Building and Running Instructions

Bobcat Bioinformatians, Ohio University

Please contact Yichao Li, yl079811@ohio.edu, if you have any questions to run our code.

Dependencies:

Python 2.7, Ubuntu System

MEME suite and the commands should be in the environment PATH.

Bedtools

Samtools

Sklearn 0.17

Pandas

I. Feature vectors preparation --- Data pre-processing

i. DNA motif feature vectors (it can take >5 days to finish, depending on your CPU cores)

1) Motif databases

CISBP, ***HOMER***, ***HOCOMOCO***, ***FactorBook***, and ***Kellis lab ENCODE motif*** databases were used in building the feature vectors. The number of motifs is huge, so we applied a motif similarity analysis to construct a non-redundant motif list (totally 1715 motifs). See step (2).

An epi-genetically derived motif list (totally, 2034 motifs) from ***epigram*** was also used. We didn't remove the redundancies in this motif list.

2) Construct non-redundant TF binding motif list

```
Python rmRedunantMotif.py
```

The outputs are:

a. non_redundant_motif.pwm - contains a combined pwm file for all non-redundant motifs.
b. indPwmfiles - contain individual pwm files for Non-redundant motifs.
c. Results - contain the comparison results of TOMTOM for each iteration // use for debugging.
d. Redundant - contains the motifs removed in each iteration.

3) Motif scanning

a. First, split the entire PWM files into 50 motifs per file. This is because of the memory issues when making feature vectors.

```
Python split_meme.py nonredundant_1715.pwm

Python split_meme.py epiMotifs_2034 .pwm
```

b. Second, split the genome file into smaller files; go to the folder where it contains the train_regions and ladder_regions bed files, copy the two python scripts to that folder, and run the following commands:

```
split -l 10 train_regions.blacklistfiltered.merged.bed train_regions.blacklistfiltered.merged.bed.copy.

python split_train_region.py

split -l 3 ladder_regions.blacklistfiltered.merged.bed ladder_regions.blacklistfiltered.merged.bed.copy.

python split_ladder_region.py
```

c. Then, use the bedtools to extract the genomic sequence for each copy:

```
for i in *copy*;do bedtools getfasta -fi /your/path/hg19.genome.fa -bed $i -fo  $i.fa";done
```

d. Next, let's use fimo to do motif scanning:

```
Python batch_fimo.py   integer1   integer2   motif_database   motif_dir    fasta_dir
```

Note: this script can be paralleled by open more terminals and run it. Integer1 and integer2 is used to control the number of pwm files used in each run; motif_database is either epiMotifs_2034 or nonredundant_1715; motif_dir and fasta_dir is the directories without the last character "/".

Also note that the output (*.cut) will be in the fasta_dir folder.

As an example, what I did is open 8 terminals and run: `Python batch_fimo.py   0   10   motif_database   motif_dir fasta_dir; Python batch_fimo.py   10   20   motif_database   motif_dir    fasta_dir; … etc.`

4) Motifs to feature vectors

The feature vectors are represented in a space separated file. The first column is the coordinates and rest of the columns is the value of motif scanning scores. The order of the motif column is the same order as in the corresponding PWM file.

```
Python   batch.fimo.out.to.csv.py  motif_database   motif_dir   bed_dir  cut_dir
```

Note: motif_database is either epiMotifs_2034 or nonredundant_1715; motif_dir and bed_dir is the directories for motif pwm files and ladder_regions.blacklistfiltered.bed or train_regions.blacklistfiltered.bed; cut_dir is the directory of the output in step (3).

This step gives the final DNA motif feature vector for the entire genome.

ii. Dnase signal feature vectors

1) We are using ***bigWigAverageOverBed*** to extract signal statistics over the 200bp genomic bins.

First, we need to add a line number to the train_regions and ladder_regions bed files.

```
awk -F'\t' 'NR>0{$0=$0"\t"NR-1} 1' ladder_regions.blacklistfiltered.bed > ladder_regions.blacklistfiltered.bed.add.number.bed
```

2) Second, use bigWigAverageOverBed.

```
bigWigAverageOverBed  DNASE.A549.fc.signal.bigwig  ladder_regions.blacklistfiltered.bed.add.number.bed  ladder_regions.blacklistfiltered.bed.add.number.bed.A549.Dnase.out
```

3) Last, we build the Dnase signal feature vectors.

```
Python  to_final_csv.py  bed_add_number_file_dir  Dnase_signal_out_dir
```

Note: bed_add_number_file_dir is the directory of step (1) output. Dnase_signal_out_dir is the directory of step (2) outputs.

iii. DNA shape signal feature vectors

The pre-computed DNA shape signal is downloaded from ftp://rohslab.usc.edu/hg19/

The feature vector building procedure is same as for Dnase.

1) Use ***bigWigAverageOverBed.***

```
bigWigAverageOverBed  hg19.HelT.2nd.wig.bw  ladder_regions.blacklistfiltered.bed.add.number.bed  ladder_regions.blacklistfiltered.bed.add.number.bed.HelT.DNAshape.out
```

2) Build the Shape signal feature vectors.

```
Python  to_final_csv.py  bed_add_number_file_dir  DNAshape_signal_out_dir
```

iv. Gene expression feature vectors

Gene annotation file is downloaded from http://www.gencodegenes.org/releases/19.html

1) PCA feature learning

```
Python  extract_pca_input_matrix.py  gene_expression_tsv_dir

Rscript   pca_feature_learning.R
```

The resulting pca_output_matrix.csv is the final output; this is the ePC feature. (See the 1-page method summary)

2) Nearest gene expression TPM

First, convert the genome annotation file into bed file.

```
Python extract.py  gencode.v19.chr_patch_hapl_scaff.annotation.gtf  > human_gft.bed
```

Note: May need to sort this bed file.

Next, use bedtools closest to get the nearest gene ID for the train_regions and ladder_regions.

```
for i in train_regions.blacklistfiltered.bed.copy.a[a-z] ;do echo $i;bedtools closest -d -a $i -b
human_gft.sorted.bed -t first> "$i.closest";done

for i in ladder_regions.blacklistfiltered.bed.copy.a[a-z];do bedtools closest -d -a $i -
b ../genome_annotation/human_gft.sorted.bed -t first> "$i.closest";done
```

Last, build closest gene feature vector. A 14 cell line gene expression hash object has been pre-computed; this is "gene_expression_dict.pckl".

```
Python to_closest_gene_expression_tsv.py
```

3) TF regulators expression TPM

The TF regulators are extracted from STRING database at http://string-db.org/

The average gene expression levels of the regulators are used in building the feature vector.

```
Python extract_regulators_exp.py
```

II. Sampling of the entire training set --- Data pre-processing

The sampling method will keep 100% Bound locations and the corresponding flanking unbound regions. A threshold X is used to control the sampling percentage of other unbound regions. "A" labeled genomic bins will not be sampled.

```
Python   final_single_file.py  MAFK.train.labels.tsv
```

III. Random Forest Motif Selection for each TF --- Data pre-processing

We have more than 3.7K motifs. We trained a random forest classifier and used the feature importance to select top 120 from non-redundant motifs and top 40 from epi-motifs.

First, build a small training file (all the genomic bins are from just one chromosome) for either the non-redundant motifs or the epi-motifs.

```
Python   to_nonred_motif_selection_training_file.py
```

```
Python   to_epimotif_motif_selection_training_file.py
```

Next, train a random forest classifier and output the feature importance.

```
Python motif_selection.py   training_file
```

Last, get the motif name for the top motifs.

```
Python top_motif_list.py
```

IV. Training files preparation --- Data pre-processing

Since you have created all the feature vectors, you now just need to provide: (1) your sub-sampled training genomic bins; (2) your top motifs for that TF; (3) your feature vectors directories.

The file path and directory path are hard coded in the script. You need to change it before running on your machine.

```
Python   to_final_train_sampled_all_features_csv.py    sub-sampled_training_bins(e.g.
FOXA1.final.all.features.csv)
```

V. Prediction files preparation --- Data pre-processing

For the whole genome prediction, the one file would be too big to process. As we previously splitted the train_regions and ladder_regions into small pieces, we then need to build the prediction file for each of these small copies for each TF/cell combination. The final file is very large; approximately 40G for each TF/cell combination.

```
Python batch_to_prediction_file.py
```

VI. Training models

This is the simplest step. Just provide the training file and do:

```
Python   train.py   your_training_file.csv
```

VII. Making predictions

Since we have many small pieces of prediction files for each TF/cell combination, it is better that their locations are following certain pattern. The script we are using just need the TF name and the CELL name. The file location patterns are hard coded. Change them on your machine.

```
Python  making_prediction_by_name.py    TF  CELL
```

VIII. Making submission files

The prediction files are unordered. To make submission, you need to follow the template. So this script is used to create the final submission file. Again, file path is hard coded.

```
Python make_final_submission.py  TF_CELL(e.g. CTCF_PC-3)
```