Package 'weightedKmeans'

July 2, 2014

Version 1.2.0

Date 2012-04-27

Title Weighted KMeans Clustering

Author Graham Williams, Joshua Z Huang, Xiaojun Chen, Qiang Wang,Longfei Xiao

Maintainer Graham Williams <Graham.Williams@togaware.com>

Depends lattice, latticeExtra, clv

Description Entropy weighted kmeans (ewkm) is a weighted subspace clustering algorithm that is well suited to very high dimensional data. Weights are calculated as the importance of a variable with regard to cluster membership. The feature group weighted kmenas (fgkm) extends this concept by grouping features and weighting the group in addition to weihgting individual features.

License GPL (>= 3)

Copyright 2011-2012 Shenzhen Institutes of Advanced Technology Chinese Academy of Sciences

LazyLoad yes

LazyData yes

URL http://www.siat.ac.cn

Repository CRAN

Date/Publication 2012-04-27 06:28:29

NeedsCompilation yes

R topics documented:

ewkm *Entropy Weighted K-Means*

Description

Perform an entropy weighted subspace k-means.

Usage

```
ewkm(x, k, lambda=1, maxiter=100, delta=0.00001, maxrestart=10)
```

Arguments

x	numeric matrix of observations and variables.
k	target number of clusters.
lambda	parameter for variable weight distribution.
maxiter	maximum number of iterations.
delta	maximum change allowed between iterations for convergence.
maxrestart	maximum number of restarts. Default is 10 so that we stand a good chance of getting a full set of clusters. Normally, any empty clusters that result are removed from the result, and so we may obtain fewer than k clusters if we don't allow restarts (i.e., maxrestart=0). If < 0 then there is no limit on the number of restarts and we are much more likely to get a full set of k clusters.

Details

The entopy weighted k-means clustering algorithm is a subspace clusterer ideal for high dimensional data. Along with each cluster we also obtain variable weights that provide a relative measure of the importance of each variable to that cluster.

The algorithm is based on the k-means approach to clustering. An initial set of k means are identified as the starting centroids. Observartions are clustered to the nearest centroid according to a distance measure. This defines the initial clustrering. New centroids are then identified based on these clusters.

Weights are then calculated for each variable within each cluster, based on the current clustering. The weights are a measure of the relative importance of each variable with regard to the membership of the observations to that cluster. These weights are then incorporated into the distance function, typically reducing the distance for the more important variables.

New centroids are then calculated, and using the weighted distance measure each observation is once again clustered to its nearest centroid.

The process continues until convergence (using a measure of dispersion and stopping when the change becomes less than delta) or until a specified number of iterations has been reached (maxiter).

Large lambda (e,g, > 3) lead to a relatively even distribution of weights across the variables. Small lambda (e.g., < 1) lead to a more uneven distribution of weights, giving more discrimintation between features. Recommended values are between 1 and 3.

Always check the number of iterations, the number of restarts, and the total number of iterations as they give a good indication of whether the algorithm converged.

As with any distance based algorithm, be sure to rescale your numeric data so that large values do not bias the clustering. A quick rescaling method to use is `scale`.

Value

Returns an object of class "kmeans" and "ewkm", compatible with other functions that work with kmeans objects, such as the 'print' method. The object is a list with the following components in addition to the components of the kmeans object:

weights: A matrix of weights recording the relative importance of each variable for each cluster.

iterations: This reports on the number of iterations before termination. Check this to see whether the maxiters was reached. If so then the algroithm may not be converging,and thus the resulting clustering may not be particularly good.

restarts: The number of times the clustering restarted because of a disappearing cluster resulting from one or more k-means having no observations associated with it. An number here greater than 0 indicates that the algorithm is not converging on a clustering for the given k. It is recommended that k be reduced.

total.iterations: The total number of iterations over all restarts.

Author(s)

Qiang Wang, Xiaojun Chen, Graham J Williams, Joshua Z Huang

References

Jing Liping, M. K. Ng, J. Z. Huang, An Entropy Weighting k-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data, IEEE Transactions on Knowledge and Data Engineering, 19 (8), Aug 2007 pp 1026–1041.

See Also

`plot.ewkm`.

Examples

```
myewkm <- ewkm(iris[1:4], k=3, lambda=0.5, maxiter=100)

plot(iris[1:4], col=myewkm$cluster)

# For comparative testing

mykm <- kmeans(iris[1:4], 3)
```

```
plot(iris[1:4], col=mykm$cluster)
```

`fgkm` *Feature Group Weighting K-Means for Subspace clustering*

Description

Perform an feature group weighting subspace k-means.

Usage

```
fgkm(x, k, strGroup, lambda, eta, maxiter=100, delta=0.000001, maxrestart=10)
```

Arguments

`x`	numeric matrix of observations and features.
`k`	target number of clusters.
`strGroup`	a string give the group information, formated as "0-9:10-19:20-49"
`lambda`	parameter of feature weight distribution.
`eta`	parameter of group weight distribution.
`delta`	maximum change allowed between iterations for convergence.
`maxiter`	maximum number of iterations.
`maxrestart`	maximum number of restarts. Default is 10 so that we stand a good chance of getting a full set of clusters. Normally, any empty clusters that result are removed from the result, and so we may obtain fewer than k clusters if we don't allow restarts(i.e., maxrestart=0). If < 0, then there is no limit on the number of restarts and we are much likely to get a full set of k clusters.

Details

The feature group weighting k-means clustering algorithm is a extension to ewkm, which itself is a soft subspace clustering method.

The algorithm weights subspaces in both feature groups and individual features.

Always check the number of iterations, the number of restarts, and the total number of iterations as they give a good indication of whether the algorithm converged.

As with any distance based algorithm, be sure to rescale your numeric data so that large values do not bias the clustering. A quick rescaling method to use is `scale`.

Value

Return an object of class "kmeans" and "fgkm", compatible with other function that work with kmeans objects, such as the 'print' method. The object is a list with the following components in addition to the components of the kmeans object:

`cluster`	A vector of integer (from 1:k) indicating the cluster to which each point is allocated.
`centers`	A matrix of cluster centers.
`featureWeight`	A matrix of weights recording the relative importance of each feature for each cluster.
`groupWeight`	A matrix of group weights recording the relative importance of each feature goup for each cluster.
`iterations`	This report on the number of iterations before termination. Check this to see whether the maxiters was reached. If so then teh algorithm may not be converging, and thus the resulting clustering may not be particularly good.
`restarts`	The number of times the clustering restarted because of a disappearing cluster resulting from one or more k-means having no observations associated with it. An number here greater than zero indicates that the algorithm is not converging on a clustering for the given k. It is recommeded that k be reduced.
`totalIterations`	The total number of iterations over all restarts.
`totolCost`	The total cost calculated in the cost function.

Author(s)

Longfei Xiao <lf.xiao@siat.ac.cn>

References

X. Chen, et al., A feature group weighting method for subspace clustering of high-dimensional data, Pattern Recognition(2011), doi:10.1016/j.patcog.2011.06.004

See Also

kmeans ewkm

Examples

```
# The data fgkm.sample has 600 objects and 50 dimensions.
# Scale the data before clustering
x <- scale(fgkm.sample)

# Group information is formated as below.
# Each feature is separated by ':'.
strGroup <- "0-9:10-19:20-49"

# Use the fgkm algorithm.
```

```
myfgkm <- fgkm(x, 3, strGroup, 3, 1)

# You can print the clustering result now.
myfgkm$cluster
myfgkm$featureWeight
myfgkm$groupWeight
myfgkm$iterations
myfgkm$restarts
myfgkm$totiters
myfgkm$totss

# Use a cluster validation method from package 'clv'.

# real.cluster is the real class label of the data 'fgkm.sample'.
real.cluster <- rep(1:3, each=200)

# std.ext() returns four values SS, SD, DS, DD.
std <- std.ext(as.integer(myfgkm$cluster), real.cluster)

# Rand index
clv.Rand(std)

# Jaccard coefficient
clv.Jaccard(std)
```

`fgkm.sample` *Sample dataset to illustrate the fgkm algorithm.*

Description

A sample dataset of 50 variables and 600 observations.

Usage

```
fgkm.sample
```

Format

The `fgkm.sample` dataset is a data frame with 600 observations and 50 variables.

See Also

`fgkm`.

plot.ewkm *Plot Entropy Weighted K-Means Weights*

Description

Plot a heatmap showing the variable weights from the subspace clustering.

Usage

```
## S3 method for class 'ewkm'
plot(x, ...)
## S3 method for class 'ewkm'
levelplot(x, ...)
```

Arguments

x	an object of class ewkm.
...	arguments passed on through to heatmap.

Details

The entopy weighted k-means clustering algorithm is a subspace clusterer ideal for high dimensional data. Along with each cluster we also obtain variable weights that provide a relative measure of the importance of each variable to that cluster.

This plot visualises these relative measures of variable importance for each of the clusters using a heatmap. The top dendrogram highlights the relationship between the clusters and the right side dendrogram provides a visual clue to the correlation between the variables.

The plot.ewkm() function uses heatmap() to display the weights. The levelplot.ewkm() uses levelplot() with dendrogramGlobs from the lattice package. Note that plot() will immediately draw the plot while levelplot() does not draw immediately but returns a result object which must be plot()ed.

Author(s)

Graham J Williams

Examples

```
myewkm <- ewkm(iris[1:4], k=3, lambda=0.5, maxiter=100)

plot(myewkm)
```

Index