
LOFS User's Manual in MATLAB

Public domain version 1.0 for MATLAB

Release date: 21/12/2015

Kui Yu¹, Wei Ding², and Xindong Wu³

¹University of South Australia, Australia

²University of Massachusetts Boston, USA

³University of Vermont, USA

Kui.Yu@unisa.edu.au, ding@cs.umb.edu, xwu@uvm.edu.cn

LOFS is a software toolbox for online streaming feature selection. It provides the first open-source library for use in MATLAB that implements the state-of-the-art algorithms of online streaming feature selection.

Copyright (C) 2015 Kui Yu, Wei Ding, and Xindong Wu.

The library is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

CONTENTS

1. Overview of LOFS	1
1.1 Introduction	1
1.2 Architecture of LOFS	1
1.3 Core function overview	2
2. Setup in MATLAB	2
2.1 Getting and installing LOFS	2
2.2 Data format	2
3. Core functions in LOFS	3
3.1 Learning features added individually	3
3.1.1 Alpha-Investing	3
3.1.2 osfs_d	4
3.1.3 osfs_z	5
3.1.4 fast_osfs_d	6
3.1.5 fast_osfs_z	7
3.1.6 saola_mi	7
3.1.7 saola_z_test	8
3.2 Learning grouped features added sequentially	9
3.2.1 group_f	9
3.2.2 saloa_group_mi	10
3.2.3 saloa_group_z_test	11
3.3 Correlation module	12
3.3.1 my_cond_indep_chisquare	12
3.3.2 my_cond_indep_fisher_z	13
3.3.3 mi	14
3.3.4 cmi	15
3.3.5 SU	15
3.4 Evaluation module	16
3.4.1 perfcurve	16
3.4.2 cal_kappa	17
3.4.3 Friedmantest	18
3.4.4 Nemenyitest	19

1. Overview of LOFS

1.1 Introduction

Traditional online feature selection deals with the observations sequentially added while the total dimensionality is fixed. In contrast, as a novel research direction, online streaming feature selection deals with sequentially added dimensions in feature space while the number of data instances is fixed. Many big data applications call for online streaming feature selection to consume sequentially added dimensions over time. Online streaming feature selection provides a new, complementary algorithmic methodology to enrich online feature selection, especially dealing with high dimensionality in big data analytics. Currently, there are two active research topics in this existing research direction. One is to online learning features added individually, and the other is to mine grouped features added sequentially over time.

The library provides the first open-source library for use in MATLAB that implements the state-of-the-art algorithms of online streaming feature selection. It is designed to facilitate the development of new algorithms in this exciting research direction and make the comparisons between new methods and existing ones available.

1.2 Architecture of LOFS

The LOFS architecture is based on a separation of three modules, that is, CM (Correlation Measure), Learning, and SC (Statistical Comparison), as shown in Figure 1. The learning module consists of two submodules, LFI (Learning Features added Individually) and LGF (Learning Grouped Features added sequentially).

The three modules in the LOFS architecture are designed independently, and all codes follow the MATLAB standards. This makes that the LOFS library is simple, easy to implement, and extendable flexibly. One can easily add a new algorithm to the LOFS library and share it through the LOFS framework without modifying the other modules.

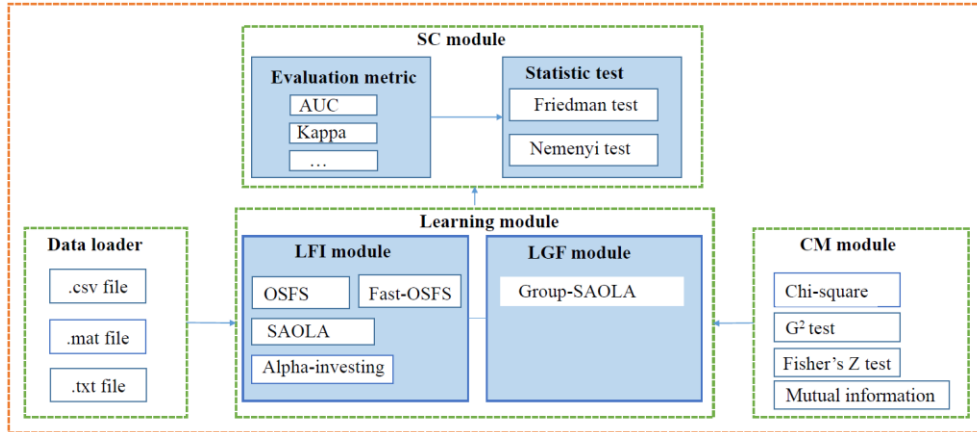


Figure 1 The architecture of LOFS

In the CM module, the library provides four measures, Chi-square test, G^2 test, the Fisher's Z test, and mutual information, to compute correlations between features, where Chi-square test, G^2 test, and mutual information to deal with discrete data while the Fisher's Z test to handle continuous data.

In the learning module, the library provides the state-of-the-art algorithms of online streaming feature selection, including Alpha-Investing, OSFS, Fast-OSFS, SAOLA, and Group-SAOLA, for facilitating the development of new algorithms in this exciting research problem and making comparisons between the new and existing methods available.

1.3 Core function overview

The core functions provided in the LOFS library are listed in Table 1.

Table 1 Core functions of LOFS

MATLAB Function	Corresponding Algorithm
Alpha_Investing	Alpha-Investing algorithm
osfs_d	OSFS algorithm with Chi-square or G^2 test
osfs_z	OSFS algorithm with Fisher's Z test
fast-osfs_d	Fast-OSFS algorithm with Chi-square or G^2 test
fast-osfs_z	Fast-OSFS algorithm with Fisher's Z test
saola_mi	SAOLA algorithm with mutual information
saola_z_test	SAOLA algorithm with Fisher's Z test
group_f	feature grouping algorithm
group-saola_mi	group-SAOLA algorithm with mutual information
group-saola_z_test	group-SAOLA algorithm with Fisher's Z test
my_cond_indep_chisquare	Chi-square test and G^2 test
my_cond_indep_fisher_z	Fisher's Z test
mi	mutual information
cmi	conditional mutual information
SU	symmetrical uncertainty
perfcurve	calculating AUC
cal_kappa	kappa statistic
Friedmantest	Friedman test
Nemenyi test	Nemenyi test

2. Setup in MATLAB

2.1 Getting and installing LOFS

The LOFS website is at <https://github.com/uiy/LOFS>. LOFS needs to calculate mutual information between variables by calling functions in the library of MIToolbox¹. To run LOFS, it is required that (1) Windows 7 operating system or above version; (2) MATLAB12a, or above to be installed; (3) The folders and subfolders in the LOFS package should be added to the search path in MATLAB, as shown in Figure 2.

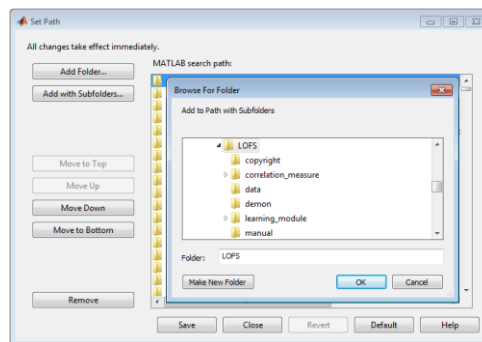


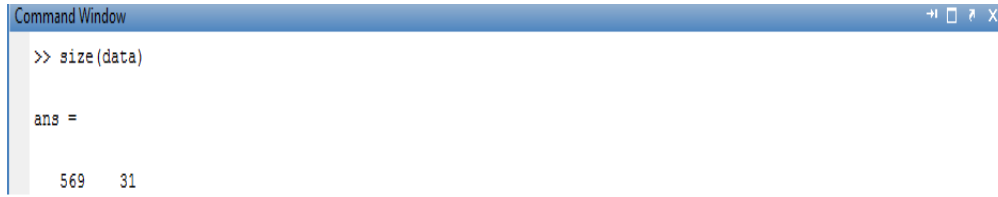
Figure 2 Adding the LOFS package to the MATLAB search path

2.2 Data format

Firstly, it is required that a data set should be correctly imported to MATLAB. Currently, LOFS supports the mat file, the txt file, and the csv file. To read and write the ARFF (Weka Attribute-Relation File Format) files and LIBSVM data files will be supported in our future plan.

¹ <http://www.cs.man.ac.uk/~gbrown/fstoolbox/>

Secondly, in a data set, columns represent features², rows denote data observations. If a data set includes N features in total, in general, LOFS assumes that the first $N - 1$ columns denote the predictive feature matrix, and the last column represents the class attribute vector. An example in MATLAB is given in Figure 3 as follows.



```

Command Window
>> size(data)

ans =

    569    31
  
```

Figure 3 Example of Data format

This denotes that the data set contains 569 data instances, and 31 features. The first 30 columns are predictive features, and the last column is the class attribute.

3. Core functions

3.1 Learning features added individually

3.1.1 Alpha-Investing

Description

Implement the Alpha_Investing algorithm.

Usage

[selectFeatures, time]= Alpha_Investing(X, Y)

Arguments

Inputs	X	Matrix of the predictive features in data matrix
	Y	Vector of the class attribute in data matrix
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

This algorithm was proposed by Zhou et al. (2006). It sequentially considers new features as additions to a predictive model by modelling the candidate feature set as a dynamically generated feature stream. Alpha-investing only calculates whether a new coming feature is added to the current feature set, and never considers removing features from the currently selected feature set.

Reference

Jing Zhou, Dean P. Foster, Robert A. Stine, and Lyle H. Ungar. (2006) Streamwise feature selection. *Journal of Machine Learning Research*, 7:1861-1885.

Example

```

>> load('wdbc.mat')

>> [selectFeatures, time] =Alpha_Investing(wdbc(:,1:30), wdbc(:,31))
  
```

² In the manual, we use the terms “attribute”, “variable”, and “feature” interchangeably.

Results

The figure shows a MATLAB Command Window and Workspace. The Command Window contains the following code and output:

```
>> clear
>> load('wdbc.mat')
>> [selectFeatures,time] =Alpha_Investing(wdbc(1:1:30), wdbc(1:1:31))

selectFeatures =

     1     2     3     4     5     6     7     8     9    10    12    16    17    20    21    22    24    25    26    27

time =

    0.0853
```

The Workspace shows the following variables:

Name	Value
selectFeatures	<1x20 double>
time	0.0853
wdbc	<569x31 double>

Figure 4 Example of the Alpha_Investing algorithm

3.1.2 osfs_d

Description

Implement the OSFS algorithm for discrete data using Chi-square test or G^2 test.

Usage

[selectFeatures, time]=osfs_d(data, class_index, alpha, test)

Arguments

Inputs	data	The discrete data used for training (matrix). For Chi-square test and G^2 test in LOFS, discrete data have to be in a special format: feature X has to take consecutive integer values starting from 1, that is, $1 \dots \max_value(X)$. For example, if $\max_value(X)=3$, this means that feature X takes values {1,2,3}.
	class_index	Index of the class attribute in data matrix. In general, LOFS assumes that the last column represents the class attribute vector in the input data set.
	alpha	Threshold on statistic (the significance level). It is always set to 0.01 or 0.05.
	test	Statistical test desired to use. The parameter test= 'chi2' for Chi-square test and test='g2' for G^2 test.
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

The OSFS algorithm maintains a best feature subset from the features available so far by processing each feature upon its arrival with a two-phase subset discovery scheme: online relevance analysis and online redundancy analysis. OSFS not only determines whether to add a new arriving features to the current feature set, but also removes features from the selected feature set currently to keep it as small as possible.

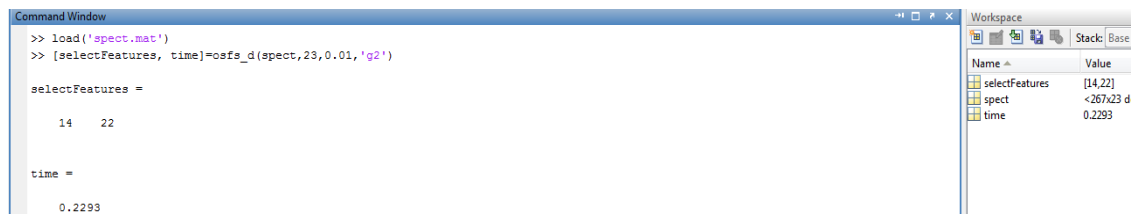
Reference

Xindong Wu, Kui Yu, Hao Wang, and Wei Ding. (2010) Online streaming feature selection. In Proceedings of the 27th international conference on machine learning (ICML'10), 1159–1166.

Example

```
>> load('spect.mat')
>> [selectFeatures, time]=osfs_d(spect,23,0.01,'g2')
```

Results



```
Command Window
>> load('spect.mat')
>> [selectFeatures, time]=osfs_d(spect,23,0.01,'g2')

selectFeatures =

    14    22

time =

    0.2293

Workspace
Name Value
selectFeatures [14,22]
spect <267x23 d
time 0.2293
```

Figure 5 Example of the osfs_d function

3.1.3 osfs_z

Description

Implement the OSFS algorithm for continuous (numeric) data using Fisher's Z test.

Usage

```
[selectFeatures, time]=osfs_z(data, class_index, alpha)
```

Arguments

Inputs	data	The continuous data used for training matrix
	class_index	Index of the class attribute in data matrix. In general, LOFS assumes that the last column represents the class attribute vector in the input data set.
	alpha	Threshold on statistic (the significance level). It is always set to 0.01 or 0.05.
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

Please refer to the details of the osfs_d function mentioned above.

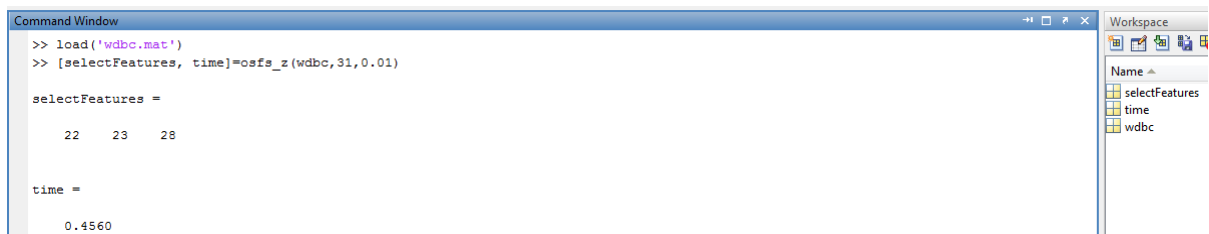
Reference

Xindong Wu, Kui Yu, Hao Wang, and Wei Ding. (2010) Online streaming feature selection. In Proceedings of the 27th international conference on machine learning (ICML'10), 1159–1166.

Example

```
>> load('wdbc.mat')
>> [selectFeatures, time]=osfs_z(wdbc,31,0.01)
```

Results



```
Command Window
>> load('wdbc.mat')
>> [selectFeatures, time]=osfs_z(wdbc,31,0.01)

selectFeatures =

    22    23    28

time =

    0.4560

Workspace
Name Value
selectFeatures
time
wdbc
```

Figure 6 Example of the osfs_z function

3.1.4 fast_osfs_d

Description

Implement the Fast-OSFS algorithm for discrete data using Chi-square test or G^2 test.

Usage

```
[selectFeatures, time]=fast_osfs_d(data, class_index, alpha, test)
```

Arguments

Inputs	data	The discrete data used for training (matrix). For Chi-square test and G^2 test in LOFS, discrete data have to be in a special format: feature X has to take consecutive integer values starting from 1, that is, $1 \dots \max_value(X)$. For example, if $\max_value(X)=3$, this means that feature X takes values $\{1,2,3\}$.
	class_index	Index of the class attribute in data matrix. In general, LOFS assumes that the last column represents the class attribute vector in the input data set.
	alpha	Threshold on statistic (the significance level). It is always set to 0.01 or 0.05.
	test	Statistical test desired to use. The parameter test= 'chi2' for Chi-square test and test='g2' for G^2 test.
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

It is a fast version of OSFS. Fast-OSFS divides the process of handling redundant features in OSFS into two steps: (1) determining whether to keep an incoming new feature or not, and (2) identifying which of the selected features observed so far may become redundant once the inclusion of the new feature occurs.

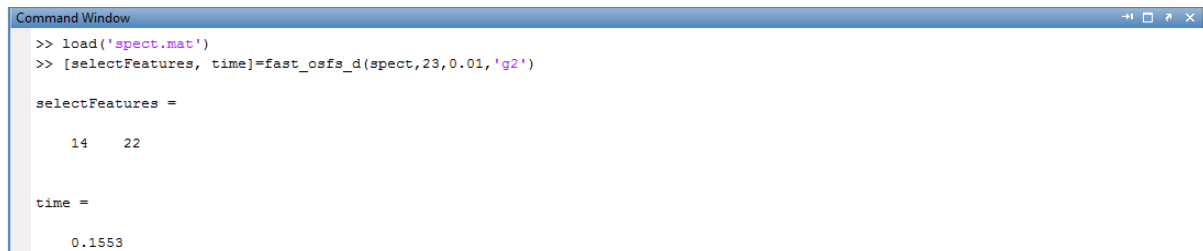
Reference

Xindong Wu, Kui Yu, Wei Ding, HaoWang, and Xingquan Zhu. (2013) Online feature selection with streaming features. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35:1178–1192.

Example

```
>> load('spect.mat')
>> [selectFeatures, time]=fast_osfs_d(spect,23,0.01,'g2')
```

Results



```
Command Window
>> load('spect.mat')
>> [selectFeatures, time]=fast_osfs_d(spect,23,0.01,'g2')

selectFeatures =

    14    22

time =

    0.1553
```

Figure 7 Example of the fast_osfs_d function

3.1.5 fast_osfs_z

Description

Implement the Fast-OSFS algorithm for continuous (numeric) data using Fisher's Z test.

Usage

```
[selectFeatures, time]=fast_osfs_z(data, class_index, alpha)
```

Arguments

Inputs	data	The continuous data used for training matrix
	class_index	Index of the class attribute in data matrix. In general, LOFS assumes that the last column represents the class attribute vector in the input data set.
	alpha	Threshold on statistic (the significance level). It is always set to 0.01 or 0.05.
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

Please refer to the detail of the fast_osfs_d function mentioned above.

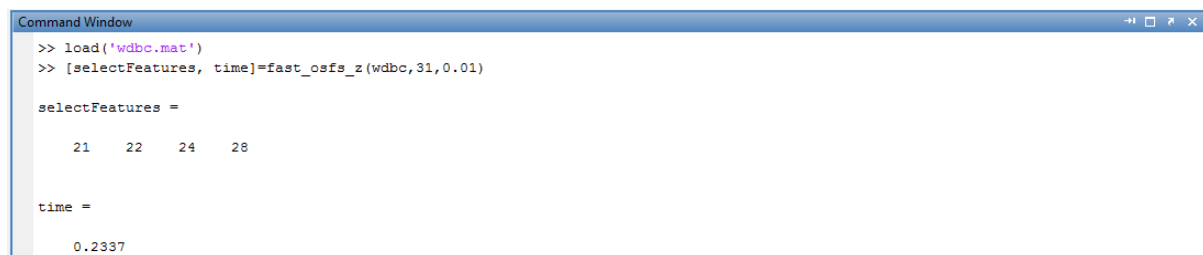
Reference

Xindong Wu, Kui Yu, Wei Ding, HaoWang, and Xingquan Zhu. (2013) Online feature selection with streaming features. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35:1178–1192.

Example

```
>> load('wdbc.mat')
>> [selectFeatures, time]=fast_osfs_z(wdbc,31,0.01)
```

Results



```
Command Window
>> load('wdbc.mat')
>> [selectFeatures, time]=fast_osfs_z(wdbc,31,0.01)

selectFeatures =

    21    22    24    28

time =

    0.2337
```

Figure 8 Example of the fast_osfs_z function

3.1.6 saola_mi

Description

Implement the SAOLA algorithm for discrete data using mutual information.

Usage

```
[selectFeatures, time]=saola_mi(data, threshold)
```

Arguments

Inputs	data	The discrete data used for training (matrix). Each feature in data matrix has to take consecutive integer values starting from 0 or 1. In general, LOFS assumes that the last column represents the class attribute vector in the input data set.
	threshold	Threshold on statistic for mutual information measure
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

To tackle dimensionality in the scale of millions or more, compared to OSFS and Fast-OSFS, the SAOLA algorithm employs online pairwise comparisons to maintain a parsimonious model over time in an online manner to make online streaming feature selection scalable in big data analytics.

Reference

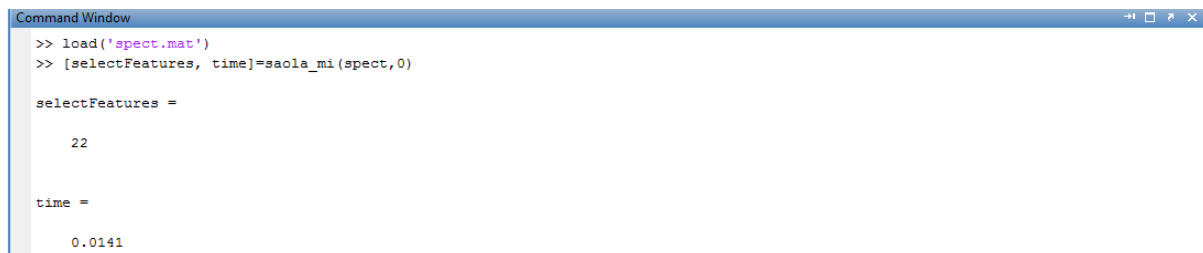
Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2014) Towards scalable and accurate online feature selection for big data. In Proceedings of the 14th IEEE International Conference on Data Mining (ICDM'14), 660-669.

Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2015) Scalable and accurate online feature selection for big data. arXiv:1511.09263v1 [cs.LG], 2015.

Example

```
>> load('spect.mat')
>> [selectFeatures, time]=saola_mi(spect, 0)
```

Results



```
Command Window
>> load('spect.mat')
>> [selectFeatures, time]=saola_mi(spect,0)

selectFeatures =

    22

time =

    0.0141
```

Figure 9 Example of the saola_mi function

3.1.7 saola_z_test

Description

Implement the SAOLA algorithm for continuous (numeric) data using Fisher's Z test.

Usage

```
[selectFeatures, time]=saola_z_test(data, alpha)
```

Arguments

Inputs	data	The continuous data used for training (matrix), and the class attribute in data matrix has to take consecutive integer values starting from 0 or 1 for classification. In general, LOFS assumes that the last column represents the class attribute vector in the input data set. The class attribute has to take consecutive integer values starting from 0 for classification.
	alpha	Threshold on statistic (the significance level). It is always set to 0.01 or 0.05.
Outputs	selectFeatures	Vector with the indices of the selected features in data matrix
	time	Computational cost (in seconds)

Details

Please refer to the details of the `saola_mi` function mentioned above.

Reference

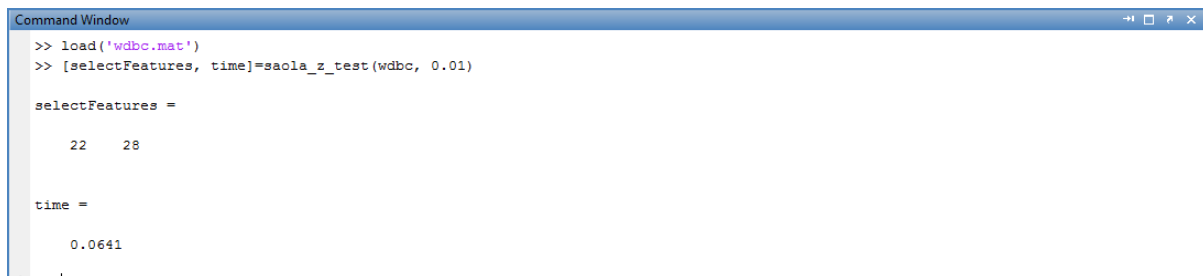
Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2014) Towards scalable and accurate online feature selection for big data. In Proceedings of the 14th IEEE International Conference on Data Mining (ICDM'14), 660-669.

Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2015) Scalable and accurate online feature selection for big data. arXiv:1511.09263v1 [cs.LG], 2015.

Example

```
>> load('wdbc.mat')
>> [selectFeatures, time]=saola_z_test(wdbc, 0.01)
```

Results



```
Command Window
>> load('wdbc.mat')
>> [selectFeatures, time]=saola_z_test(wdbc, 0.01)

selectFeatures =

    22    28

time =

    0.0641
```

Figure 10 Example of the `saola_z_test` function

3.2 Learning grouped features added sequentially

3.2.1 group_f

Description

Divide features in different groups randomly.

Usage

```
group_feature=group_f(features_index, numberGroups)
```

Arguments

Inputs	features_index	Vector with indices of attributes except for the class attribute
	numberGroups	Number of groups desired to divide
Output	group_feature	Feature groups (a cell array of group indices). If the dimensionality of a data set is divided into k distinct groups, each group includes the indices of features assigned to this group.

Example

```
>> load('spect.mat')
>> group_feature=group_f([1:22], 4)
```

Results

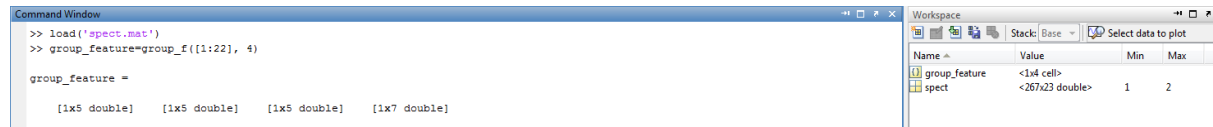


Figure 11 Example of group splitting

3.2.2 saloa_group_mi

Description

Performing the group-SAOLA for discrete data using mutual information.

Usage

```
[selectFeatures, selectGroups, time] = saloa_group_mi(group_feature,data, class_index,threshold)
```

Arguments

Inputs	group_feature	Cell array of group indices. You can implement the group_f.m function in the library to get the input parameter.
	data	Discrete data used for training (matrix)
	class_index	Index of the class attribute in data matrix
	threshold	Threshold on statistic for mutual information measure
Outputs	selectFeatures	Vector with the indexes of the selected features
	selectGroups	Number of selected groups
	time	Computational cost (in seconds)

Details

In some applications, group information is embedded in feature space. For instance, in image analysis, features are generated in groups which represent colour, texture and other visual information. The group-SAOLA algorithm online yields a set of feature groups that is sparse between groups as well as within each group for maximizing its predictive performance for classification, even the dimensionality in the scale of millions or more.

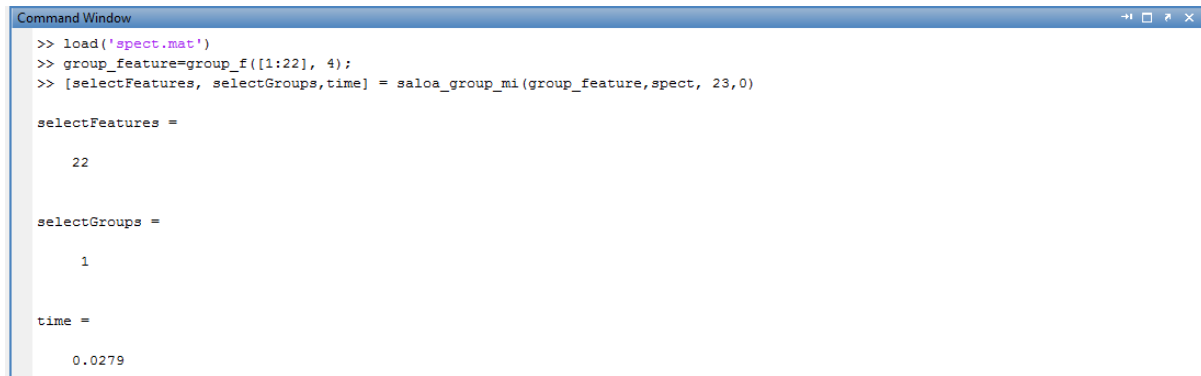
Reference

Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2015) Scalable and accurate online feature selection for big data. arXiv:1511.09263v1 [cs.LG], 2015.

Example

```
>> load('spect.mat')
>> group_feature=group_f([1:22], 4);
>> [selectFeatures, selectGroups, time] = saloa_group_mi(group_feature, spect, 23, 0)
```

Results



```
Command Window
>> load('spect.mat')
>> group_feature=group_f([1:22], 4);
>> [selectFeatures, selectGroups, time] = saloa_group_mi(group_feature, spect, 23, 0)

selectFeatures =

    22

selectGroups =

     1

time =

    0.0279
```

Figure 12 Example of the saloa_group_mi function

3.2.3 saloa_group_z_test

Description

Perform the group-SAOLA for continuous (numeric) data using Fisher's Z test.

Usage

[selectFeatures, selectGroups, time] = saloa_group_z_test (group_feature, data, class_index, alpha)

Arguments

Inputs	group_feature	Cell array of group indices. You can implement the group_f.m function in the library to get the input parameter.
	data	Continuous data used for training (matrix), and the class attribute in data matrix has to take consecutive integer values starting from 0.
	class_index	Index of the class attribute in data matrix
	alpha	Threshold on statistic (the significance level). It is always set to 0.01 or 0.05.
Outputs	selectFeatures	Vector with the indexes of the selected features
	selectGroups	Number of selected groups
	time	Computational cost (in seconds)

Details

Please refer to the description of the detail of the saloa_group_mi function mentioned above.

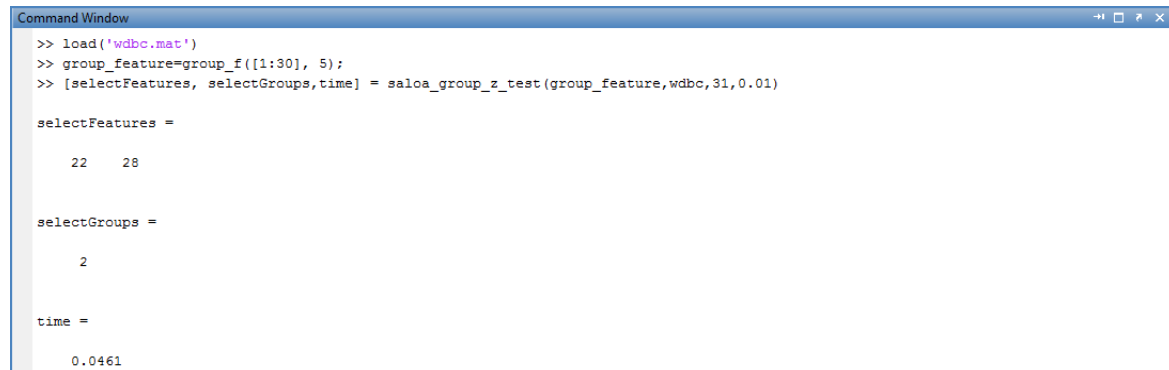
Reference

Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2015) Scalable and accurate online feature selection for big data. arXiv:1511.09263v1 [cs.LG], 2015.

Example

```
>> load('wdbc.mat')
>> group_feature=group_f([1:30], 5);
>> [selectFeatures, selectGroups,time] = saloa_group_z_test(group_feature,wdbc,31,0.01)
```

Results



```
Command Window
>> load('wdbc.mat')
>> group_feature=group_f([1:30], 5);
>> [selectFeatures, selectGroups,time] = saloa_group_z_test(group_feature,wdbc,31,0.01)

selectFeatures =

    22    28

selectGroups =

     2

time =

    0.0461
```

Figure 13 Example of the saloa_group_z_test function

3.3 Correlation Module

3.3.1 my_cond_indep_chisquare

Description

Calculate whether two variables are independent or not conditioned on a subset using Chi-square or G^2 test.

Usage

[CI] = my_cond_indep_chisquare(data,X, Y, S, test, alpha, ns)

Arguments

Inputs	data	The data used for training (matrix). For Chi-square test and G^2 test in LOFS, discrete data have to be in a special format: feature X has to take consecutive integer values starting from 1, that is, $1 \dots \max_value(X)$. For example, if $\max_value(X)=3$, this means that feature X takes values $\{1,2,3\}$.
	X	Index of feature X in data matrix
	Y	Index of feature Y in data matrix
	S	Conditioning set. It includes the indices of features in data matrix.
	alpha	Threshold on statistic (the significance level). The parameter is always set to 0.01 or 0.05.
	test	Statistical test desired to use. The parameter <i>test</i> = 'chi2' for Pearson's Chi-square test and <i>test</i> ='g2' for G^2 likelihood ratio test.
	ns	Vector with the sizes of the corresponding maximum values for all variables in data matrix (default: max(data), as shown in Figure 14). For example, $ns = [2 \ 2 \ 3]$. This specifies that the values of the first and the second feature which can take is $\{1, 2\}$ respectively, and the values of the third one that takes is $\{1, 2, 3\}$. The parameter <i>ns</i> should be empty (i.e. []) if Fisher's Z test is used (since variables are continuous.).
Output	CI	Test result (1=conditional independency, 0=dependency)

Reference

R. Neapolitan. (2003) Learning Bayesian Networks. Prentice Hall (pp. 611-615).

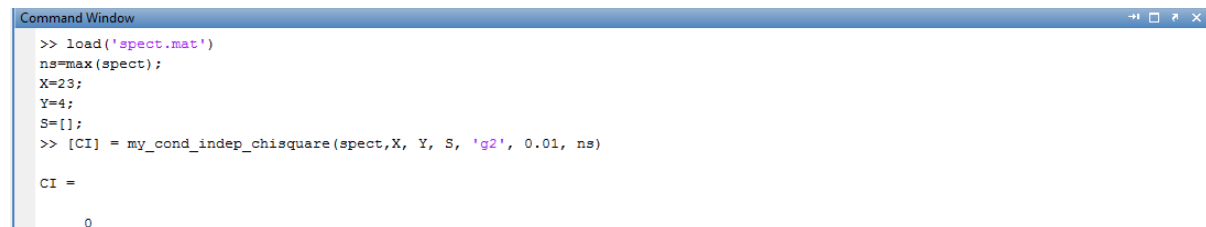
P. Spirtes, C. Glymour, and R. Scheines. (2000) Causation, Prediction, and Search, second ed. MIT Press (pp. 148- 151).

Kevin Murphy. (2001) The bayes net toolbox for MATLAB. Computing science and statistics, 33(2), 1024-1034.

Example

```
>> load('spect.mat')
>> ns=max(spect);
>> X=23;
>> Y=4;
>> S=[];
>> [CI] = my_cond_indep_chisquare(spect, X, Y, S, 'g2', 0.01, ns)
```

Results



```
Command Window
>> load('spect.mat')
ns=max(spect);
X=23;
Y=4;
S=[];
>> [CI] = my_cond_indep_chisquare(spect,X, Y, S, 'g2', 0.01, ns)

CI =

0
```

Figure 14 Example of calculating the my_cond_indep_chisquare function

3.3.2 my_cond_indep_fisher_z

Description

Calculate whether two variables are independent or not conditioned on a subset using Fisher's Z test.

Usage

```
[CI] = my_cond_indep_fisher_z(data,X,Y,S,N,alpha)
```

Arguments

Inputs	data	The data used for training (matrix)
	X	Index of feature X in data matrix
	Y	The index of variable Y in data matrix. The variable Y in data matrix should take consecutive integer values starting from 0 for classification if Y is the class attribute.
	S	Conditioning set. It includes the indices of features in data matrix.
	N	Total number of data observations
	alpha	Threshold on statistic (the significance level). The parameter is always set to 0.01 or 0.05.
Outputs	CI	Test result (1=conditional independency, 0=dependency)
	dep	Dependency value of X with respect to Y conditioned on S
	p-value	p-value at the significance level of alpha

Reference

Peña, J. M. (2008). Learning Gaussian graphical models of gene networks with false discovery rate control. In European conference on evolutionary computation, machine learning and data mining in bioinformatics (pp. 165-176). Springer Berlin Heidelberg.

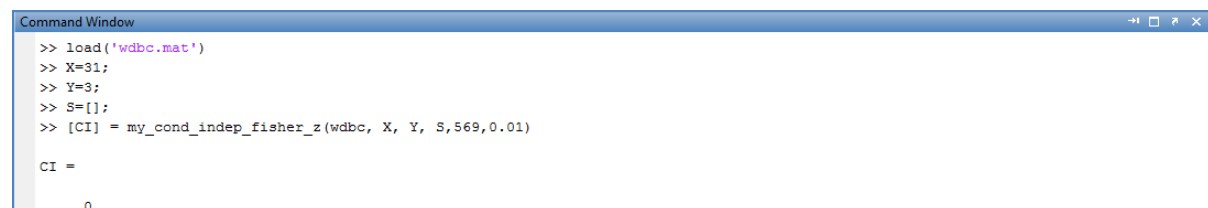
Kevin Murphy. (2001) The bayes net toolbox for MATLAB. Computing science and statistics, 33(2), 1024-1034.

Example

```
>> load('wdbc.mat')
```

```
>> X=31;  
>> Y=3;  
>> S=[];  
>> [CI] = my_cond_indep_fisher_z(wdbc, X, Y, S, 569, 0.01)
```

Results



```
Command Window  
>> load('wdbc.mat')  
>> X=31;  
>> Y=3;  
>> S=[];  
>> [CI] = my_cond_indep_fisher_z(wdbc, X, Y, S, 569, 0.01)  
  
CI =  
  
0
```

Figure 15 Example of calculating the my_cond_indep_fisher_z function

3.3.3 mi

Description

Compute mutual information between variables.

Usage

```
score=mi(X,Y)
```

Arguments

Inputs	X	A feature vector (or a column) in data matrix
	Y	A feature vector in data matrix. It is distinct from X.
Output	score	Mutual information

Reference

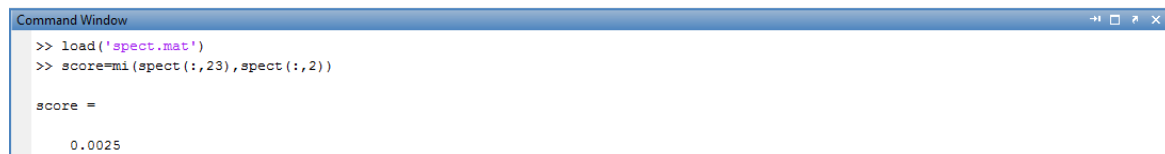
Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Lujan. (2012). Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. Journal of Machine Learning Research, 13:27–66.

Example

```
>> load('spect.mat')
```

```
>> score=mi(spect(:,23),spect(:,2))
```


Results



```
Command Window
>> load('spect.mat')
>> score=mi(spect(:,23),spect(:,2))

score =

    0.0025
```

Figure 16 Example of the mi function

3.3.4 cmi

Description

Calculate conditional mutual information.

Usage

score = cmi(X,Y,S)

Arguments

Inputs	X	A feature vector (or a column) in data matrix
	Y	A feature vector in data matrix. It is distinct from X.
	S	Conditioning set
Output	score	Conditional mutual information

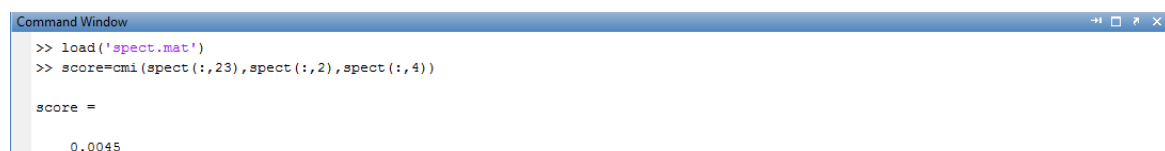
Reference

Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Lujan. (2012). Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *Journal of Machine Learning Research*, 13:27–66.

Example

```
>> load('spect.mat')
>> score=cmi(spect(:,23), spect(:,2), spect(:,4))
```

Results



```
Command Window
>> load('spect.mat')
>> score=cmi(spect(:,23),spect(:,2),spect(:,4))

score =

    0.0045
```

Figure 17 Example of the cmi function

3.3.5 SU

Description

Calculate symmetrical uncertainty between variables.

Usage

[score] = SU(firstVector,secondVector)

Arguments

Inputs	firstVector	A feature vector (or a column) in data matrix
	secondVector	A feature vector in data matrix. It is distinct from the first parameter.
Output	score	Symmetrical uncertainty

Reference

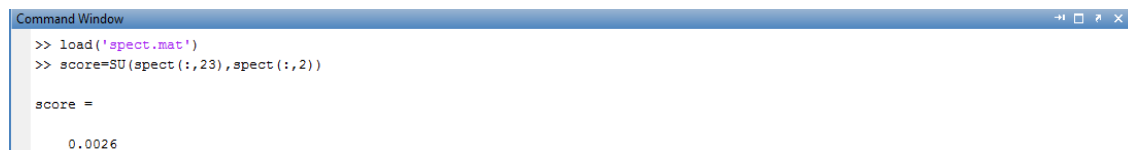
William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. 1996. Numerical recipes in C. Vol. 2. Citeseer.

Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. (2015) Scalable and accurate online feature selection for big data. arXiv:1511.09263v1 [cs.LG], 2015.

Example

```
>> load('spect.mat')
>> score=SU(spect(:,23), spect(:,2))
```

Results



```
Command Window
>> load('spect.mat')
>> score=SU(spect(:,23), spect(:,2))

score =

    0.0026
```

Figure 18 Example of the SU function

3.4 Evaluation module

Except for using the efficiency (running time), prediction accuracy, and compactness (the ratio of the number of selected features and the total number of features seen so far), in the SC module, AUC (the metric embedded in MATLAB), kappa statistic³, Friedman test, and Nemenyi test are employed to conduct comprehensively empirical comparisons.

3.4.1 perfcurve

Description

Compute AUC⁴.

Usage

[X,Y,T, AUC] = perfcurve(Labels, Scores, Posclass)

Arguments

Inputs	Labels	True test class labels
	Scores	Predicted class labels
	Posclass	The positive class label
Output	AUC	The area under curve

³ <http://au.mathworks.com/matlabcentral/fileexchange/15365-cohen-s-kappa>

⁴ More information about the function, please use “help perfcurve” in MATLAB.

Example

```
>> load('wdbc.mat')
>> testdata=wdbc(501:end,:);
>> traindata=wdbc(1:500,:);
>> class_index=31;
>> [selectedFeatures, time]=fast_osfs_z(traindata,class_index,0.01);
%use KNN classifier (k=3)
>> test_class = knnclassify(testdata(:,selectedFeatures),traindata(:,selectedFeatures),traindata(:,class_index),3);
%calculate AUC, prediction accuracy, and kappa
>> [X,Y,T,AUC] = perfcurve(testdata(:,class_index),test_class,1)
>> accuracy=length(find(testdata(:,class_index) == test_class))/length(test_class)
```

Results



```
Command Window

>> load('wdbc.mat')
>> testdata=wdbc(501:end,:);
>> traindata=wdbc(1:500,:);
>> class_index=31;
>> [selectedFeatures, time]=fast_osfs_z(traindata,class_index,0.01);
>> test_class = knnclassify(testdata(:,selectedFeatures),traindata(:,selectedFeatures),traindata(:,class_index),3);
>> [X,Y,T,AUC] = perfcurve(testdata(:,class_index),test_class,1)

X =

     0
0.0588
1.0000

Y =

     0
0.9808
1.0000

T =

     1
     1
     0

AUC =

     0.9610

>> accuracy=length(find(testdata(:,class_index) == test_class))/length(test_class)

accuracy =

     0.9710
```

Figure 19 Example of computing AUC and classification accuracy using KNN and Fast-OSFS

3.4.2 cal_kappa

Description

Compute Cohen's kappa coefficient.

Usage

Kappa = cal_kappa(Labels,PreLabels,par)

Arguments

Inputs	Labels	True (actual) class labels in test data
	Prelabels	Predicted class labels for test data
	par	The parameter is set to 'class' for classification or 'regress' for regression.
Output	kappa	Estimated Cohen's Kappa coefficient

Details

The kappa statistic is a measure of consistency among different raters, taking into account the agreement occurring by chance [Cohen 1960]. The kappa statistic is standardized to lie on a -1 to 1 scale, where 1 is perfect agreement, 0 is exactly what would be expected by chance, and negative values indicate agreement less than chance. The other values of kappa statistics and their corresponding kappa agreements are shown in the table below [Landis and Koch 1977].

Kappa statistic	<0	0.01–0.20	0.21–0.40	0.61–0.80	0.81–0.99
Kappa agreement	less than chance agreement	slight agreement	moderate agreement	substantial agreement	almost perfect agreement

Reference

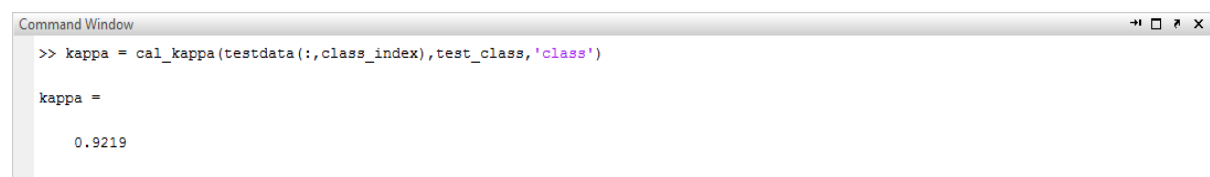
Jacob Cohen. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1): 37–46.

J. Richard Landis and Gary G. Koch. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 159–174.

Example

```
kappa = cal_kappa(testdata(:,class_index),test_class,'class')
```

Results



```
Command Window
>> kappa = cal_kappa(testdata(:,class_index),test_class,'class')

kappa =

    0.9219
```

Figure 20 Example of computing the kappa statistic

3.4.3 Friedmantest

Description

Friedman test for comparing multiple methods over multiple data sets.

Usage

```
[acceptF, rankCl]= Friedmantest(error, alpha)
```

Arguments

Inputs	error	Error rate matrix of methods
	alpha	Threshold on statistic (the significance level).
Outputs	acceptF	If the parameter equals to 0, then the null hypothesis is rejected.
	rankCl	Vectors of ranks for compared methods

Reference

Janez Demšar (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1-30.

Yifeng Li. <https://sites.google.com/site/sparsereptool/>

3.4.4 Nemenyitest

Description

Nemenyi test for comparing multiple methods over multiple data sets.

Usage

[CD]= Nemenyitest(error, alpha)

Arguments

Inputs	error	Error rate matrix of methods
	alpha	Threshold on statistic (the significance level)
Output	CD	Critical values

Details

In statistics, the Nemenyi test is a post-hoc test intended to find the groups of data that differ after a statistical test of multiple comparisons, such as the Friedman test, has rejected the null hypothesis that the performance of the comparisons on the groups of data is similar. In the package, when the null-hypothesis at the Friedman test is rejected, the Nemenyi test is proceeded as a post-hoc test. With the Nemenyi test, the performance of two methods is significantly different if the corresponding average ranks differ by at least the critical difference (how to calculate the critical difference, please see Section 3.2.2 of the reference below [Demšar, 2006]).

Reference

Janez Demšar (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1-30.

Yifeng Li. (2013) Sparse representation for high-dimensional data analysis, in "Sparse Machine Learning Models in Bioinformatics", PhD Thesis, School of Computer Science, University of Windsor, Canada, 2013. Thesis Available at <http://scholar.uwindsor.ca/etd/5023>.

Yifeng Li. <https://sites.google.com/site/sparsereptool/>