03-713 GroupC User Manual

PC Parvangada, Gayatri Joshi, Bernie Zhao

Description

GroupC is a pipeline of tools that takes as input a fastq file or bam file and detects regions that contain m1A modified transcripts on human genome. The software is written in python. (Currently the software requires python 2.7).

Installation

Download

```
$ git clone (some repository)
$ unzip GroupC.zip
$ cd GroupC
```

Install pysam

```
$ pip install pysam
```

For details, refer to https://pysam.readthedocs.io/en/latest/

Install scikit-learn

Instructions can be found on http://scikit-learn.org/stable/install.html

Requires numpy and scipy:

```
$ pip install numpy
$ pip install scipy
```

Usage

Alignment of Reads

$ python main.py align *in.fastq in.fa out.bam*

in.fastq:

sequencing reads ;
Currently, it needs to be in the GroupC directory

in.fa:

reference human genome in FA or FASTA format;
Currently, it needs to be in the GroupC directory

out.bam:

output sorted BAM file in GroupC/bowtie2_aligment;
its index file will be created in the same directory;
Save both files for future use

This command calls Bowtie2 and Samtools to create a BAM file containing the alignments of input reads to the human genome using hard-coded parameters for quality control. This BAM file can be used as the input for detecting m1A modified regions.

Detection of m1A

```
$ python main.py find in.bam in.fa in.bed out.txt
```

in.bam:

alignment file for human sequencing data;
BAM file needs to be sorted, with index file in the same directory;

in.fa:

reference human genome in FA or FASTA format;

in.bed:

file containing list of regions for for detection, with no header;
each row has tab separated chromName, regionStart, regionEnd;
eg. chr1	880000	880200
The classifier will not detect m1A modified windows in any region with size less than 40.

out.txt:

output file containing list of windows that contain m1A modifications

This command uses a pre-trained classifier to detect windows that contain m1A modification. Training data was obtained from Dominissini et al, 2016. Each window in the output file is 40-nt wide.

SAMPLE RUN:

main.py calls either ‘align’ or ‘find” as the second argument. Toy RNAseq data, reference and bedfile of suspected windows have been provided as ‘microglia1.chr19.fastq’, ‘chr19.fa’ and ‘trial_microglia_region1’ respectively. Use the first two to run the align procedure. Use the resulting bamfile to run the find procedure by passing it ‘trial_microglia_region1’.

a. First run the alignment procedure with the following command:

python main.py align microglia1.chr19.fastq chr19.fa microglia

This is a sequence of 6 operations:

1. **CutAdapt**: is run with the following preset parameters. Minimum Read length =30, trim end bases with PHRED score < 20. Maximum mismatch rate of adapters to reads is 6%.
2. **FastQC**: generates a quality report html in the local folder. Also a log of the process call is saved in ‘fastqc_log.txt’ in the ‘FastQC_report’ folder.

3. **Bowtie Build**: Indexes the reference genome for use by the bowtie aligner. Set of dictionaries are created in ‘foo_bowtie_index’ folder. Do not modify!

4. **Bowtie_Aligner**: This is the core function that aligns the reads to the reference genome. The user is asked to input the number of threads.Please use sufficient number of threads to speed up the alignment. Depending on the

size of the reference and the fastq file, this step may even take several hours.

5. **Samtools-conversion to bam**: Final alignment is converted to a bam file using samtools.

6. **Sorting and Indexing the bam file** : This step sorts the bam file. The index file is created in the bowtie2_alignment folder. Please supply this bam file when the find method is called from main.

b. Now call the find command by supplying the bam file and a bed file of suspected regions. Output contains overlapping windows with associated probabilities. Consider highest probability window as the m1A site. The sample command is:

```
python main.py find ./bowtie2_alignment/microglia.bam chr19.fa trial_microglia_region1.bed predicted_windows.bed
```

Notes about the Classifier:

Training
The training file has been provided for the purpose of understanding the method in which the training has been done and the exact classifiers that have been used.
Classifier 1: Logistic regression (scipy)
It has max iterations set to 500.
Logistic regression takes the features as input after we convert them into a quadratic features. Ie. convert them into polynomial degree 2.

Classifier 2:SVM (scipy)
order=3
input features as they are without polynomial degree 2 fit

The trained classifiers have been saved as pickle files.
Log Reg : LogReg_model.sav
SVM : SVM_Model.sav
Joblib from scikit learn has been used to create and save the pickle files.(no need for extra installation , comes with scipy)

Here are the evaluation statistics of the trained model:

Testing.py
Input**:**

A list of 4 strings, each string is the name of the filename generated by the previous stage containing the features needed for training.

Features are created from the input files ,if you wish to see the csv file formed from the features you may uncomment this part from the program. The instructions for doing so have been written in a comment.

The pickle files for the Logistic Regression and the SVM are loaded and the models are used for predicting using the feature set. (These pickle files are python version specific. Currently, our pre-trained classifier only works on python 2.7)

Output: list of windows where there is a possibility of detecting a M1A along with the average of the probabilities of SVM and Logistic Regression.