Supplementary S6 - GCFM Documentation

Release 1.0

Qi Zhang

Oct 22, 2020

CONTENTS

CHAPTER ONE
INTRODUCTION

GCFM is a tool for discriminating pairwise ncRNA sequence relationships. It consists of a series of scripts that can be executed in stages. In this tool, we firstly apply multi-view structure representations, including sequence-structure alignment encoding representation (SSR), RNA structure image representation (SIR), and RNA shape alignment representation (SAR), to capture multi-view features of ncRNAs. Specifically, SSR integrates secondary structure information into ncRNA sequences, avoiding the time-consuming of secondary structure alignment. The novel transformational gray image feature (i.e., SIR) transfers the probability of base pairing (i.e., the possibility of binding between two bases during the formation of a secondary structure) into a gray image, and the gray images of pairwise ncRNAs are aligned by subsampling or upsampling. Similar to SSR, SAR integrates the local shape structure information (i.e., stem, loop) into the alignments of ncRNA sequences. Then, we combine CNN module with GcForest module, and construct a deep fusion model, for the classification of pairwise alignments of ncRNA sequences.

Type Script

Version 1.0

Maintainer Qi Zhang <qizhang18@mails.jlu.edu.cn>

Depends Python(>=3.6)

Note Because part of the computation-heavy process was done on an off-line server (some of the base compilation libraries were too old and could not be updated, affecting subsequent tool installations), multiple deep learning tools were used in the collaboration between different devices, and the whole process was done in separate scripts that could be staged on different devices.

1.1 Environment

GCFM consists of a series of scripts that can be executed in stages, requires a basic UNIX/Linux environment. The gcc compiler with version 4.8.4 or higher is required to be installed. Currently, GCFM does not support Mac or Windows system. Due to the large memory requirement, we recommend users run GCFM on a high-performance computer (HPC), rather than local computers.

1.2 Install

```
# Simplified tree diagram of files
./GCFM_sources
├── requirements.txt
├── scripts
│   ├── CNN
│   ├── gcforest
│   └── WK_NetArch
└── tools-non-py
    ├── dafs
    ├── lib
    │   ├── parse_secondary_structure
    │   └── parse_secondary_structure.cpp
    └── RNAfold
```

Download the source from Download page of web server or:

```
cd /your/working/path/
wget http://bmbl.sdstate.edu/gcfm/static/clf/code/GCFM_sources.zip
```

Unzip the file:

```
unzip GCFM_sources.zip && rm -rf GCFM_sources.zip
```

Install Python packages according to the `requirements.txt` manually or:

```
cd ./GCFM_sources
pip install -r requirements.txt
```

Change the permissions (need to be executable) of `dafs` and `RNAfold` in directory tools-non-py or:

```
cd ./GCFM_sources/tools-non-py
chmod 777 dafs
chmod 777 RNAfold
```

Check if `parse_secondary_structure` is running, and if not, recompile its cpp file:

```
cd ./GCFM_sources/tools-non-py/lib
g++ parse_secondary_structure.cpp -o parse_secondary_structure
```

1.3 Usage

```
# Simplified tree diagram of files
./GCFM_sources
├── scripts
│   ├── make_profile.py
│   ├── prediction.py
│   ├── splice_profile.py
│   ├── train_cnn_muilt.py
│   ├── train_gc.py
│   ├── utils.py
│   └── WK_NetArch
└── tools-non-py
```

1. First, use the `make_profile.py` to extract the features

   ```
   cd ./GCFM_sources/scripts
   python make_profile.py -i "../data/sequence_all.fa" -o "../features/profile/"
   ```

2. These features are then used to build a dataset using `splice_profile.py`

   ```
   python splice_profile.py -d "../features/profile/"
   ```

3. Use `train_cnn_muilt.py` to train the convoluation module of GCFM and turn the features into what is needed for cascade module

   ```
   python train_cnn_muilt.py -d "../features/profile/"
   ```

4. Use `train_gc.py` to train the cascade module and get the final result

   ```
   python train_gc.py
   ```

5. `prediction.py` is used to apply the model to measure the function of the pairwise sequences relationship

   ```
   python train_gc.py -s1 ">ncRNA1\nTGCAATGATGTC" -s2 ">
   ↪ncRNA2\nGGGCATACTCGTAGACCTTGCCGGGACTCT"
   ```

1.4 Requirements

```
# requirements.txt
numpy==1.18.1
torch==1.3.1
psutil==5.6.7
pandas==1.0.1
scipy==1.4.1
kombu==4.6.8
joblib==0.14.1
tqdm==4.42.1
xgboost==0.90
chainer==6.3.0
torchvision==0.4.2
Bio==0.1.0
librosa==0.7.2
Pillow==7.1.2
scikit_learn==0.22.2.post1
```

CHAPTER

TWO

SCRIPTS

2.1 make_profile - a scripts for multi-view feature representation construction

make_profile is a Python script that provides calling utility functions to construct multi-view feature representations.

2.1.1 Main Functions

Here are just a few of the things that **make_profile** does well:

- Read gray image features.
- One-hot encodes nucleotide secondary structure fusion sequence and structural subunit sequence.
- Combination of multi-view feature representations.

2.1.2 Main Program Functions

`scripts.make_profile.`**`get_image_fea`** (*features_dir: str*, *tag: str*, *i: int*, *j: int*)

Get the image features extracted from the pre-training model.

Args:

features_dir: Image feature path.

tag: The name of the pre-trained image model.

i: The ith a ncRNA.

j: The jth a ncRNA.

Returns:

list(array like): Image features.

`scripts.make_profile.`**`make_pairFASTA`** (*dataset*, *itr1*, *outpath*)

Construct the ith ncRNA and the remaining unmatched multi-view features.

According to the process shown in the paper (alignment sequence, extraction of features, etc.), the ith ncRNA and the remaining unmatched multi-perspective features were constructed.

Args:

dataset: All the ncRNA data.

itr1: The ith a ncRNA.

outpath: Feature storage path.

Returns:

Note: This is the multithreaded version, so you should be careful when debugging.

2.2 splice_profile - a scripts for dataset construction

splice_profile is a Python script for building dataset with multi-view feature representations.

2.2.1 Main Functions

Here are just a few of the things that **splice_profile** does well:

- Multi-process building data sets in equal parts.
- A single process builds a data set.

2.2.2 Main Program Functions

`scripts.splice_profile.`**`assembledata`**`(`*ddir*`)`

Build a data set that contains all the data.

Args:

ddir: Data path.

Returns:

`scripts.splice_profile.`**`assembledata_sp10`**`(`*ddir: str*, *n=10*, ***, *mulit=True*, *stop_num=1*`)`

Multi-process builds j times ten equal data sets.

Args:

ddir: Data path.

n: The number of equal parts.

mulit: Whether to multiprocess.

stop_num: Processing rounds.

Returns:

2.3 train_cnn_multi - a training, testing and feature extraction scripts for CNN module

train_cnn_multi is a Python script used to provide CNN module training, testing and feature extraction. In this section, the existing model can be used to multi-process the features required by the backend GCForest classifier. The CNN module can also be retrained and tested.

2.3.1 Main Functions

Here are just a few of the things that train_cnn_multi does well:

- Test the CNN module, or retrain.
- Use CNN module to extract features.

2.3.2 Main Program Functions

`scripts.train_cnn_multi.`**`rc_extract`**(*data*, *model_path='../model/cnn_5.model'*)

Use CNN module to extract the features of the data.

Args:

data: Data for features to be extracted.

model_path: CNN Model store and read paths.

Returns:

The data processed by CNN.

`scripts.train_cnn_multi.`**`run_cnn`**(*X_train*, *X_test*, *y_train*, *y_test*, *i*, *cnn_model_path*, *retrain=False*)

Training and testing CNN module.

Args:

X_train: The training set.

X_test: The testing set.

y_train: The training set label.

y_test: The testing set label.

i: i th number.

cnn_model_path: Model store and read paths.

retrain: Retraining or not.

Returns:

CNN model.

2.4 train_gc - a training, testing scripts for gcforest module

train_gc is a Python script for gcforest module training, testing.

2.4.1 Main Functions

Here are just a few of the things that **train_gc** does well:

- Configuration and training of GCForest modules.

2.4.2 Main Program Functions

`scripts.train_gc.`**`get_ca_config`**`()`

Get the structure of gcForest cascading modules.

Returns:

Configuration information.

`scripts.train_gc.`**`run_gc`**`(`*X_train*, *X_test*, *y_train*, *gc_model_path*`)`

Train gcForest models, predict results and save models.

Args:

X_train: The training set.

X_test: The testing set

y_train: The training set label.

gc_model_path: Model saving path.

Returns: The prediction of y.

2.5 prediction - a script that predicts a pairwise ncRNA

prediction is a Python script for pairwise ncRNA prediction.

2.5.1 Main Functions

Here are just a few of the things that **prediction** does well:

- Complete pairwise ncRNA relation prediction.

2.5.2 Main Program Functions

`scripts.prediction.`**`gcfm_predict`**`(`*seq1*, *seq2*`)`

Predicting the results of pairwise sequences.

Args:

seq1: ncRNA nucleotide sequence A.

seq2: ncRNA nucleotide sequence B.

Returns: Predicted results.

`scripts.prediction.`**`get_image_fea`**`(`*dataset*, *features_dir*, *tag*`)`

Obtain the image features obtained by the pre-training model.

Args:

dataset: ncRNA data sets.

features_dir: The path of the image feature.

tag: The name of the pre-training model.

Returns: Image features.

`scripts.prediction.`**`get_seq_pair`**(*data*, *path_to_pairFasta*)

Build all the ncRNA data sets.

Args:

data: A pairwise sequence of string formats.

path_to_pairFasta: Files that temporarily store pairs of sequences.

Returns:

ncRNA data sets.

`scripts.prediction.`**`make_pairFASTA`**(*dataset*, *env*, *fa_path*)

Construct multi-view feature representation.

Args:

dataset: All the ncRNA data.

env: The environment in which the predicted results are stored.

fa_path: Files that temporarily store pairs of sequences.

Returns:

multi-view feature representation.

`scripts.prediction.`**`rc_extract`**(*data*, *model='../model/cnn_4.model'*)

Use CNN module to extract the features of the data.

Args:

data: Data for features to be extracted.

model: CNN Model store and read paths.

Returns:

The data processed by CNN.

CHAPTER

THREE

UTILS

3.1 utils - a utils scripts for GCFM

utils is a Python script used to provide generic utility functions. In this part, there are many functions involving ncRNA sequence, secondary structure, structural subunit, grayscale transformation image construction, pairing probability calculation and so on.

3.1.1 Main Functions

Here are just a few of the things that **utils** does well:

- Folder path checking and creation.
- Call DAFS, RNAfold, structure subunit computing module.
- Alignment of ncRNA sequence, secondary structure sequence and structural subunit sequence.
- Base pairing probability image construction, grayscale, alignment and feature extraction.

3.1.2 Usages

See Function calls.

3.1.3 Main Program Functions

`scripts.utils.`**`align_anno_seq`**`(`*anno_struct: str*, *seq: str*`)` → str

Align the structure subunits sequences.

Use alignment information of nucleotide sequences, alignment of structural subunits sequences.

Args:

anno_struct: A sequence of structural subunits.

seq: Nucleotide sequence.

Returns:

str: The sequence of structural subunits after alignment.

`scripts.utils.`**`cal_bp_pro`**`(`*mat: list*`)`

Calculate the probability of left and right and unpaired.

The data was extracted from the files generated by RNAfold, and the left and right and unpaired probabilities were calculated.

Args:

mat: List(array) like, containing ncRNA nucleotide pairing were generated by RNAfold.

Returns:

tuple: Result array. This array contains the probability of left and right and unpaired.

`scripts.utils.`**`check_path`**(*path: str*)

Check the target path.

Check if the target path exists and create the path if it does not.

Args:

path: Folder path. Not a file path.

Returns:

None

`scripts.utils.`**`get_annotated_struct`**(*centroid_struct: str*, *pss_path: str*) → str

Calculate the structural subunit.

Args:

centroid_struct: Secondary structure sequence.

pss_path: The path of a dynamic programming program.

Returns:

str: A sequence of structural subunits.

Note: Dynamic programming program from https://github.com/cookkate/rnascan

`scripts.utils.`**`get_matrix`**(*input: str*, *n: int*)

Extract the paired probabilities of nucleotide, and the transformed grayscale image.

The data was extracted from the files generated by RNAfold, and the probabilities of nucleotide were extracted, as well as the transformed grayscale images.

Args:

input: File path. Probability files containing ncRNA nucleotide pairing were generated by RNAfold.

n: Image size.

Returns:

tuple: Result tuple. This tuple contains the paired probabilities of nucleotide, and the transformed grayscale image.

`scripts.utils.`**`make_8bit`**(*pair: str*, *ss: str*)

The fusion secondary structure transforms the four-bit nucleotide sequence into the eight-bit. (Actually five and nine bit)

The four-bit nucleotide sequence is converted into an eight-bit sequence according to whether the corresponding position matches.

Args:

pair: Nucleotide sequence.

ss: Secondary structure sequence.

Returns:

str: A transformed eight-bit nucleotide sequence.

`scripts.utils.`**`run_DAFS`**(*path: str*)

Call DAFS to align pairwise sequences.

Call DAFS to compare the two sequences. The DAFS called by this function is an executable program under Linux.

Args:

path: File path. Subject to DAFS, this file must be in the form of only two sequences.

Returns:

tuple: Result tuple. This tuple contains two aligned sequences, and the gap is filled with '-'.

Note: The file contents are as follows:

>RF00003_ENST00000621107.1

ATACTTACGTAACAGGAGAAAATACGGCCATGAAGTTGGTGTTTCTCGGGGGCGATTT…

>RNU6_928P_ENST00000516876.1

GTGCTCTCTGAAGCAGCACAAATACAAAACTTGGAGTGAAACAGAGATGAG…

Result like this:

ATACTTACGTAACAGGAGAAAATACGGCCATGAAGTTGGTGTTTCTCGGGGGCGATTT…

GTG-CTCT-CT-GA-AGCAGCACAAATACAAA-ACTTGGAGTGAAA-CAG-AGATGAG…

`scripts.utils.`**`run_RNAfold`**(*path*)

Call RNAfold to get information of secondary structure and base pairing.

Call RNAfold to get the secondary structure of the sequence and the probability of base pairing.

Args:

path: File path. Contains ncRNA sequences.

Returns:

tuple: Result tuple. This tuple contains two secondary structure sequences.

Note: Result like this:

…(((…..(((……..))..)).))…

..(…..(.(((…..(((……..))..)).)).)..)..

`scripts.utils.`**`save_image`**(*dataset: list*, *image_mat: list*, *output_path: str*)

Save the transformed grayscale image for use by the pre-training model.

The gray image is obtained according to the transformed image matrix, and the features are extracted with the pre-training model after saving.

Args:

dataset: A list (array like) containing all the data.

image_mat: A list (array like) containing image data.

output_path: The output path.

Returns:

str: A symbol that indicates whether or not it is complete.

`scripts.utils.`**`wk_image`**`(`*output_dir: str*, *image_dir: str*`)`

Feature extraction using pre-trained image network.

Use the pre-trained image network screened by multiple classification ability to extract features.

Args:

output_dir: Feature output path.

image_dir: Grayscale image path.

Returns:

str: A symbol that indicates whether or not it is complete.

CHAPTER

FOUR

CNN

4.1 CNNmodel- Define the relevant content of CNN module

CNNmodel training, prediction, and feature extraction.

Note: The CNN module is referenced from Aoki et al., and fixes some problems. See the original version in http://www.dna.bio.keio.ac.jp/cnn/

4.1.1 Main Functions

Here are just a few of the things that **CNNmodel** does well:

- CNN model training, prediction, and feature extraction.

4.1.2 Usages

See Function calls.

4.1.3 Main Program Functions

class `scripts.CNN.CNNmodel.`**CNN**(*output_ch1*, *output_ch2*, *filter_height*, *filter_width*, *n_units*, *n_label*)

Model structure definition and computational definition.

extract(*x*)

Use CNN module to extract features.

Args: x: The data.

Returns: A feature of fixed length.

class `scripts.CNN.CNNmodel.`**Runchainer**(*batchsize*, *outdir*, *output_ch1*, *output_ch2*, *filter_height*, *width*, *n_units*, *n_label*)

Runchainer.

extract(*data*, *predict_path*)

Extract features.

Args:

data: Data for features to be extracted.

predict_path:

Returns: Extracted features.

`learn``(`*train*, *test*, *gpu*, *epoch*, *cnn_model_path*`)`

Training model.

Args:

train: The training set.

test: The testing set.

gpu: GPU configuration.

epoch: Epoch.

cnn_model_path: CNN model path.

Returns: Model.

`predict``(`*test_data*, *test_label*, *predict_path*`)`

CNN model prediction.

Args:

test_data: The testing set.

test_label: The testing set label.

predict_path:

Returns: ACC, F1

`class` `scripts.CNN.CNNmodel.`**`TestModeEvaluator`**`(`*iterator*, *target*, *converter=<function concat_examples>*, *device=None*, *eval_hook=None*, *eval_func=None*`)`

Model evaluation.

`evaluate``()`

Evaluates the model and returns a result dictionary.

This method runs the evaluation loop over the validation dataset. It accumulates the reported values to `DictSummary` and returns a dictionary whose values are means computed by the summary.

Note that this function assumes that the main iterator raises `StopIteration` or code in the evaluation loop raises an exception. So, if this assumption is not held, the function could be caught in an infinite loop.

Users can override this method to customize the evaluation routine.

Note: This method encloses `eval_func` calls with `function.no_backprop_mode()` context, so all calculations using `FunctionNode`s inside `eval_func` do not make computational graphs. It is for reducing the memory consumption.

Returns: dict: Result dictionary. This dictionary is further reported via `report()` without specifying any observer.

CHAPTER

FIVE

GCFOREST

5.1 GcForest - Define, initialize, and call GcForest

GcForest shows the definition, initialization, construction, and invocation of the GCForest algorithm.

Note: This is a modified version based on the original author and fixes some possible bugs. See the original version in https://github.com/kingfengji/gcForest

5.1.1 Main Functions

Here are just a few of the things that **GcForest** does well:

- GcForest training, prediction.

5.1.2 Usages

See Function calls.

5.1.3 Main Program Functions

class `scripts.gcforest.gcforest.`**`GCForest`**(*config*)

Build the GCForest module.

Args:

ca_config: Parameters.

Note: early_stopping_rounds: int

when not None , means when the accuracy does not increase in early_stopping_rounds, the cascade level will stop automatically growing

max_layers: int

maximum number of cascade layers allowed for exepriments, 0 means use Early Stoping to automatically find the layer number

n_classes: int

Number of classes

est_configs:

List of CVEstimator's config

look_indexs_cycle (list 2d): default=None

specification for layer i, look for the array in look_indexs_cycle[i % len(look_indexs_cycle)] defalut = None <=> [range(n_groups)] .e.g.

look_indexs_cycle = [[0,1],[2,3],[0,1,2,3]] means layer 1 look for the grained 0,1; layer 2 look for grained 2,3; layer 3 look for every grained, and layer 4 cycles back as layer 1

data_save_rounds: int [default=0]

data_save_dir: str [default=None]

each data_save_rounds save the intermidiate results in data_save_dir if data_save_rounds = 0, then no savings for intermidiate results

---

`fit_transform`(*X_train*, *y_train*, *X_test=None*, *y_test=None*, *train_config=None*)

Use GcForest to transform data.

Args:

X_train: The training set.

y_train: The training set label.

X_test: The testing set

y_test: The testing set label.

train_config: Gcforest model configuration file.

Returns: The transformed data.

`predict`(*X*)

Predict the label.

Args:

X: The data.

Returns:

The label.

`predict_proba`(*X*)

Predict the probability of each label.

Args:

X: The data.

Returns:

The probability of each label.

`set_keep_data_in_mem`(*flag*)

flag (bool): if flag is 0, data will not be keeped in memory. this is for the situation when memory is the bottleneck

`set_keep_model_in_mem`(*flag*)

> **flag (bool):** if flag is 0, model will not be keeped in memory. this is for the situation when memory is the bottleneck

`transform`(*X*)

> **return:** if only finegrained proviede: return the result of Finegrained if cascade is provided: return N x (n_trees in each layer * n_classes)

`class` `scripts.gcforest.cascade.cascade_classifier.`**`CascadeClassifier`**(*ca_config*)

Cascade classifier.

Args:

> ca_config: Parameters.

`fit_transform`(*X_groups_train, y_train, X_groups_test, y_test, stop_by_test=False, train_config=None*)

fit until the accuracy converges in early_stop_rounds stop_by_test: (bool)

> When X_test, y_test is validation data that used for determine the opt_layer_id, use this option

`scripts.gcforest.cascade.cascade_classifier.`**`calc_accuracy`**(*y_true, y_pred, name, prefix=''*)

Calculate accuracy.

`scripts.gcforest.cascade.cascade_classifier.`**`get_opt_layer_id`**(*acc_list*)

Return layer id with max accuracy on training data

CHAPTER

SIX

WK_NETARCH

6.1 WK_NetArch - Provides calls for pre-training models to process grayscale images

The model structure of three pre-training models (Alexnet, Resnet and Vgg16) was defined by **WK_NetArch** to provide their forward calculation call and gray image feature extraction.

6.1.1 Main Functions

Here are just a few of the things that **WK_NetArch** does well:

- Model definition, model invocation, feature extraction.

6.1.2 Usages

See Function calls.

6.1.3 Main Program Functions

class `scripts.WK_NetArch.alexnet_features.`**`EncoderCNN`**(*model*)

Alexnet pre-training model call, network structure adaptation modification.

`forward`(*x*)

Forward calculation to extract the required features.

Args: x: The gray image data.

Returns: The SIR feature.

class `scripts.WK_NetArch.resnet101_features.`**`EncoderCNN`**(*model*)

ResNet pre-training model call, network structure adaptation modification.

`forward`(*x*)

Forward calculation to extract the required features.

Args: x: The gray image data.

Returns: The SIR feature.

class `scripts.WK_NetArch.vgg16_features.`**`EncoderCNN`**(*model*)

Vgg16 pre-training model call, network structure adaptation modification.

forward(*x*)

Forward calculation to extract the required features.

Args: x: The gray image data.

Returns: The SIR feature.

`scripts.WK_NetArch.wk_tools.`**extract_features**(*net*, *tag*, *features_dir*, *files_list*)

Batch extraction of features.

Args:

net: Pre-training model.

tag: Name of the pre-training model.

features_dir: Feature output path.

files_list: Batch image file path.

Returns:

`scripts.WK_NetArch.wk_tools.`**extractor**(*net*, *img_path*, *use_gpu=True*)

Use a pre-training model to extract features.

Args:

net: Pre-training model.

img_path: The path of the image.

use_gpu: Whether to use GPU or not.

Returns:

Extracted features.

`scripts.WK_NetArch.wk_tools.`**get_image**(*data_dir*)

Gets the path of all images.

Args:

data_dir: Path to images.

Returns:

Path of all images.

PYTHON MODULE INDEX