grenadine Documentation

Release 1

Sergio Peignier, Pauline Schmitt

Sep 07, 2022

Contents:

GReNaDIne

CHAPTER 1

grenadine package

1.1 Subpackages

1.1.1 grenadine.Evaluation package

Submodules

grenadine.Evaluation.evaluation module

Module contents

This submodule contains several evaluation measures to assess the quality of putative GRNs with respect to a gold standard network

1.1.2 grenadine.Inference package

Submodules

grenadine.Inference.classification_predictors module

This module allows to infer Gene Regulatory Networks using gene expresion data (RNAseq or Microarray). This module implements several inference algorithms based on classification, using scikit-learn.

`grenadine.Inference.classification_predictors.`**`AdaBoost_classifier_score`**(*X*, *y*, ***adab_parameters*)

AdaBoost Classifier, score predictor function based on scikit-learn AdaBoostClassifier.

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors

- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****adab_parameters** – Named parameters for the sklearn AdaBoostClassifier

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the AdaBoostClassifier to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = AdaBoost_classifier_score(tfs,tg)
>>> scores
array([0.24, 0.44, 0.32])
```

grenadine.Inference.classification_predictors.**ComplementNB_classifier_score**(*X*, *y*, ***nb_parameters*)

Complement Naive Bayes Classifier, score predictor function based on scikit-learn ComplementtNB.

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****nb_parameters** – Named parameters for the sklearn MultinomialNB

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the ComplementNB to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
```

(continues on next page)

(continued from previous page)

```
                        columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = ComplementNB_classifier_score(tfs,tg)
>>> scores
array([0.28113447, 0.39096368, 0.45629413])
```

`grenadine.Inference.classification_predictors.`**`GB_classifier_score`**(*X*, *y*, ***gb_parameters*)

Gradient Boosting Classifier, score predictor function based on scikit-learn GradientBoostingClassifier.

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****gb_parameters** – Named parameters for the sklearn _sklearn_ExtraTreesClassifier

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the GradientBoostingClassifier to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                        index =["c1","c2","c3","c4","c5"],
                        columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = GB_classifier_score(tfs,tg)
>>> scores
 array([0.33959125, 0.21147015, 0.4489386 ])
```

`grenadine.Inference.classification_predictors.`**`MultinomialNB_classifier_score`**(*X*, *y*, ***nb_parameters*)

Multinomial Naive Bayes Classifier, score predictor function based on scikit-learn MultinomialNB.

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****nb_parameters** – Named parameters for the sklearn MultinomialNB

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the MultinomialNB to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = MultinomialNB_classifier_score(tfs,tg)
>>> scores
array([0.3010284 , 0.41871716, 0.4272386 ])
```

`grenadine.Inference.classification_predictors.`**`RF_classifier_score`**(*X*, *y*, ***rf_parameters*)

Random Forest Classifier, score predictor function based on scikit-learn RandomForestClassifier.

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****rf_parameters** – Named parameters for the sklearn _sklearn_RandomForestClassifier

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the RandomForestClassifier to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = RF_classifier_score(tfs,tg)
```

(continues on next page)

(continued from previous page)

```
>>> scores
array([0.21071429, 0.4       , 0.28928571])
```

`grenadine.Inference.classification_predictors.`**`SVM_classifier_score`**(*X*, *y*, ***svm_parameters*)

SVM Classifier, score predictor function based on scikit-learn SVC (Support Vector Classifier).

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****svm_parameters** – Named parameters for the sklearn SVC

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the SVC to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = SVM_classifier_score(tfs,tg)
>>> scores
array([0.58413783, 0.5448345 , 0.31764191])
```

`grenadine.Inference.classification_predictors.`**`XRF_classifier_score`**(*X*, *y*, ***xrf_parameters*)

Randomized decision trees Classifier, score predictor function based on scikit-learn ExtraTreesClassifier.

Parameters

- **X** (*pandas.DataFrame*) – Transcription factor gene expressions (discretized or not) where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector (discretized) where rows are experimental conditions
- ****xrf_parameters** – Named parameters for the sklearn _sklearn_ExtraTreesClassifier

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the ExtraTreesClassifier to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,3,size=5), index=["c1","c2","c3","c4","c5
↪"])
>>> scores = XRF_classifier_score(tfs,tg)
>>> scores
array([0.31354167, 0.35520833, 0.33125   ])
```

`grenadine.Inference.classification_predictors.`**`bagging_classifier_score`**(*X*, *y*, ***bagging_parameters*)

Apply the bagging technique to a regression algorithm, based on scikit-learn BaggingClassifier.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****adab_parameters** – Named parameters for the sklearn AdaBoostRegressor

Returns

co-regulation scores.

The i-th element of the score array represents the average score assigned by the Base Regressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn.svm import SVR
>>> np.random.seed(0)
>>> svc = SVC(kernel="linear",decision_function_shape='ovr')
>>> nb_conditions = 10
>>> tfs = pd.DataFrame(np.random.randn(nb_conditions,3),
               index =["c"+str(i) for i in range(nb_conditions)],
               columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randint(0,2,size=nb_conditions),
                  index =["c"+str(i) for i in range(nb_conditions)])
>>> bagging_parameters = {"base_estimator":svc,
                          "n_estimators":5,
                          "max_samples":0.9}
>>> scores = bagging_classifier_score(tfs,tg,**bagging_parameters)
```

(continues on next page)

(continued from previous page)

```
>>> scores
array([0.269231,0.412219,0.299806])
```

grenadine.Inference.inference module

This module allows to infer co-expression Gene Regulatory Networks using gene expression data (RNAseq or Microarray).

`grenadine.Inference.inference.`**`clean_nan_inf_scores`**(*scores*)

Replaces nan and -inf scores by the (minimum_score - 1), and inf scores by (maximum_score + 1)

Parameters

- **scores** (*pandas.DataFrame*) – co-regulation score matrix.
- **are target genes and columns are transcription factors.** (*Rows*) –
- **value at row i and column j represents the score assigned by the** (*The*) –
- **to the regulatory relationship between target gene i** (*score_predictor*) –
- **transcription factor j.** (*and*) –

Returns

co-regulation score matrix.

Rows are target genes and columns are transcription factors. The value at row i and column j represents the score assigned by the score_predictor to the regulatory relationship between target gene i and transcription factor j.

Return type pandas.DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(5, 5),
                    index=["gene1", "gene2", "gene3", "gene4", "gene5"],
                    columns=["c1", "c2", "c3", "c4", "c5"])
>>> tf_list = ["gene1", "gene2", "gene5"]
```

```
>>> # Example with a regression method
>>> from grenadine.Inference.regression_predictors import GENIE3
>>> scores1 = score_links(gene_expression_matrix=data,
                          score_predictor=GENIE3,
                          tf_list=tf_list)
>>> scores1
          gene2     gene5     gene1
gene1  0.484081  0.515919       NaN
gene2       NaN  0.653471  0.346529
gene3  0.245136  0.301229  0.453634
```

(continues on next page)

(continued from previous page)

```
gene4  0.309982  0.306964  0.383054
gene5  0.529839       NaN  0.470161
>>> clean_nan_inf_scores(scores1)
          gene2     gene5     gene1
gene1  0.484081  0.515919  0.245126
gene2  0.245126  0.653471  0.346529
gene3  0.245136  0.301229  0.453634
gene4  0.309982  0.306964  0.383054
gene5  0.529839  0.245126  0.470161
```

`grenadine.Inference.inference.`**`ensemble_score_links`**(*score_links_matrices*, *score_links_weights=None*)

Makes an ensemble co-regulation score matrix from a list of co-regulation score matrices obtained using different methods, and possibly a list of weights for each method

Parameters

- **score_links_matrices** (*list*) – list of co-regulation score matrices (pandas DataFrames)
- **score_links_weights** (*list*) – list of weights for each method (the higher the more confidence on the method). If no value is provided each method as a unitary weight

Returns

co-regulation score matrix.

Rows are target genes and columns are transcription factors. The value at row i and column j represents the score assigned by the score_predictor to the regulatory relationship between target gene i and transcription factor j.

Return type pandas.DataFrame

`grenadine.Inference.inference.`**`join_rankings_scores_df`**(***rank_scores*)

Join rankings and scores data frames generated by different methods.

Parameters ****rank_scores** – Named parameters, where arguments names should be the methods names and arguments values correspond to pandas.DataFrame output of rank_GRN

Returns

joined ranks and joined scores where rows represent possible regulatory links and columns represent each method. Values at row i and column j represent resp. the rank or the score of edge i computed by method j.

Return type (pandas.DataFrame, pandas.DataFrame)

Examples

```
>>> import pandas as pd
>>> method1_rank = pd.DataFrame([[1,1.3, "gene1", "gene2"],
                                 [2,1.1, "gene1", "gene3"],
                                 [3,0.9, "gene3", "gene2"]],
                                 columns=['rank', 'score', 'TF', 'TG'])
>>> method1_rank.index = method1_rank['TF']+'_'+method1_rank['TG']
>>> method2_rank = pd.DataFrame([[1,1.4, "gene1", "gene3"],
                                 [2,1.0, "gene1", "gene2"],
                                 [3,0.9, "gene3", "gene2"]],
                                 columns=['rank', 'score', 'TF', 'TG'])
```

(continues on next page)

(continued from previous page)

```
>>> method2_rank.index = method2_rank['TF']+'_'+method2_rank['TG']
>>> ranks, scores = join_rankings_scores_df(method1=method1_rank, method2=method2_
↪rank)
>>> ranks
             method1  method2
gene1_gene2        1        2
gene1_gene3        2        1
gene3_gene2        3        3
>>> scores
             method1  method2
gene1_gene2      1.3      1.0
gene1_gene3      1.1      1.4
gene3_gene2      0.9      0.9
```

`grenadine.Inference.inference.`**`rank_GRN`**(*coexpression_scores_matrix*, *take_abs_score=False*, *clean_scores=True*, *pyscenic_format=False*)

Ranks the co-regulation scores between transcription factors and target genes.

Parameters

- **`coexpression_scores_matrix`** (*`pandas.DataFrame`*) – co-expression score matrix where rows are target genes and columns are transcription factors. The value at row i and column j represents the score assigned by a score_predictor to the regulatory relationship between target gene i and transcription factor j.
- **`take_abs_score`** (*`bool`*) – take the absolute value of the score instead of taking scores themselves

Returns

ranking matrix.

A ranking matrix contains a row for each possible regulatory link, it also contains 4 columns, namely the rank, the score, the transcription factor id, and the target gene id.

Return type pandas.DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(3, 2),
                    index=["gene1", "gene2", "gene3"],
                    columns=["gene1", "gene3"])
>>> # scores associated to self loops are set to nan
>>> data.iloc[0,0]=np.nan
>>> data.iloc[2,1]=np.nan
>>> ranking_matrix = rank_GRN(data)
>>> ranking_matrix
             rank     score     TF     TG
gene3_gene2   1.0  2.240893  gene3  gene2
gene1_gene3   2.0  1.867558  gene1  gene3
gene1_gene2   3.0  0.978738  gene1  gene2
gene3_gene1   4.0  0.400157  gene3  gene1
```

`grenadine.Inference.inference.score_links`(*gene_expression_matrix*, *score_predictor*, *tf_list=None*, *tg_list=None*, *normalize=False*, *discr_method=None*, *progress_bar=False*, ***predictor_parameters*)

Scores transcription factors-target gene co-expressions using a predictor.

Parameters

- **gene_expression_matrix** (*pandas.DataFrame*) – gene expression matrix where rows are genes and columns ares samples (conditions). The value at row i and column j represents the expression of gene i in condition j.
- **score_predictor** (*function*) – function that receives a pandas.DataFrame X containing the transcriptor factor expressions and a pandas.Series y containing the expression of a target gene, and scores the co-expression level between each transcription factor and the target gene.
- **tf_list** (*list or numpy.array*) – list of transcription factors ids.
- **tg_list** (*list or numpy.array*) – list of target genes ids.
- **normalize** (*boolean*) – If True the gene expression of genes is z-scored
- **discr_method** – discretization method to use, if discretization of target gene expression is desired
- **progress_bar** – bool, if true include progress bar
- ****predictor_parameters** – Named parameters for the score predictor

Returns

co-regulation score matrix.

Rows are target genes and columns are transcription factors. The value at row i and column j represents the score assigned by the score_predictor to the regulatory relationship between target gene i and transcription factor j.

Return type pandas.DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(5, 5),
                    index=["gene1", "gene2", "gene3", "gene4", "gene5"],
                    columns=["c1", "c2", "c3", "c4", "c5"])
>>> tf_list = ["gene1", "gene2", "gene5"]
```

```
>>> # Example with a regression method
>>> from grenadine.Inference.regression_predictors import GENIE3
>>> scores1 = score_links(gene_expression_matrix=data,
                          score_predictor=GENIE3,
                          tf_list=tf_list)
>>> scores1
          gene2     gene5     gene1
gene1  0.484081  0.515919       NaN
gene2       NaN  0.653471  0.346529
gene3  0.245136  0.301229  0.453634
```

(continues on next page)

(continued from previous page)

```
gene4  0.309982  0.306964  0.383054
gene5  0.529839       NaN  0.470161
```

```
>>> # Example with a classification method
>>> from grenadine.Inference.classification_predictors import RF_classifier_score
>>> from grenadine.Preprocessing.discretization import discretize_genexp
>>> discr_method = lambda X: discretize_genexp (X, "efd", 5, axis=1)
>>> scores2 = score_links(gene_expression_matrix=data,
                              score_predictor=RF_classifier_score,
                              tf_list=tf_list,
                              discr_method=discr_method)
>>> scores2
          gene2     gene5     gene1
gene1  0.512659  0.487341       NaN
gene2       NaN  0.463122  0.536878
gene3  0.368175  0.317341  0.314484
gene4  0.302738  0.346799  0.350463
gene5  0.524815       NaN  0.475185
```

grenadine.Inference.regression_predictors module

This module allows to infer co-expression Gene Regulatory Networks using gene expression data (RNAseq or Microarray). This module implements severall inference algorithms based on regression, using scikit-learn.

`grenadine.Inference.regression_predictors.`**`AdaBoost_regressor`**(*X*, *y*, ***adab_parameters*)

AdaBoost regressor, score predictor function based on scikit-learn AdaBoostRegressor.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****adab_parameters** – Named parameters for the sklearn AdaBoostRegressor

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the AdaBoostRegressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
               index =["c1","c2","c3","c4","c5"],
               columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
```

(continues on next page)

(continued from previous page)

```
>>> scores = AdaBoost_regressor(tfs,tg)
>>> scores
array([0.32978247, 0.3617295 , 0.28896647])
```

grenadine.Inference.regression_predictors.**BayesianRidgeScore**(*X*, *y*, ***brr_parameters*)

Score predictor based on scikit-learn BayesianRidge regression.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****brr_parameters** – Named parameters for sklearn BayesianRidge regression

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the sklearn BayesianRidge regressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = BayesianRidgeScore(tfs,tg)
>>> scores
array([1.32082000e-03, 6.24177371e-05, 3.32319918e-04])
```

grenadine.Inference.regression_predictors.**Elastica**(*X*, *y*, ***elastica_parameters*)

ElasticNetCV regressor, score predictor function based on scikit-learn ElasticNetCV.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****elastica_parameters** – Named parameters for the sklearn ElasticNetCV

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the AdaBoostRegressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
               index =["c1","c2","c3","c4","c5"],
               columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = Elastica(tfs,tg)
>>> scores
array([0.05512459, 0.34453337, 0.        ])
```

`grenadine.Inference.regression_predictors.`**`GENIE3`**(*X*, *y*, ***rf_parameters*)

GENIE3, score predictor function based on scikit-learn RandomForestRegressor.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****rf_parameters** – Named parameters for the sklearn RandomForestRegressor

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the RandomForestRegressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                      index =["c1","c2","c3","c4","c5"],
                      columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = GENIE3(tfs,tg)
>>> scores
array([0.11983888, 0.28071399, 0.59944713])
```

`grenadine.Inference.regression_predictors.`**`GRNBoost2`**(*X*, *y*, ***boost_parameters*)

GRNBoost2 score predictor based on scikit-learn GradientBoostingRegressor.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****boost_parameters** – Named parameters for GradientBoostingRegressor

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the GradientBoostingRegressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = GRNBoost2(tfs,tg)
>>> scores
array([0.83904506, 0.01783977, 0.14311517])
```

`grenadine.Inference.regression_predictors.`**`LassoLars_score`**(*X*, *y*, ***l1_parameters*)

Score predictor based on scikit-learn LassoLars regression.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****l1_parameters** – Named parameters for sklearn Lasso regression

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the sklearn LassoLars regressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = LassoLars_score(tfs,tg, alpha=0.01)
>>> scores
array([0.12179406, 0.92205553, 0.15503451])
```

`grenadine.Inference.regression_predictors.`**`Lasso_score`**(*X*, *y*, ***l1_parameters*)

Score predictor based on scikit-learn Lasso regression.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****l1_parameters** – Named parameters for sklearn Lasso regression

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the sklearn Lasso regressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = Lasso_score(tfs,tg, alpha=0.01)
>>> scores
array([0.13825495, 0.94939204, 0.19118214])
```

`grenadine.Inference.regression_predictors.`**`SVR_score`**(*X*, *y*, ***svr_parameters*)

Score predictor based on scikit-learn SVR (Support Vector Regression).

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****svr_parameters** – Named parameters for sklearn SVR regression

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the sklearn SVR regressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
```

(continues on next page)

(continued from previous page)

```
                        index =["c1","c2","c3","c4","c5"],
                        columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = SVR_score(tfs,tg)
>>> scores
array([[-0.38156814,  0.28128811, -1.0230867 ]])
```

`grenadine.Inference.regression_predictors.`**`TIGRESS`**(*X, y, nsplit=100, nstepsLARS=5, alpha=0.4, scoring='area'*)

TIGRESS score predictor based on stability selection.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- **nsplit** (*int*) – number of splits applied, i.e., randomization tests, the highest the best
- **nstepsLARS** (*int*) – number of steps of LARS algorithm, i.e., number of non zero coefficients to keep (Lars parameter)
- **alpha** – Noise multiplier coefficient, Each transcription factor expression is multiplied by a random variable $in [lpha,1]$
- **scoring** (*str*) – option used to score each possible link only "area" and "max" options are available

Returns

co-regulation scores

The i-th element of the score array represents the score assigned by the sklearn randomizedlasso stability selection to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                        index =["c1","c2","c3","c4","c5"],
                        columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = TIGRESS(tfs,tg)
>>> scores
array([349.   , 312.875, 588.125])
```

`grenadine.Inference.regression_predictors.`**`XGENIE3`**(*X, y, **rf_parameters*)

XGENIE3, score predictor function based on scikit-learn ExtraTreesRegressor.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors

- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****rf_parameters*** – Named parameters for the sklearn RandomForestRegressor

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the ExtraTreesRegressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = XGENIE3(tfs,tg)
>>> scores
array([0.24905241, 0.43503283, 0.31591477])
```

grenadine.Inference.regression_predictors.**bagging_regressor**(*X*, *y*, ***bagging_parameters*)

Apply the bagging technique to a regression algorithm, based on scikit-learn BaggingRegressor.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****adab_parameters*** – Named parameters for the sklearn AdaBoostRegressor

Returns

co-regulation scores.

The i-th element of the score array represents the average score assigned by the Base Regressor to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn.svm import SVR
>>> np.random.seed(0)
>>> svr = SVR(kernel="linear")
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                 index =["c1","c2","c3","c4","c5"],
```

(continues on next page)

(continued from previous page)

```
                columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> bagging_parameters = {"base_estimator":svr,
                          "n_estimators":100,
                          "max_samples":0.7}
>>> scores = bagging_regressor(tfs,tg,**bagging_parameters)
>>> scores
array([0.32978247, 0.3617295 , 0.28896647])
```

`grenadine.Inference.regression_predictors.`**`stability_randomizedlasso`**(*X*, *y*, ***rl_parameters*)

Score predictor based on scikit-learn randomizedlasso stability selection.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****rl_parameters** – Named parameters for sklearn randomizedlasso

Returns

co-regulation scores.

The i-th element of the score array represents the score assigned by the sklearn randomizedlasso stability selection to the regulatory relationship between the target gene and transcription factor i.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = stability_randomizedlasso(tfs,tg)
>>> scores
array([0.11 , 0.17 , 0.085])
```

grenadine.Inference.statistical_predictors module

This module allows to infer co-expression Gene Regulatory Networks using gene expression data (RNAseq or Microarray). This module implements severall inference algorithms based on statistical predictors, using scipy-stats and scikit-learn.

`grenadine.Inference.statistical_predictors.`**`CLR`**(*X*, *y*, ***mi_parameters*)

Score predictor function based on scikit-learn mutual_info_regression score.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****mi_parameters** – Named parameters for sklearn mutual_info_regression

Returns

co-regulation scores.

The i-th element of the score array represents the score of the sklearn mutual_info_regression computation between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = CLR(tfs,tg)
>>> scores
array([6.66666667e-02, 1.16666667e-01, 2.22044605e-16])
```

`grenadine.Inference.statistical_predictors.`**`abs_pearsonr_coef`**(*X*, *y*)

Score predictor function based on the scipy-stats absolute Pearson correlation.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions

Returns

co-regulation scores.

The i-th element of the score array represents the absolute value of the correlation between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
```

(continues on next page)

(continued from previous page)

```
>>> scores = abs_pearsonr_coef(tfs,tg)
>>> scores
array([0.41724166, 0.02212467, 0.23708491])
```

`grenadine.Inference.statistical_predictors.`**`abs_spearmanr_coef`**(*X*, *y*)

Score predictor function based on the scipy-stats absolute Spearman correlation.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions

Returns

co-regulation scores.

The i-th element of the score array represents the absolute value of the correlation between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = abs_spearmanr_coef(tfs,tg)
>>> scores
array([0.5, 0.3, 0.3])
```

`grenadine.Inference.statistical_predictors.`**`energy_distance_score`**(*X*, *y*, ***energy_distance_parameters*)

Score predictor function based on the scipy-stats energy distance between 1D distributions.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****energy_distance_parameters** – Named parameters for the scipy-stats energy distance

Returns

co-regulation scores.

The i-th element of the score array represents the score between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = energy_distance_score(tfs,tg)
>>> scores
array([0.40613705, 0.6881455 , 0.72786711])
```

`grenadine.Inference.statistical_predictors.`**`f_regression_score`**(*X*, *y*)

Score predictor function based on the scikit-learn f_regression score.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions

Returns

co-regulation scores.

The i-th element of the score array represents the score of the f_regression linear test between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = f_regression_score(tfs,tg)
>>> scores
array([0.63235967, 0.00146922, 0.17867071])
```

`grenadine.Inference.statistical_predictors.`**`kendalltau_score`**(*X*, *y*, ***kendalltau_parameters*)

Score predictor function based on the scipy-stats Kendall's tau correlation measure.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****kendalltau_parameters** – Named parameters for the scipy-stats kendall's tau correlation measure

Returns

co-regulation scores.

The i-th element of the score array represents the score of the score between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = kendalltau_score(tfs,tg)
>>> scores
array([0.8487997 , 1.30065214, 0.20467198])s
```

`grenadine.Inference.statistical_predictors.`**`mannwhitneyu_score`**(*X*, *y*, ***mannwhitneyu_parameters*)

Score predictor function based on the scipy-stats Mann-Whitney rank test.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****mannwhitneyu_parameters** – Named parameters for the scipy-stats Mann-Whitney rank test

Returns

co-regulation scores.

The i-th element of the score array represents the score between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = mannwhitneyu_score(tfs,tg)
>>> scores
array([1.52213525, 0.47101693, 0.3795872 ])
```

`grenadine.Inference.statistical_predictors.`**`theilslopes_score`**(*X*, *y*, ***theilslopes_parameters*)

Score predictor function based on the scipy-stats Theil-Sen robust slope estimator.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****theilslopes_parameters** – Named parameters for the scipy-stats Theil-Sen robust slope estimator

Returns

co-regulation scores.

The i-th element of the score array represents the score between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = theilslopes_score(tfs,tg)
>>> scores
array([0.92309299, 0.90933202, 0.26451817])
```

`grenadine.Inference.statistical_predictors.`**`wasserstein_distance_score`**(*X*, *y*, ***wasserstein_distance_parameters*)

Score predictor function based on the scipy-stats Wasserstein distance between 1D distributions.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****wasserstein_distance_parameters** – Named parameters for the scipy-stats Wasserstein distance

Returns

co-regulation scores.

The i-th element of the score array represents the score between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),
                       index =["c1","c2","c3","c4","c5"],
                       columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = wasserstein_distance_score(tfs,tg)
>>> scores
array([0.36457586, 0.72057084, 0.81207932])
```

`grenadine.Inference.statistical_predictors.`**`wilcoxon_score`**(*X*, *y*, ***wilcoxon_parameters*)

Score predictor function based on the scipy-stats Wilcoxon signed-rank test.

Parameters

- **X** (*pandas.DataFrame*) – Transcriptor factor gene expressions where rows are experimental conditions and columns are transcription factors
- **y** (*pandas.Series*) – Target gene expression vector where rows are experimental conditions
- ****wilcoxon_parameters** – Named parameters for the scipy-stats Wilcoxon signed-rank test

Returns

co-regulation scores.

The i-th element of the score array represents the score between target gene expression and the i-th transcription factor gene expression.

Return type numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> tfs = pd.DataFrame(np.random.randn(5,3),index =["c1","c2","c3","c4","c5"],
↪columns=["tf1","tf2","tf3"])
>>> tg = pd.Series(np.random.randn(5),index=["c1","c2","c3","c4","c5"])
>>> scores = wilcoxon_score(tfs,tg)
>>> scores
array([1.36537718, 0.64797987, 0.30086998])
```

Module contents

This submodule contains different data-driven scoring functions to infer GRNs from gene expression datasets

1.1.3 grenadine.Preprocessing package

Submodules

grenadine.Preprocessing.discretization module

This module allows to discretize gene expression datasets. It is mostly based on scikit-learn library. Different discretization methods are available : EWD (equal width, uniform), EFD (equal frequency, quantile), kmeans, bikmeans (Li et al., 2010).

`grenadine.Preprocessing.discretization.`**`bikmeans_original`**(*data*, *nb_bins*)

Discretize data into nb_bins intervals, with method bikmeans, from the publication by Li et al, 2010.

Parameters

- **data** (*pandas.DataFrame*) – dataset to discretize
- **nb_bins** (*int*) – number of intervals in which to discretize data

Returns dataframe of discretized data

Return type pandas.DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(3, 5),
                        index=["gene1", "gene2", "gene3"],
                        columns=["c1", "c2", "c3", "c4", "c5"])
>>> data
             c1        c2        c3        c4        c5
gene1  1.764052  0.400157  0.978738  2.240893  1.867558
gene2 -0.977278  0.950088 -0.151357 -0.103219  0.410599
gene3  0.144044  1.454274  0.761038  0.121675  0.443863
>>> discr_data = bikmeans_original(data=data, nb_bins=2)
>>> discr_data
        c1   c2   c3   c4   c5
gene1  1.0  0.0  0.0  1.0  1.0
gene2  0.0  1.0  0.0  0.0  0.0
gene3  0.0  1.0  0.0  0.0  0.0
```

`grenadine.Preprocessing.discretization.`**`bikmeans_simple`**(*data*, *nb_bins*)

Discretize data into nb_bins intervals, with method bikmeans, simplified. From the publication by Li et al, 2010. See function bikmeans_original() for the full implementation of bikmeans as described in the paper.

Parameters

- **data** (*pandas.DataFrame*) – dataset to discretize
- **nb_bins** (*int*) – number of intervals in which to discretize data

Returns dataframe of discretized data

Return type pandas.DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(3, 5),
                        index=["gene1", "gene2", "gene3"],
                        columns=["c1", "c2", "c3", "c4", "c5"])
>>> data
             c1        c2        c3        c4        c5
gene1  1.764052  0.400157  0.978738  2.240893  1.867558
gene2 -0.977278  0.950088 -0.151357 -0.103219  0.410599
gene3  0.144044  1.454274  0.761038  0.121675  0.443863
>>> discr_data = bikmeans_simple(data=data, nb_bins=2)
>>> discr_data
        c1   c2   c3   c4   c5
gene1  2.0  1.0  1.0  2.0  2.0
gene2  1.0  2.0  1.0  1.0  1.0
gene3  1.0  2.0  1.0  1.0  1.0
```

`grenadine.Preprocessing.discretization.`**`discretize_genexp`**(*data*, *method*, *nb_bins=2*, *axis=0*)

Discretize data into nb_bins intervals, with specified method, along specified axis.

Parameters

- **data** (*pandas.DataFrame or pandas.Series*) – dataset to discretize
- **method** (*str*) – method used for discretization, amongst: 'kmeans', 'bikmeans', 'ewd', 'efd'
- **nb_bins** (*int*) – (default 2) number of intervals in which to discretize data
- **axis** (*int*) – (default 0) indicates if discretization should be done on each column (0) or each line (1) of data. Ignore this parameter if method is bikmeans

Returns dataframe or series of discretized data, depending on the dimension of passed data

Return type pandas.DataFrame or pandas.Series

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(3, 5),
                        index=["gene1", "gene2", "gene3"],
                        columns=["c1", "c2", "c3", "c4", "c5"])
>>> data
             c1        c2        c3        c4        c5
gene1  1.764052  0.400157  0.978738  2.240893  1.867558
gene2 -0.977278  0.950088 -0.151357 -0.103219  0.410599
gene3  0.144044  1.454274  0.761038  0.121675  0.443863
>>> discr_data = discretize_genexp(data=data, method='efd')
>>> discr_data
        c1   c2   c3   c4   c5
gene1  1.0  0.0  1.0  1.0  1.0
gene2  0.0  1.0  0.0  0.0  0.0
gene3  1.0  1.0  1.0  1.0  1.0
```

grenadine.Preprocessing.rnaseq_normalization module

This module allows to normalize RNAseq gene expression data.

`grenadine.Preprocessing.rnaseq_normalization.DEseq2`(*raw_counts*, *col_data*, *rlog=True*)

Apply R DEseq2 normalization.

Parameters

- **raw_counts** (*pandas.DataFrame*) – raw RNAseq counts where rows are genes and columns are conditions
- **col_data** (*pandas.DataFrame*) – Two columns, one corresponding to ids of each condition (individuals), and one with the experiment id (if many repetitions)

Returns Normalized counts

Return type pandas.DataFrame

Example

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> raw_counts = pd.DataFrame(np.random.randint(0,1000,(20,10)),
                              columns = ["Z"+str(i) for i in range(10)])
>>> col_data = pd.DataFrame([["Z0","1"],
                             ["Z1","2"],
                             ["Z2","3"],
                             ["Z3","4"],
                             ["Z4","5"],
                             ["Z5","6"],
                             ["Z6","7"],
                             ["Z7","8"],
                             ["Z8","9"],
                             ["Z9","10"]
                             ],columns=["individuals","conditions"])
>>> raw_counts.columns = col_data["individuals"]
>>> col_data.index = col_data['individuals']
>>> DEseq2(raw_counts,col_data,rlog=False)
individuals          X0          X1     ...              X8          X9
0            408.025477  382.991634     ...        7.745300  611.474516
1            165.238388  516.593367     ...      270.224902  596.251084
2            289.912839  377.510537     ...      727.197585   60.893728
3            463.502625  627.585575     ...      385.543809  718.884285
4             59.056319  674.174898     ...      364.029087  243.574911
5            573.263865  181.561329     ...      129.948918  570.878697
6            304.229522  314.477925     ...      802.068816   44.824550
7            537.472156  376.825400     ...       36.144732  373.819828
8            323.914962  608.401737     ...      748.712307  100.643800
9            464.695682  294.608949     ...      781.414683  535.357356
10           559.543710   57.551516     ...      112.737140  822.065324
11           517.786716  123.324676     ...      618.763389  768.783312
12           222.505121  584.421939     ...       81.755942  166.612005
13           361.496256  175.395095     ...      333.047888  515.905193
14           330.476775  666.638390     ...      779.693506  312.926101
15           331.073304  653.620785     ...      493.978005  787.389729
```

(continues on next page)

(continued from previous page)

```
16          437.851901   84.271862     ...      483.650938  347.601696
17          466.485268   28.090621     ...      750.433484    9.303208
18          459.326926  210.337087     ...      149.742461  468.543405
19          221.312064  126.065225     ...      662.653421  435.559302
```

`grenadine.Preprocessing.rnaseq_normalization.`**`RPK`**(*raw_counts*, *seq_lengths*, *seq_in_kb=False*)

Reads Per Kilobase normalization.

Parameters

- **raw_counts** (*pandas.DataFrame*) – raw RNAseq counts where rows are genes and columns are conditions
- **seq_lengths** (*pandas.Series*) – sequences DNA lengths
- **seq_in_kb** (*bool*) – True if lengths in kb, False otherwise

Returns Normalized counts

Return type pandas.DataFrame

Examples

```
>>> import numpy as np
>>> np.random.seed(0)
>>> import pandas as pd
>>> nb_genes = 1000
>>> nb_conditions = 5
>>> raw_counts = np.random.randint(0,1e6,(nb_genes,nb_conditions))
>>> raw_counts = pd.DataFrame(raw_counts)
>>> seq_lengths = np.random.randint(100,20000,nb_genes)
>>> seq_lengths = pd.Series(seq_lengths)
>>> rpk = RPK(raw_counts, seq_lengths)
>>> rpk.head()
               0              1              2              3              4
0  321202.997719   99612.577387  142010.101010   38433.365917  313911.697621
1   26853.843441  155566.114245   63431.417489   53620.768688   21611.248237
2   97195.319962   71353.390640   59624.960204  117133.237822  117212.034384
3  132006.796941   72465.590484  356436.703483  256785.896347  229981.733220
4   48384.227419   34354.424576   18889.143614   37956.492944   45220.490091
```

`grenadine.Preprocessing.rnaseq_normalization.`**`RPKM`**(*raw_counts*, *seq_lengths*, *seq_in_kb=False*)

Reads Per Kilobase Million (also known as FPM: Fragments per kilobase).

Parameters

- **raw_counts** (*pandas.DataFrame*) – raw RNAseq counts where rows are genes and columns are conditions
- **seq_lengths** (*pandas.Series*) – sequences DNA lengths
- **seq_in_kb** (*bool*) – True if lengths in kb, False otherwise

Returns Normalized counts

Return type pandas.DataFrame

Examples

```
>>> import numpy as np
>>> np.random.seed(0)
>>> import pandas as pd
>>> nb_genes = 1000
>>> nb_conditions = 5
>>> raw_counts = np.random.randint(0,1e6,(nb_genes,nb_conditions))
>>> raw_counts = pd.DataFrame(raw_counts)
>>> seq_lengths = np.random.randint(100,20000,nb_genes)
>>> seq_lengths = pd.Series(seq_lengths)
>>> rpkm = RPKM(raw_counts, seq_lengths)
>>> rpkm.head()
            0           1           2           3           4
0  649.733415  201.368439  291.638511   76.398582  628.676848
1   54.320288  314.479420  130.265692  106.588393   43.281252
2  196.607901  144.242035  122.448576  232.839698  234.742741
3  267.024989  146.490365  731.994898  510.443933  460.588733
4   97.872216   69.448026   38.791619   75.450645   90.563924
```

`grenadine.Preprocessing.rnaseq_normalization.`**`RPM`**(*raw_counts*)

Reads Per Million.

Parameters **raw_counts** (*pandas.DataFrame*) – raw RNAseq counts where rows are genes and columns are conditions

Returns Normalized counts

Return type pandas.DataFrame

Examples

```
>>> import numpy as np
>>> np.random.seed(0)
>>> import pandas as pd
>>> nb_genes = 1000
>>> nb_conditions = 5
>>> raw_counts = np.random.randint(0,1e6,(nb_genes,nb_conditions))
>>> raw_counts = pd.DataFrame(raw_counts)
>>> rpm = RPM(raw_counts)
>>> rpm.head()
            0            1            2            3            4
0  1994.031850   617.999738   895.038590   234.467249  1929.409246
1   308.104674  1783.727269   738.867008   604.569366   245.491264
2  1235.090833   906.128463   769.221953  1462.698984  1474.653899
3   628.576824   344.838319  1723.115991  1201.585019  1084.225878
4  1921.133736  1363.195297   761.440687  1481.020714  1777.679267
```

`grenadine.Preprocessing.rnaseq_normalization.`**`TPM`**(*raw_counts*, *seq_lengths*, *seq_in_kb=False*)

Transcript Per Million normalization.

Parameters

- **raw_counts** (*pandas.DataFrame*) – raw RNAseq counts where rows are genes and columns are conditions
- **seq_lengths** (*pandas.Series*) – sequences DNA lengths

- **seq_in_kb** (*bool*) – True if lengths in kb, False otherwise

Returns Normalized counts

Return type pandas.DataFrame

Examples

```
>>> import numpy as np
>>> np.random.seed(0)
>>> import pandas as pd
>>> nb_genes = 1000
>>> nb_conditions = 5
>>> raw_counts = np.random.randint(0,1e6,(nb_genes,nb_conditions))
>>> raw_counts = pd.DataFrame(raw_counts)
>>> seq_lengths = np.random.randint(100,20000,nb_genes)
>>> seq_lengths = pd.Series(seq_lengths)
>>> tpm = TPM(raw_counts, seq_lengths)
>>> tpm.head()
             0            1            2            3            4
0  2455.468465   739.530213  1103.147117   265.510632  2397.256398
1   205.286894  1154.932887   492.740902   370.430324   165.039097
2   743.019352   529.732184   463.172003   809.195846   895.115733
3  1009.139172   537.989227  2768.832068  1773.963432  1756.306584
4   369.878069   255.049468   146.732550   262.216233   345.336316
```

`grenadine.Preprocessing.rnaseq_normalization.`**`log`**(*X, base=10, pseudocount=1*)

Add a pseudocount and apply the log transformation with a given base.

Parameters

- **X** (*pandas.DataFrame or numpy.array*) – gene expression matrix
- **base** (*float*) – logarithm base
- **pseudocount** (*float*) – pseudocount value

Returns log transformed gene expression matrix

Return type pandas.DataFrame or numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(5, 5),
                    index=["c1", "c2", "c3", "c4", "c5"],
                    columns=["gene1", "gene2", "gene3", "gene4", "gene5"])
>>> pseudocount = -np.min(data.values)+1
>>> log_data = log(data, pseudocount=pseudocount)
>>> log_data
       gene1     gene2     gene3     gene4     gene5
c1  0.725670  0.596943  0.656264  0.762970  0.734043
c2  0.410897  0.653509  0.531687  0.537790  0.598089
c3  0.567853  0.699600  0.634883  0.565218  0.601718
c4  0.589577  0.703039  0.524764  0.587268  0.431186
c5  0.000000  0.623932  0.645169  0.448834  0.765128
```

grenadine.Preprocessing.standard_preprocessing module

This module allows to pre-process gene expression data.

`grenadine.Preprocessing.standard_preprocessing.`**`cat_gene_expression_dfs`**(*gene_expression_dfs*)

Concatenate different gene expression datasets, based on gene id (rows).

Parameters **gene_expression_dfs** (*list of pandas.DataFrame*) – Expression datasets list

Returns concatenated gene expression datasets

Return type pandas.DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data1 = pd.DataFrame(np.random.randn(3, 3),
                    index=["gene1", "gene2", "gene3"],
                    columns=["c1", "c2", "c3"])
>>> data1
             c1        c2        c3
gene1  1.764052  0.400157  0.978738
gene2  2.240893  1.867558 -0.977278
gene3  0.950088 -0.151357 -0.103219
>>> data2 = pd.DataFrame(np.random.randn(3, 3),
                    index=["gene2", "gene3", "gene4"],
                    columns=["c4", "c5", "c6"])
>>> data2
             c4        c5        c6
gene2  0.410599  0.144044  1.454274
gene3  0.761038  0.121675  0.443863
gene4  0.333674  1.494079 -0.205158
>>> data=cat_gene_expression_dfs([data1, data2])
>>> data
             c1        c2        c3        c4        c5        c6
gene1  1.764052  0.400157  0.978738       NaN       NaN       NaN
gene2  2.240893  1.867558 -0.977278  0.410599  0.144044  1.454274
gene3  0.950088 -0.151357 -0.103219  0.761038  0.121675  0.443863
gene4       NaN       NaN       NaN  0.333674  1.494079 -0.205158
```

`grenadine.Preprocessing.standard_preprocessing.`**`columns_matrix_OT_norm`**(*X, reference=None, bins=None, **Sinkhorn-Transport_para*)

Use optimal transport in order to make all conditions disributions alike.

Parameters

- **X** (*pandas.DataFrame*) – gene expression matrix
- **r_percentile** (*numpy.array*) – reference distribution
- **bins** (*numpy.array*) – bins for percentiles computation

- **SinkhornTransport_para** – ot.da.SinkhornTransport parameters

Returns Normalized matrix

Return type pandas.DataFrame

Examples

```
>>> import numpy as np
>>> import pandas as pd
>>> a = pd.DataFrame(np.random.randn(10000,10))
>>> b = pd.DataFrame(np.random.randn(10000,10)*3+4)
>>> bins = list(range(1,100))
>>> b_ = columns_matrix_OT_norm(b,a.iloc[:,0],bins,reg_e=5e-1)
```

`grenadine.Preprocessing.standard_preprocessing.`**`mean_std_polishing`**(*A*, *nb_iterations=5*)

Iterative z-score on rows and columns.

Parameters

- **A** (*pandas.DataFrame or numpy.array*) – matrix
- **nb_iterations** (*int*) – number of polishing iterations

Returns Polished matrix

Return type pandas.DataFrame or numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(5, 5),
                    index=["c1", "c2", "c3", "c4", "c5"],
                    columns=["gene1", "gene2", "gene3", "gene4", "gene5"])
>>> norm_data = mean_std_polishing(data)
>>> norm_data
       gene1     gene2     gene3     gene4     gene5
c1  0.336095 -1.618781  0.187436  1.109617 -0.014367
c2 -0.321684  0.586608 -1.606905  0.484159  0.857821
c3  0.139260  0.860934  0.976541 -1.395814 -0.580921
c4  1.243263  0.421752 -0.585940  0.282319 -1.361394
c5 -1.363323 -0.161066  0.826375 -0.421998  1.120013
```

`grenadine.Preprocessing.standard_preprocessing.`**`median_outliers_filter`**(*X*, *threshold=3*)

Ensures that all the values of data_set are within: $median(X) \pm \tau \times MAD(X))$

Parameters

- **X** (*pandas.DataFrame or numpy.array*) – gene expression matrix (for instance)
- **threshold** (*float*) – τ threshold

Returns X without outliers (outliers set to the extreme values allowed)

Return type pandas.DataFrame or numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(5, 5),
                    index=["c1", "c2", "c3", "c4", "c5"],
                    columns=["gene1", "gene2", "gene3", "gene4", "gene5"])
>>> median_outliers_filter(data)
       gene1     gene2     gene3     gene4     gene5
c1  1.764052  0.400157  0.978738  0.674682  1.867558
c2 -0.977278  0.950088 -0.653101 -0.103219  0.410599
c3  0.144044  1.454274  0.761038  0.121675  0.443863
c4  0.333674  1.494079 -0.653101  0.313068 -0.854096
c5 -2.552990  0.653619  0.864436 -0.674682  2.269755
```

`grenadine.Preprocessing.standard_preprocessing.`**`z_score`**(*A*, *axis=0*)

Compute the z-score along the specified axis.

Parameters

- **A** (*pandas.DataFrame or numpy.array*) – matrix
- **axis** (*int*) – 0 for columns and 1 for rows

Returns Normalized matrix

Return type pandas.DataFrame or numpy.array

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(0)
>>> data = pd.DataFrame(np.random.randn(5, 5),
                    index=["c1", "c2", "c3", "c4", "c5"],
                    columns=["gene1", "gene2", "gene3", "gene4", "gene5"])
>>> norm_data = z_score(data)
>>> norm_data
       gene1     gene2     gene3     gene4     gene5
c1  1.254757 -1.222682  0.914682  1.672581  0.828015
c2 -0.446591 -0.083589 -1.038607 -0.418644 -0.331945
c3  0.249333  0.960749  0.538403 -0.218012 -0.305461
c4  0.367024  1.043200 -1.131598 -0.047267 -1.338834
c5 -1.424523 -0.697678  0.717120 -0.988659  1.148225
```

Module contents

This submodule contains several pre-processing techniques for gene expression datasets (standardizations, discretizations and RNAseq normalization)

1.2 Module contents

GReNaDIne: Gene Regulatory Network Data-driven Inference

This package allows to infer Gene Regulatory Networks through several Data-driven methods. Pre-processing and evaluation methods are also included.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index