dynGENIE3: documentation

Author: Vân Anh Huynh-Thu, vahuynh@uliege.be

This is the documentation for the python implementation of dynGENIE3. This implementation is a research prototype and is provided "as is". No warranties or guarantees of any kind are given.

The dynGENIE3 method is described in the following paper:
Huynh-Thu V. A. and Geurts P. (2018) dynGENIE3: dynamical GENIE3 for the inference of gene networks from time series expression data. *Scientific Reports* 8:3384.

1 Installation

To be able to run dynGENIE3, you have to install the **scikit-learn** module ($\geq$ version 0.16). Instructions for installing sciki-learn are provided here:
`http://scikit-learn.org/dev/install.html`

2 Load the dynGENIE3 package

```
from dynGENIE3 import *
```

3 Run dynGENIE3

The dynGENIE3 method is meant to be run on time series data (with or without steady-state data). A file 'TS_data.pkl', containing an example of time series data, is provided for this tutorial.

```
import _pickle
f = open('TS_data.pkl','rb')
(TS_data, time_points, decay_rates, gene_names) = _pickle.load(f)
f.close()
```

- `TS_data` is a list of 3 arrays, each array corresponding to a time series experiment. The number of time points (rows) does not need to be the same in all the experiments, but the number of genes (columns) must be the same.
- `time_points` is a list of 3 vectors, containing the corresponding time points.
- `decay_rates` is a vector containing gene decay rates.
- `genes_names` is a list containing the names of the genes.

Run dynGENIE3 with its default parameters

The following input arguments are mandatory to run the function `dynGENIE3()`:

- Time series expression data
- The corresponding time points

```
(VIM, alphas, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points)
```

`dynGENIE3()` returns a tuple `(VIM, alphas, prediction_score, stability_score, treeEstimators)` :

- `VIM` is an array containing the scores of the putative regulatory links. *VIM(i,j)* is the weight of the link directed from the i-th gene to j-th gene.
- `alphas` is a vector in which the i-th element is the decay rate of the i-th gene. By default, the decay rate α_i of gene i is estimated from its time series data, by assuming an exponential decay $e^{-\alpha_i t}$ between the highest and lowest observed expression values of gene i.
- `prediction_score` is an empty list by default.
- `stability_score` is an empty list by default.
- `treeEstimators` is an empty list by default.

Set the values of the decay rates

The gene decay rates can be specified with the input argument `alpha`. `alpha` can be either a single positive number (in that case, the decay rates of all the genes are assumed to have the same value) or a vector of positive numbers (in which the i-th element specifies the value of the decay rate of the i-gene).

```
(VIM2, alphas2, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points,alpha=decay_rates)
```

In this case, the returned vector `alphas2` is the same as `decay_rates`.

Run dynGENIE3 on time series data and steady-state data

`dynGENIE3()` can be used to learn networks jointly from time series and steady-state data. The steady-state data must be contained in an array where the element (n, i) is the expression of the i-th gene in the n-th steady-state condition.

```
# Load some steady-state data
SS_data = loadtxt('SS_data.txt',skiprows=1)

(VIM3, alphas3, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points,SS_data=SS_data)
```

Restrict the candidate regulators to a subset of genes

```
# Genes that are used as candidate regulators
regulators = ['CD19', 'CDH17','RAD51','OSR2','TBX3']
(VIM4, alphas4, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points, ...
    gene_names=gene_names,regulators=regulators)
```

In `VIM4`, the links that are directed from genes that are not candidate regulators have a score equal to 0.

Change the tree-based method and its settings

```
# Use Extra-Trees method
tree_method='ET'

# Number of randomly chosen candidate regulators at each node of a tree
K = 7

# Number of trees per ensemble
ntrees = 50

# Run the method with these settings
(VIM5, alphas5, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points, ...
    tree_method=tree_method,K=K,ntrees=ntrees)
```

Compute the ranking quality scores

Two scores can be computed (in an attempt) to estimate the quality of the returned edge ranking:

- The “prediction score" is a measure of the predictive performance of the learned tree models. In practice, the prediction score is the Pearson correlation (averaged over all the target genes and all the trees) between the predicted and true expression values of a target gene in the out-of-bag samples (i.e. the samples left out when learning a tree).
- The “stability score" compares the rankings of candidate regulators respectively returned by the different trees of an ensemble. In practice, the stability score is the average size of the intersection of the two sets of top-5 candidate regulators respectively returned by two trees.

To compute both scores, the input argument `compute_quality_scores` must be set to `True`. Moreover, to compute the prediction score, the tree method must be Random Forests.

```
# The prediction score can only be computed when using Random Forests
tree_method = 'RF'

(VIM6, alphas6, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points, ...
    tree_method=tree_method,compute_quality_scores=True)
```

Note that we show in the paper that optimising the prediction score or the stability score does not allow to identify the values of the decay rates that optimise the quality of the network reconstruction. We leave here the possibility to compute the prediction and stability scores, so that the results presented in the paper can be reproduced.

Save the tree models

The tree models can be saved, e.g. to be later used for the prediction of gene expression profiles. To save the models, the input argument `save_models` must be set to `True`.

```
(VIM7, alphas7, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points,save_models=True)
```

`treeEstimators` is a list containing the different tree models (one for each gene).

Obtain more information

```
help(dynGENIE3)
```

Predict gene expression profiles in double knockout experiments

First, tree models must be learned using the function `dynGENIE3()`, where the input argument `save_models` is set to `True`.

```
# Learn models
(VIM8, alphas8, prediction_score, stability_score, treeEstimators) = ...
  dynGENIE3(TS_data,time_points, ...
    gene_names=gene_names,regulators=regulators,save_models=True)
```

To make predictions, initial expression data are needed. Here, we will use the data contained in the file 'WT_data.npy'.

```
WT_data = load('WT_data.npy')
```

`WT_data` is a one-dimensional array containing the wild-type expression of each gene.

The function `dynGENIE3_predict_doubleKO()` can be used to make predictions in double knockout experiments. For example, the following command line allows to obtain predicted gene expressions profiles at 10 successive time points, with 50 units of time between two time points, in an experiment where genes $CDH17$ and $CD19$ are knocked-out:

```
TS_predict = ...
  dynGENIE3_predict_doubleKO(WT_data,treeEstimators, alphas8, ...
    gene_names,regulators,'CDH17','CD19',10,50)
```

Note that the candidate regulators (input argument `regulators`) must be the same as the ones used when learning the tree models using the function `dynGENIE3()`.
The function `dynGENIE3_predict_doubleKO()` returns an array, where the element (n, i) is the predicted expression of the i-th gene at the n-th time point. The first row of the array contains the initial gene expressions, where the expressions of the two knocked-out genes were set to 0.

Obtain more information

```
help(dynGENIE3_predict_doubleKO)
```

4 Write the predictions

Get the predicted ranking of all the regulatory links

```
get_link_list(VIM)
```

The output will look like this:

```
## G1 G5 0.527481
## G5 G1 0.517994
## G6 G8 0.384952
## G8 G6 0.343328
## G9 G10 0.320162
## G2 G8 0.257154
## G9 G7 0.240072
## ...
```

Each line corresponds to a regulatory link. The first column shows the regulator, the second column shows the target gene, and the last column indicates the score of the link.
If the gene names are not provided, the i-th gene is named "Gi".
Note that the ranking that is obtained will be slightly different from one run to another. This is due to the intrinsic randomness of the Random Forest and Extra-Trees methods. The variance of the ranking can be decreased by increasing the number of trees per ensemble.

Important note on the interpretation of the scores: The weights of the links returned by `dynGENIE3()` **do not have any statistical meaning** and only provide a way to rank the regulatory links. There is therefore no standard threshold value, and caution must be taken when choosing one.

Show the names of the genes

```
get_link_list(VIM,gene_names=gene_names)
```

```
## TBX3 XRCC2 0.527481
## XRCC2 TBX3 0.517994
## CD93 CREB5 0.384952
## CREB5 CD93 0.343328
## CD19 RAD51 0.320162
## GATA5 CREB5 0.257154
## ...
```

Show only the links that are directed from the candidate regulators

```
get_link_list(VIM,gene_names=gene_names,regulators=regulators)
```

Show the first 5 links only

```
get_link_list(VIM,gene_names=gene_names,regulators=regulators,maxcount=5)
```

Write the predicted links in a file

```
get_link_list(VIM,gene_names=gene_names, ...
  regulators=regulators,file_name='ranking.txt')
```

Obtain more information

```
help(get_link_list)
```