Package 'subspace'

October 12, 2015

Title Interface to OpenSubspace

Version 1.0.4

Date 2015-09-30

Description An interface to 'OpenSubspace', an open source framework for evaluation and exploration of subspace clustering algorithms in WEKA (see <http://dme.rwth-aachen.de/de/opensubspace> for more information). Also performs visualization.

Depends R(>= 3.1.0)

Imports ggvis (>= 0.4.2), colorspace (>= 1.0), stringr (>= 1.0.0), rJava(>= 0.9)

SystemRequirements Java (>= 6)

Encoding UTF-8

License GPL-2

NeedsCompilation no

Author Marwan Hassani [aut, cre],
Matthias Hansen [aut],
Emmanuel Müller [ctb],
Ira Assent [ctb],
Stephan Günnemann [ctb],
Timm Jansen [ctb],
Thomas Seidl [ctb],
University of Waikato [ctb, cph]

Maintainer Marwan Hassani <rsubspace@cs.rwth-aachen.de>

Repository CRAN

Date/Publication 2015-10-12 17:03:17

R topics documented:

CLIQUE *The CLIQUE Algorithm for Subspace Clustering*

Description

The CLIQUE Algorithm finds clusters by first dividing each dimension into xi equal-width intervals and saving those intervals where the density is greater than tau as clusters. Then each set of two dimensions is examined: If there are two intersecting intervals in these two dimensions and the density in the intersection of these intervals is greater than tau, the intersection is again saved as a cluster. This is repeated for all sets of three, four, five,... dimensions. After every step adjacent clusters are replaced by a joint cluster and in the end all of the clusters are output.

Usage

```
CLIQUE(data, xi = 10, tau = 0.2)
```

Arguments

`data`	A Matrix of input data.
`xi`	Number of Intervals.
`tau`	Density Threshold.

References

Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications*. In Proc. ACM SIGMOD, 1999.

See Also

Other subspace.clustering.algorithms: `FIRES`; `P3C`; `ProClus`; `SubClu`

Examples

```
data("subspace_dataset")
CLIQUE(subspace_dataset,xi=40,tau=0.06)
```

clustering_from_file *Reads a Clustering Object from a File*

Description

Reads a File and creates an object of class *subspace_clustering* from that.

Usage

```
clustering_from_file(file_path, index_starts_at = 0, dim = NULL)
```

Arguments

file_path	The path to the File from which the clustering should be read
index_starts_at	The index used in the file to refer to the first object in The data matrix
dim	if provided, overrides any value for dim that is found in the first line of the file

Note

Files must have the following Format: The first line contains the substring "DIM=*dim*;" where *dim* is the number of dimensions of the data set.

Each subsequent line corresponds to a cluster and contains only numbers separated by spaces. The first *dim* of these values have to be either '0' or '1' and indicate in which subspace a cluster exists. All other values in the line have to be the row numbers of the objects that the cluster contains. Row numbers in the file are assumed to be 0-indexed and are changed to 1-indexed as they are loaded into R. This can be changed with the parameter *index_starts_at*. E.g. a clustering for a three-dimensional dataset with one cluster that is in the first and third dimension and contains the first, second and 1337-th object has to be represented as:

DIM=3;
1 0 1 0 1 1336

See Also

clustering_to_file

clustering_to_file *Write a Subspace Clustering to Disk*

Description

Write a Subspace Clustering to Disk

Usage

```
clustering_to_file(clustering, file_path, index_should_start_at = 0)
```

Arguments

clustering	a subspace clustering object as generated by one of the functions from the *subspace* package
file_path	the path to the file into which the clustering should be written
index_should_start_at	the value that is used to refer to the first value in the dataset.

Note

By default, R uses the value *1* when referring to the first object in a data frame or array, while most other languages use *0*. To make working with this convention easy, clusterings written to disk are converted to this 0-indexing System. The standard parameter for the corresponding function clustering_from_file is set in such a way that files read will automatically be converted to 1-indexes, which means that you should never need to change this parameter if you work exclusively with the *subspace* package.

See Also

clustering_from_file

FIRES *The FIRES Algorithm for Subspace Clustering*

Description

The FIRES Algorithm follows a three phase framework: In a first phase, base-clusters are generated using a clustering-algorithm on each dimension in isolation. Then these base-clusters are merged in a second phase to find multidimensional cluster-approximations. These approximations are then refined in the third phase.

Usage

```
FIRES(data, base_dbscan_epsilon = 1, base_dbscan_minpts = 4,
  minimumpercent = 25, k = 1, mu = 1, minclu = 1, split = 0.66,
  post_dbscan_epsilon = 1, post_dbscan_minpts = 1)
```

Arguments

`data`	A matrix or data frame of input data.
`base_dbscan_epsilon`	parameter for the dbscan execution that generates the base clusters
`base_dbscan_minpts`	parameter for the dbscan execution that generates the base clusters
`minimumpercent`	size a base-cluster must have relative to the average size of base clusters so that it is not discarded
`k`	amount of base-clusters that every base-cluster is compared to for merging purposes
`mu`	number of most similar clusters in which two clusters must overlap in order to be considered best-merge-clusters of each other
`minclu`	number of best-merge-candidates a cluster must have to be considered a best-merge-cluster
`split`	a base-cluster is split to merge it with two different clusters iff both clusters resulting from the split have at least this size in proportion to the average size of base-clusters
`post_dbscan_epsilon`	parameter for the dbscan execution that turns the cluster-approximations into the clusters that are output at the end
`post_dbscan_minpts`	parameter for the dbscan execution that turns the cluster-approximations into the clusters that are output at the end

Details

In this implementation, the first phase consists of a run of DBSCAN with the parameters *base_dbscan_epsilon* and *base_dbscan_minpts* on the objects as they appear from each particular dimension. Then, all of these base-clusters whose size is smaller than *minimumpercent* of the average cluster size, e.g. 25 because they are not likely to contain important information for the clustering.

In the second phase, these base clusters are merged to produce subspace cluster approximations. This is achieved by computing the *k*-most-similar clusters for each base-cluster. Then the set of best-merge-candidates for each base-cluster is determined, which contains those clusters whose *k*-most-similar clusters overlap the *k*-most similar clusters of the cluster by at least *mu*. If a cluster has at least *minclu* best-merge-candidates,it is considered a best-merge cluster. Finally, every pair of best-merge-clusters that are best-merge-candidates of each other is grouped together with all of their best-merge-candidates to form the cluster approximations.

Note that some clusters need to be split and merged with two different other clusters. This is done before the merging by determining whether the intersection between a cluster and its most similar

cluster as well as the size of the cluster without this intersection are both larger than *split* times the average size of the base clusters.

In the third phase, base-clusters are repeatedly removed from cluster-approximations if their removal improves the amount of objects that are shared by all base-clusters in the approximation. Finally, to generate the definitive clusters from the cluster approximation, for each approximation all base-clusters in the approximation are combined and the a clustering algorithm is performed on these points. In this implementation, DBSCAN was chosen for this purpose and will be performed with the parameters *post_dbscan_epsilon* and *post_dbscan_minpts*.

References

Hans-Peter Kriegel, Peer Kröger, Matthias Renz and Sebastian Wurst *A Generic Framework for Efficient Subspace Clustering of High-Dimensional Data* In Proc. 5th IEEE International Conference on Data Mining, 2005

See Also

Other subspace.clustering.algorithms: `CLIQUE`; `P3C`; `ProClus`; `SubClu`

Examples

```
data("subspace_dataset")
FIRES(subspace_dataset)
```

`P3C` *The P3C Algorithm for Projected Clustering*

Description

The main idea of the P3C algorithm is to use statistical distributions for the task of finding clusters. To this end each dimension is first split into 1+log_2(nrow(data)) bins and the chi-square test is used to compute the probability that the sizes of these bins are uniformly distributed. If this probability is bigger than 1-*ChiSquareAlpha*, nothing happens. Otherwise the largest bins will be removed until this is the case. The bins that were removed in this way are then used to find clusters. To this end, bins that are adjacent are merged. Then clusters are formed by taking a bin from one dimension and determining the probability of sharing as many points as it does with each bin from another dimension. The bin is then intersected with the bin from another dimension where this probability is the lowest, given that it is at least lower than 1.00E-*PoissonThreshold* and this is repeated until no such bin is found.

Usage

```
P3C(data, ChiSquareAlpha = 0.005, PoissonThreshold = 19)
```

Arguments

data	A Matrix of input data.
ChiSquareAlpha	probability of not being uniformly distributed that the points in a dimension are allowed to have without assuming that there is a cluster visible from this dimension
PoissonThreshold	maximum probability for a bin in another dimension to deviate from the observed bin as much as it does that is allowed. The value used for this will be 1.00*10^-PoissonThreshold.

References

Gabriela Moise, Jörg Sander and Martin Ester *P3C: A Robust Projected Clustering Algorithm* In Proc. 6th IEEE International Conference on Data Mining 2006

See Also

Other subspace.clustering.algorithms: CLIQUE; FIRES; ProClus; SubClu

Examples

```
data("subspace_dataset")
P3C(subspace_dataset,PoissonThreshold=3)
```

---

plot.subspace_clustering

Plotting Subspace Clusterings

Description

Plotting for Subspace clusterings as generated by the package *subspace*.

Generates a 2d-scatterplot with interactive controls to select the dimensions that should be plotted. This visualization is created using the ggvis package and is therefore also compatible with shiny.

Usage

```
## S3 method for class 'subspace_clustering'
plot(x, data, color_by = "mix",
  standardcolors = c("#1F77B4", "#FF7F0E", "#2CA02C", "#D62728", "#9467BD",
  "#8C564B", "#E377C2", "#7F7F7F", "#BCBD22", "#17BECF", "#000000"),
  tooltip_on = "hover", ...)
```

Arguments

x	an S3-Object of type *subspace_clustering* as generated by any of the functions of the *subspace* package
data	The original data matrix on which the clustering was performed.
color_by	a parameter indicating how a point that is in multiple clusters should be colored. If "mix" is selected, the point will be colored as a mixture of the colors of both of the clusters that the point is in. If "any" is selected, a random color is selected from the colors of all the clusters that the point is in.
standardcolors	a vector of strings representing HTML-Colors that will be used to color the points by cluster assignment. Noise will be colored with the last color in the vector.
tooltip_on	decides if tooltips should be shown on "hover" or on "click"
...	this is passed on to ggvis::layer_points and can be used to change, for example the size of the points

Value

a ggvis object. If the return value is not used, a plot will be shown, but the returned plot can also be altered using ggvis

Note

When passing ellipsis parameters, the ":=" syntax from ggvis may get in your way, but you can work around this by manually creating a props object as seen in the example.

Examples

```
#Load the example dataset for this package
data("subspace_dataset")
#Load the true clustering for this dataset
path_to_clustering <- paste(path.package("subspace"),"/extdata/subspace_dataset.true",sep="")
clustering <- clustering_from_file(file_path=path_to_clustering)
#also generate a clustering with one of the algorithms
clustering2 <- CLIQUE(subspace_dataset,tau=0.2)

#now plot the generated clustering
plot(clustering2,subspace_dataset)
#plot the true clustering with small points
plot(clustering,subspace_dataset,size=0.1)

#Now plot the points with a different shape.
#This requires the workaround that was discussed in "Notes"
p <- ggvis::prop(property="shape",x="cross")
plot(clustering,subspace_dataset,props=p)
```

ProClus *The ProClus Algorithm for Projected Clustering*

Description

The ProClus algorithm works in a manner similar to K-Medoids. Initially, a set of medoids of a size that is proportional to k is chosen. Then medoids that are likely to be outliers or are part of a cluster that is better represented by another medoid are removed until k medoids are left. Clusters are then assumed to be around these medoids.

Usage

```
ProClus(data, k = 4, d = 3)
```

Arguments

data	A Matrix of input data.
k	Number of Clusters to be found.
d	Average number of dimensions in which the clusters reside

References

C. C. Aggarwal and C. Procopiuc *Fast Algorithms for Projected Clustering*. In Proc. ACM SIGMOD 1999.

See Also

Other subspace.clustering.algorithms: CLIQUE; FIRES; P3C; SubClu

Examples

```
data("subspace_dataset")
ProClus(subspace_dataset,k=12,d=2.5)
```

SubClu *The SubClu Algorithm for Subspace Clustering*

Description

The SubClu Algorithm follows a bottom-up framework, in which one-dimensional clusters are generated with DBSCAN and then each cluster is expanded one dimension at a time into a dimension that is known to have a cluster that only differs in one dimension from this cluster. This expansion is done using DBSCAN with the same parameters that were used for the original DBSCAN that produced the clusters.

Usage

```
SubClu(data, epsilon = 4, minSupport = 4)
```

Arguments

data	A Matrix of input data.
epsilon	size of environment parameter for DBSCAN
minSupport	minimum number of points parameter for DBSCAN

References

Karin Kailing, Hans-Peter Kriegel and Peer Kröger *Density-Connected Subspace Clustering for High-Dimensional Data*

See Also

Other subspace.clustering.algorithms: CLIQUE; FIRES; P3C; ProClus

Examples

```
data("subspace_dataset")
SubClu(subspace_dataset,epsilon=1,minSupport=5)
```

---

subspace *Subspace: An R-Interface to the Subspace and Projected Clustering Algorithms of the OpenSubspace package*

Description

This package provides access to five of the most popular clustering algorithms in the subspace paradigm.

Details

The algorithms CLIQUE, P3C, ProClus, SubClu and FIRES can be applied to data.frames and matrices and will return S3 objects representing clusterings. For example, using the built-in demo dataset, you can do:

```
>data("subspace_dataset")
>clustering <-P3C(subspace_dataset,PoissonThreshold=2)
>clustering
Subspace clustering generated by the package Subspace, containing 12 clusters.
```

These *subspace_clustering* objects are actually just lists of *subspace_cluster* objects, which can be accessed as follows.

```
>clustering[[1]]
Subspace cluster generated by the package Subspace. This cluster consists of 140 objects in a 3 dimens
```

Each of these clusters then holds a vector representing its subspace and a vector with the indexes of the objects the belong in this cluster. In this example, these could be accessed as `clustering[[1]]$objects` and `clustering[[1]]$subspace`.

This package also provides a `plot` method for *subspace_clustering* objects:

```
>plot(clustering,subspace_dataset)
Showing dynamic visualisation. Press Escape/Ctrl + C to stop.
```

These plots are created using the `ggvis` package.

Finally, you can save clusterings to a file using the `clustering_from_file` and `clustering_to_file` functions.

For example you could save the clustering from this example to a file and load the true clustering of the demo dataset:

```
>clustering_to_file(clustering,file_path="clustering.txt")
>path_to_clustering <- paste(path.package("subspace"),"/extdata/subspace_dataset.true",sep="")
true_clustering <- clustering_from_file(file_path=path_to_clustering)
```

References

Müller E., Günnemann S., Assent I., Seidl T.: Evaluating Clustering in Subspace Projections of High Dimensional Data http://dme.rwth-aachen.de/OpenSubspace/ In Proc. 35th International Conference on Very Large Data Bases (VLDB 2009), Lyon, France. (2009)

`subspace_dataset` *Synthetic Subspace Clustering Dataset*

Description

This is a synthetic dataset to demonstrate the subspace clustering algorithms in the package *subspace*

Index