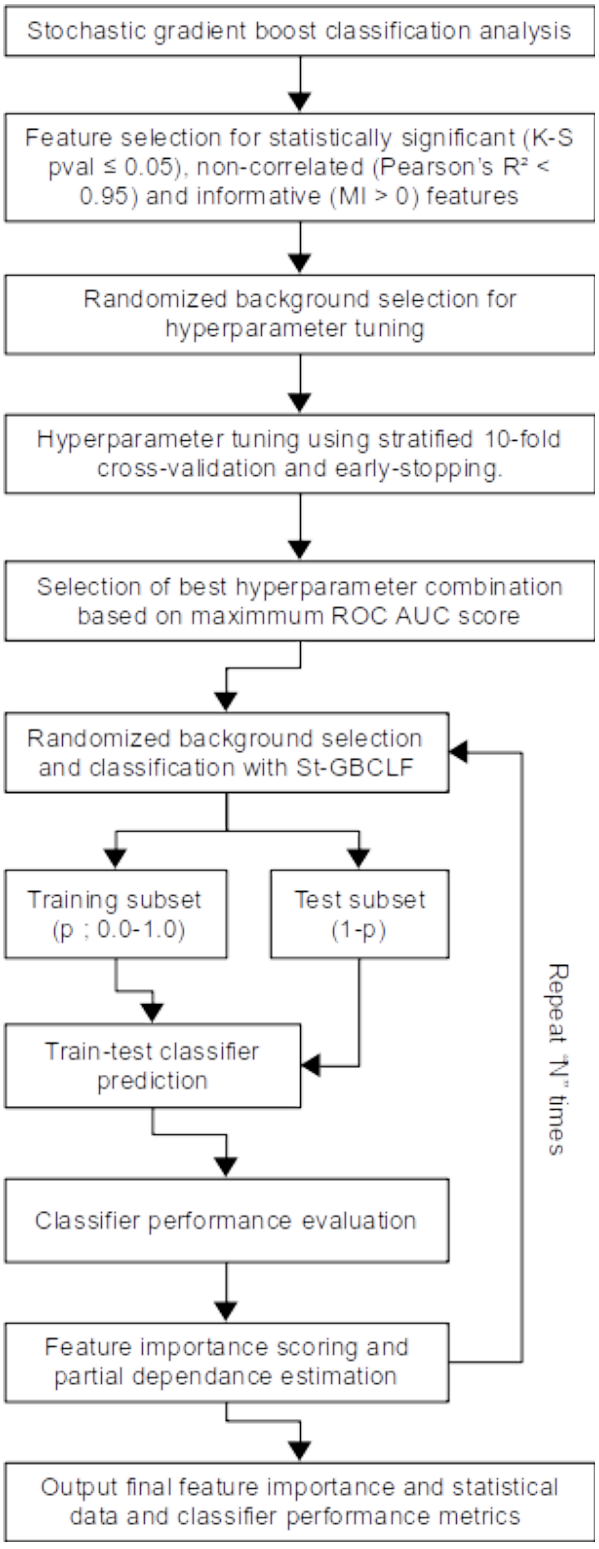
```
analyze_features.py \
-i ./rbfox2_sample.bed \
-m ./rbfox2.test.datamatrix/rbfox2.test.datamatrix.tsv \
-o rbfox2.test \
-p bonferroni \
-mi 0.01 \
-c 0.8 \
-s
```

How it works

build_datamatrix.py



analyze_features.py



Output files

"somefile".datamatrix.tsv - Tabular matrix generated by "build_datamatrix.py" containing biological data transformed in numerical values and/or scales.

best_params.txt - Text file with a dictionary-like structure containing the best set of hyperparameters for the classification step.

classifier_importance.xlsx/.tsv - Table containing the final information for each feature analyzed. Contains data for KS-test and Stochastic Gradient Boost classifier.

classifier_results.pdf - Graphical representation (barchart) containing KS score and relative feature importance found for the top 10 most-important features identified.

mean_deviance.pdf - Graphical representation for deviance scores per iteration found for test and training sets

overall_classifier_metrics.pdf/.tsv/.xlsx - Graphical and tabular representation for overall classifier performance scoring.

statistical_analysis.tsv/.xlsx - Table containing the statistical information from KS-test.

features_ami_and_variance_scores.tsv/.xlsx - Tabular file containing information for mutual information and variance scores found for each feature.

correlation_matrix.tsv/.xlsx - Correlation matrix containing Pearson's R^2 values for each pair of features in the datamatrix.

Change logs

- v1.1.4 - analyze_features.py now plots additional visualization options (total of 4: CDF, violin plot, histogram and KDE) in a single multi-page PDF file; analyze_gtf_regions now outputs a clustermap with rudimentary hierarchical clustering using centroid algorithm.
- v1.1.3 - Added early-stopping to GradientBoostClassifier.
- v1.1.2 - Added mutual information score (MI) and linear correlation as feature selection tools.
- v1.1.1 - Fixed issue with hyperparameter tuning when using 10-fold cross validation.
- v1.1.0 - Re-structure of main scripts to support both Python 3.4 and 2.7.
- v1.0.3 - Added test dataset and readme file.
- v1.0.2 - Fixed bug in classifier metrics barchart plotting.
- v1.0.1 - Fixed bug in KS test.
- v1.0.0 - Original BioFeatureFinder algorithm.