Data Mining Algorithms In R

Contents

Articles

References

Article Licenses

Data Mining Algorithms In R

In general terms, Data Mining comprises techniques and algorithms, for determining interesting patterns from large datasets. There are currently hundreds (or even more) algorithms that perform tasks such as frequent pattern mining, clustering, and classification, among others. Understanding how these algorithms work and how to use them effectively is a continuous challenge faced by data mining analysts, researchers, and practitioners, in particular because the algorithm behavior and patterns it provides may change significantly as a function of its parameters. In practice, most of the data mining literature is too abstract regarding the actual use of the algorithms and parameter tuning is usually a frustrating task. On the other hand, there is a large number of implementations available, such as those in the R project, but their documentation focus mainly on implementation details without providing a good discussion about parameter-related trade-offs associated with each of them.

This Wikibook aims to fill this gap by integrating three pieces of information for each technique: description and rationale, implementation details, and use cases. The description and rationale of each technique provide the necessary background for understanding the implementation and applying it to real scenarios. The implementation details not only expose the algorithm design, but also explain its parameters, in the light of the rationale provided previously. Finally, the use cases provide an experience of the algorithms use on synthetic and real datasets.

The choice of the R project as the computational platform associated with this Wikibook stems from its popularity (and thus critical mass), ease of programming, good performance, and an increasing use in several fields, such as bioinformatics and finances, among others.

If you want to learn how to program in the R language, read the book R Programming.

Contents

External links

- R Reference Card for Data Mining [1]
- R Data Mining [2]
- All basics of R [3]
- Online course in Data Mining in R [4]
- Computing for Data Analysis (Free online course) [5]

References

[1] http://www.rdatamining.com
[2] http://rdatamining.wordpress.com
[3] http://www.cran.r-project.org
[4] http://www.statistics.com/data-mining-r/
[5] https://class.coursera.org/compdata-003

Dimensionality Reduction

1. Principal Component Analysis
2. Singular Value Decomposition
3. Feature Selection

Frequent Pattern Mining

Contents

1. The Eclat Algorithm
2. arulesNBMiner
3. The Apriori Algorithm
4. The FP-Growth Algorithm

Sequence Mining

1. SPADE
2. DEGSeq

Clustering

1. K-Means
2. Hybrid Hierarchical Clustering
3. Expectation Maximization (EM)
4. Dissimilarity Matrix Calculation
5. Hierarchical Clustering
6. Bayesian Hierarchical Clustering
7. Density-Based Clustering
8. K-Cores
9. Fuzzy Clustering - Fuzzy C-means
10. RockCluster
11. Biclust
12. Partitioning Around Medoids (PAM)
13. CLUES
14. Self-Organizing Maps (SOM)
15. Proximus
16. CLARA

Classification

1. SVM
2. penalizedSVM
3. kNN
4. Outliers
5. Decision Trees
6. Naïve Bayes
7. adaboost
8. JRip

R Packages

In this section, we will discuss about R packages that are related to datamining.

Contents

1. RWeka
2. gausspred
3. optimsimplex
4. CCMtools
5. FactoMineR
6. nnet

Principal Component Analysis

Introduction

This chapter presents the Principal Component Analysis (PCA) technique as well as its use in R project for statistical computing. First we will introduce the technique and its algorithm, second we will show how PCA was implemented in the R language and how to use it. Finally, we will present an example of an application of the technique in a data mining scenario. In the end of the chapter you will find references for further information.

Principal Component Analysis

PCA is a dimensionality reduction method in which a covariance analysis between factors takes place. The original data is remapped into a new coordinate system based on the variance within the data. PCA applies a mathematical procedure for transforming a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

PCA is useful when there is data on a large number of variables, and (possibly) there is some redundancy in those variables. In this case, redundancy means that some of the variables are correlated with one another. And because of this redundancy, PCA can be used to reduce the observed variables into a smaller number of principal components that will account for most of the variance in the observed variables.

PCA is recommended as an exploratory tool to uncover unknown trends in the data. The technique has found application in fields such as face recognition and image compression, and is a common technique for finding patterns in data of high dimension.

Algorithm

The PCA algorithm consists of 5 main steps:

1. Subtract the mean: subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension. This produces a data set whose mean is zero.
2. Calculate the covariance matrix:

$C^{n \times n} = (c_{i,j}, c_{i,j} = cov(Dim_i, Dim_j))$

where $C^{n \times n}$ is a matrix which each entry is the result of calculating the covariance between two separate dimensions.

3. Calculate the eigenvectors and eigenvalues of the covariance matrix.
4. Choose components and form a feature vector: once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest. So that the components are sorted in order of significance. The number of eigenvectors that you choose will be the number of dimensions of the new data set. The objective of this step is construct a feature vector (matrix of vectors). From the list of eigenvectors take the eigenvectors selected and form a matrix with them in the columns:

FeatureVector = (eig_1, eig_2, ..., eig_n)

5. Derive the new data set. Take the transpose of the FeatureVector and multiply it on the left of the original data set, transposed:

FinalData = RowFeatureVector x RowDataAdjusted

where RowFeatureVector is the matrix with the eigenvectors in the columns transposed (the eigenvectors are now in the rows and the most significant are in the top) and RowDataAdjusted is the mean-adjusted data transposed (the data items are in each column, with each row holding a separate dimension).

Implementation

We choose princomp method from stats package for this tutorial.

- R package: **stats**
- Method: **princomp**
- Documentation: princomp [1]

princomp is a generic method with "**formula**" and "**default**" methods from stats package which performs a principal components analysis on the given numeric data matrix and returns the results as an object of class **princomp**.

Usage

```
princomp(x, ...)
```

- S3 method for class 'formula:

```
princomp(formula, data = NULL, subset, na.action, ...)
```

- Default S3 method:

```
princomp(x, cor = FALSE, scores = TRUE, covmat = NULL, subset =
rep(TRUE, nrow(as.matrix(x))), ...)
```

- S3 method for class 'princomp':

```
predict(object, newdata, ...)
```

Arguments

- **formula**

a formula with no response variable, referring only to numeric variables.

- **data**

an optional data frame containing the variables in the formula **formula**. By default the variables are taken from **environment(formula)**

- **subset**

an optional vector used to select rows (observations) of the data matrix x.

- **na.action**

a function which indicates what should happen when the data contain NAs. The default is set by the **na.action** setting of options, and is **na.fail** if that is unset. The 'factory-fresh' default is **na.omit**.

- **x**

a numeric matrix or data frame which provides the data for the principal components analysis.

- **cor**

a logical value indicating whether the calculation should use the correlation matrix or the covariance matrix. (The correlation matrix can only be used if there are no constant variables.)

- **scores**

a logical value indicating whether the score on each principal component should be calculated.

- **covmat**

a covariance matrix, or a covariance list as returned by **cov.wt** (and **cov.mve** or **cov.mcd** from package MASS). If supplied, this is used rather than the covariance matrix of x.

- **...**

arguments passed to or from other methods. If x is a formula one might specify **cor** or **scores**.

- **object**

Object of class inheriting from "princomp"

- **newdata**

An optional data frame or matrix in which to look for variables with which to predict. If omitted, the scores are used. If the original fit used a formula or a data frame or a matrix with column names, newdata must contain columns with the same names. Otherwise it must contain the same number of columns, to be used in the same order.

Value

The **princomp** method returns a list with class "**princomp**" containing the following components:

- **sdev**

a formula with no response variable, referring only to numeric variables.

- **loadings**

the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).

- **center**

the means that were subtracted.

- **scale**

the scalings applied to each variable.

- **n.obs**

the number of observations.

- **scores**

if scores = TRUE, the scores of the supplied data on the principal components. These are non-null only if x was supplied, and if covmat was also supplied if it was a covariance list. For the formula method, napredict() is applied to handle the treatment of values omitted by the na.action.

- **call**

the matched call.

- **na.action**

if relevant.

Visualization

The print method for these objects prints the results in a nice format and the plot method produces a screen plot. There is also a biplot method.

Examples

```
require(graphics)

summary(pc.cr <- princomp(USArrests, cor = TRUE))
loadings(pc.cr)
plot(pc.cr)
biplot(pc.cr)
```

Case Study

To illustrate the PCA technique for dimensionality reduction, a simple case study will be shown.

Scenario

In the field of information retrieval (IR), queries and documents can be represented in a vector space. Generally, several features are used to describe a document retrieved by a query such as TF-IDF and PageRank measures. Occasionally, it can be necessary to visualize documents in a 2-dimensional space. To do this, PCA could be used.

Dataset

More recently, Microsoft released the LETOR benchmark data sets for research on LEarning TO Rank, which contains standard features, relevance judgments, data partitioning, evaluation tools, and several baselines. LETOR contains several datasets for ranking settings derived from the two query sets and the Gov2 web page collection. The 5-fold cross validation strategy is adopted and the 5-fold partitions are included in the package. In each fold, there are three subsets for learning: training set, validation set and testing set. The datasets can be downloaded from LETOR site [2].

A typical document in a LETOR dataset is described as follows:

```
0 qid:1 1:1.000000 2:1.000000 3:0.833333 4:0.871264 5:0 6:0 7:0 8:0.941842 9:1.000000 10:1.000000
11:1.000000 12:1.000000 13:1.000000 14:1.000000 15:1.000000 16:1.000000 17:1.000000 18:0.719697 19:0.729351 20:0
21:0 22:0 23:0.811565 24:1.000000 25:0.972730 26:1.000000 27:1.000000 28:0.922374 29:0.946654 30:0.938888
31:1.000000 32:1.000000 33:0.711276 34:0.722202 35:0 36:0 37:0 38:0.798002 39:1.000000 40:1.000000
41:1.000000 42:1.000000 43:0.959134 44:0.963919 45:0.971425 #docid = 244338
```

Execution

LETOR files can not be imported 'as is', but using the following script, it is possible to convert LETOR format to other accepted by R, as can be seen below:

```
cat train.txt | grep "qid:1 " | gawk '{ printf("doc_"$50); for
(i=3;i<=47;i++) { split($i,a,":"); printf(", "a[2]); } printf("\n");
 }' > letor.data
```

After that, 'letor.data' will be seen as follows:

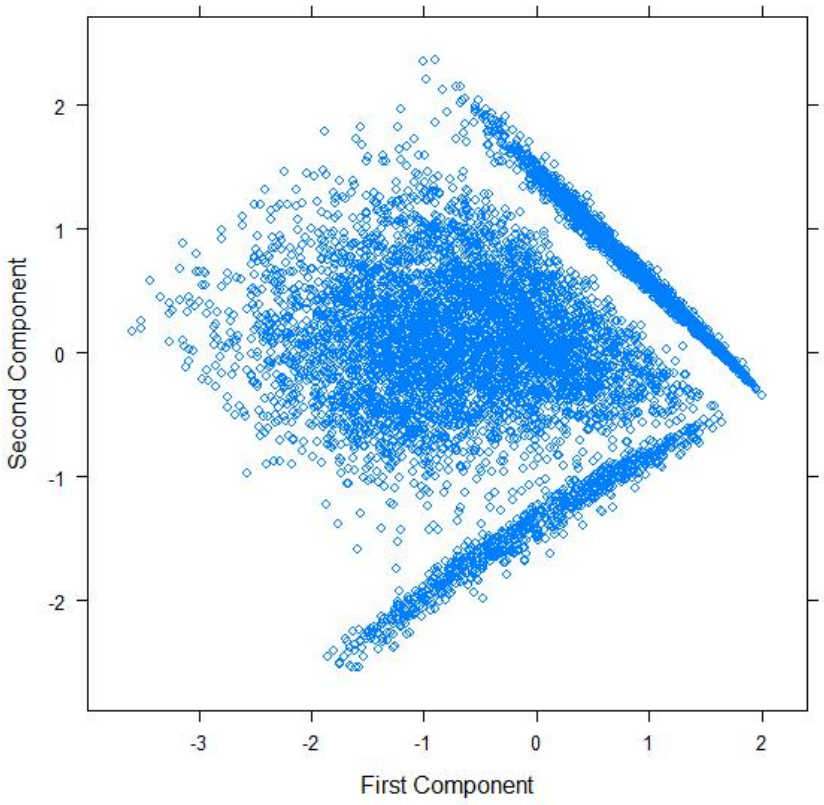
```
doc_244338, 1.000000, 1.000000, 0.833333, 0.871264, 0, 0, 0, 0.941842, 1.000000, 1.000000,
1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 0.719697, 0.729351,
0, 0, 0, 0.811565, 1.000000, 0.972730, 1.000000, 1.000000, 0.922374, 0.946654, 0.938888,
1.000000, 1.000000, 0.711276, 0.722202, 0, 0, 0, 0.798002, 1.000000, 1.000000, 1.000000,
1.000000, 0.959134, 0.963919, 0.971425
```

Now, it is possible to load 'letor.data' file to R and run PCA to plot documents in a 2-dimensional space, as follows:

```
data=read.table("letor.data",sep=",")
summary(pc.cr <- princomp(data[,2:46]))
loadings(pc.cr)

library(lattice)
pc.cr$scores
pca.plot <- xyplot(pc.cr$scores[,2] ~ pc.cr$scores[,1])
pca.plot$xlab <- "First Component"
pca.plot$ylab <- "Second Component"
pca.plot
```

Results

The script above generates the following chart:



Analysis

As we can see, despite documents being represented by multiple features (>45), PCA was able to find 2 **principal components** that were used to plot all documents in a 2-dimensional chart.

References

1. ^ Mardia, K. V., J. T. Kent and J. M. Bibby (1979). "Multivariate Analysis", London: Academic Press.
2. ^ Venables, W. N. and B. D. Ripley (2002). "Modern Applied Statistics with S", Springer-Verlag.
3. Principal Components Analysis and Redundancy Analysis Lab, Montana State University. Link [3].
4. Visualising and exploring multivariate datasets using singular value decomposition and self organising maps from Bioinformatics Zen. Link [4].

References

[1] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/princomp.html
[2] http://research.microsoft.com/en-us/um/beijing/projects/letor/
[3] http://ecology.msu.montana.edu/labdsv/R/labs/lab7/lab7.html
[4] http://www.bioinformaticszen.com/2007/07/exploring-multivariate-data-using-svd-and-som/

Singular Value Decomposition

In this chapter we will take a look at Singular Value Decomposition (SVD), a matrix's factorization method that uses the knowledge of [[w:Linear AlgebraLinear Algebra in order to make such decompositions.

Singular Value Decomposition

In data mining, this algorithm can be used to better understand a database by showing the number of important dimensions and also to simplify it, by reducing of the number of attributes that are used in a data mining process. This reduction removes unnecessary data that are linearly dependent in the point of view of Linear Algebra. For example, imagine a database which contains a field that stores the water's temperature on several samples and another that stores its state (solid,liquid or gas). Its easy to see that the second field is dependent from the first and, therefore, SVD could easily show us that it is not important for the analysis.

Principal Component Analysis (PCA) is a specific case of SVD.

Algorithm

SVD is the factorization of a matrix X of real or complex numbers, that has n rows and d columns, into:

$X = UAV^\top$

where U is a matrix whose dimensions are $n \times n$, V is another matrix whose dimensions are $d \times d$, and A is a matrix whose dimensions are $n \times d$, the same dimensions as X.

Besides:

$U^\top U = I_n$ and $V^\top V = I_d$

where I_n and I_d are Identity matrix whose size are respectively n and d.

The columns of U are the **left singular vectors** of the matrix X, and the columns of V (or the rows of $V^\top$) are the **right singular vectors**.

The matrix A is a diagonal matrix, whose diagonal values are the **singular values** of the matrix X. The singular value in a row of A is never less than the value of a row below. All singular values are greater than 0.

To compute the SVD is to find the eigenvalues and the eigenvectors of $XX^\top$ and $X^\top X$. The eigenvectors of $X^\top X$ are the columns of V and the eigenvectors of $XX^\top$ are the columns of U. The singular values of X, in the diagonal of matrix A, are the square root of the common positive eigenvalues of $XX^\top$ and $X^\top X$.

If $XX^\top$ and $X^\top X$ have he same number of eigenvalues, X is a square matrix; else, the eigenvalues of the matrix that has less eigenvalues are eigenvalues of the matrix that has more eigenvalues. Therefore, the singular values of X are the eigenvalues of the matrix, between $XX^\top$ and $X^\top X$, that has less eigenvalues.

The number of singular values of a matrix is the rank of that matrix, that is the number of linearly independent columns or rows of a matrix. The rank is not greater than $\min(n, d)$, because this is the number of elements of the diagonal of the matrix. The singular values are elements of the diagonal of the matrix A. The number of positive singular values equals the rank of the matrix.

Therefore, the algorithm is:

1. Compute $XX^\top$ normally.
2. Compute the eigenvalues and the eigenvectors of $XX^\top$ normally.
3. Compute $X^\top X$.
4. Compute the eigenvalues and the eigenvectors of $X^\top X$.
5. Compute the square root of the common positive eigenvalues of $XX^\top$ and $X^\top X$.
6. Finally, assign the computed values to U, V and A.

Example

$$X = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 1 & 1 & 2 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix}$$

$$XX^T = \begin{bmatrix} 6 & 2 & 4 \\ 2 & 6 & 4 \\ 4 & 4 & 6 \end{bmatrix}$$

The eigenvalues of XX^T are:

12.744563, 4.000000, and 1.255437

The eigenvectors of XX^T are:

1. 0.5417743, 0.5417743, 0.6426206
2. 0.7071068, -0.7071068, -2.144010 x 10^{-16}
3. 0.4544013, 0.4544013, -0.7661846

$$X^TX = \begin{bmatrix} 6 & 2 & 4 & 4 \\ 2 & 2 & 2 & 2 \\ 4 & 2 & 5 & 1 \\ 4 & 2 & 1 & 5 \end{bmatrix}$$

The eigenvalues of X^TX are:

12.744563, 4.000000, 1.255437, and (5.940557 x 10^{-18})

The eigenvectors of X^TX are:

1. 0.6635353, 0.3035190, 0.4835272, 0.4835272
2. 0.0000000, 0.5181041 x 10^{-16}, -0.7071068, 0.7071068
3. -0.5565257, 0.8110957, 0.1272850, 0.1272850
4. 0.5000000, 0.5000000, -0.5000000, -0.5000000

The singular values of X are the square root of the common eigenvalues of X^TX and XX^T:

$$\sqrt{12.744563} = 3.569953$$
$$\sqrt{4.000000} = 2.000000$$
$$\sqrt{1.255437} = 1.120463$$

Therefore:

$$A = \begin{bmatrix} 3.569953 & 0 & 0 & 0 \\ 0 & 2.000000 & 0 & 0 \\ 0 & 0 & 1.120463 & 0 \end{bmatrix}$$

Finally, X is decomposed:

$$X = UAV^T =$$

$$\begin{bmatrix} 0.5417743 & 0.7071068 & 0.4544013 \\ 0.5417743 & -0.7071068 & 0.4544013 \\ 0.6426206 & -2.144010 \cdot 10^{-16} & -0.7661846 \end{bmatrix} \begin{bmatrix} 3.569953 & 0 & 0 & 0 \\ 0 & 2.000000 & 0 & 0 \\ 0 & 0 & 1.120463 & 0 \end{bmatrix} \begin{bmatrix} 0.6635353 & 0.000000 & -0.5565257 & 0.5 \\ 0.3035190 & 5.181041 \cdot 10^{-16} & 0.8110957 & 0.5 \\ 0.4835272 & -0.7071068 & 0.1272850 & -0.5 \\ 0.4835272 & 0.7071068 & 0.1272850 & -0.5 \end{bmatrix}$$

Implementation

R has a built in function which calculates SVD, called 'svd()'.

It, by default, receives a R's native matrix as argument and returns an frame, that contains U, A and V.

Arguments

If it is not necessary that all singular values and vectors are computed, one can tell svd() the exact number of needed elements.

This can be achieved by assigning these values to **nu** and **nv** which represent the number of left and right singular vectors needed, respectively.

For example, in order to calculate only half of these vectors, one could do:

```
svd(X, nu = min(nrow(X), ncol(X)) / 2, nv = min(nrow(X), ncol(X))/2)
```

Returned object

```
s = svd(X)
```

Considering:

$X = UAV^T$

The returned object is a data structure that contains three fields:

s$d is the vector that contains the singular values of X, that was got from the diagonal of matrix A.

s$u is the matrix whose columns contain the left singular vectors of X. Its number of rows is the same number of rows of X and its number of columns is the number passed to the parameter nu. Note that if nu is 0, this matrix is not created.

s$v is the matrix whose columns contain the right singular vectors of X. Its number of rows is the same number of columns of X and its number of columns is the number passed to the parameter nv. Note that if nv is 0, this matrix is not created.

Examples

Execute the following command sequences in the R terminal or as a R program and see the results:

Example 1:

```
dat <- seq(1,240,2)

X <- matrix(dat,ncol=12)

s <- svd(X)

A <- diag(s$d)

s$u %*% A %*% t(s$v) #  X = U A V'
```

Example 2:

```
dat <- seq(1,240,2)

X <- matrix(dat,ncol=12)
```

```
s <- svd(X, nu = nrow(X), nv = ncol(X))

A <- diag(s$d)

A <- cbind(A, 0)  # Add two columns with zero, in order to A have the same dimensions of X.

A <- cbind(A, 0)

s$u %*% A %*% t(s$v) #  X = U A V'
```

Case Study

Scenario

In order to better visualize the results of SVD, in ths case study we will work with a matrix that represents an image, so any change on the matrix can be easily observed.

Input Data

To work with an image on R, one should install the package **rimage**:

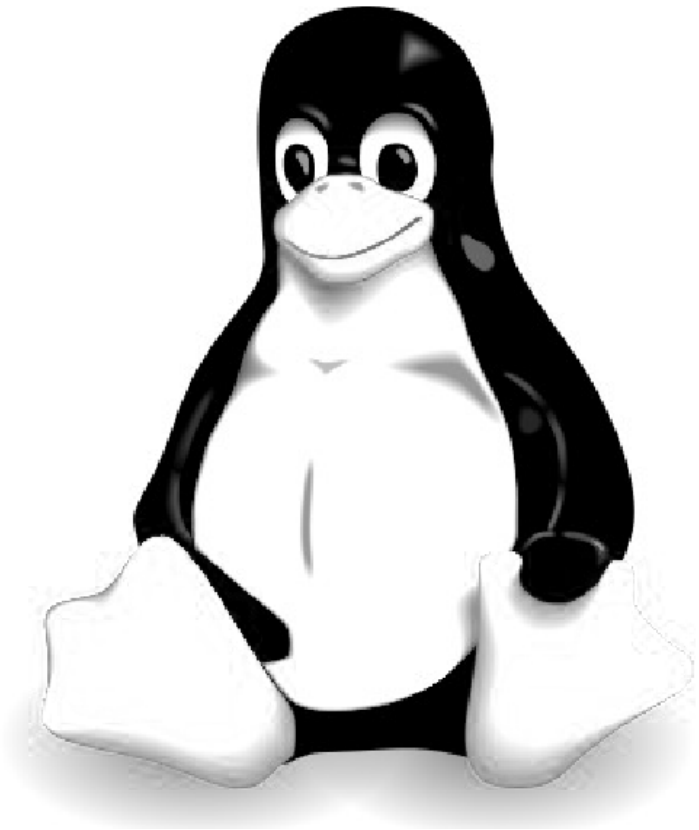
```
> install.packages("rimage")
```

Let's then load the image into R, converting it to a greyscale one. This way, we will end up with an binary matrix, where 1 means white, and 0 black.

```
library(rimage)
tux <- read.jpeg('tux.jpeg')
tux <- imagematrix(tux,type='grey')
```

We can see the result of this import:

```
plot(tux)
```



In order to help us with this dimension reduction, lets make a little help function, which will receive our **tux** and the numbers of dimension we want and return our new **tux**.

```
reduce <- function(A,dim) {
    #Calculates the SVD
    sing <- svd(A)

    #Approximate each result of SVD with the given dimension
    u<-as.matrix(sing$u[, 1:dim])
    v<-as.matrix(sing$v[, 1:dim])
    d<-as.matrix(diag(sing$d)[1:dim, 1:dim])

    #Create the new approximated matrix
    return(imagematrix(u%*%d%*%t(v),type='grey'))
}
```

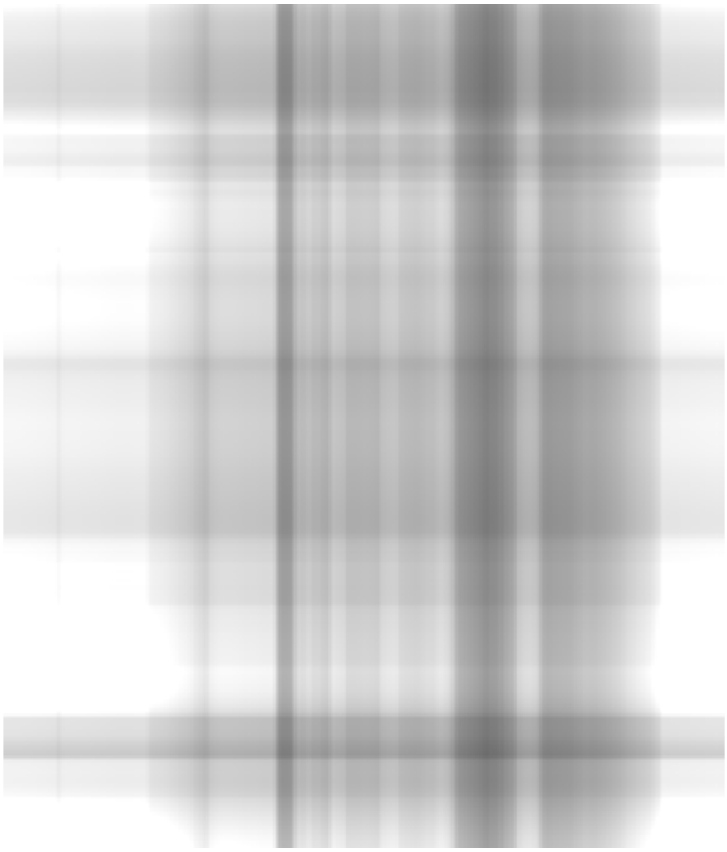
Execution and output

Now that we have our matrix, lets se how many singular values it has:

```
tux_d <- svd(tux)
length(tux_d$d)
[1] 335
```

So we have 335 singular values on this matrix. Lets first try to reduce it to only one singular value:

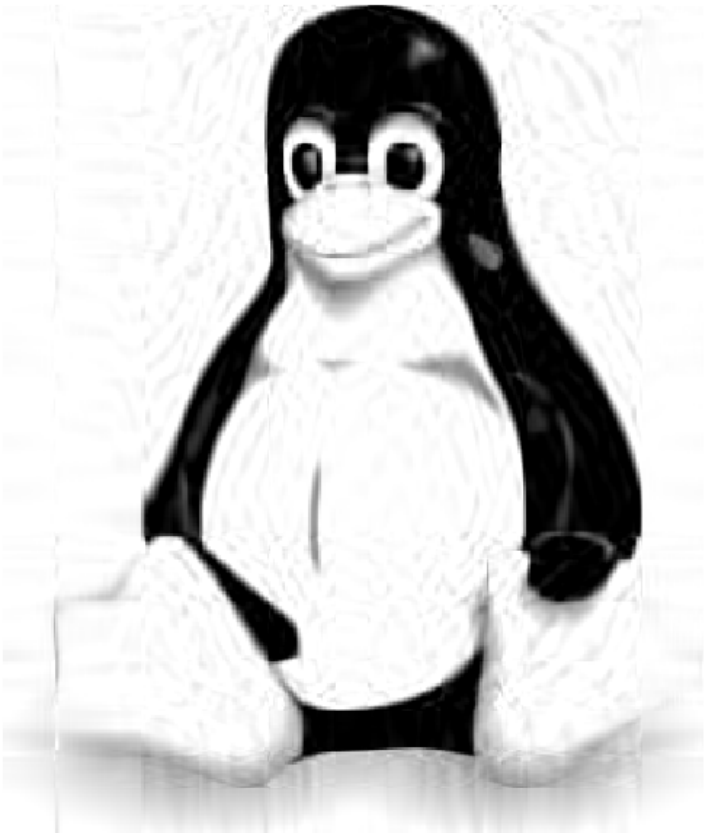
```
plot(reduce(tux,1))
```



As we can see, this approximation removes a lot of useful information from our matrix. If it was a database, we surely would have lost important data.

Lets try with 10% (35) singular values:

```
plot(reduce(tux,35))
```



Analysis

With only 10% of the real data we are able to create a very good approximation of the real data. Moreover, with this method we can remove noises and linear dependent elements by using only the most important singular values. This is very useful on data mining, since is hard to identify if a database is **clear** and, if not, which elements are useful or not to our analysis.

References

- Fundamentals of Data Mining Algorithms - Mohammed J. Zaki, Wagner Meira Jr. - Section 7.4
- Singular value decomposition
- http://www.ats.ucla.edu/stat/r/code/svd_demos.htm
- http://stat.ethz.ch/R-manual/R-patched/library/base/html/svd.html

Feature Selection

Feature Selection in R with the FSelector Package

Introduction

In Data Mining, Feature Selection is the task where we intend to reduce the dataset dimension by analyzing and understanding the impact of its features on a model. Consider for example a predictive model $C_1A_1 + C_2A_2 + C_3A_3 = S$, where C_i are constants, A_i are features and S is the predictor output. It is interesting to understand how important are the used features (A_1, A_2 and A_3) and what are their relevance to the model and their correlation with S. Such analysis allow us to select a subset of the original features, reducing the dimension and complexity of future steps on the Data Mining process. During a subset selection, we try to identify and remove as much of the irrelevant and redundant information as possible.

Techniques for Feature Selection can be divided in two approaches: **feature ranking** and **subset selection**. In the first approach, features are ranked by some criteria and then features above a defined threshold are selected. In the second approach, one searches a space of feature subsets for the optimal subset. Moreover, the second apprach can be split in three parts:

1. **Filter approaches:** you select the features first, then you use this subset to execute a classification algorithm;
2. **Embedded approaches** the feature selection occurs as part a classification algorithm;
3. **Wrapper approaches** an algorithm for classification is applied over the dataset in order to identify the best features.

The FSelector Package for R offers algorithms for filtering attributes (e.g. cfs, chi-squared, information gain, linear correlation) and algorithms for wrapping classifiers and search attribute subset space (e.g. best-first search, back-ward search, forward search, hill climbing search). The package also makes it possible to choose subsets of features based on attributes' weights by performing different ways of cutoff.

The FSelector Package was created by Piotr Romanski and this article is based on the version 0.16, released in April 11, 2009.

The Feature Ranking Approach

As we explained, in the ranking approach, features are ranked by some criteria and those which are above a defined threshold are selected. A general algorithm can be considered for such approach where you just need to decide which one if the best ranking criteria to be used. In the F-Selector Package, such criteria is represented by a set of functions that calculate weights to your features according to a model. All of this elements will be explained in this text.

Feature Ranking Algorithm

The general algorithm for the Feature Ranking Approach is:

```
for each feature F_i

   wf_i = getFeatureWeight(F_i)

   add wf_i to weight_list

sort weight_list

choose top-k features
```

We can translate such algorithm idea to R language by these commands:

```
#load a dataset and use it as the main source of data

library(mlbench)

data(HouseVotes84)

#calculate weights for each atribute using some function

weights <- SOME_FUNCTION(Class~., HouseVotes84)

print(weights)

#select a subset of 5 features with the lowest weight

subset <- cutoff.k(weights, 5)

#print the results

f <- as.simple.formula(subset, "Class")

print(f)
```

On the first part of the code above, we use the function *library* to load the package *mlbench* which contains a collection of artificial and real-world machine learning benchmark problems, including, e.g., several data sets from the UCI repository.(http://cran.r-project.org/web/packages/mlbench/index.html).After that, we use the *mlbench* dataset *HouseVotes84* (United States Congressional Voting Records 1984) as the working data source for the later steps. Instead of using the *HouseVotes84*, you can also define your own dataset and provide it as the input for the algorithm.

The *HouseVotes84* data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA contains 16 variables, and consider ninve different types of votes represented by three classes: yea (voted for, paired for, announced for), nay (voted against, paired against, announced against) and unknown (voted present, voted present to avoid conflict of interest, did not vote or otherwise make a position known).

On the second part of the code above, we calculate weights for each attribute using some function. This function must be selected by the user and they are listed later on this text. The output of those functions will be a list of features and its weights according to the chosen technique, and that will be available in the *weights* array. You can print the weights like we do using the command *print*.

On the third part of the code, we define a cut-off of the top-5 features of the *weights* array. By using the function *cutoff.k*, we inform the array name and the *k* value, that is 5 in our case. The result will be stored in another array, called *subset*, cointaining the 5 features with the highest rank weight.

On the fourth part of the code, you can print the final model as a simple formula, composed by the features subset present in the *subset* array, and the predictable feature names *Class*.

Available Feature Ranking Techniques in FSelector Package

All the listed techniques below can be used to generate rank weights for features. The first parameter, *formula*, is a symbolic description of a model (e.g. Class~. for a model where all the features are used do predict the feature *Class*). The second parameter, *data*, is the target data to be considered in the model.

Chi-squared Filter

Usage:

```
chi.squared(formula, data)
```

Correlation Filter

Usage for Pearson's correlation:

```
linear.correlation(formula, data)
```

Usage for Spearman's correlation:

```
rank.correlation(formula, data)
```

Entropy-Based Filter

Usage for Information Gain:

```
information.gain(formula, data)
```

Usage for Gain Ratio:

```
gain.ratio(formula, data)
```

Usage for Symmetrical Uncertainty:

```
symmetrical.uncertainty(formula, data)
```

OneR Algorithm

Usage:

```
oneR(formula, data)
```

Random Forest Filter

Usage:

```
random.forest.importance(formula, data, importance.type = 1)
```

Where the *importance.type* parameter specify the type of importance measure (1=mean decrease in accuracy, 2=mean decrease in node impurity).

The Feature Subset Selection Approach

In the feature subset selection approach, one searches a space of feature subsets for the optimal subset. Such approach is present on the FSelector package by wrappers techniques (e.g. best-first search, back-ward search, forward search, hill climbing search). Those techniques works by informing a function that takes a subset and generate an evaluation value for that subset. A search is performed in the subsets space until the best solution can be found.

Feature Subset Selection Algorithm

The general algorithm for the Feature Subset Selection approach is:

```
S = all subsets

for each subset s in S

   evaluate(s)

return (the best subset)
```

We can translate the algorithm idea in R by using these commands:

```
#load a dataset and use it as the main source of data

library(mlbench)

data(HouseVotes84)

#define the evaluation function
evaluator <- function(subset) {

  #here you must define a function that returns a double value to evaluate the given subset

  #consider high values for good evaluation and low values for bad evaluation.

  return(something)

}
```

```
#perform the best subset search

subset <- MY_FUNCTION(data, evaluator)
```

```
#prints the result

f <- as.simple.formula(subset, "Class")

print(f)
```

As in the first example on this text, we use again the HouseVotes84 dataset from the mlbench library. We must define a evaluation function that wil do some calculus over a given subset and return a high value for a good subset. Later, all you have to do is choose a search strategy and you can also print the selection.

Feature Subset Search Techniques Available in FSelector Package

The FSelector Package offers some functions to search for the best subset selection. In most of them, the *attributes* parameters is a character vector of all attributes to search (the set of features that will be parted in subsets during the search), and the *eval.fun* parameter is a function taking as first parameter a character vector of all attributes and returning a numeric indicating how important a given subset is. The CFS and the Consistency techniques encapsulate such process by using the best first approach, so, you just have to inform the symbolic model and the data, like in the ranking approach.

Best First Search

Usage:

```
best.first.search(attributes, eval.fun)
```

Exhaustive Search

Usage:

```
exhaustive.search(attributes, eval.fun)
```

Greedy Search

Usage for forward greedy search:

```
forward.search(attributes, eval.fun)
```

Usage for backward greedy search:

```
backward.search(attributes, eval.fun)
```

Hill Climbing Search

Usage:

```
hill.climbing.search(attributes, eval.fun)
```

CFS Filter

Usage:

```
cfs(formula, data)
```

Consistency Based Filter

Usage:

```
consistency(formula, data)
```

The Eclat Algorithm

The Eclat Algorithm

The Eclat algorithm is used to perform itemset mining. Itemset mining let us find frequent patterns in data like if a consumer buys milk, he also buys bread. This type of pattern is called association rules and is used in many application domains.

The basic idea for the eclat algorithm is use tidset intersections to compute the support of a candidate itemset avoiding the generation of subsets that does not exist in the prefix tree.

Algorithm

The Eclat algorithm is defined recursively. The initial call uses all the single items with their tidsets. In each recursive call, the function IntersectTidsets verifies each itemset-tidset pair $\langle X, t(X) \rangle$ with all the others pairs $\langle Y, t(Y) \rangle$ to generate new candidates N_{XY}. If the new candidate is frequent, it is added to the set P_X. Then, recursively, it finds all the frequent itemsets in the X branch. The algorithm searches in a DFS manner to find all the frequent sets.

Implementation

The eclat algorithm can be found in the arule package of R system.

```
   * R package: arules
   * Method: eclat
   * Documentation : arules package [1]
```

Usage

```
eclat(data, parameter = NULL, control = NULL)
```

Arguments

data

object of class transactions or any data structure which can be coerced into transactions (e.g., binary matrix, data.frame).

parameter

object of class ECparameter or named list (default values are: support 0.1 and maxlen 5)

control

object of class ECcontrol or named list for algorithmic controls.

Value

Returns an object of class itemsets

Example

```
data("Adult")

## Mine itemsets with minimum support of 0.1.
itemsets <- eclat(Adult, parameter = list(supp = 0.1, maxlen = 15))
```

Visualization

The arules package implements some visualization methods for itemsets, which are the return type for the eclat algorithm. Here are some examples:

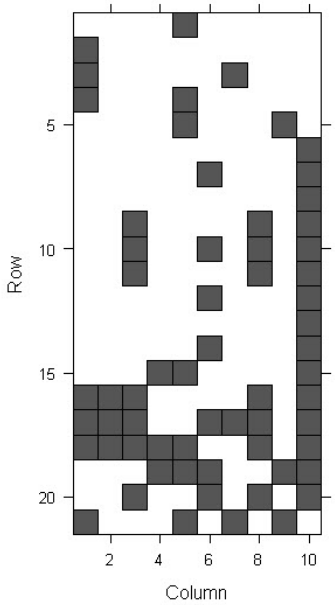
Example 1

```
data("Adult")

## Mine frequent itemsets with Eclat.
fsets <- eclat(Adult, parameter = list(supp = 0.5))

## Display the 5 itemsets with the highest support.
fsets.top5 <- SORT(fsets)1:5?
inspect(fsets.top5)

## Get the itemsets as a list
as(items(fsets.top5), "list")

## Get the itemsets as a binary matrix
as(items(fsets.top5), "matrix")

## Get the itemsets as a sparse matrix, a ngCMatrix from package Matrix.
## Warning: for efficiency reasons, the ngCMatrix you get is transposed
as(items(fsets.top5), "ngCMatrix")
```

Example 2

```
## Create transaction data set.
data <- list(
  c("a","b","c"),
  c("a","b"),
  c("a","b","d"),
  c("b","e"),
  c("b","c","e"),
  c("a","d","e"),
  c("a","c"),
  c("a","b","d"),
```

```
  c("c","e"),
  c("a","b","d","e")
)

t <- as(data, "transactions")

## Mine itemsets with tidLists.
f <- eclat(data, parameter = list(support = 0, tidLists = TRUE))

## Get dimensions of the tidLists.
dim(tidLists(f))

## Coerce tidLists to list.
as(tidLists(f), "list")

## Inspect visually.
image(tidLists(f))

##Show the Frequent itemsets and respectives supports
inspect(f)
```



Result of example 2

#	items	support
1	{b, c, e}	0.1
2	{a, b, c}	0.1
3	{a, c}	0.2
4	{b, c}	0.2
5	{c, e}	0.2
6	{a, b,d, e}	0.1
7	{a, d, e}	0.2
8	{b, d, e}	0.1
9	{a, b, d}	0.3
10	{a, d}	0.4
11	{b, d}	0.3
12	{d, e}	0.2
13	{a, b, e}	0.1
14	{a, e}	0.2
15	{b, e}	0.3
16	{a, b}	0.5
17	{a}	0.7
18	{b}	0.7
19	{e}	0.5
20	{d}	0.4
21	{c}	0.4

Use Case

To see some real example of the use of the Eclat algorithm it will be used some data from the northwind [2] database. The northwind database is freely available for download and represents data from an enterprise. In this example it will be used the table *order details* from the database. The order details table is used to relate the orders with products (in a n to n relationship). The Eclat algorithm will be used to find frequent patterns from this data to see if there are any products that are bought together.

Scenario

Given the data from the order details table from the northwind database, find all the frequent itemsets with support = 0.1 and length of at least 2.

Input Data

The order details table has the fields:

ID

: primary key

Order ID

: foreign key from table Orders

Product ID

foreign key from table Products

Quantity

the quantity bought

Discount

the discount offered

Unit Price

the unit price of the product

To use the data, some pre-processing is necessary. The table may have many rows that belongs to the same order, so the table was converted in a way that all the rows for one order became only one row in the new table containing the product id's of the products belonging to that order. The fields ID, order id, quantity, discount and unit price was discarded. The data was saved in a txt file called *northwind-orders.txt*. The file was scripted in a way ready to be loaded as a list object in R.

Implementation

To run the example the package arules need to be loaded in R.

First, the data is loaded in a list object in R

```
## 1041 transactions is loaded, the listing below was shortened
## some duplicates transactions was introduced to produce some results
for the algorithm
data = list(

c("2","3","4","6","7","8","10","12","13","14","16","20","23","32","39","41","46","52","55","60","64","66","73","75","77"),
 c("11","42","72"),
 c("14","51"),
 c("41","51","65"),
 c("22","57","65"),
 ...)
```

Second, the eclat algorithm is used.

```
itemsets <- eclat(data, parameter = list(support = 0.1, minlen=2, tidLists = TRUE, target="frequent itemsets"))
```

```
parameter specification:
 tidLists support minlen maxlen            target   ext
     TRUE     0.1      2      5 frequent itemsets FALSE

algorithmic control:
 sparse sort verbose
      7   -2    TRUE

eclat - find frequent item sets with the eclat algorithm
version 2.6 (2004.08.16)          (c) 2002-2004   Christian Borgelt
create itemset ...
set transactions ...[78 item(s), 1041 transaction(s)] done [0.00s].
sorting and recoding items ... [3 item(s)] done [0.00s].
creating bit matrix ... [3 row(s), 1041 column(s)] done [0.00s].
writing  ... [4 set(s)] done [0.00s].
```

```
Creating S4 object  ... done [0.00s].
```

Output Data

The itemsets object holds the output of the execution of the eclat algorithm. As can be seen above, 4 sets was generated. To see the results it can be used:

```
inspect(itemsets)
```

```
  items   support
1 {11,
   42,
   72}  0.1940442
2 {42,
   72}  0.1940442
3 {11,
   42}  0.1940442
4 {11,
   72}  0.1959654
```

Analysis

As can be seen above, there are 4 frequent itemsets as result of the eclat algorithm. This output was induced by the replication of the transaction {11, 42, 72} many times in the data. This result shows that the tuples {11,42,72},{42,72} and {11,42} has a support of 19,40%; and the tuple {11,72} has a support of 19,60%.

The product id's 11, 42 and 72 represents the products *Queso Cabrales*, *Singaporean Hokkien Fried Mee* and *Mozzarella di Giovanni*, respectively. So, the output of the eclat algorithm suggests a strong frequent shop pattern of buying this items together.

Variations

PPV, PrePost, and FIN Algorithm

These three algorithms were propsed by Deng et al , and are based on three novel data structures called Node-list , N-list , and Nodeset respectively for facilitating the mining process of frequent itemsets. They are sets of nodes in a FP-tree with each node encoding with pre-order traversal and post-order traversal. Compared with Node-lists, N-lists and Nodesets are more efficient. This causes the efficiency of PrePost and FIN is higher than that of PPV . See for more details.

References

- Hahsler, M.; Buchta, C.; Gruen, B. & Hornik, K. **arules: Mining Association Rules and Frequent Itemsets**. 2009.
- Hahsler, M.; Gruen, B. & Hornik, K. **arules -- A Computational Environment for Mining Association Rules and Frequent Item Sets**. Journal of Statistical Software, 2005, 14, 1-25
- Zaki, M. J.;Meira Jr., W. **Fundamentals of Data Mining Algorithms**. Cambridge University Press, 2009
- Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. **New algorithms for fast discovery of association rules**. Technical Report 651, Computer Science Department, University of Rochester, Rochester, NY 14627. 1997.
- Christian Borgelt (2003) **Efficient Implementations of Apriori and Eclat**. Workshop of Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL, USA).

- Deng, Z. & Wang, Z. A New Fast Vertical Method for Mining Frequent Patterns [3]. International Journal of Computational Intelligence Systems, 3(6): 733 - 744, 2010.
- Deng, Z.; Wang, Z. & Jiang, J. A New Algorithm for Fast Mining Frequent Itemsets Using N-Lists [4]. SCIENCE CHINA Information Sciences, 55 (9): 2008 - 2030, 2012.
- Deng, Z. & Lv, S. Fast mining frequent itemsets using Nodesets [5]. Expert Systems with Applications, 41(10): 4505–4512, 2014.

References

[1] http://r-forge.r-project.org/projects/arules/
[2] http://www.microsoft.com/downloads/details.aspx?FamilyID=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en
[3] http://www.tandfonline.com/doi/abs/10.1080/18756891.2010.9727736
[4] http://info.scichina.com:8084/sciFe/EN/abstract/abstract508369.shtml
[5] http://www.sciencedirect.com/science/article/pii/S0957417414000463

arulesNBMiner

Introduction

The technique to be discussed in this chapter is used in frequent itemset mining. There are several situations which people are interested on co-occurrence of two or more item of a set. It is important to establish which items co-occur, since based on them, association rules can be extracted between itemsets [1]. A typical example of an application is about a supermarket when one discovery customers who buy meat and beer also tend to buy coal. Thus, a frequent item set would be meat-beer-coal and an association rule would be customers, in general, who buy meat and beer, have more chances to buy coal.

Many works have dealt with the problem of frequent itemset mining. Most of them show the necessity of a min_support threshold, which is an itemset minimum frequency in the data and in general defined by the miner user. Besides, these studies have as goal to mine a complete set of frequent itemsets that satisfy min_support [2]. The application of a minimum support results in several assumptions which are rarely discussed or verified. One assumptions is items occur in the database following a, possibly unknown, but stable process and that the items occur in the database with roughly similar frequencies. Nevertheless, in the real world, transactions data have a frequency distribution highly skewed with almost items occurring in an infrequent way while just some of them occur with high frequency. In database where this phenomenon happens, interesting patterns are not found since some of the associated items are too infrequent to satisfy the user-specified minimum support [3].

Some algorithms such as TFP were developed in a way that a user do not need determine a min_support, however, he needs to inform the minimum size of itemset (min_l) and the number of itemset which he desires mining (k). Furthermore, the TFP algorithms just mine frequent closed itemset.[2]. Again, the user has a parameter (min_l) which he should specify before mining the data, what is a subtle decision.

Therefore, this chapter presents an algorithm, which is implemented at R-package and uses a simple stochastic model (Negative Binomal model or NB-model) to estimate a minimum support utilizing knowledge of the process which generates transaction data and allows for highly skewed frequency distributions. The name of package in R program is arulesNBMiner that is the Java implementation of a depth first search algorithm to mine NB-frequent itemsets of NB-precise rules [4]. Beside the algorithm utilize the information contained in own data structure to estimate the minimum support, it uses a precision limit to estimate min_support and for each k-itemset plus 1 extension it calculates a different minimum support.

Technique to be discussed

Algorithm

Function NBSelect $\left(l,\ L,\ n,\ \tilde{k},\ \tilde{a}',\ \pi\right)$

1. r_{max} = *max*(c.count) in L
2. $r_{rescale}$ = *sum*(c.count) in L
3. **for** each tuple [c,c.count] ∈ L **do** n_{obs} [c.count]++
4. **do**
5. $$\text{precision} = \frac{(1-|l|)P[R_1 \geq p \mid k=\tilde{k},\ a=r_{\text{rescale}}\cdot\tilde{a}']}{\sum_{r=p}^{r_{\max}} n_{ob}s[r]}$$
6. **while** (precision ≥ π ∧ (p--) > 0)
7. **return** {c| [c,c.count] ∈ L ∧ c.count > p}

where

l = the pattern for which candidate items are selected

L = a data structure which holds count information for items which co-occur with pattern l; we use tuples c, c.count , where c represents a candidate item and c.count the count.

n = the total number of available items

$\tilde{k}$ and $\tilde{a}'$ = estimated parameters for the data set

π = user-specified precision threshold

Implementation (description of used modules and packages)

The first thing one has to do to use NBMiner is installing the NBMiner package. This package has three dependencies packages: arules (http://cran.r-project.org/web/packages/arules/index.html) which provides the infrastructure for representing, manipulating and analyzing transaction data and patterns (frequent itemsets and association rules), Matrix (http://cran.r-project.org/web/packages/Matrix/index.html) which is classes and methods for dense and sparse matrices and operations on them using Lapack and SuiteSparse, and lattice (http://cran.r-project.org/web/packages/lattice/index.html) which is a framework for data visualization developed at the Bell Labs. Besides, the user has to install on computer the Sun Java(TM) Development Kit (JDK) 6, since the NBMiner uses a package called rJava which is part of JDK. This must be installed in the user operational system.

Installation of the dependencies packages can be performed within the R environment using the function "install.packages(“*package name*”)". The name of NBMiner package is arulesNBMiner.

```
install.packages(“arulesNBMiner”)
```

After the user installed the necessaries packages, he must load them. This can be done using the function "library(*package name*)".

```
library(arulesNBMiner)
```

However, before we show how to use the NBMiner package, it is necessary to show how to load the data. Here, we are using a data on csv format. In this format each line is a transaction like the example below:

Employment	Education	Marital	Occupation	Sex	Accounts
Private	College	Separated	Service	Female	USA
Private	Associate	Unmarried	Transport	Male	Jamaica
Private	HSgrad	Divorced	Clerical	Male	USA
Private	Bachelor	Civil	Repair	Male	USA
Private	College	Civil	Executive	Male	USA
Private	HSgrad	Civil	Service	Male	USA

Using the function "read.table()" on R environment, we are able to load the data. This function creates a "data.frame" object. Its syntax is:

```
object name<-read.table("filename", header=TRUE/FALSE,sep=",")
```

The argument header indicates the data has or does not a header while the sep indicates which character is being used to separate the elements. Later, the user has to transform this data frame object in a transaction object. For that, he can use the function "as()". Its syntax is:

```
object name<-as(data frame object,"transactions")
```

It is important to mention the user has to load the NBMiner package before he uses the function as with parameter transaction. Furthermore, there is another data format that some R functions can read and transform it in a transaction object. The user can utilize a binary format like the example below:

1	1	0	0	0	1	0	0	0
1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0
1	0	0	0	1	0	0	0	1
1	1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1
1	0	0	0	1	0	0	0	1

Here, each line represents a transaction and each column represents a term of the database. The number 1 means presence of that term in that transaction, while 0 means no presence. To read this data, user can first utilize the function "scan()". It creates a vector object, and its syntax is:

```
object name<-scan("file name",sep=",")
```

After that, it is necessary to transform the vector object in a matrix object. This is performed using the function "dim()". Its syntax is:

```
dim(vector object)<-c(dim1,dim2)
```

Dim1 and *Dim2* are the dimensions of the matrix which the user want creating. If the user wants, he can name each row and column of the matrix using the function "dimnames()". Its syntax is:

```
dimnames(matrix object)<-list(pridim=c(names),segdim=c(names))
```

Where *pridim* is a vector with first dimension names, and *segdim* is a vector with second dimension names. Finally, this object can be transformed in transaction object using the function "as()". Its syntax is:

```
object name<-as(matrix object,"ItemMatrix")
```

Soon after the user loaded the package and the data, it is necessary to create the parameters object using the function of NBMiner called "NBMinerParameters()". This function reads the data and extracts from them some parameters which will be used in the next step. The syntax of this function is:

```
object name<-NBMinerParameters(data object, pi=#, theta=#, maxlen=#, minlen=#, trim=#, verb=TRUE/FALSE, plot=TRUE/FALSE, rules=TRUE/FALSE)

where:

# = numbers

data object = the data as an object of class transaction

pi = precision threshold

theta = pruning parameter

maxlen = maximal number of items in found itemsets (default = 5)

minlen = minimum number of items in found itemsets (default = 1)

trim = fraction of incidences to trim off the tail of the frequency distribution of the data

verbose = use verbose output for the estimation procedure

plot = plot the model

rules = mine NB-precise rules instead of NB-frequent itemsets
```

After the user perform this procedure, he is able to run the algorithm to mine his data. For that, one will use the function called "**NBMiner()**". Its syntax is:

```
object name<-NBMiner(data object, parameter=object parameter, control = list(verb=TRUE, debug=FALSE))
```

If the user uses the option rules=TRUE when he created the object parameter, the algorithm will mine the NB-precise rules. Otherwise, the same algorithm will mine the NB-frequent itemsets. Finally, we summarized the necessaries commands to use the NBMiner package for mining tasks using the first data structure mentioned above, these are below:

```
library(package name)

object name<-read.table("filename", header=TRUE/FALSE,sep=",")

object name<-as(data frame object,"transaction")

object name<-NBMinerParameters(data object, pi=#, theta=#, maxlen=#, minlen=#, trim=#, verb=TRUE/FALSE, plot=TRUE/FALSE, rules=TRUE/FALSE)

object name<-NBMiner(data object, parameter=object parameter, control = list(verb=TRUE, debug=FALSE
```

Visualization

To visualize data information contained in a "data.frame" object just use the object name. Here, we use table as object name:

```
table = read.table("data",sep=",",header=TRUE)
table
```

#	Employment	Education	Marital	Occupation	Sex	Accounts
1	Private	College	Separated	Service	Female	USA
2	Private	Associate	Unmarried	Transport	Male	Jamaica
3	Private	HSgrad	Divorced	Clerical	Male	USA
4	Private	Bachelor	Civil	Repair	Male	USA
5	Private	College	Civil	Executive	Male	USA
6	Private	Hsgrad	Civil	Service	USA	

However, when the same principle is used in “transactions” format, we have just the number of transactions and items generated by the "as()" function:

```
tableT = as(table,"transactions")
tableT
 transactions in sparse format with
 6 transactions (rows) and
 18 items (columns)
```

Use “inspect()” function to visualize the result of the conversion to the “transactions” format:

```
inspect(tableT)
```

#	items	transactionID
1	{Employment=Private,Education=College,Marital=Separated,Occupation=Service,Sex=Female,Accounts=USA}	1
2	{Employment=Private,Education=Associate,Marital=Unmarried,Occupation=Transport,Sex=Male,Accounts=Jamaica}	2
(...)		

To summarize the data information in "transaction" format use "summary()".

```
summary(tableT)
transactions as itemMatrix in sparse format with
6 rows (elements/itemsets/transactions) and
18 columns (items) and a density of 0.3333333
most frequent items:
Employment=Private    Sex=Male    Accounts=USA
        6                5            5
Marital=Civil    Education=College  (Other)
        3                2            15
element (itemset/transaction) length distribution:
sizes
6
6
  Min.    1st Qu.  Median  Mean   3rd Qu.   Max.
     6       6       6       6       6       6
includes extended item information - examples:
       labels                  variables          levels
1        Employment=Private       Employment         Private
2       Education=Associate        Education       Associate
```

```
3        Education=Bachelor        Education        Bachelor
includes extended transaction information - examples:
transaction            ID
1                        1
2                        2
3                        3
```

To see labels of the items generated in the conversion to transaction format use “itemInfo()”:

```
itemInfo(tableT)
      labels              variables            levels
1      Employment=Private       Employment            Private
2          Education=Associate        Education                Associate
3           Education=Bachelor        Education                 Bachelor
4      Education=College        Education                College
5      Education=HSgrad        Education             HSgrad
6      Marital=Civil          Marital                  Civil
7      Marital=Divorced       Marital                 Divorced
8      Marital=Separated          Marital                Separated
9      Marital=Unmarried          Marital                Unmarried
10         Occupation=Clerical       Occupation                Clerical
11        Occupation=Executive       Occupation               Executive
12      Occupation=Repair       Occupation             Repair
13          Occupation=Service       Occupation            Service
14      Occupation=Transport       Occupation               Transport
15      Sex=Female             Sex                   Female
16      Sex=Male               Sex                    Male
17      Accounts=Jamaica         Accounts                Jamaica
18      Accounts=USA               Accounts             USA
```

Use “labels()” to see only the labels of the itens and transactions:

```
labels(tableT)
$items
[1]      "Employment=Private"      "Education=Associate"           "Education=Bachelor"
[4]      "Education=College"     "Education=HSgrad"           "Marital=Civil"
[7]      "Marital=Divorced"      "Marital=Separated"             "Marital=Unmarried"
[10]      "Occupation=Clerical"   "Occupation=Executive"        "Occupation=Repair"
[13]      "Occupation=Service"    "Occupation=Transport"        "Sex=Female"
[16]      "Sex=Male"              "Accounts=Jamaica"           "Accounts=USA"
$transactionID
[1] "1" "2" "3" "4" "5" "6"
```

To see NB mined results use the same commands above mentioned to “transactions” data:

```
paramA <- NBMinerParameters(tableT, trim = 0.01, pi = 0.999, theta = 0.8, rules = TRUE, plot = FALSE, verbose = FALSE, minlen=3, maxlen=5)

tableNB <- NBMiner(tableT, parameter = paramA, control = list(verb = FALSE, debug=FALSE))

inspect(head(tableNB))

     lhs                             rhs                         precision

1     {Education=HSgrad, Sex=Male}            => {Employment=Private}         0.9991467
```

```
2      {Sex=Male, Accounts=USA}             => {Marital=Civil}            0.9999421

3      {Sex=Male, Accounts=USA}             => {Education=HSgrad}            0.9977636

4      {Education=College, Accounts=USA}       => {Employment=Private}            0.9982934

5      {Marital=Civil, Accounts=USA}             => {Sex=Male}                    0.9999752

6      {Employment=Private, Sex=Male}            => {Accounts=USA}           0.9999948
```

Where "lhs" is the antecedent of the rules and de "rhs" is the consequent of the rules.

We can also show the data distribution in the space with “image()” function:

```
image(tableNB)
```

Case study

Scenario

It contains artificial data created by Graham Williams (developer of Rattle) and is supplied with Rattle. To quote from the Rattle documentation: "It consists of 2,000 fictional clients who have been audited, perhaps for compliance with regard to the amount of a tax refund that is being claimed. For each case an outcome is recorded (whether the taxpayer's claims had to be adjusted or not) and any amount of adjustment that resulted is also recorded."

Input data

Available on http://cs.anu.edu.au/Student/comp3420/mining/audit.csv.

Execution

```
table = read.table("audit.csv",sep=",",header=TRUE)

trans = as(table,"transactions")

paramA = NBMinerParameters(tableT, trim = 0.01, pi = 0.999, theta = 0.8, rules = TRUE, plot = FALSE, verbose = FALSE, minlen=3, maxlen=5)

transNB = NBMiner(tableT, parameter = paramA, control = list(verb = FALSE, debug=FALSE))
```

Output

```
transNB
set of 18158 rules
```

```
tableNB = as(trans,"data.frame")
write.table(tableNB,file="auditNB.csv",sep=",")
```

Analysis

rules	consequent	precedent A	precedent B	precedent C	precedent D
92	Accounts=China	Employment=PSState	Education=Master	Occupation=Professional	
4721	Accounts=China	Employment=PSState	Education=Master		
4762	Accounts=China	Employment=PSState	Occupation=Professional		
4857	Accounts=China	Education=Master	Marital=Civil	Occupation=Professional	Sex=Male
5871	Accounts=China	Education=Master	Occupation=Professional		
6131	Accounts=China	Marital=Civil	Occupation=Professional		
10269	Accounts=China	Education=Master	Occupation=Professional	Sex=Male	
10386	Accounts=China	Education=Master	Marital=Civil	Occupation=Professional	
14791	Accounts=China	Marital=Civil	Occupation=Professional	Sex=Male	

Based on the data above we can note that when the Marital state is “Civil” and the "Occupation" is "Professional" we have an "Chinese" account.

References

- [1]Mohammed j. Zaki and Wagner Meira Jr. Fundamentals of Data Mining Algorithms. Chaper 11 – Itemset Mining, 74-93
- [2]Jianyong Wang, Jiawei Han, Ying Lu and Petre Tzvetkov. TFP: An efficient algorithm for mining Top-K frequent closed itemsets. IEEE Transactions on Knowledge and Data Engineering, 17(5):652-665, May 2005
- [3]Michael Hahsler. A model-based frequency constraint for mining associations from transaction data. Data Mining and Knowledge Discovery, 13(2):137-166, September 2006.
- [4]http://cran.fiocruz.br/web/packages/arulesNBMiner/index.html

The Apriori Algorithm

Introduction

In computer science and data mining, **Apriori** is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions. As is common in association rule mining, given a set of **itemsets**, the algorithm attempts to find subsets which are common to at least a minimum number C of the itemsets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k-1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k -length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|}-1$ of its proper subsets.

The quest to mine frequent patterns appears in many domains. The prototypical application is market basket analysis, i.e., to mine the sets of items that are frequent bought together, at a supermarket by analyzing the customer shopping carts (the so-called “market baskets”). Once we mine the frequent sets, they allow us to extract association rules among the item sets, where we make some statement about how likely are two sets of items to co-occur or to conditionally occur. For example, in the weblog scenario frequent sets allow us to extract rules like, “Users who visit the sets of pages main, laptops and rebates also visit the pages shopping-cart and checkout”, indicating, perhaps, that the special rebate offer is resulting in more laptop sales. In the case of market baskets, we can find rules like, “Customers who buy Milk and Cereal also tend to buy Bananas”, which may prompt a grocery store to co-locate bananas in the cereal aisle.

Algorithm

The following is a formal statement of the problem: Let $\tau = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq \tau$. Associated with each transaction is a unique identifier, called its TID. We say that a transaction T contains X, a set of some items in τ, if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \tau$, $Y \subset \tau$, and $X \cap Y = \varnothing$. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called minsup) and minimum confidence (called minconf) respectively.

The problem is usually decomposed into two subproblems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence. Suppose one of the large itemsets is Lk, Lk = {I1, I2, … , Ik}, association rules with this itemsets are generated in the following way: the first rule is {I1, I2, … , Ik-1}⇒ {Ik}, by checking the confidence this rule can be determined as interesting or not. Then other rule are generated by deleting the last items in the antecedent and inserting it to the consequent, further the confidences of the new rules are checked to determine the interestingness of them. Those processes

iterated until the antecedent becomes empty. Since the second subproblem is quite straight forward, most of the researches focus on the first subproblem. The Apriori algorithm finds the frequent sets L In Database D .

Let X , Y ⊆ I be any two itemsets. Observe that if $X \subseteq Y$, then sup(X) ≥ sup(Y), which leads to the following two corollaries:

- If X is frequent, then any subset Y ⊆ X is also frequent.
- If X is not frequent, then any superset Y ⊇ X cannot be frequent.

Based on the above observations, we can significantly improve the itemset mining algorithm by reducing the number of candidates we generate, by limiting the candidates to be only those that will potentially be frequent. First we can stop generating supersets of a candidate once we determine that it is infrequent, since no superset of an infrequent itemset can be frequent. Second, we can avoid any candidate that has an infrequent subset. These two observations can result in significant pruning of the search space.

- Find frequent set L_{k-1}.
- Join Step.
  - C_k is generated by joining L_{k-1} with itself
- Prune Step.
  - Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k -itemset, hence should be removed.

where

- (C_k : Candidate itemset of size k)
- (L_k : frequent itemset of size k)

Apriori Pseudocode

Apriori (T, ε)

$L_1 \leftarrow \{$ *large 1-itemsets that appear in more than* ε *transactions* $\}$

$k \leftarrow 2$

while $L_{k-1} \neq \varnothing$

$C_k \leftarrow$ *Generate* (L_{k-1})

for transactions $t \in T$

$C_t \leftarrow$ Subset (C_k, t)

for candidates $c \in C_t$

$\text{count}[c] \leftarrow \text{count}[c] + 1$

$L_k \leftarrow \{c \in C_k | \ \text{count}[c] \geq \varepsilon\}$

$k \leftarrow k + 1$

return $\bigcup_k L_k$

Implementation

The R package that implements association rule mining is called arules, and it can be found at http://cran.r-project.org/web/packages/arules/index.html (Details how to install R can be found here http://cran.r-project.org/bin/linux/ubuntu). In package arules we interface free reference implementations of Apriori and Eclat by Christian Borgelt (Borgelt and Kruse, 2002; Borgelt, 2003)3. The code is called directly from R by the functions apriori() and éclat() and the data objects are directly passed from R to the C code and back without writing to external files. The implementations can mine frequent itemsets, and closed and maximal frequent itemsets. In addition, apriori() can

also mine association rules.

Installation of the dependencies packages can be performed within the R environment using the function "install.packages(“*package name*”)". The name of the package in question is “*arules*”. To use this package you need R environment with version 2.7.0 at least (details how to update R can be found in http://cran.r-project.org/bin/linux/ubuntu).

```
install.packages(“arules”)
```

After the user installed the necessaries packages, he must load them. This can be done using the function "library(*package name*)".

```
library(arules)
```

Visualization

Reading the data

Transactions can be read from files in the basket format, with the command read.transactions. The parameters for read.transactions are:

file

format read.transactions can get data structured in multiple formats, one of them being basket.

separator(sep)

One example of it’s use would be:

```
tr<-read.transactions("teste",format="basket",sep=",")
```

The object ”tr” is used to store the transactions read from the file named ”teste”, where each item is separated by a ”,”. ”teste” could be, for example:

```
A,B,C
B,C
A,B,D
A,B,C,D
A
B
```

One way to visualize the data is inspect(object). For example:

```
inspect(tr)
```

```
items
1 {A,
   B,
   C}
2 {B,
   C}
3 {A,
   B,
   D}
4 {A,
   B,
   C,
```

```
   D}
5 {A}
6 {B}
```

Additionally, you can visually inspect binary incidence matrices, or plot the frequency of items sets:

```
image(tr)
itemFrequencyPlot(tr, support = 0.1)
```

To show the number of items in transactions read from the file named ”teste” do:

```
length(tr)
```

```
[1] 6
```

Rules

The function to mine frequent itemsets, association rules or association hyperedges, using the Apriori algorithm, takes 2 parameters:

Data: the object that contains the data

parameter: a multi-dimensional parameter to set up support and confidence

For example, using the dataset gathered in the previous section:

```
rules <- apriori(tr, parameter= list(supp=0.5, conf=0.5))
```

The rules can be visualized with the command inspect:

```
inspect(rules)
```

```
    rhs    support confidence lift
1 {} => {C} 0.5000000 0.5000000 1.0
2 {} => {A} 0.6666667 0.6666667 1.0
3 {} => {B} 0.8333333 0.8333333 1.0
4 {C} => {B} 0.5000000 1.0000000 1.2
5 {B} => {C} 0.5000000 0.6000000 1.2
6 {A} => {B} 0.5000000 0.7500000 0.9
7 {B} => {A} 0.5000000 0.6000000 0.9
```

To get a summary of the rules' characteristics, the function ”summary” can be used:

```
summary(rules)
```

```
set of 7 rules

rule length distribution (lhs + rhs):sizes
1 2
3 4

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   2.000   1.571   2.000   2.000

summary of quality measures:
    support         confidence          lift
```

```
 Min.   :0.5000   Min.   :0.5000   Min.   :0.900
 1st Qu.:0.5000   1st Qu.:0.6000   1st Qu.:0.950
 Median :0.5000   Median :0.6667   Median :1.000
 Mean   :0.5714   Mean   :0.7071   Mean   :1.029
 3rd Qu.:0.5833   3rd Qu.:0.7917   3rd Qu.:1.100
 Max.   :0.8333   Max.   :1.0000   Max.   :1.200

mining info:
 data ntransactions support confidence
   tr             6     0.5        0.5
```

Outher quality measures of the rules can be displaed with:

```
interestMeasure(rules, c("support", "chiSquare", "confidence",
"conviction",
      "cosine", "coverage", "leverage", "lift", "oddsRatio"), tr)
```

```
    support chiSquare confidence conviction    cosine  coverage    leverage
1 0.5000000       NaN  0.5000000  1.0000000 0.7071068 1.0000000  0.00000000
2 0.6666667       NaN  0.6666667  1.0000000 0.8164966 1.0000000  0.00000000
3 0.8333333       NaN  0.8333333  1.0000000 0.9128709 1.0000000  0.00000000
4 0.5000000       1.2  1.0000000        Inf 0.7745967 0.5000000  0.08333333
5 0.5000000       1.2  0.6000000  1.2500000 0.7745967 0.8333333  0.08333333
6 0.5000000       0.6  0.7500000  0.6666667 0.6708204 0.6666667 -0.05555556
7 0.5000000       0.6  0.6000000  0.8333333 0.6708204 0.8333333 -0.05555556
  lift oddsRatio
1  1.0       NaN
2  1.0       NaN
3  1.0       NaN
4  1.2       Inf
5  1.2       Inf
6  0.9         0
7  0.9         0
```

To calculate a single measure and add it to the quality slot:

```
quality(rules) <- cbind(quality(rules), hyperConfidence = interestMeasure(rules, method = "hyperConfidence", Income))

inspect(head(SORT(rules, by = "hyperConfidence")))
```

```
  lhs     rhs    support confidence lift hyperConfidence
1 {C} => {B} 0.5000000  1.0000000  1.2             0.5
2 {B} => {C} 0.5000000  0.6000000  1.2             0.5
3 {}  => {C} 0.5000000  0.5000000  1.0             0.0
4 {}  => {A} 0.6666667  0.6666667  1.0             0.0
5 {}  => {B} 0.8333333  0.8333333  1.0             0.0
6 {A} => {B} 0.5000000  0.7500000  0.9             0.0
```

Finally, to send the output to a file use:

```
sink("sink-examp.txt")
inspect(head(SORT(rules, by = "hyperConfidence")))
```

```
...
```

Case study

We now present a brief case study to illustrate the use of **apriori** and the **package** arules in a real data set.

Scenario

Spam is the abuse of electronic messaging systems (including most broadcast media, digital delivery systems) to send unsolicited bulk messages indiscriminately. Many filters are designed to stop spam in its tracks, but such filters are often left behind by the spammers' obfuscating techniques. One of such filters is SpamAssassin, a well known filter that relies on machine learning and fixed rules for filtering. SpamAssassin, however, doesn't consider the possible relations between its rules.

Input Data

22406 spams were used in this case study, ranging from 1998 to 2009, collected from the public spam corpus SpamArchive (http://untroubled.org/spam/).Each spam was analyzed by SpamAssassin, and the rules found were analyzed, as well its year. For example, a particular spam would be: YEAR=1998 INVALID_DATE INVALID_TZ_EST meaning this was a spam from 1998 in which SpamAssassin accused two rules: INVALID_DATE and INVALID_TZ_EST.

Implementation

The output from each spam filtered through SpamAssassin was saved in a file named output. The following commands were issued:

```
library("arules"); #Using the package

tr<-read.transactions("output",format="basket",sep=" ") #Reading each spam from the output file, separated by ' ' (single spaces).

rules <- apriori(tr, parameter= list(supp=0.05, conf=0.3)) #Running apriori

inspect(rules) #Visualizing the association rules found.
```

Output Data

We displayed only the itemsets of size 1, for simplicity:

```
    lhs                            rhs                          support    confidence   lift
1   {}                          => {RDNS_NONE}                0.40828350  0.4082835  1.0000000
2   {}                          => {MIME_HTML_ONLY}           0.30509685  0.3050968  1.0000000
3   {}                          => {HTML_MESSAGE}             0.51347853  0.5134785  1.0000000
4   {YEAR=2008}                 => {RDNS_NONE}                0.06663394  0.7465000  1.8283864
5   {YEAR=2009}                 => {RDNS_NONE}                0.07007052  0.7850000  1.9226836
6   {YEAR=2009}                 => {HTML_MESSAGE}             0.05877890  0.6585000  1.2824295
7   {RATWARE_OUTLOOK_NONAME}    => {RATWARE_MS_HASH}          0.05404802  0.9991749 12.9482436
8   {RATWARE_MS_HASH}           => {RATWARE_OUTLOOK_NONAME}   0.05404802  0.7004049 12.9482436
9   {MISSING_DATE}              => {MISSING_MID}              0.05538695  0.7231935  4.0540087
10  {MISSING_MID}               => {MISSING_DATE}             0.05538695  0.3104829  4.0540087
11  {HELO_DYNAMIC_IPADDR}       => {FH_HELO_EQ_D_D_D_D}       0.05400339  0.8916728 10.0648972
12  {FH_HELO_EQ_D_D_D_D}        => {HELO_DYNAMIC_IPADDR}      0.05400339  0.6095718 10.0648972
13  {YEAR=2007}                 => {RDNS_NONE}                0.05543158  0.6210000  1.5210020
14  {YEAR=2007}                 => {HTML_MESSAGE}             0.05346782  0.5990000  1.1665532
15  {FORGED_OUTLOOK_TAGS}       => {HTML_MESSAGE}             0.05163795  0.9974138  1.9424644
16  {SUBJ_ILLEGAL_CHARS}        => {SUBJECT_NEEDS_ENCODING}   0.05038829  0.9982317 17.8645195
```

```
17  {SUBJECT_NEEDS_ENCODING} => {SUBJ_ILLEGAL_CHARS}     0.05038829  0.9017572 17.8645195
18  {RATWARE_MS_HASH}        => {MSGID_OUTLOOK_INVALID}  0.06667857  0.8640833  8.7604752
19  {MSGID_OUTLOOK_INVALID}  => {RATWARE_MS_HASH}        0.06667857  0.6760181  8.7604752
20  {YEAR=2002}              => {HTML_MESSAGE}           0.05636883  0.6315000  1.2298469
21  {DATE_SPAMWARE_Y2K}      => {INVALID_DATE}           0.05511916  1.0000000  5.8394579
22  {INVALID_DATE}           => {DATE_SPAMWARE_Y2K}      0.05511916  0.3218660  5.8394579
23  {RCVD_HELO_IP_MISMATCH}  => {RCVD_NUMERIC_HELO}      0.06190306  0.9992795  7.4982777
24  {RCVD_NUMERIC_HELO}      => {RCVD_HELO_IP_MISMATCH}  0.06190306  0.4645010  7.4982777
25  {MIME_QP_LONG_LINE}      => {HTML_MESSAGE}           0.07033830  0.8602620  1.6753612
26  {FH_HELO_EQ_D_D_D_D}     => {HTML_MESSAGE}           0.05819870  0.6569270  1.2793660
27  {FORGED_OUTLOOK_HTML}    => {MIME_HTML_ONLY}         0.06538427  1.0000000  3.2776477
28  {FORGED_OUTLOOK_HTML}    => {HTML_MESSAGE}           0.06538427  1.0000000  1.9475011
29  {MSGID_OUTLOOK_INVALID}  => {MISSING_MIMEOLE}        0.06261716  0.6348416  4.1301572
30  {MISSING_MIMEOLE}        => {MSGID_OUTLOOK_INVALID}  0.06261716  0.4073751  4.1301572
31  {MSGID_OUTLOOK_INVALID}  => {MIME_HTML_ONLY}         0.05752923  0.5832579  1.9117140
32  {MSGID_OUTLOOK_INVALID}  => {HTML_MESSAGE}           0.06364367  0.6452489  1.2566229
33  {YEAR=2003}              => {MIME_HTML_ONLY}         0.05766313  0.6460000  2.1173604
34  {YEAR=2003}              => {HTML_MESSAGE}           0.06895474  0.7725000  1.5044446
35  {YEAR=2004}              => {HTML_MESSAGE}           0.07042756  0.7890000  1.5365784
36  {FORGED_MUA_OUTLOOK}     => {HTML_MESSAGE}           0.05681514  0.5912680  1.1514951
37  {MIME_HTML_ONLY_MULTI}   => {MPART_ALT_DIFF}         0.06583058  0.9886059  9.9285987
38  {MPART_ALT_DIFF}         => {MIME_HTML_ONLY_MULTI}   0.06583058  0.6611385  9.9285987
39  {MIME_HTML_ONLY_MULTI}   => {MISSING_MIMEOLE}        0.05315540  0.7982574  5.1933086
40  {MISSING_MIMEOLE}        => {MIME_HTML_ONLY_MULTI}   0.05315540  0.3458188  5.1933086
41  {MIME_HTML_ONLY_MULTI}   => {MIME_HTML_ONLY}         0.06658931  1.0000000  3.2776477
42  {MIME_HTML_ONLY_MULTI}   => {HTML_MESSAGE}           0.06658931  1.0000000  1.9475011
43  {MISSING_MID}            => {RDNS_NONE}              0.07060609  0.3957968  0.9694167
44  {MISSING_MID}            => {MIME_HTML_ONLY}         0.07730072  0.4333250  1.4202867
45  {MISSING_MID}            => {HTML_MESSAGE}           0.09515308  0.5334001  1.0387972
46  {MPART_ALT_DIFF}         => {MISSING_MIMEOLE}        0.05337856  0.5360825  3.4876492
47  {MISSING_MIMEOLE}        => {MPART_ALT_DIFF}         0.05337856  0.3472706  3.4876492
48  {MPART_ALT_DIFF}         => {MIME_HTML_ONLY}         0.06591984  0.6620350  2.1699174
49  {MPART_ALT_DIFF}         => {HTML_MESSAGE}           0.09957154  1.0000000  1.9475011
50  {RCVD_NUMERIC_HELO}      => {MISSING_MIMEOLE}        0.05369098  0.4028801  2.6210603
51  {MISSING_MIMEOLE}        => {RCVD_NUMERIC_HELO}      0.05369098  0.3493031  2.6210603
52  {RCVD_NUMERIC_HELO}      => {RDNS_NONE}              0.07752388  0.5817147  1.4247812
53  {RCVD_NUMERIC_HELO}      => {MIME_HTML_ONLY}         0.05855574  0.4393838  1.4401453
54  {RCVD_NUMERIC_HELO}      => {HTML_MESSAGE}           0.07877354  0.5910918  1.1511518
55  {RDNS_DYNAMIC}           => {MIME_HTML_ONLY}         0.06257253  0.4054367  1.3288786
56  {RDNS_DYNAMIC}           => {HTML_MESSAGE}           0.09975007  0.6463274  1.2587232
57  {INVALID_DATE}           => {MIME_HTML_ONLY}         0.05467286  0.3192598  1.0464213
58  {INVALID_DATE}           => {HTML_MESSAGE}           0.06712488  0.3919729  0.7633676
59  {MISSING_MIMEOLE}        => {MIME_HTML_ONLY}         0.10381148  0.6753775  2.2136494
60  {MIME_HTML_ONLY}         => {MISSING_MIMEOLE}        0.10381148  0.3402575  2.2136494
61  {MISSING_MIMEOLE}        => {HTML_MESSAGE}           0.10648933  0.6927991  1.3492269
62  {MIME_HTML_ONLY}         => {RDNS_NONE}              0.11867357  0.3889702  0.9526963
63  {RDNS_NONE}              => {HTML_MESSAGE}           0.23216995  0.5686489  1.1074443
```

```
64  {HTML_MESSAGE}           => {RDNS_NONE}           0.23216995  0.4521512  1.1074443
65  {MIME_HTML_ONLY}         => {HTML_MESSAGE}        0.30505222  0.9998537  1.9472162
66  {HTML_MESSAGE}           => {MIME_HTML_ONLY}      0.30505222  0.5940895  1.9472162
```

Analysis

It takes a critical eye to spot the interesting information. Rules such as YEAR=2009 => HTML_MESSAGE carry information that is not surprising, deeming it uninteresting. However, some interesting patterns can be found:

MISSING_DATE => MISSING_MID

> Spammers who don't set up a date usually 'forget' about the message ID as well. Maybe spam assassin could score differently when they are found together.

HELO_DYNAMIC_IPADDR => FH_HELO_EQ_D_D_D_D

> When HELO is done using a suspicious hostname, it is usually in the d-d-d-d format. This characterizes a spam more than just one of the two rules separately.

SUBJECT_NEEDS_ENCODING => SUBJ_ILLEGAL_CHARS

> Typically this means a spam is in Chinese. These rules obviously have a strong correlation (confidence= 90%).

Many more interesting patterns can be found and studied, in order to improve the quality of the filter. Using apriori with "arules" is an easy and straightforward task.

References

- [1]http://en.wikipedia.org/wiki/Apriori_algorithm
- [2]http://rss.acs.unt.edu/Rdoc/library/arules/html/apriori.html
- [3]http://untroubled.org/spam/

The FP-Growth Algorithm

In Data Mining the task of finding frequent pattern in large databases is very important and has been studied in large scale in the past few years. Unfortunately, this task is computationally expensive, especially when a large number of patterns exist.

The FP-Growth Algorithm, proposed by Han in [1], is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In his study, Han proved that his method outperforms other popular methods for mining frequent patterns, e.g. the Apriori Algorithm [2] and the TreeProjection [3]. In some later works [4] it was proved that FP-Growth has better performance than other methods, including Eclat [5] and Relim [6]. The popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve his performance [] [] [7] [8] [9] [10] [11] [12] [13] [14].

This chapter describes the algorithm and some variations and discuss features of the R language and strategies to implement the algorithm to be used in the R. Next a briefly conclusion and future works are proposed.

The algorithm

The FP-Growth Algorithm is an alternative way to find frequent itemsets without using candidate generations, thus improving performance. For so much it uses a divide-and-conquer strategy [15]. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the itemset association information.

In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity.

In large databases, it's not possible to hold the FP-tree in the main memory. A strategy to cope with this problem is to firstly partition the database into a set of smaller databases (called projected databases), and then construct an FP-tree from each of these smaller databases.

The next subsections describe the FP-tree structure and FP-Growth Algorithm, finally an example is presented to make it easier to understand these concepts.

FP-Tree structure

The frequent-pattern tree (FP-tree) is a compact structure that stores quantitative information about frequent patterns in a database .

Han defines the FP-tree as the tree structure defined below :

1. One root labeled as "null" with a set of item-prefix subtrees as children, and a frequent-item-header table (presented in the left side of Figure 1);
2. Each node in the item-prefix subtree consists of three fields:
    1. Item-name: registers which item is represented by the node;
    2. Count: the number of transactions represented by the portion of the path reaching the node;
    3. Node-link: links to the next node in the FP-tree carrying the same item-name, or null if there is none.

1. Each entry in the frequent-item-header table consists of two fields:
    1. Item-name: as the same to the node;
    2. Head of node-link: a pointer to the first node in the FP-tree carrying the item-name.

Additionally the frequent-item-header table can have the count support for an item. The Figure 1 below show an example of a FP-tree.

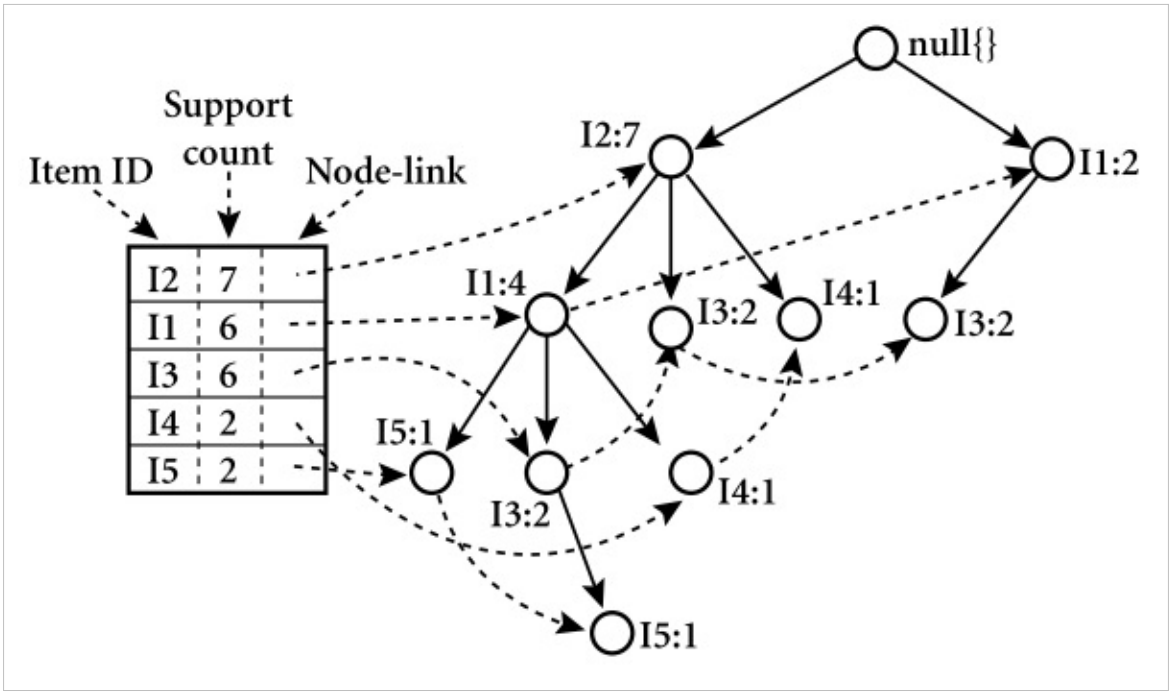


Figure 1: An example of an FP-tree from [15].

The original algorithm to construct the FP-Tree defined by Han in is presented below in Algorithm 1.

Algorithm 1: FP-tree construction

Input: A transaction database DB and a minimum support threshold ?.

Output: FP-tree, the frequent-pattern tree of DB.

Method: The FP-tree is constructed as follows.

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.
2. Create the root of an FP-tree, T, and label it as “null”. For each transaction Trans in DB do the following:
   - Select the frequent items in Trans and sort them according to the order of FList. Let the sorted frequent-item list in Trans be [p | P], where p is the first element and P is the remaining list. Call insert tree([p | P], T).
   - The function insert tree([p | P], T) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N ’s count by 1; else create a new node N , with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

By using this algorithm, the FP-tree is constructed in two scans of the database. The first scan collects and sort the set of frequent items, and the second constructs the FP-Tree.

FP-Growth Algorithm

After constructing the FP-Tree it's possible to mine it to find the complete set of frequent patterns. To accomplish this job, Han in presents a group of lemmas and properties, and thereafter describes the FP-Growth Algorithm as presented below in Algorithm 2.

Algorithm 2: FP-Growth

Input: A database DB, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold ?.

Output: The complete set of frequent patterns.

Method: call FP-growth(FP-tree, null).

```
Procedure FP-growth(Tree, a) {
    (01) if Tree contains a single prefix path then // Mining single prefix-path FP-tree {
        (02) let P be the single prefix-path part of Tree;
        (03) let Q be the multipath part with the top branching node replaced by a null root;
        (04) for each combination (denoted as ß) of the nodes in the path P do
        (05) generate pattern ß ∪ a with support = minimum support of nodes in ß;
        (06) let freq pattern set(P) be the set of patterns so generated;
    }
    (07) else let Q be Tree;
    (08) for each item ai in Q do { // Mining multipath FP-tree
        (09) generate pattern ß = ai ∪ a with support = ai .support;
        (10) construct ß's conditional pattern-base and then ß's conditional FP-tree Tree ß;
        (11) if Tree ß ≠ Ø then
            (12) call FP-growth(Tree ß , ß);
        (13) let freq pattern set(Q) be the set of patterns so generated;
    }
    (14) return(freq pattern set(P) ∪ freq pattern set(Q) ∪ (freq pattern set(P) × freq pattern set(Q)))
}
```

When the FP-tree contains a single prefix-path, the complete set of frequent patterns can be generated in three parts: the single prefix-path P, the multipath Q, and their combinations (lines 01 to 03 and 14). The resulting patterns for a single prefix path are the enumerations of its subpaths that have the minimum support (lines 04 to 06). Thereafter, the multipath Q is defined (line 03 or 07) and the resulting patterns from it are processed (lines 08 to 13). Finally, in line 14 the combined results are returned as the frequent patterns found.

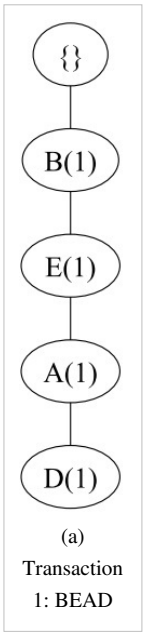
An example

This section presents a simple example to illustrate how the previous algorithm works. The original example can be viewed in [16].

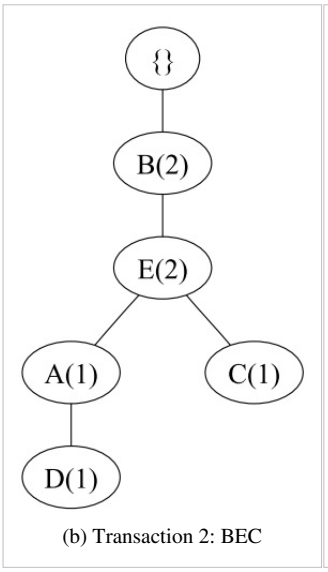
Consider the transactions below and the minimum support as 3:

	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

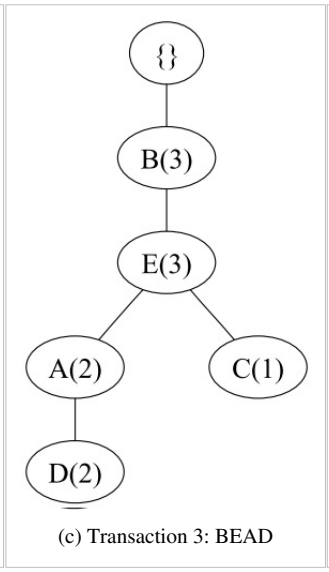
To build the FP-Tree, frequent items support are first calculated and sorted in decreasing order resulting in the following list: { B(6), E(5), A(4), C(4), D(4) }. Thereafter, the FP-Tree is iteratively constructed for each transaction, using the sorted list of items as shown in Figure 2.



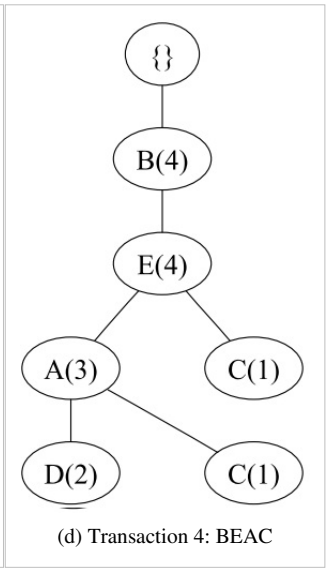
(a) Transaction 1: BEAD



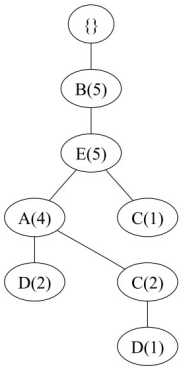
(b) Transaction 2: BEC



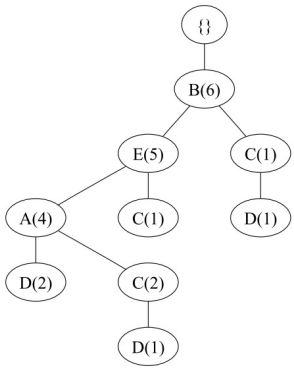
(c) Transaction 3: BEAD



(d) Transaction 4: BEAC



(e) Transaction 5: BEACD



(f) Transaction 6: BCD

Figure 2: Constructing the FP-Tree iteratively.

As presented in Figure 3, the initial call to FP-Growth uses the FP-Tree obtained from the Algorithm 1, presented in Figure 2 (f), to process the projected trees in recursive calls to get the frequent patterns in the transactions presented before.

Using a depth-first strategy the projected trees are determined to items D, C, A, E and B, respectively. First the projected tree for D is recursively processed, projecting trees for DA, DE and DB. In a similar manner the remaining items are processed. At the end of process the frequent itemset is: { DAE, DAEB, DAB, DEB, CE, CEB, CB, AE, AEB, AB, EB }.

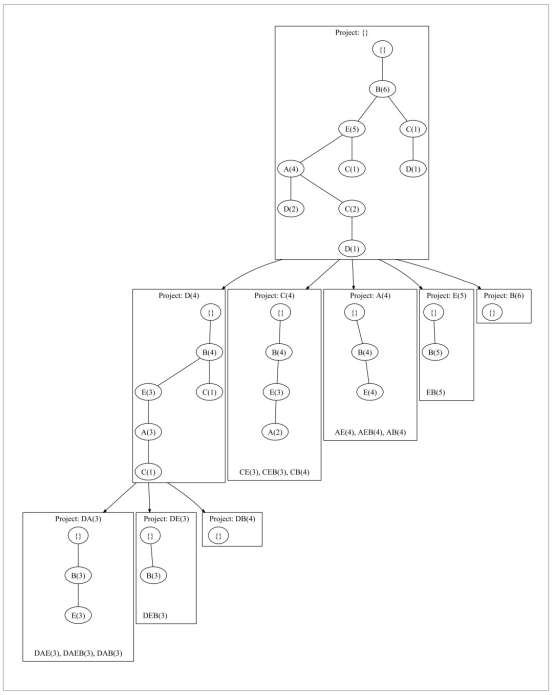


Figure 3: Projected trees and frequent patterns founded by the recursively calls to FP-Growth Algorithm.

FP-Growth Algorithm Variations

As mentioned before, the popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve its performance [17] . In this section some of them are briefly described.

DynFP-Growth Algorithm

The DynFP-Growth , has focused in improving the FP-Tree algorithm construction based on two observed problems:

1. The resulting FP-tree is not unique for the same “logical” database;
2. The process needs two complete scans of the database.

To solve the first problem Gyorödi C., et al. proposes the usage of a support descending order together with a lexicographic order, ensuring in this way the uniqueness of the resulting FP-tree for different “logically equivalent” databases. To solve the second problem they proposed devising a dynamic FP-tree reordering algorithm, and employing this algorithm whenever a “promotion” to a higher order of at least one item is detected.

An important feature in this approach is that it’s not necessary to rebuild the FP-Tree when the actual database is updated. It’s only needed to execute the algorithm again taking into consideration the new transactions and the stored FP-Tree.

Another adaptation proposed, because of the dynamic reordering process, is a modification in the original structures, by replacing the single linked list with a doubly linked list for linking the tree nodes to the header and adding a

master-table to the same header. See for more details.

FP-Bonsai Algorithm

The FP-Bonsai improve the FP-Growth performance by reducing (pruning) the FP-Tree using the ExAnte data-reduction technique . The pruned FP-Tree was called FP-Bonsai. See for more details.

AFOPT Algorithm

Investigating the FP-Growth algorithm performance Liu proposed the AFOPT algorithm in . This algorithm aims at improving the FP-Growth performance in four perspectives:

- Item Search Order: when the search space is divided, all items are sorted in some order. The number of the conditional databases constructed can differ very much using different items search orders;
- Conditional Database Representation: the traversal and construction cost of a conditional database heavily depends on its representation;
- Conditional Database Construction Strategy: constructing every conditional database physically can be expensive affecting the mining cost of each individual conditional database;
- Tree Traversal Strategy: the traversal cost of a tree is minimal using top-down traversal strategy.

See for more details.

NONORDFP Algorithm

The Nonordfp algorithm was motivated by the running time and the space required for the FP-Growth algorithm. The theoretical difference is the main data structure (FP-Tree), which is more compact and which is not needed to rebuild it for each conditional step. A compact, memory efficient representation of an FP-tree by using Trie data structure, with memory layout that allows faster traversal, faster allocation, and optionally projection was introduced. See for more details.

FP-Growth* Algorithm

This algorithm was proposed by Grahne et al , and is based in his conclusion about the usage of CPU time to compute frequent item sets using FP-Growth. They observed that 80% of CPU time was used for traversing FP-Trees . Therefore, they used an array-based data structure combined with the FP-Tree data structure to reduce the traversal time, and incorporates several optimization techniques. See for more details.

PPV, PrePost, and FIN Algorithm

These three algorithms were propsed by Deng et al , and are based on three novel data structures called Node-list , N-list , and Nodeset respectively for facilitating the mining process of frequent itemsets. They are based on a FP-tree with each node encoding with pre-order traversal and post-order traversal. Compared with Node-lists, N-lists and Nodesets are more efficient. This causes the efficiency of PrePost and FIN is higher than that of PPV . See for more details.

Data Visualization in R

Normally the data used to mine frequent item sets are stored in text files. The first step to visualize data is load it into a data-frame (an object to represent the data in R).

The function read.table could be used in the following way:

```
var <- read.table(fileName, fileEncoding=value, header = value, sep = value)

Where:
  • var: the R variable to receive the loaded data.
  • fileName: is a string value with the name of the file to be loaded.
  • fileEncoding: to be used when the file has no-ASCII characters.
  • header: indicates that the file has headers (T or TRUE) or not (F
or FALSE).
  • sep: defines the field separator character (“,”, “;” or “\t” for
example)
  • Only the filename is a mandatory parameter.
```

Another function in R to load data is called scan. See the R Data Import/Export Manual [18] for details.

The visualization of the data can be done in two ways:

- Using the variable name (var), to list the data in a tabular presentation.
- And summary(var), to list a summary of the data.

Example:

```
> data <- read.table("boolean.data", sep=",", header=T)

> data
      A    B     C     D     E
1  TRUE TRUE FALSE  TRUE  TRUE
2 FALSE TRUE  TRUE FALSE  TRUE
3  TRUE TRUE FALSE  TRUE  TRUE
4  TRUE TRUE  TRUE FALSE  TRUE
5  TRUE TRUE  TRUE  TRUE  TRUE
6 FALSE TRUE  TRUE  TRUE FALSE

> summary(data)
     A              B               C               D               E
 Mode :logical   Mode:logical   Mode :logical   Mode :logical   Mode
:logical
 FALSE:2         TRUE:6         FALSE:2         FALSE:2         FALSE:1
 TRUE :4         NA's:0         TRUE :4         TRUE :4         TRUE :5
 NA's :0                        NA's :0         NA's :0         NA's :0
```

In the example above the data in “boolean.data”, that have a simple binary database, was loaded in the data-frame variable data. Typing the name of the variable in the command line, its content is printed, and typing the summary command the frequency occurrence of each item is printed. The summary function works differently. It depends on the type of data in the variable, see [19] [20] [21] for more details.

The functions presented previously can be useful, but to frequent item set datasets use an specific package called arules [22] [23] which is better to visualize the data.

Using arules, several functions are made available:

- read.transactions: used to load the database file into a variable.
- inspect: used to list the transactions.
- length: returns the number of transactions.
- image: plots an image with all transactions in a matrix format.
- itemFrequencyPlot: calculates the frequency of each item and plots it in a bar graphic.

Example:

```
> data <- read.transactions("basket.data", format="basket", sep = ",")

> data
transactions in sparse format with
 6 transactions (rows) and
 5 items (columns)

> inspect(data)
  items
1 {A,
   B,
   D,
   E}
2 {B,
   C,
   E}
3 {A,
   B,
   D,
   E}
4 {A,
   B,
   C,
   E}
5 {A,
   B,
   C,
   D,
   E}
6 {B,
   C,
   D}

> length(data)
[1] 6

> image(data)

> itemFrequencyPlot(data, support=0.1)
```

In this example we can see the difference in the usage of the variable name in the command line. From transactions, only the number of rows (transactions) and cols (items) are printed. The result of image(data) and itemFrequencyPlot(data, support = 0.1) are presented in the figures 4 and 5 below.

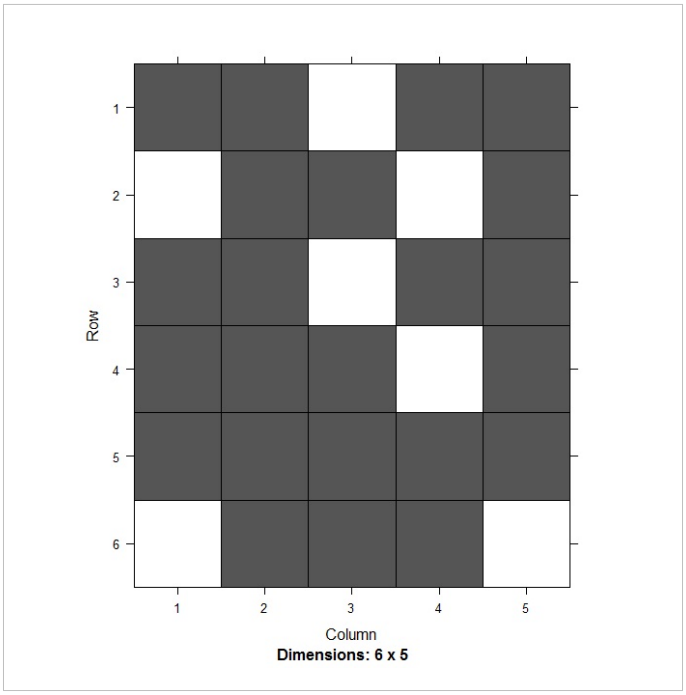


Figure 4: Result of the image(data) call.

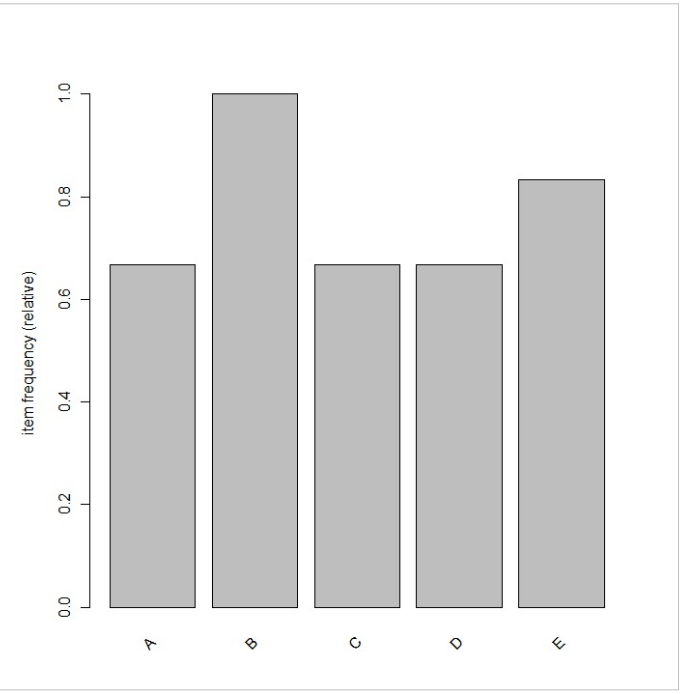


Figure 5: Result of the itemFrequencyPlot(data, support = 0.1) call.

Implementation in R

The R [24] provides several facilities for data manipulation, calculation and graphical display very useful for data analysis and mining. It can be used as both a statistical library and a programming language.

As a statistical library, it provides a set of functions to summary data, matrix facilities, probability distributions, statistical models and graphical procedures.

As a programming language, it provides a set of functions, commands and methods to instantiate and manage values of different type of objects (including lists, vectors and matrices), user interaction (input and output from console), control statements (conditional and loop statements), creation of functions, calls to external resources and create packages.

This chapter isn't accomplished to present details about R resources and will focus on the challenges to implement an algorithm using R or to be used in R. However, to better understanding the R power, some basic examples based in are presented in Appendix A.

To implement an algorithm using R, normally it would be necessary to create complex objects to represent the data structures to be processed. Also, it would be necessary to implement complex functions to process this data structures. Thinking in the specific case of implementing the FP-Growth algorithm could be very hard to represent and process an FPTree using only the R resources. Moreover, for performance reasons it could be interesting to implement the algorithm using other languages and integrate it with R. Other reasons for using other languages are to get better memory management and to use existing packages[25].

Two ways to integrate R with other languages are available and will be briefly presented below: creating a package and making an external call using interface functions . Next it is presented the FP-Growth implementation used in this work and the efforts to integrate it with R. For both would be necessary to install the RTools [26].

Creating a Package

Package is a mechanism for loading optional code implemented in other languages in R . The R distribution itself includes about 25 packages, and some extra packages used in this WikiBook can be listed:

- aRules [1]
- arulesNBMiner [27]
- arulesSequences [28]
- cluster [29]

To create a package it's necessary to follow some specifications. The sources of an R package consist in a directory structure described below:

- **Root**: the root directory containing a DESCRIPTION file and some optional files (INDEX, NAMESPACE, configure, cleanup, LICENCE, COPYING and NEWS).
- **R**: contains only R code files that could be executed by the R command source(filename) to create R objects used by users. Alternatively, this directory can have a file sysdata.rda. This file has a saved image of R objects created in an execution of R console.
- **data**: aimed to have data files, either to be made available via lazy-loading or for loading using function data(). These data files could be from three different types: plain R code (.r or .R), tables (.tab, .txt, or .csv) or saved data from R console (.RData or .rda). Some additional compressed file can be used to table's files.
- **demo**: contains scripts in pain R code (for running using function demo()) that demonstrate some of the functionality of the package
- **exec**: could contain additional executables the package needs, typically scripts for interpreters such as the shell, Perl, or Tcl.
- **inst**: its content will be copied to the installation directory after it is built and its makefile can create files to be installed. May contain all information files that intended to be viewed by end users.

- **man**: should contain only documentation files for the objects in the package (using an specific R documentation format). An empty man directory causes an installation error.
- **po**: used for files related to internalization, in other words, to translate errors and warning messages.
- **src**: contains the sources, headers, makevars and makefiles. The supported languages are: C, C++, FORTRAN 77, Fortran 9x, Objective C and Objective C++. It's not possible to mix all these languages in a single package, but mix C and FORTRAN 77 or C and C++ seems to be successful. However, there ways to make usage from other packages.
- **tests**: used for additional package-specific test code.

Once a source package is created, it must be installed by the command line in the OS console:

```
R CMD INSTALL <parameters>
```

Alternatively, packages can be downloaded and installed from within R, using the command line in the R console:

```
> install.packages(<parameters>)
```

See the Installation and Administration manual [30] [], for details.

After installed, the package needs to be loaded to be used, using the command line in the R console:

```
> library(packageName)
```

Making external call using interface functions

Making external call using interface functions is a simple way to use external implementation without complies with all rules described before to create a package to R.

First the code needs to include R.h header file that comes with R installation.

To compile a source code is needs to use the compiler R at the OS command line:

```
> R CMD SHLIB <parameters>
```

Compiled code to be used in R needs to be loaded as a shared object in Unix-like OS, or as a DLL in Windows OS. To load or unload it can be used the commands in the R console:

```
> dyn.load(fileName)
> dyn.unload(fileName)
```

After the load, the external code can be called using some of these functions:

- .C
- .Call
- .Fortran
- .External

Two simple examples are presented below, using .C function:

Example 1: Hello World

```
## C code in file example1.c

#include <R.h>

Void do_stuff ()
{
   printf("\nHello, I'm in a C code!\n");
}
```

```
## R code in file example.R

dyn.load("example1.dll")

doStuff <-
function (){
    tmp <- .C("do_stuff")
      rm(tmp)
      return(0)
}

doStuff()


## Compiling code in OS command line

C:\R\examples>R CMD SHLIB example1.c
gcc -I"C:/PROGRA~1/R/R-212~1.0/include" -O3 -Wall -std=gnu99 -c
example1.c -o example1.o
gcc -shared -s -static-libgcc -o example1.dll tmp.def example1.o
-LC:/PROGRA~1/R/R-212~1.0/bin/i386 -lR


## Output in R console

> source("example1.R")

Hello, I'm in a C code!
>
```

Example 2: Calling C with an integer vector

```
##C code in file example2.c

#include <R.h>

void doStuff(int *i) {
    i[0] = 11;
}


## Output in R console

> dyn.load("example2.dll")
> a <- 1:10
> a
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
> out <- .C("doStuff", b = as.integer(a))
> a
 [1]  1  2  3  4  5  6  7  8  9 10
> out$b
 [1] 11  2  3  4  5  6  7  8  9 10
```

The FP-Growth Implementation

The FP-Growth implementation used in this work was maded by Christian Borgelt [31] [32] a principal researcher at European Centre for Soft Computing. He also implemented the code used in arules package [22] for Eclat and Apriori algorithms. The source code can be downloaded in his personal site.

As described by Borgelt implemented two variants of the core operation of computing a projection of an FP-tree. In addition, projected FP-trees are optionally pruned by removing items that has becoming in-frequent (using FP-Bonsai approach).

The source code is divided into three main folders (packages):

- **fpgrowth**: contais the main file that implements the algorithm and manages the FP-Tree;
- **tract**: manages item sets, transactions and its reports;
- **util**: facilities to be used in fpgrowth and tract.

The syntax to call this implementation, from the OS command line, is:

```
> fpgrowth [options] infile [outfile [selfile]]
-t#      target type                              (default: s)
         (s: frequent, c: closed, m: maximal item sets)
-m#      minimum number of items per item set     (default: 1)
-n#      maximum number of items per item set     (default: no limit)
-s#      minimum support of an item set           (default: 10%)
         (positive: percentage, negative: absolute number)
-e#      additional evaluation measure            (default: none)
-d#      minimum value of add. evaluation measure (default: 10%)
-g       write output in scanable form (quote certain characters)
-H#      record header  for output                (default: "")
-k#      item separator for output                (default: " ")
-v#      output format for item set information   (default: "  (%1S)")
-q#      sort items w.r.t. their frequency        (default: 2)
         (1: ascending, -1: descending, 0: do not sort,
          2: ascending, -2: descending w.r.t. transaction size sum)
-j       use quicksort to sort the transactions   (default: heapsort)
-a#      variant of the fpgrowth algorithm to use (default: simple)
-x       do not prune with perfect extensions
-z       do not use head union tail (hut) pruning
         (only for maximal item sets, option -tm)
-b#      blank   characters                       (default: " \t\r")
-f#      field   separators                       (default: " \t,")
-r#      record  separators                       (default: "\n")
-C#      comment characters                       (default: "#")
-!       print additional option information
infile   file to read transactions from           [required]
```

```
outfile  file to write frequent item sets to      [optional]
selfile  file stating a selection of items        [optional]
```

There are options to choose the limits of items per set, the minimum support, evaluation measure, to configure the input and output format, and so on.

A simple calling to FP-Growth, and its results, using the test1.tab example file (that comes with source code) as input file, the test1.out, and minimum support as 30%, could be made as follow:

```
C:\R\exec\fpgrowth\src>fpgrowth -s30 test1.tab test1.out
fpgrowth - find frequent item sets with the fpgrowth algorithm
version 4.10 (2010.10.27)        (c) 2004-2010   Christian Borgelt
reading test1.tab ... [5 item(s), 10 transaction(s)] done [0.00s].
filtering, sorting and recoding items ... [5 item(s)] done [0.00s].
reducing transactions ... [8/10 transaction(s)] done [0.00s].
writing test1.out ... [15 set(s)] done [0.00s].
```

The presented result shows some information about Copyright and some execution data, as the number of items and transactions and the number of frequent set (21 in this example). The content of input and output files is presented below.

The input file content:

```
a b c
a d e
b c d
a b c d
b c
a b d
d e
a b c d
c d e
a b c
```

The output file content:

```
e d  (30.0)
e  (30.0)
a c b  (40.0)
a c  (40.0)
a d b  (30.0)
a d  (40.0)
a b  (50.0)
a  (60.0)
d b c  (30.0)
d b  (40.0)
d c  (40.0)
d  (70.0)
c b  (60.0)
c  (70.0)
b  (70.0)
```

Calling FP-Growth from R

As observed before, to create a package are imposed a several rules creating a standard directory structure and content to make it available an external source code. An alternative presented before is to creating a shared object, or a DLL, to be called using specific R functions (.C, .CALL, and so on).

To start a job of adapt an existing code to compose a package can be a hard job and spending too much time. An interesting approach is to iteratively create and adapt a shared object, or DLL, and make tests to validate it and after improve the adaptations in some iterations, when a satisfactory result has been done, start to work in a package version.

The intent iterations to make it available the C implementation in R are:

1. Create a simple command line call, without parameters making only two changes in the original source (the fpgrowth.c file):
   - Rename the main function to FP-Growth with the same signature;
   - Create a function to be called from R, creating the parameters from a configuration file (containing only a string with the same syntax of the command line, broken it in an array to be used as the argument array to FP-Growth function;
2. Compile the code project within the R compile command, including the R.h reader file and call it using R;
3. Implement the input parameters from the R call, eliminating the usage of a configuration file, including the change to define a input file name to data-frames in R;
4. Preparing the output in a R data-frame to be returned to R;
5. Create the R package.

The first iteration could be done easily, without any surprise.

Unfortunately, the second iteration, that sounds to be ease to be done either, in a practice proved to be very hard. The R compile command does not work with makefiles and the compile original code with it could not be done. After some experiments, the strategy was changed to build a library with the adapted code, without the function created to be called from R, and then create a new code containing this function and making use of the compiled library. Next, calling the new code, compiled as a DLL, from R raises execution errors. Debugging the execution, wasting several time, was detected that some compile configurations to create the library was wrong. To solving this problem, some tests are made creating an executable version to be run using OS command line until all execution errors are solved. However, solved this errors, another unexpected behavior was founded. Calling the version compiled using R command from R console the incompatible cygwin version error was rised in loading DLL function. Several experiments, changing the compilation parameters, different versions of cygwin, and so on were tried, but have no success (these tests are made only under Windows OS). So, having no success in the second iteration, the next step was compromised.

The main expected challenge in third and fourth iterations is to interface the R data types and structure with its correspondents in the C language, either to dataset input and other input parameters to be converted and used internally than to output dataset needed to be created to be returned to R. An alternative is to adapt all the code to use the data received. However, it sounds to be more complex to be done.

The fifth iteration sounds to be a bureaucratic work. Once the code has been entirely adapted and validated, create the additional directory and required content should be an easy task.

Conclusion and Future Works

In this chapter an efficient and scalable algorithm to mine frequent patterns in databases was presented: the FP-Growth. This algorithm uses a useful data structure, the FP-Tree, to store information about frequent patterns. Also an implementation of the algorithm was presented. Additionally, some features of R language and experiments to adapt the algorithm source code to be used in R. We could observe that the job to make this adaptation is hard, and cannot be done in short time. Unfortunately, have no time yet to conclude this adaptation.

As a future work would be interesting to better understand the implementation of external resources on R and complete the job proposed in this work, and after comparing results with other algorithms to mining frequent itemsets available in R.

Appendix A: Examples of R statements

Some basic examples based in .

Getting help about functions

```
> help(summary)
> ?summary
```

Creating an object

```
> perceptual <- 1.2
```

Numeric expressions

```
> z <- 5
> w <- z^2
> y <- (34 + 90) / 12.5
```

Printing an object value

```
> w
[1] 25
```

Creating a vector

```
> v <- c(4, 7, 23.5, 76.2, 80)
```

Vector operations

```
> x <- sqrt(v)
> v1<- c(4, 6, 87)
> v2 <- c(34, 32.4, 12)
> v1 + v2
[1] 38.0 38.4 99.0
```

Categhoric data

```
> s <- c("f", "m", "m", "f")
> s
[1] "f" "m" "m" "f"
> s <- factor(s)
> s
[1] f m m f
```

```
Levels: f m
> table(s)
s
f m
2 2
```

Sequences

```
> x <- 1:1000
> y <- 5:0
> z <- seq(-4, 1, 0.5) # create a sequence starting in -4, stopping in 1
                       # with an increment of 0.5
> w <- rnorm(10) # create a random sequence of 10 numeric values
> w <- rnorm(10, mean = 10, sd = 3) # create a normal distribution of
                                    # 10 numeric values with mean of 10
                                    # and standard deviation of 3
```

Matrices

```
> m1 <- matrix(c(45,  23,  66,  77,  33,  44),  2,  3)
> m1
        [,1]   [,2]   [,3]
[1,]     45     66     33
[2,]     23     77     44
> m2 <- matrix(c(12,  65,  32,  7,  4,  78),  2,  3)
> m2
        [,1]   [,2]   [,3]
[1,]     12     32      4
[2,]     65      7     78
> m1 + m2
        [,1]   [,2]   [,3]
[1,]     57     98     37
[2,]     88     84    122
```

Lists

```
> student <- list(nro = 34453, name = "Marie", scores = c(9.8, 5.7, 8.3))
> student[[1]]
[1] 34353
> student$nro
[1] 34353
```

Data Frames (represents database tables)

```
> scores.inform <- data.frame(nro = c(2355, 3456, 2334, 5456),
+ team = c("tp1", "tp1", "tp2", "tp3"),
+ score = c(10.3, 9.3, 14.2, 15))
> scores.inform
    nro    team   score
1  2355     tp1    10.3
2  3456     tp1     9.3
```

```
3  2334     tp2    14.2
4  5456     tp3    15.0
> scores.inform[score > 14,]
    nro    team   score
3  2334     tp2    14.2
4  5456     tp3    15.0
> team
[1]  tp1  tp1  tp2  tp3
Levels:  tp1  tp2  tp3
```

Conditional statement

```
> if (x > 0) y <- z / x else y <- z
> if (x > 0) {
+    cat('x is positive.\n')
+    y <- z / x
+} else {
+    cat('x isn't positive!\n')
+    y <- z
+}
```

Case statement

```
> sem <- "green"
> switch(sem, green = "continue", yellow = "attention", red = "stop")
[1] "continue"
```

Loop statements

```
> x <- rnorm(1)
> while (x < -0.3) {
+    cat("x=", x, "\t")
+    x <- rnorm(1)
+  }

> text  <-  c()
> repeat  {
+   cat('Type a phrase?  (empty to quit) ')
+   fr  <-  readLines(n=1)
+   if  (fr  ==  '')  break  else  texto  <-  c(texto,fr)
+}

> x <- rnorm(10)
> k <- 0
> for(v in x)  {
+   if(v > 0)
+     y <- v
+   else y <- 0
+     k <- k + y
+  }
```

Creating and calling functions

```
> cel2far <- function(cel) {
+   res  <-  9/5  *  cel  +  32
+   res
+ }
> cel2far(27.4)
[1]  81.32
> cel2far(c(0,  -34.2,  35.6,  43.2))
[1]  32.00  -29.56    96.08  109.76
```

References

[1] J. Han, H. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In: Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX). ACM Press, New York, NY, USA 2000.

[2] Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. In Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), Santiago, Chile, pp. 487–499.

[3] Agarwal, R., Aggarwal, C., and Prasad, V.V.V. 2001. A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing, 61:350–371.

[4] B.Santhosh Kumar and K.V.Rukmani. Implementation of Web Usage Mining Using APRIORI and FP Growth Algorithms. Int. J. of Advanced Networking and Applications, Volume: 01, Issue:06, Pages: 400-404 (2010).

[5] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97), 283–296. AAAI Press, Menlo Park, CA, USA 1997.

[6] Christian Borgelt. Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination. Workshop Open Source Data Mining Software (OSDM'05, Chicago, IL), 66-70. ACM Press, New York, NY, USA 2005

[7] Aiman Moyaid, Said and P.D.D., Dominic and Azween, Abdullah. A Comparative Study of FP-growth Variations. international journal of computer science and network security, 9 (5). pp. 266-272.

[8] Liu,G. , Lu ,H. , Yu ,J. X., Wang, W., & Xiao, X.. AFOPT: An Efficient Implementation of Pattern Growth Approach, In Proc. IEEE ICDM'03 Workshop FIMI'03, 2003.

[9] Grahne, G. , & Zhu, J. Fast Algorithm for frequent Itemset Mining Using FP-Trees. IEEE Transactions on Knowledge and Data Engineer,Vol.17,NO.10, 2005.

[10] Gao, J. Realization of new Association Rule Mining Algorithm. Int. Conf. on Computational Intelligence and Security ,IEEE, 2007.

[11] Cornelia Gyorödi, Robert Gyorödi, T. Cofeey & S. Holban. Mining association rules using Dynamic FP-trees. in Proceedings of The Irish Signal and Systems Conference, University of Limerick, Limerick, Ireland, 30th June 2nd July 2003, ISBN 0-9542973-1-8, pag. 76-82.

[12] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Exante: Anticipated data reduction in constrained pattern mining. In Proc. of PKDD03.

[13] Balázes Rácz. Nonordfp: An FP-Growth Variation without Rebuilding the FP-Tree. 2nd Int'l Workshop on Frequent Itemset Mining Implementations FIMI2004.

[14] Grahne O. and Zhu J. Efficiently Using Prefix-trees in Mining Frequent Itemsets, In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining, 2004.

[15] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques. 2nd edition, Morgan Kaufmann, 2006.

[16] M. Zaki and W. Meira Jr. Fundamentals of Data Mining Algorithms, Cambridge, 2010 (to be published)

[17] Grahne, G. , & Zhu, J. Fast Algorithm for frequent Itemset Mining Using FP-Trees. IEEE Transactions on Knowledge and Data Engineer,Vol.17,NO.10, 2005.

[18] http://cran.r-project.org/doc/manuals/R-data.html

[19] W. N. Venables, D. M. Smith and the R Development Core Team. An Introduction to R: Notes on R: A Programming Environment for Data Analysis and Graphics. Version 2.11.1 (2010-05-31).

[20] R Development Core Team. R Language Definition. Version 2.12.0 (2010-10-15) DRAFT.

[21] R Development Core Team. Writing R Extensions. Version 2.12.0 (2010-10-15).

[22] https://r-forge.r-project.org/projects/arules/

[23] Michael Hahsler and Bettina G and Kurt Hornik and Christian Buchta. Introduction to arules – A computational environment for mining association rules and frequent item sets. March 2010.

[24] Luís Torgo. Introdução à Programação em R. Faculdade de Economia, Universidade do Porto, Outubro de 2006.

[25] Sigal Blay. Calling C code from R an introduction. Dept. of Statistics and Actuarial Science Simon Fraser University, October 2004.

[26] http://www.murdoch-sutherland.com/Rtools/

[27] http://cran.fiocruz.br/web/packages/arulesNBMiner/index.html

[28] http://cran.r-project.org/web/packages/arulesSequences/index.html

[29] http://cran.r-project.org/web/packages/cluster/index.html
[30] http://cran.r-project.org/doc/manuals/R-admin.html
[31] http://www.borgelt.net//fpgrowth.html
[32] Christian Borgelt. An Implementation of the FP-growth Algorithm. Workshop Open Source Data Mining Software (OSDM'05, Chicago, IL), 1-5. ACM Press, New York, NY, USA 2005.

SPADE

Frequent Sequence Mining is used to discover a set of patterns shared among objects which have between them a specific order. For instance, a retail shop may possess a transaction database which specifies which products were acquired by each customer over time. In this case, the store may use Frequent Sequence Mining to find that 40% of the people who bought the first volume of Lord of the Rings came back to buy the second volume a month later. This kind of information may be used to support directed advertising campaigns or recommendation systems.

Another application domain where Frequent Sequence Mining may be used is Web click log analysis in Information Retrieval systems, in which case the system performance may be refined by analyzing the sequence of interactions that the user exposed while searching or browsing for a specific information. This kind of usage becomes especially clear when we consider the huge amount of data obtained by industrial search engines in the form of query logs. Google alone was reported to answer 5.42 billion queries during December 2008 (Telecom Paper, 2009)

In biology Frequent Sequence Mining may be used to extract information hidden in DNA sequences. Sequence databases in biology are often huge and complex due to variations from genetic mutations and evolution (Li et al., 2007). For example, Frequent Sequence Mining can be used to extract patterns which may be determinant to the development of genetic conditions.

A sequence α is an ordered list of events <a1,a2,...,am>. An event is a non-empty unordered set of items ai $\subseteq$ i1,i2,...,ik. A sequence α = <a1,a2,...,am> is a subsequence of β = < b1, b2,...,bn > if and only if exists i1,i2,...,im such that 1 $\leq$ i1 < i2 < ... < im $\leq$ n and a1 $\subseteq$ bi1, a2 $\subseteq$ bi2 and am $\subseteq$ bim. Given a sequence database D = s1,s2,...,sn, the support of a sequence α is the number of sequences of D which contains α as a subsequence. If the support of α is bigger than a threshold maxsup, then α is a frequent sequence (Peng and Liao, 2009).

Algorithm

An algorithm to Frequent Sequence Mining is the SPADE (Sequential PAttern Discovery using Equivalence classes) algorithm. It uses a vertical id-list database format, where we associate to each sequence a list of objects in which it occurs. Then, frequent sequences can be found efficiently using intersections on id-lists. The method also reduces the number of databases scans, and therefore also reduces the execution time.

The first step of SPADE is to compute the frequencies of 1-sequences, which are sequences with only one item. This is done in a single database scan. The second step consists of counting 2-sequences. This is done by transforming the vertical representation into an horizontal representation in memory, and counting the number of sequences for each pair of items using a bidimensional matrix. Therefore, this step can also be executed in only one scan.

Subsequent n-sequences can than be formed by joining (n-1)-sequences using their id-lists. The size of the id-lists is the number of sequences in which an item appears. If this number is greater than minsup, the sequence is a frequent one. The algorithm stops when no frequent sequences can be found anymore. The algorithm can use a breadth-first or a depth-first search method for finding new sequences (Zaki, 2001)

Implementation

The first step is to install the arulesSequences package (Buchta and Hahsler, 2010).

```
> install.packages("arules")
> install.packages("arulesSequences")
```

To illustrate the use of CSPADE, we are going to use an example file that can be found inside the package arulesSequence. This file is listed bellow:

```
$ cat /usr/local/lib/R/site-library/arulesSequences/misc/zaki.txt
1 10 2 C D
1 15 3 A B C
1 20 3 A B F
1 25 4 A C D F
2 15 3 A B F
2 20 1 E
3 10 3 A B F
4 10 3 D G H
4 20 2 B F
4 25 3 A G H
```

Each line in the zaki.txt file is an event. The first column is the sequence id, that is, the sequence to which this event belongs. The second column is the event timestamp, which is the moment in time when the event has occurred. The third column is the number n of items in the event, and it is followed by n additional columns, one for each item.

First, we need to load the necessary packages:

```
> library(Matrix)
> library(arules)
> library(arulesSequences)
```

To read the zaki.txt file, issue the following commands:

```
> x <- read_baskets(con = system.file("misc", "zaki.txt", package = "arulesSequences"), info = c("sequenceID","eventID","SIZE"))
> as(x, "data.frame")


       items sequenceID eventID SIZE
1      {C,D}          1      10    2
2    {A,B,C}          1      15    3
3    {A,B,F}          1      20    3
4  {A,C,D,F}          1      25    4
5    {A,B,F}          2      15    3
6        {E}          2      20    1
7    {A,B,F}          3      10    3
8    {D,G,H}          4      10    3
9      {B,F}          4      20    2
10   {A,G,H}          4      25    3
```

Next, we execute the CSPADE algorithm:

```
> s1 <- cspade(x, parameter = list(support = 0.4), control = list(verbose = TRUE))
```

Note that we executed the cpade algorithm with the data contained in zaki object. We have set the support parameter to 0.4, and also have instructed the algorithm to show a verbose output.

The algorithm output will be the following:

```
preprocessing ... 1 partition(s), 0 MB [0.046s]
mining transactions ... 0 MB [0.022s]
reading sequences ... [0.07s]

total elapsed time: 0.138s
```

Visualization

We can use the command summary and as to see the results:

```
cspade> summary(s1)
set of 18 sequences with

most frequent items:
      A       B       F       D (Other)
     11      10      10       8      28

most frequent elements:
    {A}     {D}     {B}     {F}   {B,F} (Other)
      8       8       4       4       4       3

element (sequence) size distribution:
sizes
1 2 3
8 7 3

sequence length distribution:
lengths
1 2 3 4
4 8 5 1

summary of quality measures:
    support
 Min.   :0.5000
 1st Qu.:0.5000
 Median :0.5000
 Mean   :0.6528
 3rd Qu.:0.7500
 Max.   :1.0000

mining info:
 data ntransactions nsequences support
    x            10          4     0.4

cspade> as(s1, "data.frame")
```

```
          sequence support
1            <{A}>    1.00
2            <{B}>    1.00
3            <{D}>    0.50
4            <{F}>    1.00
5          <{A,F}>    0.75
6          <{B,F}>    1.00
7        <{D},{F}>    0.50
8      <{D},{B,F}>    0.50
9        <{A,B,F}>    0.75
10         <{A,B}>    0.75
11       <{D},{B}>    0.50
12       <{B},{A}>    0.50
13       <{D},{A}>    0.50
14       <{F},{A}>    0.50
15   <{D},{F},{A}>    0.50
16     <{B,F},{A}>    0.50
17 <{D},{B,F},{A}>    0.50
18   <{D},{B},{A}>    0.50
```

This output shows (1) the list of the most frequent isolated items (A,B, ..), (2) the list of the most frequent set of items that occur in events (referred to as elements), (3) the distribution of the sizes of the set of items, (4) the distribution of the number of events in a sequence (referred to as sequence length), (5) the minimum, maximum, mean and median support values, and (6) the set of frequent sequences mined ordered by its support value.

Case Study

Scenario

In this case study we analyse the application of the CSPADE algorithm to a Tag Recommendation problem. Tags are keywords assigned by users to items in the context of the Web 2.0. A Tag Recommendation System is used to suggest new tags to users with the objective of enhancing their browsing experience and enrich item description. The dataset used in this case study was obtained from Delicious using its public time-line, which shows bookmarks from all the system users during a given period of time. The SPADE algorithms were executed and some results are presented in the discussion bellow.

Datasets

The dataset used in this case study was collected from Delicious in October 2009. We collected periodically the Delicious [1] public time-line which shows the bookmarks from all the system users. Each bookmark consists of a user, the URL which was bookmarked, and a set of tags that user chose to describe the URL.

We show some bookmark examples:

```
gmenezes http://www.traderslog.com/forum/ 5 education investment forex CFD trading

gmenezes http://bikebins.com/index.html 6 pannier bike bicycle cycling bikes commuting

osvaldo http://www.noycefdn.org/ecrwresources.php 6 literacy math foundation education webdesign professionaldevelopment
```

In this example, the user *gmenezes* has bookmarked the URL http://www.traderslog.com/forum/" *and used 5 tags to describe its content:* education investment forex CFD trading.

The public time-line shows the system bookmarks in a time sequence, so that we can obtain sequences of bookmarks for specific users. We can use these sequences as input for the CSPADE algorithm. In this case, each bookmark is an event, and each tag is an item. In other words, we will mine temporal patterns with the objective of generating rules that predict useful tags for a specific user by using the wisdom of crowds.

Before appling the algorithm to the data we had to perform some pre-processing in our raw data. First, we extracted a sample, which consisted of 31,289 sequential bookmarks. Next, we performed data cleansing and duplicate removal. We ended up with 7,559 sequential bookmarks. We grouped each user`s bookmarks as sequences and each individual bookmark as an event. For instance, the example above yields:

```
1 1 5 education investment forex CFD trading
1 2 6 pannier bike bicycle cycling bikes commuting
2 1 6 literacy math foundation education webdesign professionaldevelopment
```

The dataset used can be downloaded from here [2].

Execution

We used the dataset described above in the experiments. To execute the algorithm we first execute *read_baskets()* to load the dataset file from disk as temporal transaction data. Note that we need to load the required libraries (as above).

```
n <- read_baskets(con = system.file("misc", "delicious.sequence", package = "arulesSequences"), info = c("sequenceID","eventID","SIZE"))
```

We can see data statistics with the command *summary()*:

```
> summary(n)
transactions as itemMatrix in sparse format with
 7559 rows (elements/itemsets/transactions) and
 7496 columns (items) and a density of 0.0004482878

most frequent items:
     design       tools        blog   webdesign inspiration     (Other)
        469         301         233         229         220       23949

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
2283 1432 1172  825  560  343  230  273  171  100   60   34   25   14    5    5
  17   18   19   20
   5    7    8    7

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.00    3.00    3.36    4.00   20.00

includes extended item information - examples:
  labels
1      |
2      -
3      ,
```

```
includes extended transaction information - examples:
  sequenceID eventID SIZE
1          1       1    2
2          2       1    5
3          3       1    3
```

Next, we execute *cspade()* to generate the rules. We have set a support of 0.002 to obtain a larger number of patterns.

```
s1 <- cspade(n, parameter = list(support = 0.002), control = list(verbose = TRUE))
```

Caveats

The use of higher support levels in this dataset does not generate any rule and can cause non-intuitive errors. For example, if the support is set to 0.1, the system outputs:

```
> s1 <- cspade(n, parameter = list(support = 0.1), control = list(verbose = TRUE))
parameter specification:
support : 0.1
maxsize :  10
maxlen  :  10

algorithmic control:
bfstype : FALSE
verbose :  TRUE
summary : FALSE

preprocessing ... 1 partition(s), 0.18 MB [0.05s]
mining transactions ... can't open data file: No such file or directory
Error in cspade(n, parameter = list(support = 0.1), control = list(verbose = TRUE)) :
  system invocation failed
```

Output

We can see the generated rules by issuing the command *as()*.

```
as(s1, "data.frame")
(...)
845                         <{webdesign},{design}> 0.004675877
846             <{inspiration,webdesign},{design}> 0.001912859
847                  <{design,webdesign},{design}> 0.004250797
848                 <{design,typography},{design}> 0.002337938
849                      <{design,tools},{design}> 0.002337938
850         <{inspiration},{inspiration},{design}> 0.001912859
851              <{inspiration},{design},{design}> 0.002125399
852                <{design,inspiration},{design}> 0.004675877
853              <{design},{inspiration},{design}> 0.001912859
854                   <{inspiration,art},{design}> 0.001912859
855 <{design,inspiration},{inspiration},{design}> 0.001912859
856      <{design},{design,inspiration},{design}> 0.001912859
857                    <{design},{design},{design}> 0.004250797
858                       <{design,blog},{design}> 0.002337938
```

```
859                        <{design,art},{design}> 0.002550478
860         <{design},{design},{design},{design}> 0.002337938
861                       <{design},{design,blog}> 0.001912859
862                            <{design,art,blog}> 0.001912859
863                        <{design},{design,art}> 0.002763018
864                           <{art},{design,art}> 0.002125399
865                                <{culture,art}> 0.001912859
866                                  <{css},{css}> 0.001912859
867                               <{design},{css}> 0.002125399
868                                 <{blog,blogs}> 0.003400638
869                               <{blog,blogging}> 0.001912859
(...)
```

To see the complete set of rules, download it from here [3].

Analysis

We have observed that CSPADE found many trivial sequences from user behaviour. For example, it has found many unitary sequences, such as <{design}>, <{ajax}>, <{css}>, among others. These unitary sequences are really frequently used, but they may not be useful in the particular application, which is Tag Recommendation.

Furthermore, other trivial sequences were found, such as <{design}.{design}> and <{webdesign},{design}>. These sequences indicates that the same users tend to bookmarks pages in the same subject subsequently. However, some interesting patters were also found. We can cite <{library},{books}>, <{javascript},{ajax}> and <{video},{youtube}>.

We can also observe that many frequent patterns are related to *design*, *art* and *web_development*. These tags are also the most popular tags in the whole Delicious system, as can be seen here [4].

References

1. ^ Buchta, C., Hahsler, M., 2010. "arulesSequences: Mining frequent sequences". R package version 0.1-11. http://CRAN.R-project.org/package=arulesSequences
2. ^ Li, K.-C., Chang, D.-J., Rouchka, E. C., Chen, Y. Y., 2007. "Biological sequence mining using plausible neural network and its application to exon/intron boundaries prediction". In: CIBCB. IEEE, pp. 165–169.
3. ^ Peng, W.-C., Liao, Z.-X., 2009. "Mining sequential patterns across multiple sequence databases". Data Knowl. Eng. 68 (10), 1014–1033.
4. ^ Telecom Paper, January 2009. "Google query volume [5]".
5. ^ Zaki, M. J., 2001. "Spade: An efficient algorithm for mining frequent sequences [6]". In: Machine Learning. Vol. 42. pp. 31–60.

References

[1] http://www.delicious.com
[2] http://homepages.dcc.ufmg.br/~gmenezes/delicious-sequence/delicious-sequence.txt
[3] http://homepages.dcc.ufmg.br/~gmenezes/delicious-sequence/delicious-out.txt
[4] http://delicious.com/tag
[5] http://www.telecompaper.com/news/article.aspx?cid=653579
[6] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.113.6042

DEGSeq

Working with DEGseq

Introduction

First we have to choose the way to organize your data. There is two ways to do that: using the MA-plot-based method with random sampling model or MA-plot-based method with technical replicates. **MA-plot-based method with random sampling model**

The MA-plot is a statistical analysis tool that having been used to detect and visualize intensity-dependent ratio of microarray data. Let C1 and C2 denote the counts of reads mapped to a specific gene obtained from two samples, with Ci ~ binomial(ni, pi), I = 1,2, where ni denotes the total number of mapped reads and pi denotes the probability of a read coming from that gene. We define M = log2C1 - log2C2, and A = (log2C1 + log2C2)/2. It can be proven that under the random sampling assumption the conditional distribution of M given that A = a (a is an observation of A), follows an approximate normal distribution. For each gene on the MA-plot, we do the hypothesis test of H0: p1 = p2 versus H1: p1 ≠ p2. Then a P-value could be assigned based on the conditional normal distribution.

MA-plot-based method with technical replicates

Though it has been reported that sequencing platform has low background noise, technical replicates would still be informative for quality control and to estimate the variation due to different machines or platforms. MA-plot-based is a another method which estimates the noise level by comparing technical replicates in the data (if available). In this method, a sliding-window is first applied on the MA-plot of the two technical replicates along the A-axis to estimate the random variation corresponding to different expression levels. A smoothed estimate of the intensity-dependent noise level is done by loess regression, and converted to local standard deviations of M conditioned on A, under the assumption of normal distribution. The local standard deviations are then used to identify the difference of the gene expression between the two samples.

Multiple testing correction

For the above methods, the P-values calculated for each gene are adjusted to Q-values for multiple testing corrections. Users can set either a P-value or a false discovery rate (FDR) threshold to identify differentially expressed genes.

Once you choose your data, you can apply the R package DEGseq. There are five different methods to run the program. They are DEGexp, DEGseq, readGeneExp, samWrapper and getGeneExp.

Functions of Package

DEGexp

Description

This function is used to identify differentially expressed genes when users already have the gene expression values (such as the number of reads mapped to each gene).

Usage

```
DEGexp( geneExpFile1, geneCol1=1, expCol1=2, depth1=rep(0, length(expCol1) ),
groupLabel1="    geneExpFile2,    geneCol2=1,    expCol2=2,    depth2=rep(0,
length(expCol2) ), groupLabel2=" header=TRUE, sep="", method=c("LRT", "CTR",
"FET",   "MARS",   "MATR",   "FC"),   pValue=1e-3,   zScore=4,   qValue=1e-3,
foldChange=4,   thresholdKind=1,   outputDir="none",   normalMethod=c("none",
"loess", "median"), replicate1="none", geneColR1=1, expColR1=2, depthR1=rep(0,
length(expColR1) ), replicate2="none", geneColR2=1, expColR2=2, depthR2=rep(0,
length(expColR2) ), rawcount=TRUE )
```

Arguments

Argument	Description
geneExpFile1	file containing gene expression values for replicates of sample1 (or replicate1 when method="CTR").
geneCol1	gene id column in geneExpFile1.
expCol1	expression value columns in geneExpFile1 for replicates of sample1 (numeric vector). Note: Each column corresponds to a replicate of sample1.
depth1	the total number of reads uniquely mapped to genome for each replicate of sample1 (numeric vector), default: take the total number of reads mapped to all annotated genes as the depth for each replicate.
groupLabel1	label of group1 on the plots.
GeneExpFile2	file containing gene expression values for replicates of sample2 (or replicate2 when method="CTR").
geneCol2	gene id column in geneExpFile2.
expCol2	expression value columns in geneExpFile2 for replicates of sample2 (numeric vector). Note: Each column corresponds to a replicate of sample2.
depth2	the total number of reads uniquely mapped to genome for each replicate of sample2 (numeric vector), default: take the total number of reads mapped to all annotated genes as the depth for each replicate.
groupLabel2	label of group2 on the plots.
header	a logical value indicating whether geneExpFile1 and geneExpFile2 contain the names of the variables as its first line.
sep	the field separator character. If sep = "" (the default for read.table) the separator is white space, that is one or more spaces, tabs, newlines or carriage returns.
method	method to identify differentially expressed genes. Possible methods are: • "LRT": Likelihood Ratio Test (Marioni et al. 2008),• "CTR": Check whether the variation between Technical Replicates can be explained by the random sampling model (Wang et al. 2009), • "FET": Fisher's Exact Test (Joshua et al. 2009), • "MARS": MA-plot-based method with Random Sampling model (Wang et al. 2009), • "MATR": MA-plot-based method with Technical Replicates (Wang et al.2009), • "FC" : Fold-Change threshold on MA-plot.
pValue	pValue threshold (for the methods: LRT, FET, MARS, MATR). only used when thresholdKind=1.
zScore	zScore threshold (for the methods: MARS, MATR). only used when thresholdKind=2.
qValue	qValue threshold (for the methods: LRT, FET, MARS, MATR). only used when thresholdKind=3 or thresholdKind=4.
thresholdKind	the kind of threshold. Possible kinds are: • 1: pValue threshold, • 2: zScore threshold, • 3: qValue threshold (Benjamini et al. 1995), • 4: qValue threshold (Storey et al. 2003).

foldChange	fold change threshold on MA-plot (for the method: FC).
outputDir	the output directory.
normalMethod	the normalization method: "none", "loess", "median". recommend: "none".
replicate1	file containing gene expression values for replicate batch1 (only used when method="MATR"). Note: replicate1 and replicate2 are two (groups of) technical replicates of a sample.
GeneColR1	gene id column in the expression file for replicate batch1 (only used when method="MATR").
expColR1	expression value columns in the expression file for replicate batch1 (numeric vector) (only used when method="MATR").
depthR1	the total number of reads uniquely mapped to genome for each replicate in replicate batch1 (numeric vector), default: take the total number of reads mapped to all annotated genes as the depth for each replicate (only used when method="MATR").
ReplicateLabel1	label of replicate batch1 on the plots (only used when method="MATR").
replicate2	file containing gene expression values for replicate batch2 (only used when method="MATR"). Note: replicate1 and replicate2 are two (groups of) technical replicates of a sample.
geneColR2	gene id column in the expression file for replicate batch2 (only used when method="MATR").
expColR2	expression value columns in the expression file for replicate batch2 (numeric vector) (only used when method="MATR").
depthR2	the total number of reads uniquely mapped to genome for each replicate in replicate batch2 (numeric vector), default: take the total number of reads mapped to all annotated genes as the depth for each replicate (only used when method="MATR").
ReplicateLabel2	label of replicate batch2 on the plots (only used when method="MATR").
rawCount	a logical value indicating the gene expression values are based on raw read counts or normalized values.

Example:

```
> library(DEGseq)
> geneExpFile <- system.file("data", "GeneExpExample5000.txt",
+ package = "DEGseq")
> if (geneExpFile == "") {
+ zipFile <- system.file("data", "Rdata.zip", package = "DEGseq")
+ if (zipFile != "") {
+ unzip(zipFile, "GeneExpExample5000.txt", exdir = tempdir())
+ geneExpFile <- file.path(tempdir(), "GeneExpExample5000.txt")
+ }
+ }
> layout(matrix(c(1, 2, 3, 4, 5, 6), 3, 2, byrow = TRUE))
> par(mar = c(2, 2, 2, 2))
> DEGexp(geneExpFile1 = geneExpFile, expCol1 = c(7, 9, 12, 15, 18),
groupLabel1 = "kidney", geneExpFile2 = geneExpFile, expCol2 = c(8, 10,
11, 13, 16), groupLabel2 = "liver", method = "MARS")

Please wait...

geneExpFile1:
D:/myrpackage/DEGseq.Rcheck/DEGseq/data/GeneExpExample5000.txt
gene id column in geneExpFile1: 1
expression value column(s) in geneExpFile1: 7 9 12 15 18
total number of reads uniquely mapped to genome obtained from sample1:
345504 354981 334557
```

```
geneExpFile2:
D:/myrpackage/DEGseq.Rcheck/DEGseq/data/GeneExpExample5000.txt
gene id column in geneExpFile2: 1
expression value column(s) in geneExpFile2: 8 10 11 13 16
total number of reads uniquely mapped to genome obtained from sample2:
274430 274486 264999

method to identify differentially expressed genes: MARS
pValue threshold: 0.001
output directory: none
The outputDir is not specified!

Please wait ...
Identifying differentially expressed genes ...
Please wait patiently ...
output ...
The results can be observed in directory: none
```

From Wang et. Al (2009) [1]

DEGseq

Description

This function is used to identify differentially expressed genes from RNA-seq data. It takes uniquely mapped reads from RNA-seq data for the two samples with a gene annotation as input. So users should map the reads (obtained from sequencing libraries of the samples) to the corresponding genome in advance.

Usage

DEGseq (mapResultBatch1, mapResultBatch2, fileFormat="bed", readLength=32, strandInfo=FALSE, refFlat, groupLabel1="group1", groupLabel2="group2", method=c("LRT", "CTR", "FET", "MARS", "MATR", "FC"), pValue=1e-3, zScore=4, qValue=1e-3, foldChange=4, thresholdKind=1, outputDir="none", normalMethod=c("none", "loess", "median"), depthKind=1, replicate1="none", replicate2="none", replicateLabel1="replicate1", replicateLabel2="replicate2")

Arguments

Argument	Description
mapResultBatch1	uniquely mapping result files for technical replicates of sample1 (or replicate1 when method="CTR").
MapResultBatch2	uniquely mapping result files for technical replicates of sample2 (or replicate2 when method="CTR").
fileFormat	file format: "bed" or "eland". example of "bed" format: chr12 7 38 readID 2 example of "eland" format: readID chr12.fa 7 U2 F Note: The field separator character is TAB. And the files must follow the format as one of the examples.
ReadLength	the length of the reads (only used if fileFormat="eland"). strandInfo whether the strand information was retained during the cloning of the cDNAs. "TRUE" : retained, "FALSE": not retained.
RefFlat	gene annotation file in UCSC refFlat format.
GroupLabel1	label of group1 on the plots.
GroupLabel2	label of group2 on the plots.

Method	method to identify differentially expressed genes. Possible methods are: "LRT": Likelihood Ratio Test, "CTR": Check whether the variation between two Technical Replicates can be explained by the random sampling model, "FET": Fisher's Exact Test, "MARS": MA-plot-based method with Random Sampling model, "MATR": MA-plot-based method with Technical Replicates, "FC" : Fold-Change threshold on MA-plot.
pValue	pValue threshold (for the methods: LRT, FET, MARS, MATR). only used when thresholdKind=1.
zScore	zScore threshold (for the methods: MARS, MATR). only used when thresholdKind=2.
qValue	qValue threshold (for the methods: LRT, FET, MARS, MATR). only used when thresholdKind=3 or thresholdKind=4.
ThresholdKind	the kind of threshold. Possible kinds are: • 1: pValue threshold, • 2: zScore threshold, • 3: qValue threshold, • 4: qValue threshold .
foldChange	fold change threshold on MA-plot (for the method: FC).
outputDir	the output directory.
normalMethod	the normalization method: "none", "loess", "median". recommend: "none".
DepthKind	1- take the total number of reads uniquely mapped to genome as the depth for each replicate, 0: take the total number of reads uniquely mapped to all annotated genes as the depth for each replicate. We recommend taking depthKind=1, especially when the genes in annotation file are part of all genes.
replicate1	files containing uniquely mapped reads obtained from replicate batch1 (only used when method="MATR").
replicate2	files containing uniquely mapped reads obtained from replicate batch2 (only used when method="MATR").
ReplicateLabel1	label of replicate batch1 on the plots (only used when method="MATR").
ReplicateLabel2	label of replicate batch2 on the plots (only used when method="MATR").

Example:

> kidneyR1L1 <- system.file("data", "kidneyChr21.bed.txt", package = "DEGseq") > liverR1L2 <- system.file("data", "liverChr21.bed.txt", package = "DEGseq") > refFlat <- system.file("data", "refFlatChr21.txt", package = "DEGseq") > mapResultBatch1 <- c(kidneyR1L1) > mapResultBatch2 <- c(liverR1L2) > outputDir <- file.path(tempdir(), "DEGseqExample") > DEGseq(mapResultBatch1, mapResultBatch2, fileFormat = "bed",refFlat = refFlat, outputDir = outputDir, method = "LRT")

Please wait...

mapResultBatch1: D:/myrpackage/DEGseq.Rcheck/DEGseq/data/kidneyChr21.bed.txt mapResultBatch2: D:/myrpackage/DEGseq.Rcheck/DEGseq/data/liverChr21.bed.txt file format: bed refFlat: D:/myrpackage/DEGseq.Rcheck/DEGseq/data/refFlatChr21.txt Ignore the strand information when count the reads mapped to genes! Count the number of reads mapped to each gene ... This will take several minutes, please wait patiently!

Please wait...

SampleFiles: D:/myrpackage/DEGseq.Rcheck/DEGseq/data/kidneyChr21.bed.txt Count the number of reads mapped to each gene. This will take several minutes.

Please wait …

total 259 unique genes processed 0 reads (kidneyChr21.bed.txt) processed 10000 reads (kidneyChr21.bed.txt) processed 20000 reads (kidneyChr21.bed.txt) processed 30000 reads (kidneyChr21.bed.txt) processed 34304 reads (kidneyChr21.bed.txt) total used 0.328000 seconds!

Please wait...

SampleFiles: D:/myrpackage/DEGseq.Rcheck/DEGseq/data/liverChr21.bed.txt Count the number of reads mapped to each gene. This will take several minutes.

Please wait …

total 259 unique genes processed 0 reads (liverChr21.bed.txt) processed 10000 reads (liverChr21.bed.txt) processed 20000 reads (liverChr21.bed.txt) processed 30000 reads (liverChr21.bed.txt) processed 30729 reads (liverChr21.bed.txt) total used 0.297000 seconds!

Please wait...

geneExpFile1: C:\DOCUME~1\wanglk\LOCALS~1\Temp\RtmpUvf7Fw/DEGseqExample/group1.exp gene id column in geneExpFile1: 1 expression value column(s) in geneExpFile1: 2 total number of reads uniquely mapped to genome obtained from sample1: 34304 geneExpFile2: C:\DOCUME~1\wanglk\LOCALS~1\Temp\RtmpUvf7Fw/DEGseqExample/group2.exp gene id column in geneExpFile2: 1 expression value column(s) in geneExpFile2: 2 total number of reads uniquely mapped to genome obtained from sample2: 30729 method to identify differentially expressed genes: LRT pValue threshold: 0.001 output directory: C:\DOCUME~1\wanglk\LOCALS~1\Temp\RtmpUvf7Fw/DEGseqExample

Please wait …

Identifying differentially expressed genes ... Please wait patiently ... output ...

Done ... The results can be observed in directory: C:\DOCUME~1\wanglk\LOCALS~1\Temp\RtmpUvf7Fw/DEGs

getGeneExp

Description

This function is used to count the number of reads and calculate the RPKM for each gene. It takes uniquely mapped reads from RNA-seq data for a sample with a gene annotation file as input. So users should map the reads (obtained from sequencing library of the sample) to the corresponding genome in advance.

Usage

getGeneExp(mapResultBatch, fileFormat="bed", readLength=32, strandInfo=FALSE, refFlat, output=paste(mapResultBatch[1],".exp",sep=""), min.overlapPercent= 1)

Arguments

Argument	Description
mapResultBatch	a vector containing uniquely mapping result files for a sample. Note: The sample can have multiple technical replicates.
fileFormat	file format: "bed" or "eland". example of "bed" format: chr12 7 38 readID 2 example of "eland" format: readID chr12.fa 7 U2 F Note: The field separator character is TAB. And the files must follow the format as one of the examples.
readLength	the length of the reads (only used if fileFormat="eland"). strandInfo whether the strand information was retained during the cloning of the cDNAs. • "TRUE" : retained, • "FALSE": not retained.
refFlat	gene annotation file in UCSC refFlat format.
output	the output file.
min.overlapPercent	the minimum percentage of the overlapping length for a read and an exon over the length of the read itself, for counting this read from the exon. should be <=1. 0: at least 1 bp overlap between a read and an exon.

Example:

> kidneyR1L1 <- system.file("data", "kidneyChr21.bed.txt", package = "DEGseq") > refFlat <- system.file("data", "refFlatChr21.txt", package = "DEGseq") > mapResultBatch <- c(kidneyR1L1) > output <- file.path(tempdir(), "kidneyChr21.bed.exp") > getGeneExp(mapResultBatch, refFlat = refFlat, output = output) Please wait... SampleFiles: D:/myrpackage/DEGseq.Rcheck/DEGseq/data/kidneyChr21.bed.txt Count the number of reads mapped to each gene. This will take several minutes. Please wait ... total 259 unique genes processed 0 reads (kidneyChr21.bed.txt) processed 10000 reads (kidneyChr21.bed.txt) processed 20000 reads (kidneyChr21.bed.txt) processed 30000 reads (kidneyChr21.bed.txt) processed 34304 reads (kidneyChr21.bed.txt) total used 0.328000 seconds! > exp <- readGeneExp(file = output, geneCol = 1, valCol = c(2, + 3), label = c("raw count", "RPKM")) >

exp[30:32,] raw count RPKM C21orf131 0 0.000 C21orf15 0 0.000 C21orf2 51 665.789

readGeneExp

Description

This method is used to read gene expression values from a file to a matrix in R workspace. So that the matrix can be used as input of other packages, such as edgeR. The input of the method is a file that contains gene expression values.

Usage

readGeneExp(file, geneCol=1, valCol=2, label = NULL, header=TRUE, sep="")

Arguments

Argument Descripition file file containing gene expression values. GeneCol gene id column in file. valCol expression value columns to be read in the file. label label for the columns. Header a logical value indicating whether the file contains the names of the variables as its first line. sep the field separator character. If sep = "" (the default for read.table) the separator is white space, that is one or more spaces, tabs, newlines or carriage returns.

Example:

> geneExpFile <- system.file("data", "GeneExpExample1000.txt", + package = "DEGseq") > exp <- readGeneExp(file = geneExpFile, geneCol = 1, valCol = c(7, + 9, 12, 15, 18, 8, 10, 11, 13, 16)) > exp[30:32,]

R1L1Kidney

```
R1L3Kidney
```

R1L7Kidney

```
R2L2Kidney
```

R2L6Kidney ENSG00000188976 73 77 68 70 82 ENSG00000187961 15 15 13 12 15 ENSG00000187583 1 1 3 0 3

R1L2Liver

```
R1L4Liver
```

R1L6Liver

```
R1L8Liver
```

R2L3Liver ENSG00000188976 34 56 45 55 42 ENSG00000187961 8 13 11 12 20 ENSG00000187583 0 1 0 0 2

samWrapper

Description

This function is a wrapper of the functions in samr. It is used to identify differentially expressed genes for two sets of samples with multiple replicates or two groups of samples from different individuals (e.g. disease samples vs. control samples).

Usage

samWrapper(geneExpFile1, geneCol1=1, expCol1=2, measure1=rep(1, length(expCol1)), geneExpFile2, geneCol2=1, expCol2=2, measure2=rep(2, length(expCol2)), header=TRUE, sep="", paired=FALSE, s0=NULL, s0.perc=NULL, nperms=100, testStatistic= c("standard","wilcoxon"), max.qValue=1e-3, in.foldchange=logged2=FALSE, output)

Arguments

Argument Description geneExpFile1 file containing gene expression values for group1. geneCol1 gene id column in geneExpFile1. expCol1 expression value columns in geneExpFile1. See the example. measure1

```
numeric vector of outcome measurements for group1. like c(1,1,1...) when paired=FALSE, or like c(-1,-2,-3,...) when paired=TRUE.
```

geneExpFile2 file containing gene expression values for group2. geneCol2 gene id column in geneExpFile2. ExpCol2 expression value columns in geneExpFile2. See the example. Measure2 numeric vector of outcome measurements for group2. like c(2,2,2...) when paired=FALSE, or like c(1,2,3,...) when paired=TRUE. header a logical value indicating whether geneExpFile1 and geneExpFile2 contain the names of the variables as its first line. sep the field separator character. If sep = "" (the default for read.table) the separator is white space, that is one or more spaces, tabs, newlines or carriage returns. paired a logical value indicating whether the samples are paired. s0 exchangeability factor for denominator of test statistic; Default is automatic choice. s0.perc percentile of standard deviation values to use for s0; default is automatic choice; -1 means s0=0 (different from s0.perc=0, meaning s0=zeroeth percentile of standard deviation values= min of sd values. Nperms number of permutations used to estimate false discovery rates. TestStatistic test statistic to use in two class unpaired case. Either "standard" (t-statistic) or "wilcoxon" (Two-sample wilcoxon or Mann-Whitney test). recommend "standard". max.qValue the max qValue desired; shoube be <1. min.foldchange the minimum fold change desired; should be >1. default is zero, meaning no fold change criterion is applied. logged2 a logical value indicating whether the expression values are logged2. output the output file.

Example

> geneExpFile <- system.file("data", "GeneExpExample1000.txt", + package = "DEGseq") > set.seed(100) > geneExpFile1 <- geneExpFile > geneExpFile2 <- geneExpFile > output <- file.path(tempdir(), "samWrapperOut.txt") > expCol1 = c(7, 9, 12, 15, 18) > expCol2 = c(8, 10, 11, 13, 16) > measure1 = c(-1, -2, -3, -4, -5) > measure2 = c(1, 2, 3, 4, 5) > samWrapper(geneExpFile1 = geneExpFile1, geneCol1 = 1, expCol1 = expCol1, measure1 = measure1, geneExpFile2 = geneExpFile2, geneCol2 = 1, expCol2 = expCol2, measure2 = measure2, nperms = 100, min.foldchange = 2, max.qValue = 1e-04, output = output, paired = TRUE)

References

Jiang,H. and Wong,W.H. (2009) Statistical inferences for isoform expression in RNASeq. Bioinformatics, 25, 1026–1032.

Likun Wang , Zhixing Feng , Xi Wang , Xiaowo Wang , and Xuegong Zhang. DEGseq: an R package for identifying differentially expressed genes from RNA-seq data Bioinformatics Advance Access published on October 24, 2009, DOI 10.1093/bioinformatics/btp612.

Wang, Likun; Feng, Zhixing; Wang, Xi; Wang, Xiaowo; Zhang, Xuegong. DEGseq Manual. Tsinghua University. Beijing, China. 2009(B). Available at <http://www.bioconductor.org/packages/devel/bioc/manuals/DEGseq/man/DEGseq.pdf> Access on: 18 November 2009

Wang, Likun; Wang, Xi. How to use the Degseq Package. Laboratory of Bioinformatics and Bioinformatics Division, TNLIST / Department of Automation, Tsinghua University. Beijing, China. 2009(A). Available at <http://bioinfo.au.tsinghua.edu.cn/software/degseq/DEGseq.pdf> Access on: 18 November 2009

References

[1] http://bioinformatics.oxfordjournals.org/cgi/content/abstract/26/1/136|

K-Means

Introduction

Clustering techniques have a wide use and importance nowadays. This importance tends to increase as the amount of data grows and the processing power of the computers increases. Clustering applications are used extensively in various fields such as artificial intelligence, pattern recognition, economics, ecology, psychiatry and marketing.

The main purpose of clustering techniques is to partitionate a set of entities into different groups, called clusters. These groups may be consistent in terms of similarity of its members. As the name suggests, the representative-based clustering techniques uses some form of representation for each cluster. Thus, every group has a member that represents it. The motivation to use such clustering techniques is the fact that, besides reducing the cost of the algorithm, the use of representatives makes the process easier to understand. There are many decisions that have to be made in order to use the strategy of representative-based clustering. For example, there is an obvious trade-off between the number of clusters and the internal cohesion of them. If there are few clusters, the internal cohesion tends to be small. Otherwise, a large number of clusters makes them very close, so that there is little difference between adjacent groups. Another decision is whether the clusters should be mutually exclusive or not, that is, if an entity can co-exist in more than one cluster at the same time.

Technique to be discussed

In this work, we focus on K-Means algorithm, which is probably the most popular technique of representative-based clustering. In the first section, we give a brief explanation of how the algorithm works.

Algorithm

K-Means is a simple learning algorithm for clustering analysis. The goal of K-Means algorithm is to find the best division of *n* entities in *k* groups, so that the total distance between the group's members and its corresponding centroid, representative of the group, is minimized. Formally, the goal is to partition the *n* entities into *k* sets S_i, *i=1, 2, ..., k* in order to minimize the within-cluster sum of squares (WCSS), defined as:

$$\sum_{j=1}^{k} \sum_{i=1}^{n} \| x_i^j - c_j \|^2$$

where term $\| x_i^j - c_j \|$ provides the distance between an entity point and the cluster's centroid.

The most common algorithm, described below, uses an iterative refinement approach, following these steps:

- Define the initial groups' centroids. This step can be done using different strategies. A very common one is to assign random values for the centroids of all groups. Another approach is to use the values of *K* different entities as being the centroids.
- Assign each entity to the cluster that has the closest centroid. In order to find the cluster with the most similar centroid, the algorithm must calculate the distance between all the entities and each centroid.
- Recalculate the values of the centroids. The values of the centroid's fields are updated, taken as the average of the values of the entities' attributes that are part of the cluster.
- Repeat steps 2 and 3 iteratively until entities can no longer change groups.

The K-Means is a greedy, computationally efficient technique, being the most popular representative-based clustering algorithm. The pseudocode of the K-Means algorithm is shown below.

Algorithm 1: K-Means Algorithm

Input: $E = \{e_1, e_2, \ldots, e_n\}$ (set of entities to be clustered)
k (number of clusters)
$MaxIters$ (limit of iterations)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l(e) \mid e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

```
foreach c_i ∈ C do
    c_i ← e_j ∈ E (e.g. random selection)
end
foreach e_i ∈ E do
    l(e_i) ← argminDistance(e_i, c_j) j ∈ {1...k}
end

changed ← false;
iter ← 0;
repeat
    foreach c_i ∈ C do
        UpdateCluster(c_i);
    end
    foreach e_i ∈ E do
        minDist ← argminDistance(e_i, c_j) j ∈ {1...k};
        if minDist ≠ l(e_i) then
            l(e_i) ← minDist;
            changed ← true;
        end
    end
    iter++;
until changed = true and iter ≤ MaxIters ;
```

Implementation

In order to use the K-Means algorithm in R, one must have the *stats* package installed. This package includes a function that performs the K-Mean process, according to different algorithms. These algorithms are described below:

- **Lloyd**

  Given any set of k centers Z, for each center z in Z, let V(z) denote its neighborhood. That is the set of data points for which z is the nearest neighbor. Each stage of Lloyd's algorithm moves every center point z to the centroid of V(z) and then updates V(z) by recomputing the distance from each point to its nearest center. These steps are repeated until convergence. Note that Lloyd's algorithm can get stuck in locally minimal solutions that are far from the optimal. For this reason it is common to consider heuristics based on local search, in which centers are swapped in and out of an existing solution (typically at random). Such a swap is accepted only if it decreases the average distortion, otherwise it is ignored.

- **Forgy**

  Forgy's algorithm is a simple alternating least-squares algorithm consisting of the following steps:

  - Initialize the codebook vectors. (Suppose that when processing a given training case, N cases have been previously assigned to the winning codebook vector.)
  - Repeat the following two steps until convergence:
    1. Read the data, assigning each case to the nearest (using Euclidean distance) codebook vector.
    2. Replace each codebook vector with the mean of the cases that were assigned to it.

- **MacQueen**

  This algorithm works by repeatedly moving all cluster centers to the mean of their respective Voronoi sets.

- **Hartigan and Wong**

Given n objects with p variables measured on each object x(i,j) for i = 1,2,...,n; j = 1,2,...,p; K-means allocates each object to one of K groups or clusters to minimize the within-cluster sum of squares:

$$Sum(k) = \sum_{i=0}^{n} \sum_{j=0}^{p} (x(i,j) - \overline{x(k,j)})^2$$

where $\overline{x(k,j)}$ is the mean variable j of all elements in group K.

In addition to the data matrix, a K x p matrix giving the initial cluster centers for the K clusters is required. The objects are then initially allocated to the cluster with the nearest cluster mean. Given the initial allocation, the procedure is to iteratively search for the K-partition with locally optimal within-cluster sum of squares by moving points from one cluster to another.

The K-Means function, provided by the *stats* package, might be used as follow:

```
kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))
```

where the arguments are:

- **x:** A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
- **centers:** Either the number of clusters or a set of initial (distinct) cluster centers. If a number, a random set of (distinct) rows in x is chosen as the initial centers.
- **iter.max:** The maximum number of iterations allowed.
- **nstart:** If *centers* is a number, *nstart* gives the number of random sets that should be chosen.
- **algorithm:** The algorithm to be used. It should be one of "Hartigan-Wong", "Lloyd", "Forgy" or "MacQueen". If no algorithm is specified, the algorithm of Hartigan and Wong is used by default.

If everything goes OK, an object of class *kmeans* is returned. This object has the following components:

- **cluster:** A vector of integers indicating the cluster to which each point is allocated.
- **centers:** A matrix of cluster centers.
- **whithnss:** The within-cluster sum of squares for each cluster.
- **size:** The number of points in each cluster.

View

There are actually two ways of viewing the result of a K-Means use. Both of them use the object of class *kmeans* returned by the function application.

The first way is to plot the object, creating a chart that represents the data. Thus, if there are N objects divided into K clusters, the chart must contain N points representing the objects, and those points must be colored in K different colors, each one representing a cluster set. For example, given the object *km*, which is a result of the function *kmeans* application, all one has to do in order to plot the object is:

```
plot(km)
```

The second way of viewing the result of a K-Means application is to simply print the components of the object of class *kmeans*. For example, given the same object *km* of the previous example, one could print its components using:

```
print(km)
```

Example

Suppose we have four objects and each object have two attributes (or features), as shown in table below.

Table 1: ***Table representing objects***

Object	Attribute X	Attribute Y
A	1	1
B	2	1
C	4	3
D	5	4

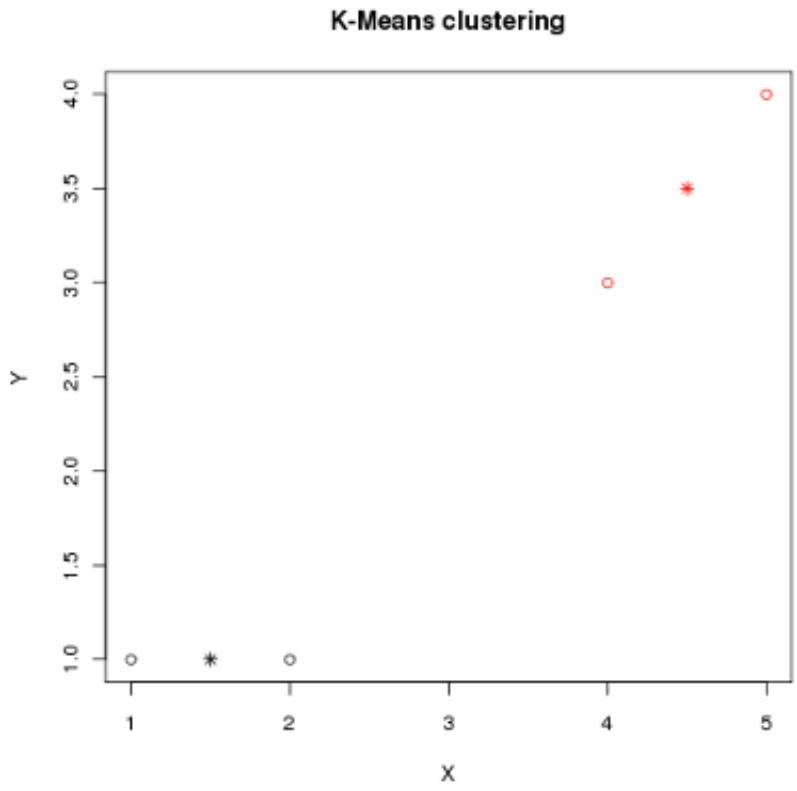
Our goal is to group these objects into K=2 groups based on their two features. The function K-Means can be used to define the groups as follow:

```
# prepare matrix of data
cells <- c(1, 1, 2, 1, 4, 3, 5, 4)
rnames <- c("A", "B", "C", "D")
cnames <- c("X", "Y")
x <- matrix(cells, nrow=4, ncol=2, byrow=TRUE, dimnames=list(rnames, cnames))

# run K-Means
km <- kmeans(x, 2, 15)

# print components of km
print(km)

# plot clusters
plot(x, col = km$cluster)
# plot centers
points(km$centers, col = 1:2, pch = 8)
```

Result of printing components of *km*:

```
K-means clustering with 2 clusters of sizes 2, 2

Cluster means:
    X   Y
1 1.5 1.0

2 4.5 3.5

Clustering vector:
A B C D
1 1 2 2

Within cluster sum of squares by cluster:
[1] 0.5 1.0

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

Result of plotting:



Case Study

In this section, we illustrate a case study using K-Means.

Scenario

The differences between countries go far beyond the physical and territorial aspects. Hence, for analytical purposes, it is very common to classify countries in groups based on some of their attributes. A traditional classification divides countries into developed, emerging and underdeveloped. In this division, many criteria, such per capita income and life expectancy, can be considered.

The k-means algorithm is a technique for grouping entities according to the similarity of their attributes. As the presenting problem consists of dividing countries into similar groups, it is plausible that K-means can be applied to this task.

Let's consider the scenario where countries need to be classified into the three already mentioned groups: developed, emerging and underdeveloped. To analyze their similarity and assign them to the groups, the following attributes should be taken into account:

- per capita income;
- literacy;
- infant mortality;
- life expectancy;

Input data

The input data is a table containing the numeric values of attributes for each considered country. The table consists of nineteen lines, representing countries and five columns (including the first containing the name of the country), representing the attributes. The table can be loaded from a spreadsheet or from a text file. To preserve the semantics of the clusters, all the values used in this example are real statistics of the countries.

Table 2: ***Input Data***

Country	Per capita income	Literacy	Infant mortality	Life expectancy
Brazil	10326	90	23.6	75.4
Germany	39650	99	4.08	79.4
Mozambique	830	38.7	95.9	42.1
Australia	43163	99	4.57	81.2
China	5300	90.9	23	73
Argentina	13308	97.2	13.4	75.3
United Kingdom	34105	99	5.01	79.4
South Africa	10600	82.4	44.8	49.3
Zambia	1000	68	92.7	42.4
Namibia	5249	85	42.3	52.9
Georgia	4200	100	17.36	71
Pakistan	3320	49.9	67.5	65.5
India	2972	61	55	64.7
Turkey	12888	88.7	27.5	71.8
Sweden	34735	99	3.2	80.9
Lithuania	19730	99.6	8.5	73
Greece	36983	96	5.34	79.5
Italy	26760	98.5	5.94	80
Japan	34099	99	3.2	82.6

Execution

The function "kmeans" can be used to define the groups of countries as follows:

```
# import data (assume that all data in "data.txt" is stored as comma separated values)
x <- read.csv("data.txt", header=TRUE, row.names=1)

# run K-Means
km <- kmeans(x, 3, 15)

# print components of km
print(km)

# plot clusters
plot(x, col = km$cluster)
```

```
# plot centers
points(km$centers, col = 1:2, pch = 8)
```

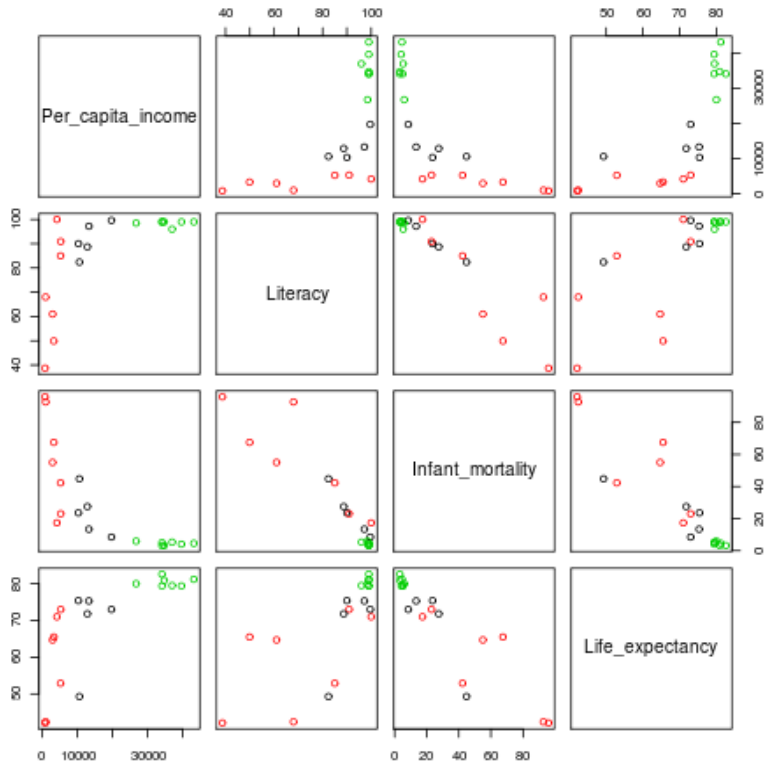
The value of the second parameter of "kmeans" was passed as the number 3, because we want to get three groups of countries.

Output

The result of printing the components of the class returned by the function application is shown below:

```
K-means clustering with 3 clusters of sizes 5, 7, 7

Cluster means:
  Per_capita_income Literacy Infant_mortality Life_expectancy
1         13370.400    91.58        23.560000        68.96000
2          3267.286    70.50        56.251429        58.80000
3         35642.143    98.50         4.477143        80.42857

Clustering vector:
        Brazil        Germany     Mozambique      Australia          China
             1              3              2              3              2
     Argentina United_Kingdom   South_Africa         Zambia        Namibia
             1              3              1              2              2
       Georgia       Pakistan          India         Turkey         Sweden
             2              2              2              1              3
     Lithuania         Greece          Italy          Japan
             1              3              3              3

Within cluster sum of squares by cluster:
[1]  57626083  20109876 158883600

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

The result of plotting the class returned by the function application is shown below:



data= 5.0 3.5 1.3 0.3 -1 5.5 2.6 4.4 1.2 0 6.7 3.1 5.6 2.4 1 5.0 3.3 1.4 0.2 -1 5.9 3.0 5.1 1.8 1 5.8 2.6 4.0 1.2 0

The implementation of k-means generated three clusters, relatively homogeneous, consisting of 5, 7 and 7 countries. Analyzing the cluster means, we can relate each group with each of the three classes of countries:

- the cluster formed by Germany, United Kingdom, Greece, Australia, Japan, Italy and Sweden, has the highest per capita income, literacy and life expectancy and the lowest infant mortality. So, this cluster represents the developed countries.
- the cluster formed by Mozambique, Georgia, Pakistan, India, Zambia and Namibia has the lowest values for all attributes and, therefore, represents the undeveloped countries.
- the cluster formed by the other countries, Brazil, South Africa, Turkish, Argentina and Lithuania represents the group of emerging countries.

To enhance the quality of the classification made by K-Means, the resulting division of the groups was compared with the classification of all countries by Human Development Index as included in the United Nations Development Program's Human Development Report released on October 5, 2009, compiled on the basis of data from 2007. The Human Development Index (HDI) is a comparative measure of well-being that considers aspects like life expectancy, literacy, and education. Compared with the division by the HDI, only four countries have been classified into different groups: Namibia, Georgia, Pakistan and India. These countries should have been placed in the "developing" rather than into "undeveloped" group.

References

1. Meira Jr., W.; Zaki, M. Fundamentals of Data Mining Algorithms. [1]
2. Cluster R package. [29]
3. J. B. MacQueen. 1967. Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press.
4. Hartigan, J.A. (1975), Clustering Algorithms, New York: John Wiley & Sons, Inc.
5. A. K. Jain, M.N. Murthy and P.J. Flynn, Data Clustering: A Review, ACM Computing Reviews, Nov 1999.

References

[1] http://www.dcc.ufmg.br/miningalgorithms/DokuWiki/doku.php

Hybrid Hierarchical Clustering

A Hybrid Hierarchical Clustering is a clustering technique that try to combine the best characteristics of both Hierarchical Techniques (Agglomerative and Divisive).

Introduction

Clustering can be considered an important unsupervised learning problem, which tries to find similar structures within an unlabeled data collection (JAIN, MURTY, FLYNN, 1999) [1].

These similar structures are data groups, better known as clusters. The data inside each cluster is similar (or close) to elements within his cluster, and is dissimilar (or further) to elements that belongs to other clusters. The clustering techniques' goal is to determinate the intrinsic grouping in a data set (JAIN, MURTY, FLYNN, 1999) .

Clustering Techniques

There are several clustering techniques but none can be considered the absolute best method. Each technique has his own merits and flaws, which usually leave the job to the user to determine what clustering method, will better satisfy his needs.

Hierarchical Clustering

The hierarchical clustering functions basically in joining closest clusters until the desired number of clusters is achieved. This kind of hierarchical clustering is named agglomerative because it joins the clusters iteratively. There is also a divisive hierarchical clustering that does a reverse process, every data item begin in the same cluster and then it is divided in smaller groups (JAIN, MURTY, FLYNN, 1999).

The distance measurement between clusters can be done in several ways, and that's how hierarchical clustering algorithms of single, average and complete differ.

In the single-link clustering, also known as minimum method, the distance between two clusters is considered to be the minimum distance between all pairs of data items. In the complete link clustering, also known as maximum method, the distance between two clusters is considered to be the maximum distance between all pairs of data items. The clusters found by the complete link algorithm are usually more compact than the ones found by the single link. However, the single link algorithm is more versatile (JAIN, MURTY, FLYNN, 1999).

In the average link clustering, the distance between two clusters is equal to the average distance between all data. A variation of this method uses median distance, which is less sensitive to greater data variation than the average distance.

Many hierarchical clustering algorithms have an appealing property that the nested sequence of clusters can be graphically represented with a tree, called a 'dendrogram' (CHIPMAN, TIBSHIRANI, 2006)[2]. Figure 1 shows an dendogram example.

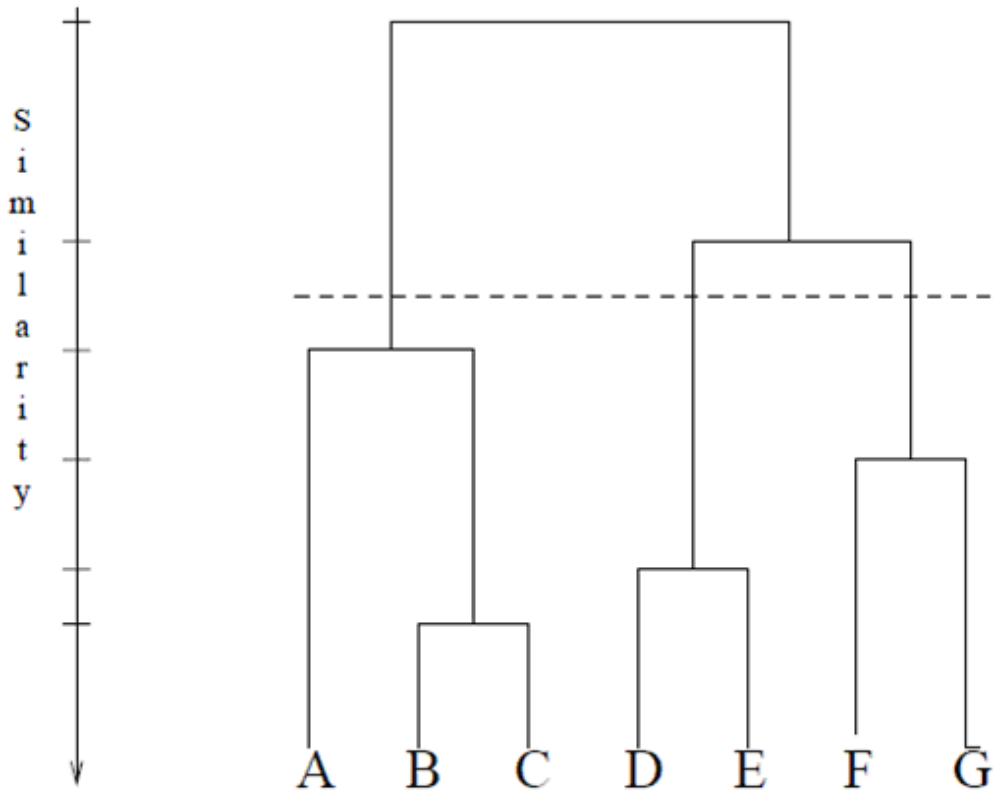


Figure 1: A dendogram obtained using a single-link agglomerative clustering algorithm. Source: Jain, Murty, Flynn (1999).

The greatest disadvantage of the hierarchical clustering is his high complexity order of O(n^2 log n) (JAIN, MURTY, FLYNN, 1999). The Hierarchical approach however has the advantage of handling clusters with different densities, much better than other clustering techniques.

Hybrid Hierarchical Clustering

Agglomerative algorithms group data into many small clusters and few large ones, which usually makes them good at identifying small clusters but not large ones. Divisive algorithms however, have reserved characteristics; making them good at identifying large clusters in general (CHIPMAN, TIBSHIRANI, 2006).

Hybrid Hierarchical Clustering Techniques try to combine the best advantages of both Agglomerative and Divisive techniques (LAAN, POLLARD, 2002; CHIPMAN, TIBSHIRANI, 2006)[3]. The way this combination is implemented depends on the chosen algorithm.

In this section it will be presented a hybrid hierarchical algorithm that uses the concept of 'mutual cluster' to combine the divisive techniques procedures with information gained from a preliminary agglomerative clustering.

Mutual Cluster

A mutual cluster can be defined as a group of data that are collectively closer to each other than to any other data, and distant from all other data. The data contained in a mutual cluster should never be separated (CHIPMAN, TIBSHIRANI, 2006).

Basically that requires in a mutual cluster that, the largest distance between data elements in S to be smaller than the smallest distance from an element in S to any data not in S:

$$d(x,y)_{x\in S, y\notin S} > diameter(S) \equiv \max_{w\in S, z\in S} d(w,z)$$

Where S is a subset of the data, d is a distance function between two data items, x is an element belonging to S, and y is an element that does not belong to S.

There is an interesting mutual cluster's property which indicates that a mutual cluster is not broken by an agglomerative clustering with any of single, average or complete linkage approaches (CHIPMAN, TIBSHIRANI, 2006). For more information on single, average or complete linkage approaches, see the agglomerative techniques in the hierarchical clustering section.

This property has several implications on mutual clusters. The most obvious is to support the idea that mutual clusters contain strong clustering information, no matter which linkage approach is used. This property also aids in the interpretation of agglomerative methods. This additional information can aid in the interpretation of mutual clusters, or in the decision of what clusters to divide.

Another implication of this property would be that mutual clusters can't be broken by agglomerative methods, which indicates that all mutual clusters can be identified by examining nested clusters in this method.

Algorithm

The hybrid hierarchical algorithm using mutual clusters can be described in three steps:

1. Compute the mutual clusters using an agglomerative technique.
2. Perform a constrained divisive technique in which each mutual cluster must stay intact. This is accomplished by temporarily replacing a mutual cluster of data elements by their centroid.
3. Once the divisive clustering is complete, divide each mutual cluster by performing another divisive clustering 'within' each mutual cluster.

The mutual cluster property and implications described earlier, indicates how it is possible to use agglomerative techniques to easily find mutual clusters in Step 1. For Step 2 any top-down method could be employed, it doesn't even need to be a divisive hierarchical algorithm. In Step 3, an agglomerative method can be used instead. Since mutual cluster are usually small, the use of either a top-down or bottom-up approach in Step 3 will give similar results. Using a top-down at both steps 2 and 3 just seems simpler.

Implementation in R

Package: hybridHclust
Descrption: hybrid hierarchical clustering via mutual clusters
Version: 1.0-3
Depends: cluster
Published: 2008-04-08
Author: Hugh Chipman, Rob Tibshirani, with tsvq code originally from Trevor Hastie
Maintainer: Hugh Chipman <hugh.chipman@acadiau.ca>
License: GPL-2
URL: http://ace.acadiau.ca/math/chipmanh/hybridHclust
Repository: CRAN

In views: Cluster, Multivariate

To download this package, visit the CRAN Mirrors page and select the mirror closest to your region. Once there select 'packages' and search for 'hybridHclust'.

Details:

The hybridHclust package uses the mutual cluster concept to construct a clustering in which mutual clusters are never broken. This is achieved by temporarily "fusing" together all points in a mutual cluster so that they have equal coordinates. The resultant top-down clusterings are then "stitched" together to form a single top-down clustering (CHIPMAN, TIBSHIRANI, 2006; 2008)[4].

"Only maximal mutual clusters are constrained to not be broken. Thus if points A, B, C, D are a mutual cluster and points A, B are also a mutual cluster, only the four points will be forbidden from being broken" (CHIPMAN, TIBSHIRANI, 2008).

This package uses squared Euclidean distance between rows of x as a distance measurement. In some instances, a desirable distance measure is d(x1,x2)=1-cor(x1,x2), if x1 and x2 are 2 rows of the matrix x. This correlation-based distance is equivalent to squared Euclidean distance once rows have been scaled to have mean 0 and standard deviation 1. This can be accomplished by pre-processing the data matrix before calling the hybrid clustering algorithm (CHIPMAN, TIBSHIRANI, 2008).

The main method to perform Hybrid clustering is called hybridHclust.

Usage

hybridHclust(x, themc=NULL, trace=FALSE)

Arguments

x - A data matrix whose rows are to be clustered

themc - An object representing the mutual clusters in x, typically generated by mutualCluster function. If it is not provided, it will be calculated.

trace – Indicates if internal steps be printed as they execute.

Return

A dendogram in hclust format

Visualization

The hybridHClust function returns a dendogram representing the algorithm clusters. A dendogram can be seem using the 'plot' function.

The mutual clusters can be calculated using the mutual cluster function. The function has a 'plot' parameter that automatically plots the mutual clusters dendogram on a graph. To print the mutual clusters is necessary to use the 'print.mutualCluster' function.

Please remember to load the hybridHclust R package before trying any of the example codes.

Example

```
# Function to show multiple graphs plotting, in this case 3 graphs in one row

par(mfrow=c(1,3))

# An Example Data

x <- cbind(c(-1.4806,1.5772,-0.9567,-0.92,-1.9976,-0.2723,-0.3153),c( -0.6283,-0.1065,0.428,-0.7777,-1.2939,-0.7796,0.012))

# Plot the example data

plot(x, pch = as.character(1:nrow(x)), asp = 1)

# Calculate the mutual clusters

mc1 <- mutualCluster(x, plot=TRUE)
```

```
print.mutualCluster(mc1) # print the mutual clusters

dist(x) # distance between data so you can verify that mc's are correct

# Calculate the Hybrid Hierarchical Cluster

hyb1 <- hybridHclust(x)

# Plot the Hybrid Clustering Dendogram

plot(hyb1)
```

Case Study

To better illustrate the clustering technique, it will be shown a simple case study.

The "Big Five", in contemporary psychology, consists of five important personality factors. These five factors have been found to contain and subsume more-or-less all known personality traits within their five domains and to represent the basic structure behind all personality traits.

The Big Five factors and their constituent traits can be summarized as follows:

- Openness - appreciation for art, emotion, adventure, unusual ideas, curiosity, and variety of experience.
- Conscientiousness - a tendency to show self-discipline, act dutifully, and aim for achievement; planned rather than spontaneous behavior.
- Extraversion - energy, positive emotions, urgency, and the tendency to seek stimulation in the company of others.
- Agreeableness - a tendency to be compassionate and cooperative rather than suspicious and antagonistic towards others.
- Neuroticism - a tendency to experience unpleasant emotions easily, such as anger, anxiety, depression, or vulnerability.

The most common way for a psychologist to measure these factors in someone's personality, is to apply a 'test' with descriptive sentences. The test scores for these traits are frequently presented as percentile scores.

Scenario/Situation/Development

Let's assume Big Five scores being applied to a population. In this case the test also comes with a socio-economic test for research purposes. The tests could be clustered in groups based on their percentile scores in each factor and the socio-economic variable, so that psychologists may better analyze the results and their possible effect on the population.

Input Data

The input data is a six dimensional numerical data consisting in the five factors percentile scores and the person's family income. The input data was randomly generated using a normal distribution on each dimension. The random generated data saves time and avoid the legal issue in using a person's possible privilege information (even anonymously).

The columns O, C, E, A, N represent respectively Openness, Conscientiousness, Extraversion, Agreableness and Neuroticism factors. The values on all of these columns are presented as percentile results from the Big Five test. The percentiles for each factor vary from a minimum of 5 up to a maximum of 95 in intervals of 5.

The column I represent the person's family income, the value is the number of minimum wages that family earn.

A csv file with the input data can be found here: HybridHClust Case Study CSV Input File [5]

	O	C	E	A	N	I
1.	10	30	40	55	45	12
2.	50	85	25	30	40	6
3.	95	90	80	50	5	10
4.	20	65	20	15	50	5
5.	50	25	65	10	95	13
6.	40	85	25	10	65	10
7.	55	50	90	90	75	9
8.	35	80	5	40	35	10
9.	30	65	85	35	80	13
10.	20	45	30	50	25	13
11.	40	30	65	10	25	14
12.	50	75	85	50	10	6
13.	65	75	50	5	50	9
14.	90	30	50	35	95	10
15.	60	80	10	75	50	6
16.	25	85	25	50	20	4
17.	10	30	90	50	35	13
18.	75	10	85	55	5	10
19.	65	65	20	50	15	16
20.	60	70	60	60	25	9
21.	35	70	30	40	45	6
22.	55	5	90	70	70	13
23.	15	20	60	40	60	10
24.	20	40	75	70	15	9
25.	30	95	25	65	20	7
26.	90	75	30	20	70	2
27.	95	20	65	80	45	15
28.	45	50	85	70	65	4
29.	60	90	70	25	5	14
30.	70	45	65	50	40	9
31.	50	50	65	65	45	7
32.	50	95	15	35	60	5
33.	70	15	90	80	50	9
34.	85	30	20	30	80	2
35.	70	40	45	85	30	2
36.	75	55	80	85	25	4
37.	20	55	25	35	90	4
38.	85	10	5	55	80	15

39.	5	15	75	35	25	10
40.	75	50	45	60	40	16
41.	65	85	35	90	10	3
42.	25	50	20	15	65	12
43.	15	15	50	75	80	3
44.	30	95	30	45	75	14
45.	80	50	85	20	25	5
46.	35	35	60	25	35	8
47.	90	55	50	15	35	5
48.	35	65	95	35	20	7
49.	30	70	60	25	45	15
50.	30	55	50	30	65	9
51.	35	90	70	20	20	6
52.	60	35	95	10	15	9
53.	25	60	35	90	25	10
54.	10	5	10	45	20	6
55.	80	5	15	75	90	14
56.	20	40	80	35	15	5
57.	80	60	95	65	70	4
58.	65	30	75	30	65	15
59.	15	45	55	50	70	6
60.	20	5	55	55	35	1
61.	40	10	75	70	30	3
62.	20	90	65	80	75	10
63.	95	40	40	20	5	4
64.	45	45	75	25	45	11
65.	80	95	50	45	10	14
66.	60	25	50	70	80	6
67.	85	60	45	95	55	9
68.	95	10	70	60	20	12
69.	65	75	25	50	40	15
70.	10	50	35	10	50	10
71.	50	85	85	40	65	8
72.	45	55	10	10	70	1
73.	5	50	95	55	90	3
74.	35	15	35	15	50	15
75.	95	40	75	50	50	4
76.	50	40	95	65	5	4
77.	40	80	50	95	5	14

78.	55	50	70	50	15	9
79.	90	45	55	30	65	1
80.	20	60	95	95	50	15
81.	85	50	95	95	90	11
82.	5	55	55	25	95	10
83.	15	15	40	20	15	9
84.	80	50	50	50	35	2
85.	65	90	65	50	35	3
86.	75	70	85	20	30	7
87.	65	10	85	10	50	1
88.	75	10	85	15	5	10
89.	50	25	15	20	30	6
90.	15	30	60	10	75	7
91.	30	15	45	25	25	11
92.	75	95	5	95	30	10
93.	55	15	70	30	40	12
94.	10	15	50	30	65	5
95.	5	50	50	60	25	7
96.	75	5	75	65	90	15
97.	80	25	90	90	50	6
98.	60	15	10	75	80	9
99.	95	15	95	95	5	7
100.	55	30	70	85	90	5

Execution

```
# Read the data file
x <- read.csv("hybridHclust_case_study_input.csv")
# calculate the mutual clusters
mc <- mutualCluster(x)
# Calculate the Hybrid Hierarchical Cluster
hyb <- hybridHclust(x, mc)
# Plot the Hybrid Clustering Dendogram
plot(hyb)
```

Output

The clustering function shows a dendogram of the clustered data, with that it is possible to decide how many clusters we want to use. The red line in the figure below demonstrates this choice for analysis, is this case the clusters below the red line will be considered.

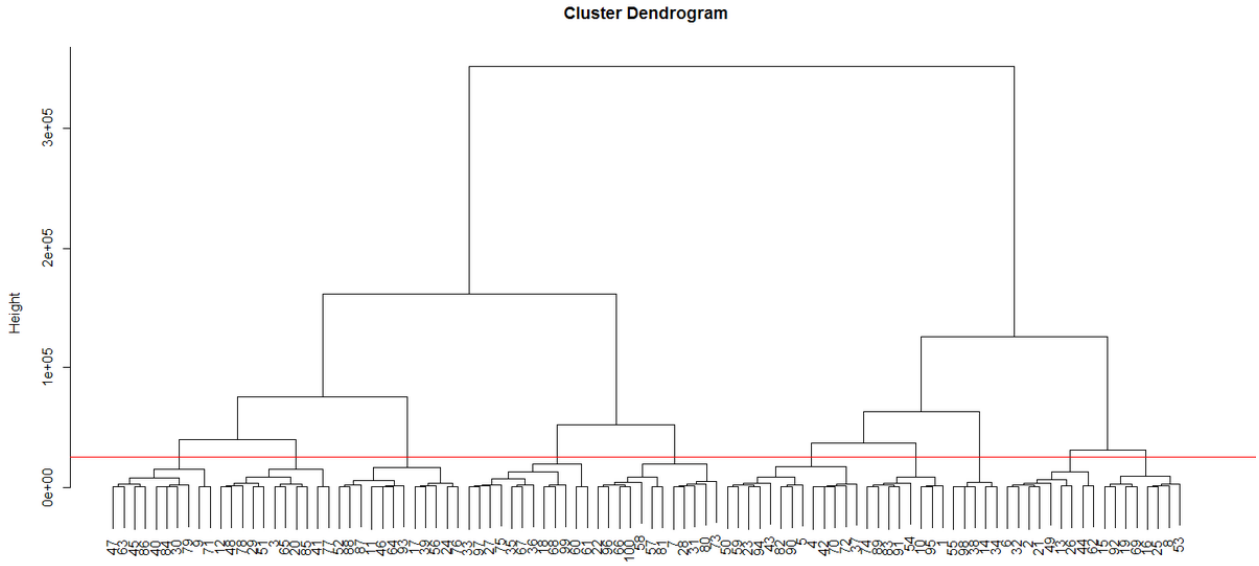


Figure 2: Hybrid clustering dendogram of the above case study data

Analysis

Because the case study has a multi-dimensional data it's harder to visualize what the clusters have in common. However, when a good number of clusters are chosen, it is possible to see their similarities and better analyze the results.

Look at the cluster formed by 50, 59, 23, 94, 43, 82, 90, 5, 4, 42, 70, 72 and 37. The cluster has mid to high values of neuroticism [50, 95], mid to low values of openness [5, 50], and low to mid high values of conscientiousness [15, 65].

	O	C	E	A	N	I
4.	20	65	20	15	50	5
5.	50	25	65	10	95	13
23.	15	20	60	40	60	10
37.	20	55	25	35	90	4
42.	25	50	20	15	65	12
43.	15	15	50	75	80	3
50.	30	55	50	30	65	9
59.	15	45	55	50	70	6
70.	10	50	35	10	50	10
72.	45	55	10	10	70	1
82.	5	55	55	25	95	10
90.	15	30	60	10	75	7

94.	10	15	50	30	65	5

Another interesting cluster is the one formed by 74, 89, 83, 91, 54, 10, 95 and 1. The cluster has mid to low values of openness [5, 50], mid to low values of conscientiousness [5, 50], mid to low values of extraversion [10, 50], and mid to low values of neuroticism [15, 50].

	O	**C**	**E**	**A**	**N**	**I**
1.	10	30	40	55	45	12
10.	20	45	30	50	25	13
54.	10	5	10	45	20	6
74.	35	15	35	15	50	15
83.	15	15	40	20	15	9
89.	50	25	15	20	30	6
91.	30	15	45	25	25	11
95.	5	50	50	60	25	7

If the values range in a cluster was an indication of a risk behavior or pathology, the psychologist would be able to pay more attention to anyone in that group for example. Or with more accurate socio-economic variables it would be possible to see correlations between the big five factors and other socio-economic factors.

References

[1] Jain, A. K., Murty, M. N., Flynn, P. J. (1999) "Data Clustering: A Review". ACM Computing Surveys (CSUR), 31(3), p.264-323, 1999.

[2] Chipman, H., Tibshirani, R. (2006) "Hybrid hierarchical clustering with applications to microarray data". Biostatistics Advance Access, 7(2), p.286-301, apr 2006.

[3] Laan, M., Pollard, K. (2002) "A new algorithm for hybrid hierarchical clustering with visualization and the bootstrap". Journal of Statistical Planning and Inference, 117 (2), p.275-303, dec 2002.

[4] Chipman, H., Tibshirani, R. (2008) "Package hybridHClust", apr 2008.

[5] http://rapidshare.com/files/321641297/hybridHclust_case_study_input.csv.html

Expectation Maximization (EM)

This chapter intends to give an overview of the technique Expectation Maximization (EM), proposed by (although the technique was informally proposed in literature, as suggested by the author) in the context of R-Project environment. The first section gives an introduction of representative clustering and mixture models. The algorithm details and a case study will be presented on the second section.

The R package that will be used is the MCLUST-v3.3.2 developed by Chris Fraley and Adrian Raftery, available in CRAN repository. The MCLUST tool is a software that includes the following features: normal mixture modeling (EM); EM initialization through an hierarchical clustering approach; estimate the number of clusters based on the Bayesian Information Criteria (BIC); and displays, including uncertainty plots and dimension projections.

The information sources of this document were divided in two groups: (i) manual and guides of R and MCLUST, which includes the technical report that is the base reference of this project and gives an overview of MCLUST with several examples and (ii) theoretical papers, surveys and books found in literature.

Introduction

Clustering consists in identifying groups of entities that have characteristics in common and are cohesive and separated from each other. Interest in clustering has increased due to several applications in distinct knowledge areas. Highlighting the search for grouping of customers and products in massive datasets, document analysis in Web usage data, gene expression from microarrays and image analysis where clustering is used for segmentation.

The clustering methods can be grouped in classes. One widely used involves hierarchical clustering, which consider, initially, that each points represent one group and at each iteration it merged two groups chosen to optimize some criterion. A popular criteria, proposed by, include the sum of within group sum of squares and is given by the shortest distance between the groups (single-link method).

Another typical class is based on iterative relocation, which data are moved from one group to another at each iteration. Also called as representative clustering due the use of a model, created to each cluster, that summarize the characteristics of the group elements. The most popular method in this class is the K-Means, proposed by, which is based on iterative relocation with the sum of squares criterion.

In statistic and optimization problems is usual to maximize or minimize a function, and its variables in a specific space. As these optimization problems may assume several different types, each one with its own characteristics, many techniques have been developed to solve them. This techniques are very important in data mining and knowledge discovery area as it can be used as basis for most complex and powerful methods.

One of these techniques is the Maximum Likelihood and its main goal is to adjust a statistic model with a specific data set, estimating its unknown parameters so the function that can describe all the parameters in the dataset. In other words, the method will adjust some variables of a statistical model from a dataset or a known distribution, so the model can “describe” each data sample and estimate others.

It was realized that clustering can be based on probability models to cover the missing values. This provides insights into when the data should conform to the model and has led to the development of new clustering methods such as Expectation Maximization (EM) that is based on the principle of Maximum Likelihood of unobserved variables in finite mixture models.

Technique to be discussed

The EM algorithm is an unsupervised clustering method, that is, don't require a training phase, based on mixture models. It follows an iterative approach, sub-optimal, which tries to find the parameters of the probability distribution that has the maximum likelihood of its attributes.

In general lines, the algorithm's input are the data set (x), the total number of clusters (M), the accepted error to converge (e) and the maximum number of iterations. For each iteration, first is executed the E-Step (E-xpectation), that estimates the probability of each point belongs to each cluster, followed by the M-step (M-aximization), that re-estimate the parameter vector of the probability distribution of each class. The algorithm finishes when the distribution parameters converges or reach the maximum number of iterations.

Algorithm

Initialization

Each class j, of M classes (or clusters), is constituted by a parameter vector (θ), composed by the mean (μj) and by the covariance matrix (P_j), which represents the features of the Gaussian probability distribution (Normal) used to characterize the observed and unobserved entities of the data set x.

$$\theta(t) = \mu_j(t),\ P_j(t),\ j\ =\ 1..M$$

On the initial instant (t = 0) the implementation can generate randomly the initial values of mean (μj) and of covariance matrix (P_j). The EM algorithm aims to approximate the parameter vector (θ) of the real distribution.

Another alternative offered by MCLUST is to initialize EM with the clusters obtained by a hierarchical clustering technique.

E-Step

This step is responsible to estimate the probability of each element belong to each cluster ($P(C_j|x_k)$). Each element is composed by an attribute vector (x_k). The relevance degree of the points of each cluster is given by the likelihood of each element attribute in comparison with the attributes of the other elements of cluster C_j .

$$P(C_j|x) = \frac{|\sum_j(t)|^{-\frac{1}{2}}\ exp^{n_j}\ P_j(t)}{\sum_{k=1}^{M}\ |\sum_j(t)|^{-\frac{1}{2}}\ exp^{n_j}\ P_k(t)}$$

M-Step

This step is responsible to estimate the parameters of the probability distribution of each class for the next step. First is computed the mean (μj) of class j obtained through the mean of all points in function of the relevance degree of each point.

$$\mu_j(t+1) = \frac{\sum_{k=1}^{N}\ P(C_j|x_k)\ x_k}{\sum_{k=1}^{N}\ P(C_j|x_k)}$$

To compute the covariance matrix for the next iteration is applied the Bayes Theorem, which implies that $P(A|B) = P(B|A) * P(A)P(B)$, based on the conditional probabilities of the class occurrence.

$$\sum_j(t+1) = \frac{\sum_{k=1}^{N}\ P(C_j|x_k)\ (x_k - \mu_j(t))\ (x_k - \mu_j(t))}{\sum_{k=1}^{N}\ P(C_j|x_k)}$$

The probability of occurrence of each class is computed through the mean of probabilities (C_j) in function of the relevance degree of each point from the class.

$$P_j(t+1) = \frac{1}{N}\sum_{k=1}^{N} P(C_j|x_k)$$

The attributes represents the parameter vector θ that characterize the probability distribution of each class that will be used in the next algorithm iteration.

Convergence Test

After each iteration is performed a convergence test which verifies if the difference of the attributes vector of an iteration to the previous iteration is smaller than an acceptable error tolerance, given by parameter. Some implementations uses the difference between the averages of class distribution as the convergence criterion.

```
if(||θ(t + 1) − θ(t)|| < ϙ)
   stop
else
   call E-Step
end
```

The algorithm has the property of, at each step, estimate a new attribute vector that has the maximum local likelihood, not necessarily the global, what reduces the its complexity. However, depending on the dispersion of the data and on its volume, the algorithm can stop due the maximum number of iterations defined.

Implementation

Packages

The expectation-maximization in algorithm in R, proposed in, will use the package mclust. This package contains crucial methods for the execution of the clustering algorithm, including functions for the E-step and M-step calculation. The package manual explains all of its functions, including simple examples. This manual can be found in.

The mclust package also provides various models for EM and also hierarchical clustering(HC), which is defined by the covariance structures. These models are presented in Table 1 and are explained in detail in.

Table 1: ***Covariance matrix structures.***

identifier	Model	HC	EM	Distribution	Volume	Shape	Orientation
E		*	*	(univariate)	equal		
V		*	*	(univariate)	variable		
EII	λI	*	*	Spherical	equal	equal	NA
VII	$\lambda_k I$	*	*	Spherical	variable	equal	NA
EEI	λA		*	Diagonal	equal	equal	coordinate axes
VEI	$\lambda_k A$		*	Diagonal	variable	equal	coordinate axes
EVI	λA_k		*	Diagonal	equal	variable	coordinate axes
VVI	$\lambda_k A_k$		*	Diagonal	variable	variable	coordinate axes
EEE	λDAD^T	*	*	Ellipsoidal	equal	equal	equal
EEV	$\lambda D_k A D_k^T$		*	Ellipsoidal	equal	equal	variable
VEV	$\lambda_k D_k A D_k^T$		*	Ellipsoidal	variable	equal	variable
VVV	$\lambda_k D_k A_k D_k^T$	*	*	Ellipsoidal	variable	variable	variable

Executing the Algorithm

The function “em” can be used for the expectation-maximization method, as it implements the method for parameterized Gaussian Mixture Models (GMM), starting in the E-step. This function uses the following parameters:

- **model-name**: the name of the model used;
- **data**: all the collected data, which must be all numerical. If the data format is represent by a matrix, the rows will represent the samples (observations) and the columns the variables;
- **parameters**: model parameters, which can assume the following values: pro, mean, variance and Vinv, corresponding to the mixture proportion for the components of mixture, mean of each component, parameters variance and the estimate hypervolume of the data region, respectively.
- **other**: less relevant parameters which wont be described here. More details can be found in the package manual.

After the execution, the function will return:

- **model-name**: the name of the model;
- **z**: a matrix whose the element in position [I,k] presents the conditional probability of the ith sample belongs to the kth mixture component;
- **parameters**: same as the input;
- **others**: other metrics which wont be discussed here. More details can be found in the package manual.

A simple example

In order to demonstrate how to use the R to execute the expectation-Maximization method, the following algorithm presents a simple example for a test dataset. This example can also be found in the package manual.

```
> modelName = ``EEE''
> data = iris[,-5]
> z = unmap(iris[,5])
> msEst <- mstep(modelName, data, z)
> names(msEst)
> modelName = msEst$modelName
> parameters = msEst$parameters
> em(modelName, data, parameters)
```

The first line executes the M-step so the parameters used in the em function can be generated. This function is called mstep and its inputs are model name, as “EEE”, the dataseta the iris dataset and finally, the z matrix, which contains the conditional probability of each class contains each data sample. This z matrix is generated by the unmap function.

After the M-step, the algorithm will show (line 2) the attributes of the object returned by this function. The third line will start the clustering process using some of the result of the M-step method as input.

The clustering method will return the parameters estimated in the process and the conditional probability of each sample falls in each class. These parameters include mean and variance, and this last one corresponds to the use of the mclustVariance method.

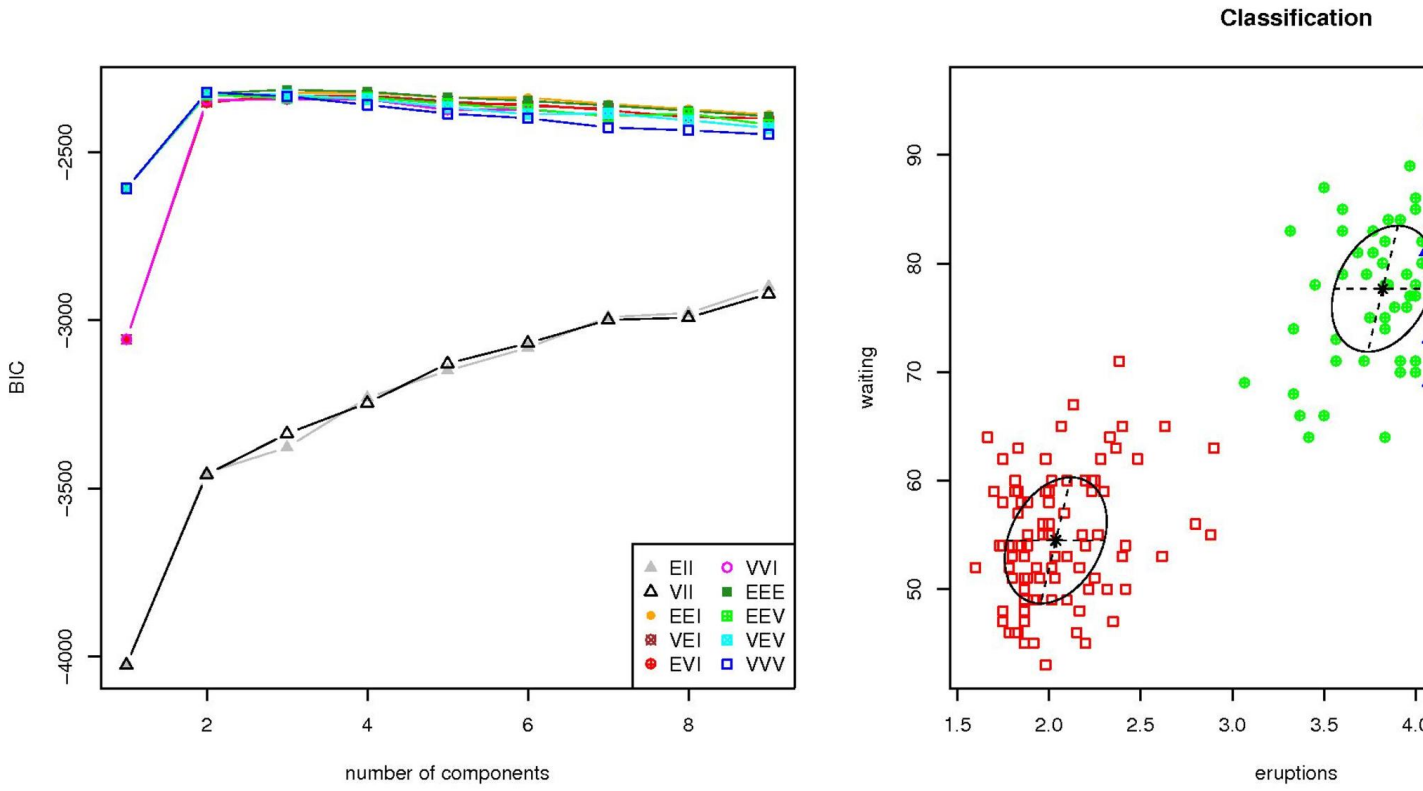
View

This section will present some examples of visualization available in MCLUST package. First will be showed a simple example of the overall process of clustering from the choice of the number of clusters, the initialization and the partitioning. Then it will be explained a didactical example using two random mixtures in comparison with two gaussian mixtures.

Basic Example

This is a simple example to show the features offered by MCLUST package. It is applied to the faithful dataset (included in R project). First the cluster analysis estimates the number of clusters that best represents this data set and also the covariance structure of the spread points. This is performed through the technique called Bayesian Information Criterion (BIC) that varies the number of cluster from 1 to 9. The BIC is the value of the maximized loglikelihood measured with a penalty for the number of parameters in the model. Then it's executed the hierarchical clustering technique (HC), which doesn't require a initialization phase. The output of the HC, that is, the cluster that each element belongs, is used to initialize the Expectation-Maximization technique (EM). After the execution of EM clustering the charts are showed below:

```
### basic_example.R ###
# usage: R --no-save < basic_example.R

library(mclust)           # load mclust library
x = faithful[,1]          # get the first column of the faithful data
set
y = faithful[,2]          # get the second column of the faithful data
set
plot(x,y)                 # plot the spread points before the
clustering
model <- Mclust(faithful) # estimate the number of cluster (BIC),
initialize (HC) and clusterize (EM)
data = faithful           # get the data set
plot(model, faithful)     # plot the clustering results
```



Didactical Example

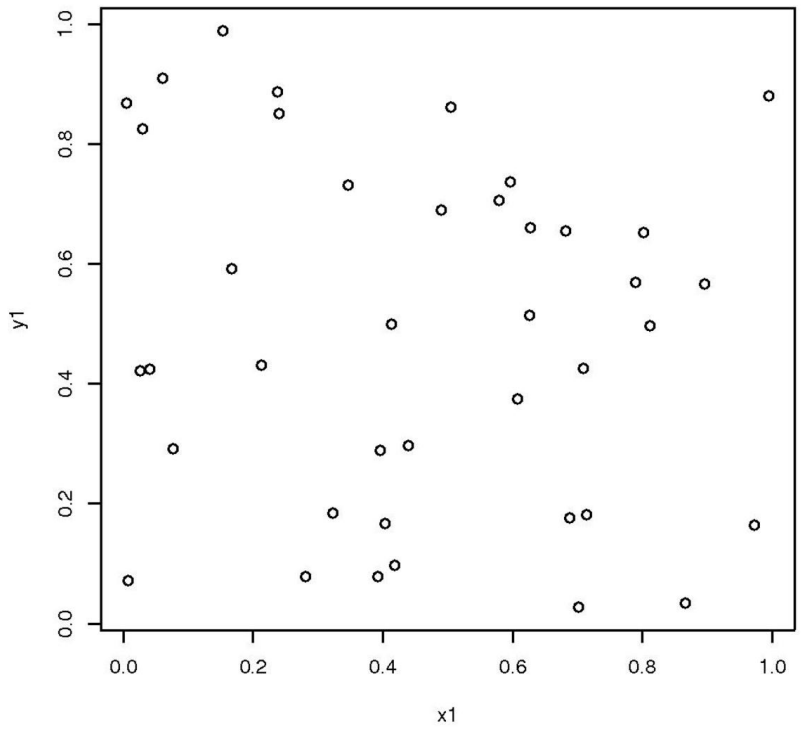
A didactical example is developed to show two distinct scenarios: (a) one that the model doesn't represents the data and (b) another that the data is conformed to the model. This example intends to show how EM behaves with noised and clean data sets.

a) Uniform random mixtures

A noised data set is generated through a uniform random function. The points spread are showed in a chart. Then the clustering analysis is executed with default parameters (i.e. varies cluster from 1 to 9). The clustering tool shows a warning with a message that the best model occurs at the min or max, in this case is the min that all points are grouped in a single cluster.

```
### random_example.R ###
# usage: R --no-save < random_example.R

library(mclust)                    # load mclust library
x1 = runif(20)                     # generate 20 random random numbers
for x axis (1st class)
y1 = runif(20)                     # generate 20 random random numbers
for y axis (1st class)
x2 = runif(20)                     # generate 20 random random numbers
for x axis (2nd class)
y2 = runif(20)                     # generate 20 random random numbers
for y axis (2nd class)
rx = range(x1,x2)                  # get the axis x range
ry = range(y1,y2)                  # get the axis y range
plot(x1, y1, xlim=rx, ylim=ry)     # plot the first class points
```

```
points(x2, y2 )                    # plot the second class points
mix = matrix(nrow=40, ncol=2)      # create a dataframe matrix
mix[,1] = c(x1, x2)                # insert first class points into the
matrix
mix[,2] = c(y1, y2)                # insert second class points into the
matrix
mixclust = Mclust(mix)             # initialize EM with hierarchical
clustering, execute BIC and EM

# Warning messages:
# 1: In summary.mclustBIC(Bic, data, G = G, modelNames = modelNames) :
#    best model occurs at the min or max # of components considered
# 2: In Mclust(mix) : optimal number of clusters occurs at min choice
```

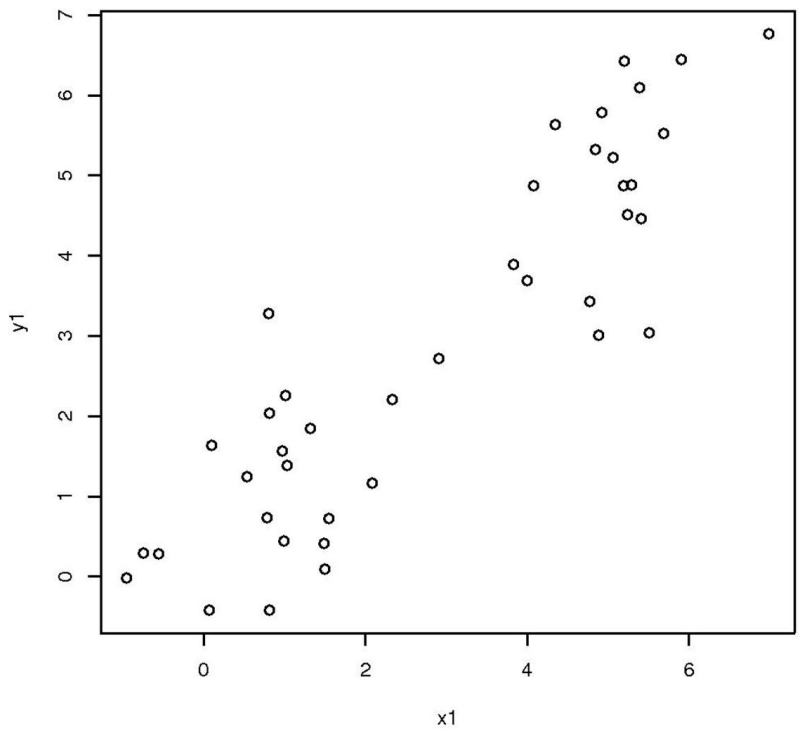


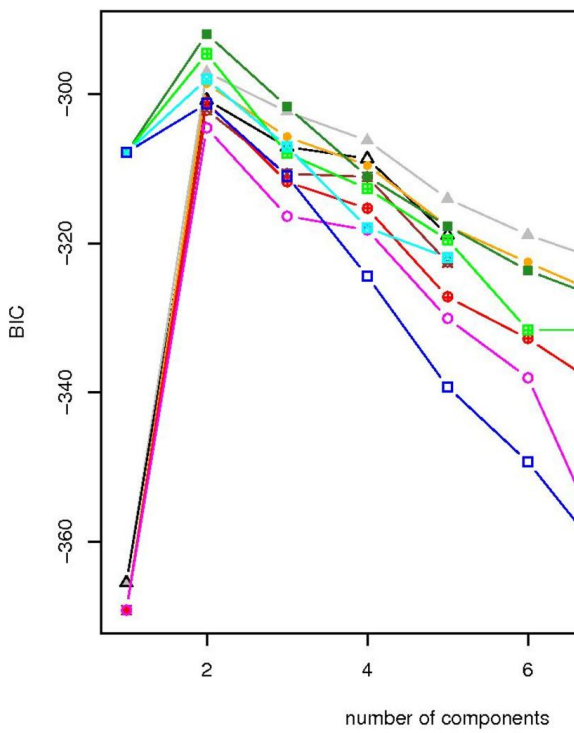
b) Two gaussian mixtures

This scenario is composed by two well separated data sets generated through a gaussian distribution function (Normal). The points are showed in the first chart. The EM clustering is applied and the results are also showed in the graphs below. As we can see, the EM clustering obtain two gaussian models that is in conformed to the data.

```
### gaussian_example.R ###
# usage: R --no-save < gaussian_example.R

library(mclust)                    # load mclust library
x1 = rnorm(n=20, mean=1, sd=1)     # get 20 normal distributed points for
 x axis with mean=1 and std=1 (1st class)
```

```
y1 = rnorm(n=20, mean=1, sd=1)    # get 20 normal distributed points for
 x axis with mean=1 and std=1 (2nd class)
x2 = rnorm(n=20, mean=5, sd=1)    # get 20 normal distributed points for
 x axis with mean=5 and std=1 (1st class)
y2 = rnorm(n=20, mean=5, sd=1)    # get 20 normal distributed points for
 x axis with mean=5 and std=1 (2nd class)
rx = range(x1,x2)                 # get the axis x range
ry = range(y1,y2)                 # get the axis y range
plot(x1, y1, xlim=rx, ylim=ry)    # plot the first class points
points(x2, y2)                    # plot the second class points
mix = matrix(nrow=40, ncol=2)     # create a dataframe matrix
mix[,1] = c(x1, x2)               # insert first class points into the
matrix
mix[,2] = c(y1, y2)               # insert second class points into the
matrix
mixclust = Mclust(mix)            # initialize EM with hierarchical
clustering, execute BIC and EM
plot(mixclust, data = mix)        # plot the two distinct clusters found
```





Case Study

Scenario

The scenario to be analized is composed by a sample data set available in the MCLUST package named "wreath". A clustering analysis is performed with more details, applied to a scenario composed by 14 point groups, that exceeds the maximum number of clusters allowed by the default MCLUST parameters. The clustering technique is executed two times: (i) the first based on the default MCLUST, (ii) with customized parameters.

Input data

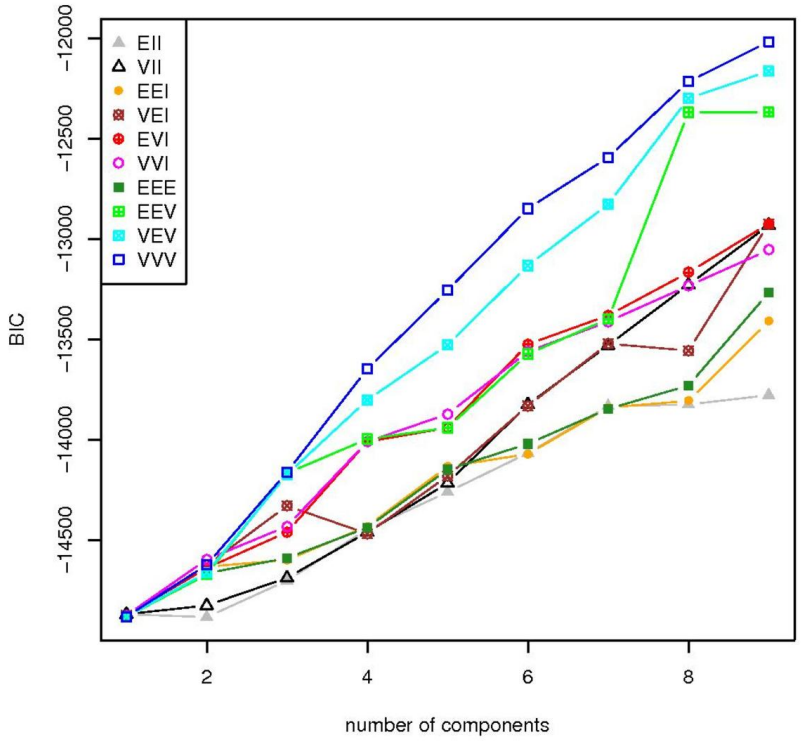
The input of the case study is the data set wreath provided by the MCLUST package. This data set consists in a 14 point group showed on the next figure, which can be modeled with Spherical or Ellipsoide that take into account the orientation of the data due its rotation.

```
### case_input.R ###
# usage: R --no-save < case_default.R

plot(wreath[,1],wreath[,2])
```

Execution

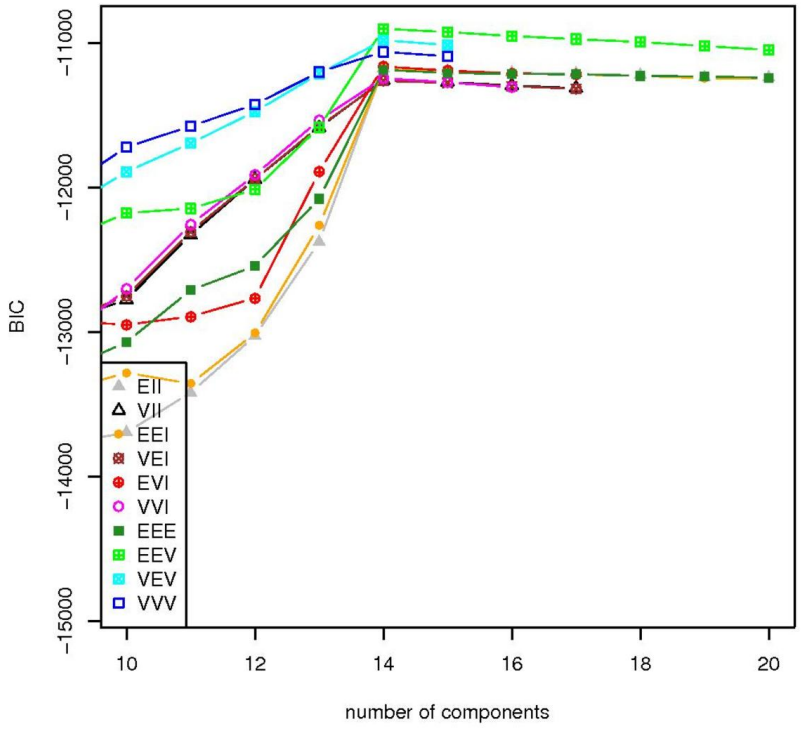
The clustering is executed two times. The first one is based on the default parameters given by the MCLUST tool, that varies the number of cluster from 1 to 9, which is smaller than the necessary to fit the case study data set. The estimation of the number of clusters is showed in a graphic that varies the number of clusters and compute the Bayesian Informatin Criterion (BIC) for each value. We can see that the BIC, using the default parameters, is divergent while is expected find a peak followed by a decrease that indicates the best number of clusters.

```
### case_default.R ###
# usage: R --no-save < case_default.R

library(mclust)

wreathBIC <- mclustBIC(wreath)
plot(wreathBIC, legendArgs = list(x = "topleft"))
```



Then the number of cluster is customized, modified to varies from 1 to 20 as showed below. The BIC technique will give the best number of clusters, in this case 14 clusters and the coefficient structure that have the properties of this data set, that is the EEV, which means that the clusters has similar shape, similar volumes but variable orientation. After executes the BIC method again we can see that 14 clusters, indicated by the peak on the graphic, is the number of cluster that present the maximum likelihood for this data.

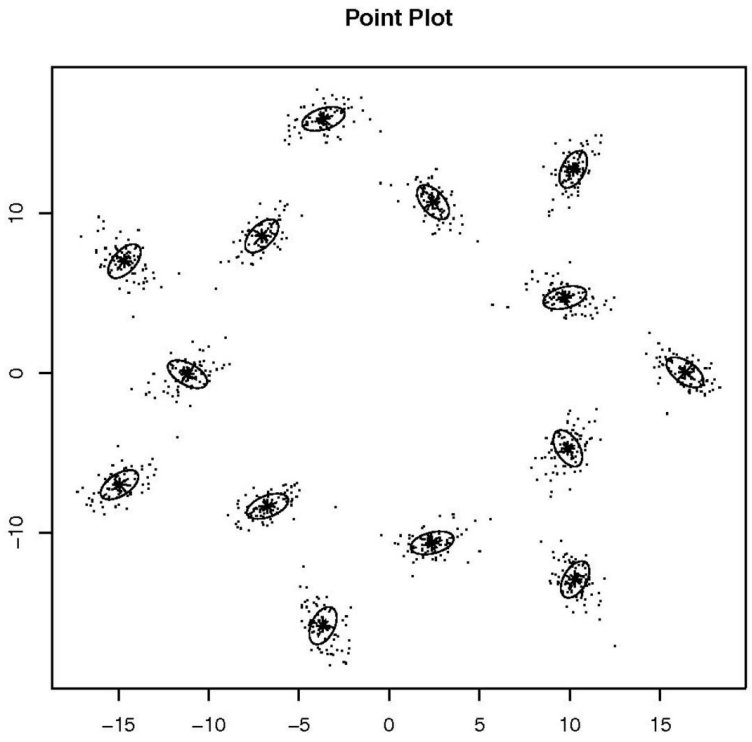
```
### case_customized.R ###
# usage: R --no-save < case_customized.R

library(mclust)
wreathDefault <- mclustBIC(wreath)
wreathCustomize <- mclustBIC(wreath, G = 1:20, x = wreathDefault)
plot(wreathCustomize, G = 10:20, legendArgs = list(x = "bottomleft"))
summary(wreathCustomize, wreath)
```



Output

The output of this clustering is analysed is obtained using the method mclust2Dplot depicted in the next figure. It was used the density visualization. The clusters found are characterized by 14 models wich have the distribution of an ellipsoides, with different orientation, beein in conformed with the data. The method summary can be executed later analyse other aspects of the clustering result.

```
### case_output.R ###
# usage: R --no-save < case_customized.R

library(mclust)
data(wreath)
wreathBIC <- mclustBIC(wreath)
wreathBIC <- mclustBIC(wreath, G = 1:20, x = wreathBIC)
wreathModel <- summary(wreathBIC, data = wreath)
mclust2Dplot(data = wreath, what = "density", identify = TRUE,
parameters = wreathModel$parameters, z = wreathModel$z)
```



Analysis

We can see that the mixture models created to represent the point are in conformation to the data set. On this case, the groups doesn't have an intersection between them, so all points were classified to the right group. The cluster orientation allows the method to find a better Ellipsoide to represent those points.

We conclude that the EM clustering technique, despite the dependence of the number of clusters and the initialization phase, is an efficient method that produces good results for several scenarios of data dispersion. The use of BIC to estimate the number of clusters and of the hierarchical clustering (HC) (which doesn't depend of the number of clusters) to initialize the clusters improves the quality of the results.

References

[] [] [] [] [] [] [] [1] [2]

[1] Leslie Burkholder. Monty hall and bayes's theorem, 2000.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological), 39(1):1{38, 1977.

Dissimilarity Matrix Calculation

Introduction

Dissimilarity may be defined as the distance between two samples under some criterion, in other words, how different these samples are. Considering the Cartesian Plane, one could say that the euclidean distance between two points is the measure of their dissimilarity. The Dissimilarity index can also be defined as the percentage of a group that would have to move to another group so the samples to achieve an even distribution.

The Dissimilarity matrix is a matrix that express the similarity pair to pair between to sets. It's square, and symmetric. The diagonal members are defined as zero, meaning that zero is the measure of dissimilarity between an element and itself. Thus, the information the matrix holds can be seen as a triangular matrix. MARCHIORO et al. (2003) [1] used the matrix of dissimilarity to determine the differences between oat specimens and discover good generators for the future generations.

The concept of Dissimilarity may be used in a more general way, to determine the pairwise difference between samples. As an example, this was used by da Silveira and Hanashiro (2009)[2] to study the impact of similarity and dissimilarity between superior and subordinate in the quality of their relationship. The similarity notion is a key concept for Clustering, in the way to decide which clusters should be combined or divided when observing sets. An appropriate metric use is strategic in order to achieve the best clustering, because it directly influences the shape of clusters. The Dissimilarity Matrix (or Distance matrix) is used in many algorithms of Density-based and Hierarchical clustering, like LSDBC.

The Dissimilarity Matrix Calculation is used, for example, to find Genetic Dissimilarity among oat genotypes. The way of arranging the sequences of protein, RNA and DNA to identify regions of similarity that may be a consequence of relationships between the sequences, in bioinformatics, is defined as sequence alignment. Sequence alignment is part of genome assembly, where sequences are aligned to find overlaps so that long sequences can be formed.

Algorithm

The matrix may be calculated by iterating over each element and calculating its dissimilarity to every other element. Let A be a Dissimilarity Matrix of size NxN, and B a set of N elements. A_{ij} is the dissimilarity between elements B_i and B_j.

```
   for i = 0 to N do
       for j = 0 to N do
           A_ij = Dissimilarity(B_i,B_j)
       end-for
   end-for
```

where the function Dissimilarity is defined as follows:

```
Dissimilarity(a,b) =
    0, if a = b
    ApplyDissimilarityCriterion(a,b), otherwise
```

ApplyDissimilarityCriterion is a function that calculates the dissimilarity between to elements based in the defined criterion. Here is a list of some criteria:

- Euclidean Distance
- Squared Euclidean Distance

- Manhattan Distance
- Maximum Distance
- Mahalanobis Distance
- Cosine Similarity

Implementation in R

Cluster Analysis, extended original from Peter Rousseeuw, Anja Struyf and Mia Hubert.

Package: cluster
Version: 1.12.1
Priority: recommended
Depends: R (>= 2.5.0), stats, graphics, utils
Published: 2009-10-06
Author: Martin Maechler, based on S original by Peter Rousseeuw, Anja.Struyf@uia.ua.ac.be and Mia.Hubert@uia.ua.ac.be, and initial R port by Kurt.Hornik@R-project.org
Maintainer: Martin Maechler <maechler at stat.math.ethz.ch>
License: GPL (>= 2)
Citation: cluster citation info
In views: Cluster, Environmetrics, Multivariate
CRAN checks: cluster results

The package can be downloaded from the CRAN [3] website. It can be installed using the install.packages() function, directly in R environment. The function daisy is used to calculate the dissimilarity matrix. It can be found in the cluster package.

The Dissimilarity Object is the representation of the Dissimilarity Matrix. The matrix is symmetric and the diagonal is not interesting, thus the lower triangle is represented by a vector to save storage space. To generate the dissimilarity matrix one must use the daisy function as follows:

Usage

daisy(x, metric = c("euclidean", "manhattan", "gower"), stand = FALSE, type = list())

Arguments

- x:numeric matrix or data frame. The dissimilarities will be computed between the rows of x.
- metric: character string specifying the metric to be used. The currently available options are "euclidian", which is the default, "manhattan" and "gower".
- stand: logical flag: If the value is true, then the measurements in x are standardized before calculating the dissimilarities.
- type: list specifying some or all of the types of the variables(columns) in x. The options are: "ordratio" (ratio scaled variables treated like ordinary variables), "logicalratio" (ratio scaled variables that must be logarithmically transformed), "asymm" (asymmetric binary variables) and "symm" (symmetric binary variables). Each entry is a vector containing the names or numbers of the corresponding columns of x.

Return

The function returns a dissimilarity object.

For further information, please refer to the daisy documentation [4].

Visualization

For the example, we will use the agriculture dataset available in R.

The dissimilarity matrix, using the euclidean metric, can be calculated with the command: daisy(agriculture, metric = "euclidean").

The result the of calculation will be displayed directly in the screen, and if you wanna reuse it you can simply assign it to an object: x <- daisy(agriculture, metric = "euclidean").

The object returned by the daisy function is a dissimilarity object, defined earlier in this text.

To visualize the matrix use the next command: as.matrix(x)

```
> as.matrix(x)
            B         DK         D         GR         E         F        IRL
B    0.000000  5.408327  2.061553 22.339651  9.818350  3.448188 12.747549
DK   5.408327  0.000000  3.405877 22.570113 11.182576  3.512834 13.306014
D    2.061553  3.405877  0.000000 22.661200 10.394710  2.657066 13.080138
GR  22.339651 22.570113 22.661200  0.000000 12.567418 20.100995  9.604166
E    9.818350 11.182576 10.394710 12.567418  0.000000  8.060397  3.140064
F    3.448188  3.512834  2.657066 20.100995  8.060397  0.000000 10.564563
IRL 12.747549 13.306014 13.080138  9.604166  3.140064 10.564563  0.000000
I    5.803447  5.470832  5.423099 17.383325  5.727128  2.773085  7.920859
L    4.275512  2.220360  2.300000 24.035391 12.121056  4.060788 14.569145
NL   1.649242  5.096077  2.435159 20.752349  8.280097  2.202272 11.150785
P   17.236299 17.864490 17.664088  5.162364  7.430343 15.164432  4.601087
UK   2.828427  8.052950  4.850773 21.485344  8.984431  5.303772 12.103718
            I         L        NL         P        UK
B    5.803447  4.275512  1.649242 17.236299  2.828427
DK   5.470832  2.220360  5.096077 17.864490  8.052950
D    5.423099  2.300000  2.435159 17.664088  4.850773
GR  17.383325 24.035391 20.752349  5.162364 21.485344
E    5.727128 12.121056  8.280097  7.430343  8.984431
F    2.773085  4.060788  2.202272 15.164432  5.303772
IRL  7.920859 14.569145 11.150785  4.601087 12.103718
I    0.000000  6.660330  4.204759 12.515990  6.723095
L    6.660330  0.000000  4.669047 19.168985  7.102112
NL   4.204759  4.669047  0.000000 15.670673  3.124100
P   12.515990 19.168985 15.670673  0.000000 16.323296
UK   6.723095  7.102112  3.124100 16.323296  0.000000
```

To obtain a summary of the data stored in the matrix, you can use: summary(x)

```
> summary(x)
66 dissimilarities, summarized :
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.6492  4.3569  7.9869  9.5936 13.2500 24.0350
Metric :  euclidean
Number of objects : 12
```

You can also use the dissimilarity matrix in the print method to obtain the lower triangle (the one that matters and is stored) of the matrix:

```
> print(x)
Dissimilarities :
            B         DK          D         GR          E          F        IRL
DK   5.408327
D    2.061553  3.405877
GR  22.339651 22.570113 22.661200
E    9.818350 11.182576 10.394710 12.567418
F    3.448188  3.512834  2.657066 20.100995  8.060397
IRL 12.747549 13.306014 13.080138  9.604166  3.140064 10.564563
I    5.803447  5.470832  5.423099 17.383325  5.727128  2.773085  7.920859
L    4.275512  2.220360  2.300000 24.035391 12.121056  4.060788 14.569145
NL   1.649242  5.096077  2.435159 20.752349  8.280097  2.202272 11.150785
P   17.236299 17.864490 17.664088  5.162364  7.430343 15.164432  4.601087
UK   2.828427  8.052950  4.850773 21.485344  8.984431  5.303772 12.103718
            I          L         NL          P
DK
D
GR
E
F
IRL
I
L    6.660330
NL   4.204759  4.669047
P   12.515990 19.168985 15.670673
UK   6.723095  7.102112  3.124100 16.323296

Metric :  euclidean
Number of objects : 12
```

Case Study

To illustrate the Dissimilarity Matrix technique, a simple case study will be shown.

Scenario

In the context of species enhancement programs, it is desirable to have very heterogeneous generations, so it is possible to induce the desired characteristics. When trying to induce a giving set of characteristics, it is necessary to choose a parent specimen that will result in the next generations.

Genetic dissimilarity measures have become the interest of many authors (Santos et al., 1997; Gaur et al., 1978; Casler,1995) in characterizing and identifying genetic contributions of different species. As dissimilarity measures to show the genetic variability intensity, the Euclidian distance and the Mahalanobis distance are the most used in plant genetic enhancement programs. The objective is to choose genetic constitutions that may result in superior combinations through their progeny.

Input Data

The data used here was extracted from MARCHIORO et al. (2003). 18 oat genotypes were measured in many aspects to be later compared. Here we use the measure of days until flowering(DUF) for genotypes subject to fungicides.

Genotype	DUF (days)
UPF 7.	104
UPF 15.	100
UPF 16.	97
UPF 17.	96
UPF 18.	102
UPF 19.	98
UFRGS 7.	95
UFRGS 14.	99
UFRGS 15.	101
UFRGS 16.	101
UFRGS 17.	100
UFRGS 18.	105
UFRGS 19.	92
URS 20.	97
URS 21.	93
IAC 7.	91
OR 2.	97
OR 3.	98

This data is the input for the daisy function to calculate the dissimilarity matrix. From the results it is possible to define the best parent to achieve very heterogeneous future generations.

Data construction in R:

```
DUF <- c(104,100, 97, 96, 102, 98, 95, 99, 101, 101, 100, 105, 92,
97, 93, 91, 97, 98)
Genotype <- c("UPF 7", "UPF 15", "UPF 16", "UPF 17", "UPF 18", "UPF
19", "UFRGS 7", "UFRGS 14", "UFRGS 15", "UFRGS 16", "UFRGS 17", "UFRGS
18", "UFRGS 19", "URS 20", "URS 21", "IAC 7", "OR 2", "OR 3")

myframe <- data.frame(DUF)

rownames(myframe) <- Genotype
```

Execution

To determine the dissimilarity matrix of the data selected in this case study, use the command below:

```
dis_mat <- daisy(myframe, metric = "euclidean", stand = FALSE)
```

Output

The dissimilarity matrix stored in dis_mat can be visualized as showed below:

```
> as.matrix(dis_mat)
```

	UPF 7	UPF 15	UPF 16	UPF 17	UPF 18	UPF 19	UFRGS 7	UFRGS 14	UFRGS 15	UFRGS 16	UFRGS 17	UFRGS 18	UFRGS 19	URS 20	URS 21	IAC 7	OR 2	OR 3
UPF 7	0	4	7	8	2	6	9	5	3	3	4	1	12	7	11	13	7	6
UPF 15	4	0	3	4	2	2	5	1	1	1	0	5	8	3	7	9	3	2
UPF 16	7	3	0	1	5	1	2	2	4	4	3	8	5	0	4	6	0	1
UPF 17	8	4	1	0	6	2	1	3	5	5	4	9	4	1	3	5	1	2
UPF 18	2	2	5	6	0	4	7	3	1	1	2	3	10	5	9	11	5	4
UPF 19	6	2	1	2	4	0	3	1	3	3	2	7	6	1	5	7	1	0
UFRGS 7	9	5	2	1	7	3	0	4	6	6	5	10	3	2	2	4	2	3
UFRGS 14	5	1	2	3	3	1	4	0	2	2	1	6	7	2	6	8	2	1
UFRGS 15	3	1	4	5	1	3	6	2	0	0	1	4	9	4	8	10	4	3
UFRGS 16	3	1	4	5	1	3	6	2	0	0	1	4	9	4	8	10	4	3
UFRGS 17	4	0	3	4	2	2	5	1	1	1	0	5	8	3	7	9	3	2
UFRGS 18	1	5	8	9	3	7	10	6	4	4	5	0	13	8	12	14	8	7
UFRGS 19	12	8	5	4	10	6	3	7	9	9	8	13	0	5	1	1	5	6
URS 20	7	3	0	1	5	1	2	2	4	4	3	8	5	0	4	6	0	1
URS 21	11	7	4	3	9	5	2	6	8	8	7	12	1	4	0	2	4	5
IAC 7	13	9	6	5	11	7	4	8	10	10	9	14	1	6	2	0	6	7
OR 2	7	3	0	1	5	1	2	2	4	4	3	8	5	0	4	6	0	1
OR 3	6	2	1	2	4	0	3	1	3	3	2	7	6	1	5	7	1	0

Analysis

The output shows that there is a high dissimilarity between the genotypes. This is in agreement with the results in MARCHIORO et al. (2003). Cross breeding these genotypes, it is possible to achieve high dissimilarity in the next generations, which is very good, since the genetic enhancement program can have more genetic combinations to explore.

References

[1] MARCHIORO, Volmir Sergio, DE CARVALHO, Fernando Irajá Félix, DE OLIVEIRA, Antônio Costa CRUZ, Pedro Jacinto, LORENCETTI, Claudir, BENIN, Giovani, DA SILVA, José Antônio Gonzales, SCHMIDT, Douglas A. M., GENETIC DISSIMILARITY AMONG OAT GENOTYPES, Ciênc. agrotec., Lavras. V.27, n.2, p.285-294, mar./abr., 2003

[2] DA SILVEIRA, Nereida Salette Paulo, HANASHIRO, Darcy Mitiko Mori, Similarity and Dissimilarity between Superiors and Subordinates and Their Implications for Dyadic Relationship Qualit, RAC, Curitiba, v. 13, n. 1, art. 7, p. 117-135, Jan./Mar. 2009

[3] http://cran.r-project.org/

[4] http://stat.ethz.ch/R-manual/R-patched/library/cluster/html/daisy.html

Hierarchical Clustering

Introduction

A hierarchical clustering method consists into grouping data objects into a tree of clusters. There are two main types of techniques: a bottom-up and a top-down approach. The first one starts with small clusters composed by a single object and, at each step, merge the current clusters into greater ones, successively, until reach a cluster composed by all data objects. The second approach use the same logic, but to the opposite direction, starting with the greatest cluster, composed by all objects, and split it successively into smaller clusters until reach the singleton groups. Besides the strategies, other important issue is the metrics used to build (merge or split) clusters. Such metrics can be different distance measures, described next section.

Inter-cluster Metrics

Four widely used measures for distance between clusters are as follows. Where p and p' are two different data object points, m_i is the mean for cluster C_i, n_i is the number of objects in cluster C_i, and $|p - p'|$ is the distance between p and p'.

Minimum distance: $d_{min}(C_i, C_j) = min_{p \in C_i, p' \in C_j} \mid p - p' \mid$

Maximum distance: $d_{max}(C_i, C_j) = max_{p \in C_i, p' \in C_j} \mid p - p' \mid$

Mean distance: $d_{mean}(C_i, C_j) = \mid m_i - m_j \mid$

Average distance: $d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i} \sum_{p' \in C_j} \mid p - p' \mid$

Algorithms

One algorithm that implements the bottom-up approach is AGNES (AGglomerative NESting). The main idea of AGNES is, at its first step, create clusters composed by one single data object, and then, using a specified metric (such the ones mentioned previous section), merge such clusters into greater ones. The second step is repeated iteratively until only one cluster is obtained, composed by all data objects. Another example of algorithm implementing this approach is Cure, where instead dealing all the time with the whole set of entities, the clusters are shrank to its centers.

DIANA (DIvisive ANAlysis) is an example of an algorithm that implements top-down approach, starting with a single big cluster with all elements and iteratively spliting the current groups into smaller ones. As happens with AGNES and Cure, is necessary to define some metric to compute the distance inter-clusters, in order to decide how (where) they have to be splitted.

Implementation

In order to use Hierarchical Clustering algorithms in R, one must install *cluster* package. This package includes a function that performs the AGNES process and the DIANA process, according to different algorithms.

AGNES Function

The AGNES function provided by *cluster* package, might be used as follow:

```
agnes(x, diss = inherits(x, "dist"), metric = "euclidean", stand
= FALSE, method = "average", par.method, keep.diss = n < 100,
keep.data = !diss)
```

where the arguments are:

- **x**: data matrix or data frame (all variables must be numeric and missing values are allowed), or dissimilarity matrix (missing values aren't allowed), depending on the value of the *diss* argument.
- **diss**: logical flag. if TRUE (default for dist or dissimilarity objects), then *x* is assumed to be a dissimilarity matrix. If FALSE, then *x* is treated as a matrix of observations by variables.
- **metric**: character string specifying the metric to be used for calculating dissimilarities between observations. The currently available options are *euclidean* and *manhattan*. Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences. If *x* is already a dissimilarity matrix, then this argument will be ignored.
- **stand**: logical flag: if TRUE, then the measurements in *x* are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation. If *x* is already a dissimilarity matrix, then this argument will be ignored.
- **method**: character string defining the clustering method. The six methods implemented are *average* ([unweighted pair-]group average method, UPGMA), *single* (single linkage), *complete* (complete linkage), *ward* (Ward'smethod), *weighted* (weighted average linkage) and its generalization *flexible* which uses (a constant version of) the Lance-Williams formula and the par.method argument. Default is *average*.
- **par.method**: if method = *flexible*, numeric vector of length 1, 3, or 4.
- **keep.diss, keep.data**: Logicals indicating if the dissimilarities and/or input data *x* should be kept in the result. Setting these to FALSE can give much smaller results and hence even save memory allocation time.

AGNES algorithm has the following features:

- yields the agglomerative coeficient which measures the amount of clustering structure found
- provides the hierarchical tree and the banner, a novel graphical display

The function AGNES returns a *agnes object* representing the clustering. This agnes object is a list with the components listed below:

- **order**: a vector giving a permutation of the original observations to allow for plotting
- **order.lab**: a vector similar to order, but containing observation labels instead of observation numbers. This component is only available if the original observations were labelled.
- **height**: a vector with the distances between merging clusters at the successive stages.
- **ac**: the agglomerative coefficient, measuring the clustering structure of the dataset.
- **merge**: an (n-1) by 2 matrix, where *n* is the number of observations.
- **diss**: an object of class *dissimilarity*, representing the total dissimilarity matrix of the dataset.
- **data**: a matrix containing the original or standardized measurements. This is available only if the input structure were different from a dissimilarity matrix.

Visualization

To visualize the AGNES function result might be used the funcionts: *print* and *plot*.

The first function simply print the components of object *agnes* and the second one plot the object, creating a chart that represents the data.

Here, it is a example of use function plot:

```
plot(x, ask = FALSE, which.plots = NULL, main = NULL,sub =
paste("Agglomerative Coefficient = ",round(x$ac, digits = 2)),adj
= 0, nmax.lab = 35, max.strlen = 5, xax.pretty = TRUE, ...)
```

The arguments used are:

- x: object agnes
- ask: If TRUE and which.plots = NULL, plot.agnes operates an interactive mode, via menu.
- which.plots: integer vector or NULL (default), the latter producing both plots
- main, sub: main and sub title for the plot, each with a convenient default.
- adj: For label adjustment in bannerplot
- nmax.lab: nteger indicating the number of labels which is considered too large for singlename labelling the banner plot.
- max.strlen: positive integer giving the length to which strings are truncated in banner plot labeling.
- xax.prety: positive integer giving the length to which strings are truncated in banner plot labeling.
- ...: graphical parameters.

And here, it is the example of use function print:

print(x, ...)

The arguments are:

- x: object agnes
- ...: potential further arguments (require by generic functin print)

DIANA Function

The DIANA function provided by *cluster* package, might be used as follow:

```
diana(x, diss = inherits(x, "dist"), metric = "euclidean", stand
= FALSE, keep.diss = n < 100, keep.data = !diss)
```

where the arguments are:

- **x**: data matrix or data frame (all variables must be numeric and missing values are allowed), or dissimilarity matrix (missing values aren't allowed), depending on the value of the *diss* argument.
- **diss**: logical flag. if TRUE (default for dist or dissimilarity objects), then *x* is assumed to be a dissimilarity matrix. If FALSE, then *x* is treated as a matrix of observations by variables.
- **metric**: character string specifying the metric to be used for calculating dissimilarities between observations. The currently available options are *euclidean* and *manhattan*. Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences. If *x* is already a dissimilarity matrix, then this argument will be ignored.
- **stand**: logical; if true, the measurements in *x* are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation. If x is already a dissimilarity matrix, then this argument will be ignored.
- **keep.diss, keep.data**: Logicals indicating if the dissimilarities and/or input data *x* should be kept in the

result. Setting these to FALSE can give much smaller results and hence even save memory allocation time.

DIANA algorithm is probably unique in computing a divisive hierarchy, whereas most other software for hierarchical clustering is ag glomerative. It has the same features as AGNES functions as follows:

- yields the agglomerative coeficient which measures the amount of clustering structure found
- provides the hierarchical tree and the banner, a novel graphical display

The function DIANA returns a *diana object* representing the clustering. This agnes object is a list with the components listed below:

- **order**: a vector giving a permutation of the original observations to allow for plotting
- **order.lab**: a vector similar to order, but containing observation labels instead of observation numbers. This component is only available if the original observations were labelled.
- **heigh**: a vector with the distances between merging clusters at the successive stages.
- **dc**: the divisive coefficient, measuring the clustering structure of the dataset.
- **merge**: an (n-1) by 2 matrix, where *n* is the number of observations.
- **diss**: an object of class *dissimilarity*, representing the total dissimilarity matrix of the dataset.
- **data**: a matrix containing the original or standardized measurements. This is available only if the input structure were different from a dissimilarity matrix.

Visualization

To visualize the DIANA function result might be used the funcionts: *print* and *plot*.

The first function simply print the components of object *agnes* and the second one plot the object, creating a chart that represents the data.

Here, it is a example of use function *plot*:

```
plot(x, ask = FALSE, which.plots = NULL, main = NULL, sub =
paste("Divisive Coefficient = ", round(x$dc, digits = 2)), adj =
0, nmax.lab = 35, max.strlen = 5, xax.pretty = TRUE, ...)
```

The arguments used are:

- **x**: object *diana*
- **ask**: If TRUE and which.plots = NULL, plot.diana operates an interactive mode, via menu.
- **which.plots**: integer vector or NULL (default), the latter producing both plots
- **main, sub**: main and sub title for the plot, each with a convenient default.
- **adj**: For label adjustment in bannerplot
- **nmax.lab**: nteger indicating the number of labels which is considered too large for singlename labelling the banner plot.
- **max.strlen**: positive integer giving the length to which strings are truncated in banner plot labeling.
- **xax.prety**: positive integer giving the length to which strings are truncated in banner plot labeling.
- **...**: graphical parameters.

And here, it is the example of use function *print*:

```
print(x, ...)
```

The arguments are:

- **x**: object *diana*
- **...**: potential further arguments (require by generic functin print)

Use Case

In this section it is illustrate a example of using hierarchical clustering in some Italian cities. In the example below, it is used mean distance, or single-linkage method.

Scenario

Given the distances in kilometers between Italian cities, the object is to group them into clusters in a agglomerative way. For examples in R is used the function *agnes*.

Input Data

The input data is a table containing the numeric values of distances between cities in Italy. The table consists of six columns (and rows), representing Italian cities. Each cell has a numeric value representing the distance in kilometers between them. The table can be loaded from a spreadsheet or from a text file.

Figure 1 illustres the cities used in this example.

Figure 1

Input distance matrix

	BA	FI	MI	VO	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
VO	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

Execution

The function *agnes* can be used to define the groups of countries as follow:

```
# import data
x <- read.table("data.txt")

# run AGNES
ag <- agnes (x, false, metric="euclidean", false, method ="single")

# print components of ag
print(ag)

# plot clusters
plot(ag, ask = FALSE, which.plots = NULL)
```

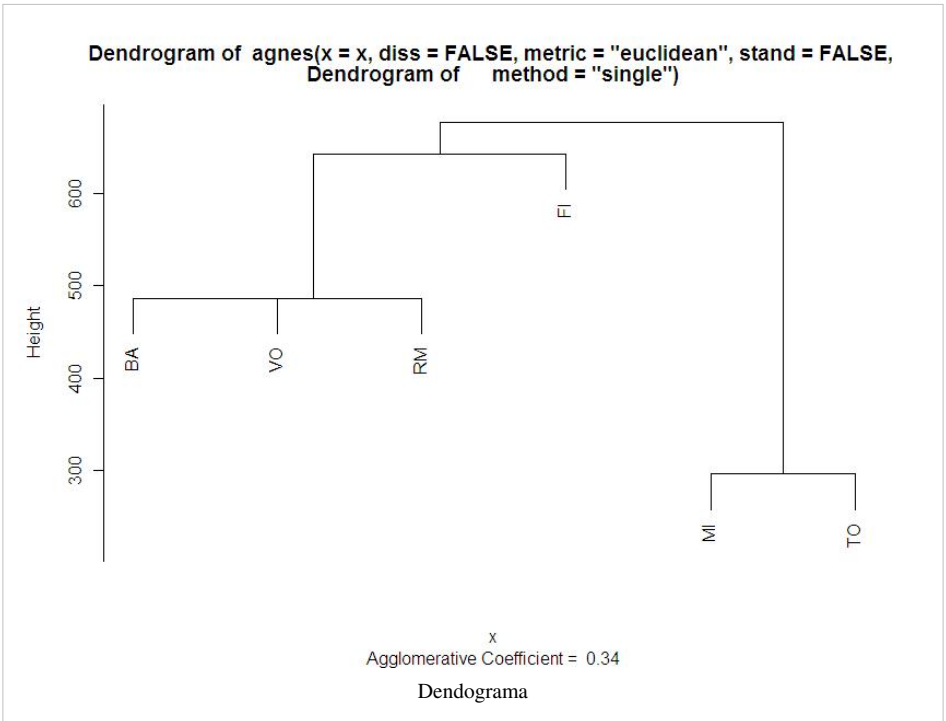
The value of the second parameter of *agnes* was FALSE, because **x** was treated as a matrix of observations by variables. The fourth parameter also is FALSE because isn't necessary standardizing each column.

Output

The result of printing the components of the class *agnes* returned by the function application is shown below:

```
Call:    agnes(x = x, diss = FALSE, metric = "euclidean", stand = FALSE,      method = "single")
Agglomerative coefficient:  0.3370960
Order of objects:
[1] BA VO RM FI MI TO
Height (summary):
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  295.8   486.0   486.5   517.6   642.6   677.0

Available components:
[1] "order"     "height"    "ac"        "merge"     "diss"      "call"
[7] "method"    "order.lab" "data"
```

The result of plotting the class returned by the function application it is shown below:



Dendograma

Analysis

The algorithm always cluster the nearest pair of cities so, in this case, the " neighborhood " cities are clusterized first. In single link clustering the rule is that the distance from the compound object to another object is equal to the shortest distance from any member of the cluster to the outside object

However, there are a few weaknesses in agglomerative clustering methods:

- they do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects;
- they can never undo what was done previously.

References

J. Han and M. Kamber. Data Mining: Concepts and Techniuqes. Morgan Kaufmann Publishers, San Francisco, CA, 2006.

Maechler, M. Package 'cluster'. Disponível em <http://cran.r-project.org/web/packages/cluster/cluster.pdf> Acesso em 12 dez 2009.

Meira Jr., W.; Zaki, M. Fundamentals of Data Mining Algorithms. Disponível em <http://www.dcc.ufmg.br/miningalgorithms/DokuWiki/doku.php> Acesso em 12 dez 2009.

A Tutorial on Clustering Algorithms. Disponível em <http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html>. Acesso em 12 dez 2009.

Density-Based Clustering

Clustering techniques have an important role in class identification of records on a database, therefore it's been established as one of the main topics of research in data mining. Spatial clustering techniques are a subset of clustering techniques applied on databases whose records have attributes intrinsically related to some spatial semantics. Most of the traditional clustering techniques described in the previous chapters can be applied to spatial databases. However, when applied to tasks like class identification on a spatial context, the traditional techniques might be unable to achieve good results, e.g. elements in the same cluster might not share enough similarity or the performance may be prohibitively poor.

The class identification task assisted by spatial clustering algorithms has a wide range of applications as finding relevant information on increasingly large spatial databases have recently become a highly demanded task. Examples include geographic data from satellite images, medical x-ray image processing or pattern recognition in machine learning samples.

Although there have been proposed an extensive number of techniques for clustering space-related data, many of the traditional clustering algorithm specified by them suffer from a number of drawbacks. Firstly, techniques based on k-partitioning such as those based on k-means are restricted to clusters structured on a convex-shaped fashion. Many databases have clusters with a broad variety of shapes (Figure 1), hence the traditional k-partitioning algorithms will fail to produce satisfactory results. Secondly, most techniques require previous knowledge of the database (domain knowledge) to determine the best input parameters. For example, k-means takes as input the number of expected clusters, k, which must be previously known for whichever database it's applied on. In many real-life databases there is not an a priori domain knowledge and therefore choosing parameters values based on guesses will probably lead to incomplete and undesirable results. Finally, most techniques are not scalable, which means they can not be used for large databases, such as those made up by hundreds of thousands of elements. Although most techniques derives algorithms belonging to a polynomial run time complexity class, the cost can become prohibitively large when they are applied on huge databases, e.g. millions or billions of elements.

The restrictions mentioned above can be overcome by using a new approach, which is based on density for deciding which clusters each element will be in. The next session will introduce this new approach, DBSCAN, which stands for density-based algorithm for discovering clusters in large spatial databases with noise.

DBSCAN: Density Based Spatial Clustering of Applications with Noise

The idea behind constructing clusters based on the density properties of the database is derived from a human natural clustering approach. By looking at the two-dimensional database showed in figure 1, one can almost immediately identify three clusters along with several points of noise. The clusters and consequently the classes are easily and readily identifiable because they have an increased density with respect to the points they possess. On the other hand, the single points scattered around the database are outliers, which means they do not belong to any clusters as a result of being in an area with relatively low concentration.

Furthermore, as will be explained in the following sections, the DBSCAN algorithm requires at most two parameters: a density metric and the minimum size of a cluster. As a result, estimating the number of clusters a priori is not a need, as opposed to other techniques, namely k-means. Finally, as will be demonstrated later, the DBSCAN is efficient even when applied on large databases.

Important Concepts

Prior to describing the DBSCAN algorithm, some concepts must be explained to its fully understanding. The elements of the database can be classified in two different types: the border points, the points located on the extremities of the cluster, and the core points, which are located on its inner region. A neighborhood of a point p is a set of all points that have a distance measure less than a predetermined value, called Eps. Therefore, the neighborhood size of the core points is generally bigger than that of the border points. A point p is directly density-reachable from another point q if it belongs to the neighborhood of q and q's neighborhood size is greater than a given constant MinPts. By deriving canonically the previous concept, one can define a generic density-reachability: a point p is density-reachable from q if there exists a chain of points p_1,..., p_n, where $p_1 = q$, $p_n = p$, and p_{i+1} is directly density-reachable from p_i. The former concepts are then combined to the definition of a cluster D:

- $\forall$ q,p : if q $\in$ D and if q is density-reachable from p and p's neighborhood is greater than the MinPts threshold, then p belongs to D.
- $\forall$ q,p : if q $\in$ D and p $\in$ D then there must be a point t such that q and p are directly reachable from t.

According to the above definition, there might be some points not belonging to any of the generated clusters, those points are outliers (noise). In other words, a cluster D is formed by a set of points that respects a certain degree of concentration, which is set by the MinPts and Eps constraints. By adjusting those values, one can find clusters of varying shapes and densities.

In other words, a cluster D is formed by a set of points that respects a certain degree of concentration, which is set by the MinPts and Eps constraints. By adjusting those values, one can and clusters of varying shapes and densities.

Algorithm

The DBSCAN algorithm (Algorithm 1) starts by randomly selecting an element P from the database. If P is not a core point, i.e. P has fewer than MinPts neighbors, it will be marked as noise. Otherwise it will be marked as being in the current cluster and the ExpandCluster (Algorithm 2) function will be called. Its purpose is to find all points that are density-reachable from P and are currently being marked as unclassified or noise. Despite being a recursive function, ExpandCluster is implemented without using recursion explicitly. The recursive behaviour is accomplished by using a set whose size varies as new density-reachable points are found. The algorithm ends when all points have been properly classified.

Finally, after identifying all clusters, one might wonder that a border point might belong to two clusters at the same time. For this matter, the currently implement algorithm will consider the ambiguous points as being part of the cluster which aggregated them firstly.

DBSCAN on R

R is a programming language and software environment for statistical computing. Besides being an widely used tool for statistical analysis, R aggregates several data mining techniques as well. Therefore it has become an major tool for simple tasks aiming to discover knowledge on databases. R's source code is freely available under the GNU General Public License and has been ported for several platforms other than Unix varieties, like Windows and MacOS. Although R uses primarily a command line interface, several GUIs are available, which increases its user-friendability. The DBSCAN technique is available on R's fpc package, by Christian Hennig, which implements clustering tasks for fixed point clusters. The DBSCAN implementation offers high-configurability, as it allows choosing several parameters and options values. Moreover, fpc's DBSCAN has a visualization interface, which make it possible to visualize the clustering process iteratively.

Installation

The fpc package can be installed by using the following command on R's command-line:

```
install.packages("fpc", dependencies = TRUE)
```

The above command shall recursivelly download and install all packages that fpc depend to along with fpc itself.

Use

To start using the fpc package, the following command must be invoked:

```
library('fpc')
```

DBSCAN Procedure

The fpc package allows the user to use the following procedure for applying dbscan technique directly:

```
dbscan(data, eps, MinPts, scale, method, seeds, showplot, countmode)
```

Parameters

The DBSCAN procedure takes the following parameters:

- data: The data that will be clustered. It can be a data matrix, a data.frame, dissimilarity matrix or dist-object.
- eps: Reachability distance (discussed before).
- MinPts: Reachability minimum number of points (discussed before).
- scale: Used for scaling the data.
- method: Confifigures memory usage by constraining performance, there are three options:
  - "raw": treats data as raw data and avoids calculating a distance matrix (saves memory but may be slow).
  - "dist": treats data as distance matrix (relatively fast but memory expensive).
  - "hybrid": expects also raw data, but calculates partial distance matrices (very fast with moderate memory requirements.
- seeds: Configuration regarding to the inclusion of the isseed-vector in the dbscan-object, can be TRUE or FALSE.
- showplot: Plotting configuration:
  - 0: no plot
  - 1: plot per iteration
  - 2: plot per subiteration
- countmode: NULL or vector of point numbers for reporting progress.

The first three parameters are, by far, the most important ones: data is the input data used for clustering; eps and MinPts variables retains the same meaning as before, that is, they are the minimum distance between elements and the necessary quantity of points to form a cluster respectivelly. The parameter showplot is related to the visualization of the results. Parameter method impacts directly the efficiency of the algorithm and can be used for calibrating memory-performance tradeoff on memory-constrained environments.

Example

The example in this section will illustrate the fpc's DBSCAN usage on the database depicted in figure 1. It will also show the clustering mechanism as a iterative visualization process. Firstly, the data to be clustered must be created:

```
x <- matrix(scan("file.dat",1926), nrow=1926, ncol=2, byrow=TRUE); # Read 1926 points on file "file.dat".

par(bg="grey40");                                                   # Set background to gray.

plot(x);                                                           # Plot original database.

d <- dbscan(x,10,showplot = 2);                                    # Calls DBSCAN routine (eps = 10 and MinPts is set to its default, which is 5).

d;                                                                 # Shows DBSCAN result.

# Notice that setting showplot to 2 will make dbscan show the result iteratively by its sub-iterations.
```

Output

The DBSCAN's output shows information about the clusters that were just created:

```
dbscan Pts=1926 MinPts=5 eps=20
       0 1 2  3 4   5 6   7 8   9 10 11 12 13 14 15 16 17 18 19
seed   0 8 8 12 8 844 8 312 8 616  8 18  8  8 10 10  8  8 12  8
border 4 0 0  0 0   0 0   0 0   0  0  0  0  0  0  0  0  0  0  0
total  4 8 8 12 8 844 8 312 8 616  8 18  8  8 10 10  8  8 12  8
```

Each cluster is represented by a single column and the rows shows how many of seed (core), border and total points it has. Notice that the tiny clusters scattered throughout the database are, in fact, formed by more than a single point. Thus, they are considered valid clusters because the MinPts parameter was not set and its default value is too low.

Visualization

Points that belong to a cluster are delimited by triangles of different colors. All points belonging to the same cluster are delimited by triangles of the same color. Points that do not belong to any cluster retain the same color as before and, as will be showed, are usually represented by black circles. Moreover, setting showplot parameter to 2 implies that on each subiteration of the DBSCAN algorithm, the routine will show the partial clusters on the screen. Following are five of these moments:

First image shows the points before the DBSCAN routine is called (command plot(x)). The second one shows the beginning of the clustering algorithm, the first major cluster is being formed in the upper center of the database. The fifth image shows the 19 clusters which were obtained when the algorithm had ended.

Case Study: Identifying Celestial Entities

The following sections will discuss a common class identification task assisted by DBSCAN and applied on spatial databases: identification of celestial entities on astronomy.

Scenario

Identifying celestial objects by capturing the radiation they emit is recurring task on astronomy. An astronomical entity might be itself the source of electromagnetic radiation (i.e. a star) or might reflect it from other sources (i.e. a planet). Typically, an entity will emit radiation on different wavelengths, which, together, will help identifying its class: it might be a planet, an star of any kind or age, or even a galaxy or other exotic entity previously unidentified. The intensity collected from each range of the eletromagnetic spectrum is stored on an individual two-dimensional grid (e.g. one for ultraviolet and other for gamma rays). Modern research groups on astronomy are capable of collecting and storing thousands of large-dimension grids, each of them representing a different view (or image) from the sky and using up to several terabytes of storage space.

Besides capturing the electromagnetic intensity emitted from an entity at a given range of the spectrum, the sensors might capture noise caused by the sensors as well as diffuse emission from the atmosphere and the space itself. Traditionally, a method for eliminating diffuse emission and some of the noise is to constrain the relevant intensity by a known threshold. For example, the intensity will only be considered relevant when surpassing 5 ρ rms. A greater problem arises due to the fact that electromagnetc waves at different spectrum ranges will yield different interference when traversing a medium. This phenomenum is called Rayleigh Scattering and will ultimately cause the wave to defleect upon being captured by the sensor. Therefore, when considering several images taken from the same region of space, but from different spectrum ranges, the following steps that must done for correctly identifying the celestial entities.

Methodology and Execution

As an example, consider figure 2, which shows an image taken from the center of a galaxy. This image consists of a pathologic example of noise on an astronomical picture. Following is the methodology for extracting the celestial objects on this image.

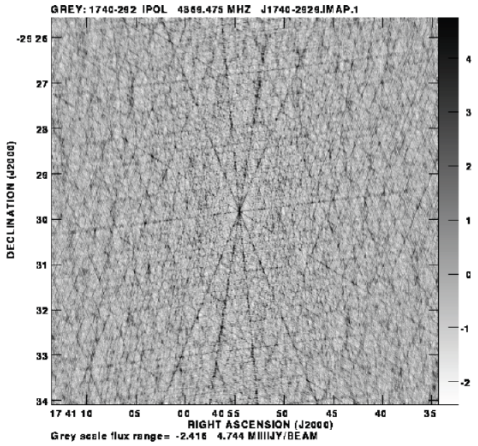


Figure 2: The original image. At a wavelength of 4,865 MHz, it shows the center of a galaxy.

Step 1: Pre-Processing

Firstly, a pre-processing step must be applied to the removal of noise and diffuse emission. As stated before, this might be accomplished by using a threshold. This step it is shown at figure 3, whereas the original image can be viewed at figure 2 and represents the original image depicting the center of a galaxy. The threshold was set to 50 which means that only pixels whose intensity are less than 50 (and consequently darker) are being considered.

Step 2: DBSCAN Clustering

Secondly, the DBSCAN algorithm can be applied on individual pixels to link together a complete emission area at the images for each channel of the electromagnetic spectrum. This is done by setting the eps parameter to some value that will define the minimum area required for a source to be considered. The eps parameter will define the distance metric in terms of pixels. Each of the generated cluster will define a celestial entity. Figure 4 shows the result of this step with eps and MinPts parameter set to 5.

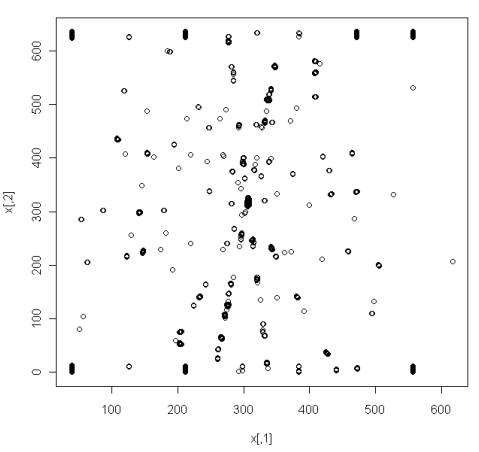


Figure 3: Source emissions after pre-processing.

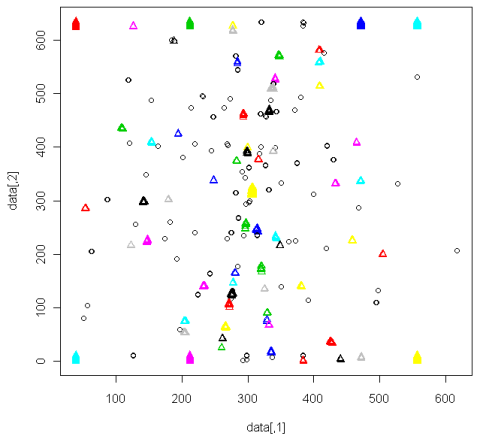


Figure 4: Clustering results for MinPts = 5 and eps = 5.

```
x <- matrix(scan("file.dat",1214), nrow=1214, ncol=2, byrow=TRUE);
dbscan(x,5,showplot = 2);
```

Results:

```
dbscan Pts=1214 MinPts=5 eps=5
         0  1  2  3  4 5 6  7 8  9 10 11 12 13 14 15 16
seed     0 28 26 26 26 6 6 18 2 10 18  8 16 16  8 28 20
border 226  0  0  0  0 0 0  0 4  0  0  0  0  0  0  0  0
total  226 28 26 26 26 6 6 18 6 10 18  8 16 16  8 28 20
        17 18 19 20 21 22 23 24 25 26 27 28 29  30 31 32
seed    14 14  8 18  6  6  6 14  6  6  6 14  8 112  6 18
border   0  0  0  0  0  0  0  0  0  0  0  0  0   0  0  0
total   14 14  8 18  6  6  6 14  6  6  6 14  8 112  6 18
        33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
seed     6 20 20 20 16 10  6  6  8 14  8 10 14 10  6 36
border   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2
total    6 20 20 20 16 10  6  6  8 14  8 10 14 10  6 38
        49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
seed    24  6  6 14  6 24 18  6 16  6 14 28 26 26 10 10  8
border   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
total   24  6  6 14  6 24 18  6 16  6 14 28 26 26 10 10  8
```

Step 3: Multi-spectral Correlation

After identifying all clusters, one can apply a multi-espectral correlation process in order to consider the results (generated clusters) from every electromagnetic wavelength. It will not be detailed here, but a common approach would be only considering clusters which has one or more counterparts close enough with respect to some threshold on the other channels of the electrogmagnetic spectrum.

Analysis

Figure 4 shows the results after the clustering has been performed by the DBSCAN algorithm with MinPts and eps set to 5. By looking at the results, we can see that many isolated points have not been clustered because MinPts parameter restrict the size of a cluster by a minimum value of elements. This restriction will remove clusters that are too small to be classified as a relevant emission source and consequently are classified as noise. By restricting the

MinPts parameter, celestial objects whose intensity is weak (e.g. they might be too far or do not emit strong radiation.) are eliminated from the results. This is generally desirable but can become a problem when emission sources defined by smaller areas must be analysed. Therefore, MinPts value must be carefully chosen since it may drastically modify the results.

Nevertheless, 64 clusters and 224 outliers were found. Most clusters do not have a large number of points, with some exceptions, like the large cluster located at the center of the database, which represents galaxy core which is an area with strong emission. The eps parameter takes an important role as it will define the minimum radius of pixels that represents the same emission source. Being set to 5 implies that points whose euclidian distance is greater than 5 do not belong to the same emission source. Although five is relatively large for the eps parameter, some clusters like those on (300,0) were not clustered together. This is because despite being close to each other they do not represent the same celestial object according to the eps value. Therefore, setting eps plays a role similar to that of MinPts, since it will define noise objects with respect to the distance measure, the same was done by the count measure of the MinPts.

Conclusion

The need to automatize the process of class identification of celestial entities is becoming increasinly important as astronomy and astrophysics become prominent areas of research. There is a growing demand for this task as technology evolves ands yields more and larger data samples to be analysed. Otherwise impossible without computational aid, this task have become easier with the assistance of the DBSCAN technique, which allowed larger samples to be analysed along with more precise results.

References

1. Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Institute for Computer Science, University of Munich. Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96).
2. FPC Package on CRAN: http://cran.r-project.org/web/packages/fpc/index.html
3. Manual for the DBSCAN algorithm implemented on FPC at http://bm2.genes.nig.ac.jp/RGM2/R_current/library/fpc/man/dbscan.html
4. Jörg Sander, Martin Ester, Hans-Peter Kriegel, Xiaowei Xu (1998). Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications

K-Cores

Introduction

In social networks analysis one of the major concerns is identification of cohesive subgroups os actores within a network. Friendship relation, publications citation, and many other more. Many studies and researches are focused on social network analysis, including in data mining. It is really important to find patterns in behavior of large online social networks, so the firms behind are able to create better mechanism to handle all that information with lower cost.

Online services such as Orkut, Facebook, Twitter and so on, have millions of users using their services simultaneously and interacting with others. Even in different services, the behavior of the network is similar.

People tend to interact in the same way as they do in real life, in a structure called “small world”, where people in a social network can reach any other person with less than seven steps. Such behavior can be studied to prevent disease propagation or to predict how fast an information can flow in society.

Several notions were introduced to formally describe cohesive groups: cliques, n–cliques, n–clans, n–clubs, k–plexes, k–cores, lambda sets, . . . For most of them it turns out that they are algorithmically difficult, classified as NP hard. However for cores very efficient algorithm exists.

The Technique

There are several tasks when handling with social networks. There are some of them listed above.

Link-based object classification – We can classify the object based on its links.

Object type prediction – We can predict the object type based on the objects linked to it.

Link type prediction – Here we want to predict the link type based on the objects linked by it.

Predicting link existence – Now we don't want to know nothing about the link, but it's existence. How to predict when a link exists between two objects? That's the point here.

Link cardinality estimation – There are several ways to estimate the cardinality of a link, such as counting the number of links of an object, or counting the numbers of smallest paths that pass through an object.

Object reconciliation – The task is to check whether two objects are the same, based on the links and attributes. Remove duplicate instances are useful in many applications.

Group detection – A clustering task. Here we want to know when a group of objects belong to the same group.

Subgraph detection - Subgraph identification is to find characteristic subgraphs in a network. This is a form of search in graphs.

Metadata mining – Metadata is data about data.

The technique that will be explained in this text is about group detection in social networks, based on the degree of each node in the network.

The Algorithm

K-cores in graph theory were introduced by Seidman in 1983 and by Bollobas in 1984 as a method of (destructively) simplifying graph topology to aid in analysis and visualization. They have been more recently defined as the following by Batagelj et al.: given a graph G = {V,E}> with vertices set V and edges set E, the k-core is computed by pruning all the vertices (with their respective edges) with degree less than k. That means that if a vertex u has degree du, and it has n neighbors with degree less than k, then u's degree becomes du – n, and it will be also pruned if k > du – n.

This operation can be useful to filter or to study some properties of the graphs. For instance, when you compute the 2-core of graph G, you are cutting all the vertices which are in a tree part of graph. (A tree is a graph with no loops).

Note that, there is a refinement (possibly empty) of the k-core of a graph G, for which there exists at least k paths between any two pairs of vertices of G. This concept is called structural cohesion in sociology[1] and vertex-connectivity in graph theory, and is equivalent via the Menger theorem to a k-component, a maximal graph that cannot be disconnected by fewer than k nodes.

Cores

The notion of core is presented in Butts (2010) as following:

Let G = (V, E) be a graph, and let f (v, S, G) for v ∈ V, S ⊆ V be a real-valued vertex property function (in the language of Batagelj and Zaversnik). Then some set H ⊆ V is a generalized k-core for f if H is a maximal set such that f (v, H, G) ≥ k for all v ∈ H. Typically, f is chosen to be a degree measure with respect to S (e.g., the number of ties to vertices in S). In this case, the resulting k-cores have the intuitive property of being maximal sets such that every set member is tied (in the appropriate manner) to at least k others within the set.

Degree-based k-cores are a simple tool for identifying well-connected structures within large graphs. Let the core number of vertex v be the value of the highest-value core containing v. Then, intuitively, vertices with high core numbers belong to relatively well-connected sets (in the sense of sets with high minimum internal degree). It is important to note that, while a given k-core need not be connected, it is composed of subsets which are themselves well-connected; thus, the k-cores can be thought of as unions of relatively cohesive subgroups. As k-cores are nested, it is also natural to think of each k-core as representing a “slice” through a hypothetical “cohesion surface” on G. (Indeed, k-cores are often visualized in exactly this manner.)

The kcores function produces degree-based k-cores, for various degree measures (with or without edge values). The return value is the vector of core numbers for V , based on the selected degree measure. Missing (i.e., NA) edge are removed for purposes of the degree calculation.

Implementation in R

Package sna

Butts (2010) presents a series of implementations in R about algorithms to analysis in social networks. The implementations uses the package "sna" as follow:

- **Package**: ‘sna’
- **Version**: 2.0-1
- **Date**: 2009-06-07
- **Title**: Tools for Social Network Analysis
- **Author**: Carter T. Butts <buttsc@uci.edu>
- **Maintainer**: Carter T. Butts <buttsc@uci.edu>
- **Depends**: R (>= 2.0.0), utils
- **Suggests**: network, rgl, numDeriv, SparseM, statnet
- **Description**: A range of tools for social network analysis, including node and graph-level indices, structural distance and covariance methods, structural equivalence detection, p* modeling, random graph generation, and 2D/3D network visualization.
- **License**: GPL (>= 2)
- **URL**: http://erzuli.ss.uci.edu/R.stuff
- **Repository**: CRAN
- **Date/Publication**: 2009-06-08 07:08:51

To install the package "sna" see http://erzuli.ss.uci.edu/R.stuff

Description

kcores calculates the k-core structure of the input network, using the centrality measure indicated in cmode.

Usage

```
   kcores(dat, mode = "digraph", diag = FALSE, cmode = "freeman", ignore.eval = FALSE)
```

Arguments

```
                        one or more (possibly valued) graphs.
   dat
                        "digraph" for directed data, otherwise "graph".
   mode
                        logical; should self-ties be included in the degree calculations?
   diag
                        the degree centrality mode to use when constructing the cores.
   cmode
                        logical; should edge values be ignored when computing degree?
   ignore.eval
```

Value

A vector containing the maximum core membership for each vertex.

```
function (dat, mode = "digraph", diag = FALSE, cmode = "freeman",
    ignore.eval = FALSE)
{
    dat <- as.edgelist.sna(dat, as.digraph = TRUE, suppress.diag = TRUE)
    if (is.list(dat))
        return(lapply(dat, kcores, dat = dat, mode = mode, diag = diag,
            cmode = cmode, ignore.eval = ignore.eval))
    if (mode == "graph")
        cmode <- "indegree"
    n <- attr(dat, "n")
    m <- NROW(dat)
    corevec <- 1:n
    dtype <- switch(cmode, indegree = 0, outdegree = 1, freeman = 2)
    if (!(cmode %in% c("indegree", "outdegree", "freeman")))
        stop("Illegal cmode in kcores.\n")
    solve <- .C("kcores_R", as.double(dat), as.integer(n), as.integer(m),
        cv = as.double(corevec), as.integer(dtype), as.integer(diag),
        as.integer(ignore.eval), NAOK = TRUE, PACKAGE = "sna")
    if (is.null(attr(dat, "vnames")))
        names(solve$cv) <- 1:n
    else names(solve$cv) <- attr(dat, "vnames")
    solve$cv
}
```

Case Study

To illustrate the k-cores algorithm a simple case study will be shown.

Scenario

Imagine a group of students dining. Each of them may choose another one to sit with him. Each student is a node and each possibility of sitting together is an edge. We want to group them as friends they are separately.

Input Data

The input data is a adjacency matrix representing the network with students and friendship.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
1	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0
11	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0
13	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1
18	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2
20	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0	0
23	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

	V22	V23	V24	V25	V26
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0

```
10   0   0   0   0   0
11   0   0   0   0   0
12   0   0   0   0   0
13   1   0   0   0   0
14   0   0   0   0   0
15   0   0   0   0   0
16   0   0   0   0   0
17   0   0   0   0   0
18   0   0   0   0   0
19   0   0   0   0   0
20   0   0   0   0   0
21   0   0   0   0   0
22   0   0   0   0   0
23   0   0   1   0   0
24   0   0   0   0   0
25   0   0   0   0   0
26   0   0   2   0   0
```

When there's value 1 it means the first choice and 2 the second one.

Execution

To determine the maximun k-core for each vertex from the network, we have:

```
kcores(people)
```

Where "people" is the matrix.

Output

A vector containing the maximum core membership for each vertex.

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
```

Analysis

As it is a small network, the maximum k-core is for k=2, so there aren't clear group definition.

Package RBGL

- **Package**: 'RBGL'
- **Version**: 1.26.0
- **Date**: 2010-10-26
- **Title**: An interface to the BOOST graph library
- **Author**: Vince Carey, Li Long, R. Gentleman
- **Maintainer**: Li Long
- **Depends**: graph, methods
- **Suggests**: Rgraphviz
- **Description**: A fairly extensive and comprehensive interface to the graph algorithms contained in the

BOOST library.

- **License**: Artistic-2.0

- **URL**: http://www.bioconductor.org
- **Repository**: CRAN
- **Date/Publication**: 2010-10-26 12:22:53

Usage

kCores(g, EdgeType=c("in", "out"))

Arguments

- **g**: an instance of the graph class'
- **EdgeType**: what types of edges to be considered when g is directed

Details

The implementation is based on the algorithm by V. Batagelj and M. Zaversnik, 2002. The example snacoreex.gxl is in the paper by V. Batagelj and M. Zaversnik, 2002.

Value

A vector of the core numbers for all the nodes in g.

Examples

```
library(RBGL)
con1 <- file(system.file("XML/snacoreex.gxl",package="RBGL"))
kcoex <- fromGXL(con1)
close(con1)
kCores(kcoex)
A C B E F D G H J K I L M N O P Q R S T U
1 2 1 2 3 3 3 3 3 3 3 3 2 2 1 1 2 2 2 2 0
```

```
con2 <- file(system.file("XML/conn2.gxl",package="RBGL"))
kcoex2 <- fromGXL(con2)
close(con2)
kCores(kcoex2)
A B C D E G H F
3 3 3 3 3 3 3 3
```

```
kCores(kcoex2, "in")
A B C D E G H F
0 0 0 0 0 0 0 0
```

```
kCores(kcoex2, "out")
A B C D E G H F
0 0 0 0 0 0 0 0
```

References

BATAGELJ, V. & MRVAR, A. (2002). Generalized Cores. Journal of ACM, v. 5, 2002.

BATAGELJ, V. & MRVAR, A. (2003). An O(m) Algorithm for Cores Decomposition of Networks.

CAREY, V. & LONG, L. & GENTLEMAN, R. (2010). An Interface to the BOOST graph library. http://www.bioconductor.org/packages/release/bioc/html/RBGL.html'".

BUTTS, C. T. (2010). Tools for Social Networks Analysis. http://erzuli.ss.uci.edu/R.stuff/'".

Fuzzy Clustering - Fuzzy C-means

Introduction

The aim of clustering is to minimize a set of data points into self-similar groups such that the points that belong to the same group are more similar than the points belonging to different groups. Each group is called a cluster.

This text will appear one more of the algorithms discussed in the literature known as Fuzzy C-Means. After the presentation of the technique will be presented an R package that implements this algorithm. Finally, a case study will be made using the algorithm implemented and presented the results obtained.

Technique/Algorithm

Fuzzy *C*-means (FCM----Frequently C Methods) is a method of clustering which allows one point to belong to one or more clusters. The method was developed by Dunn in 1973 and improved by Bezdek in 1981 and it is frequently used in pattern recognition. The FCM algorithm attempts to partition a finite collection of points into a collection of *C* fuzzy clusters with respect to some given criteria. Thus, points on the edge of a cluster, may be in the cluster to a lesser degree than points in the center of cluster. The FCM algorithm is based on minimization of the following objective function:

$$J(U, c_1, c_2, \ldots, c_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2}$$

Algorithm

The FCM is also known as fuzzy *k*-means nebulous because it uses fuzzy logic [Zadeh 1965] so that each instance is not associated with only one cluster, but has a certain degree of membership for each of the existing centroids. For this, the algorithm creates a matrix **U** associativity, where each term μ_{ij} represents the degree of membership of sample *i* to cluster *j*. In the FCM algorithm have a variable fuzziness *m* such that $1.0 < m < \infty$ where *m* and being a real number. The closer *m* is to infinity (∞), the greater the fuzziness of the solution and the closer to 1, the solution becomes increasingly similar to the clustering of binary *k*-means [Bezdek 1981]. A good choice is to set *m* = 2.0 [Hathaway and Bezdek 2001].

You can see both the *k*-means and FCM together in the same pseudo-code described in (Algorithm 1). In it, we have the *k*-means or FCM only by changing the formula to calculate the terms μ_{ij}, changing the average fuzzy [Zadeh 1965] for a binary choice, showing that FCM is indeed the K-Means cloudy.

In (Algorithm 1), the manner stating that $|A - B|^2$ is the distance away Euclidean to the a to b taking as input: set of samples x_i ($1 < i < N$), plain number of clusters K factor cloudiness me a factor of Tolerance, we leave on the: a cluster vector c_i ($1 < i < K$) and a matrix U determines the associativity of each sample with each of the clusters. It should be noted that the values of the matrix U depend only on H (array that stores the distances are the examples of clusters) and the value of m. The upgrade of the clusters depend solely on the values of the iteration matrix U in

iteration.

Implementation

The algorithm described above was implemented by the R package "e1071" released in 21/04/2010. This package has GPL-2 and can be found in the CRAN repository.

Install CBA Package

```
install.packages("e1071")
```

Importing methods and algorithm

```
library("e1071")
```

Usage:

```
cmeans(x, centers, iter.max = 100, verbose = FALSE,
       dist = "euclidean", method = "cmeans", m = 2,
       rate.par = NULL, weights = 1, control = list())
```

Arguments:

```
• x: The data matrix where columns correspond to variables and rows to observations.
• centers: Number of clusters or initial values for cluster centers.
• iter.max: Maximum number of iterations.
• verbose: If TRUE, make some output during learning.
• dist: Must be one of the following: If "euclidean", the mean square error,
if "manhattan", the mean absolute error is computed. Abbreviations are also accepted.
• method:  If "cmeans", then we have the c-means fuzzy clustering method,
if "ufcl" we have the on-line update. Abbreviations are also accepted.
• m: A number greater than 1 giving the degree of fuzzification.
• rate.par: A number between 0 and 1 giving the parameter of the learning rate for the
on-line variant. The default corresponds to 0:3.
• weights: a numeric vector with non-negative case weights. Recycled to the
number of observations in x if necessary.
• control: a list of control parameters. See Details.
```

If everything goes ok, an object fclust is returned. This object has the following components:

```
• centers: the final cluster centers.
• size: the number of data points in each cluster of the closest hard clustering.
• cluster: a vector of integers containing the indices of the clusters where the data points are assigned
to for the closest hard clustering, as obtained by assigning points to the (first) class with maximal membership.
• iter: the number of iterations performed.
• membership: a matrix with the membership values of the data points to the clusters.
• withinerror: the value of the objective function.
• call: the call used to create the object.
```

View

There is one way to show the result from this algorithm. That way would be printing the object fclust :

```
print(fclust)
```

Example

To illustrate the method implemented by the package studied, we have created two examples. The first one is created an array of two dimensions and in the second example creates an array with three dimensions. In both examples we used a normal distribution with a mean ranging and using a standard deviation of 0.3.

Example 1

```
# a 2-dimensional example
x<-rbind(matrix(rnorm(100,sd=0.3),ncol=2),
matrix(rnorm(100,mean=1,sd=0.3),ncol=2))
cl<-cmeans(x,2,20,verbose=TRUE,method="cmeans",m=2)
print(cl)
```

Example 2

```
# a 3-dimensional example
x<-rbind(matrix(rnorm(150,sd=0.3),ncol=3),
matrix(rnorm(150,mean=1,sd=0.3),ncol=3),
matrix(rnorm(150,mean=2,sd=0.3),ncol=3))
cl<-cmeans(x,6,20,verbose=TRUE,method="cmeans")
print(cl)
```

Output - Example 1

Output - Example 2

Case Study

The topics below describe the case study to demonstrate the results obtained using the Fuzzy C-Means.

Scenario

In the proposed case study, we used the database available in packages Iris R. This database consists of 150 instances each containing features of flower petals. Each instance of a database has its classification. There are three different classifications for the instances of the database (setosa, versicolor and virginica). Each of the classes present in the database studied has 50 examples that in total, summarize the 150 instances of the base. In addition to the definitions of classes in each of these instances, the other features are:

Datasets

The database of iris used in the experiment can be charged in the language R using the following command.

```
data(iris)
```

The table below has a small sample of the data contained in the iris database.

Table 1: *Samples of iris database*

Instance	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
72	6.1	2.8	4.0	1.3	versicolor
73	6.3	2.5	4.9	1.5	versicolor
114	5.7	2.5	5.0	2.0	virginica
115	5.8	2.8	5.1	2.4	virginica

Execution

To generate the results using the Fuzzy C-Means was run the script below.

```
data(iris)
x<-rbind(iris$Sepal.Length, iris$Sepal.Width, iris$Petal.Length)
x<-t(x)
result<-cmeans(x,3,50,verbose=TRUE,method="cmeans")
print(result)
s3d <- scatterplot3d(result$membership, color=result$cluster, type="h",
angle=55, scale.y=0.7, pch=16, main="Pertinence")
plot(iris, col=result$cluster)
```

Output

The results obtained by running the script are shown below.

```
Fuzzy c-means clustering with 3 clusters


Cluster centers:
      [,1]     [,2]     [,3]
1 5.003653 3.412805 1.484775
2 5.874034 2.760272 4.382520
3 6.793622 3.054510 5.644347


Memberships:
                  1            2            3
  [1,] 0.9964733414 2.388793e-03 1.137865e-03
  [2,] 0.9730096494 1.850758e-02 8.482767e-03
  [3,] 0.9776389508 1.515266e-02 7.208389e-03
  [4,] 0.9635322892 2.503070e-02 1.143701e-02
  [5,] 0.9939984763 4.051202e-03 1.950322e-03
```

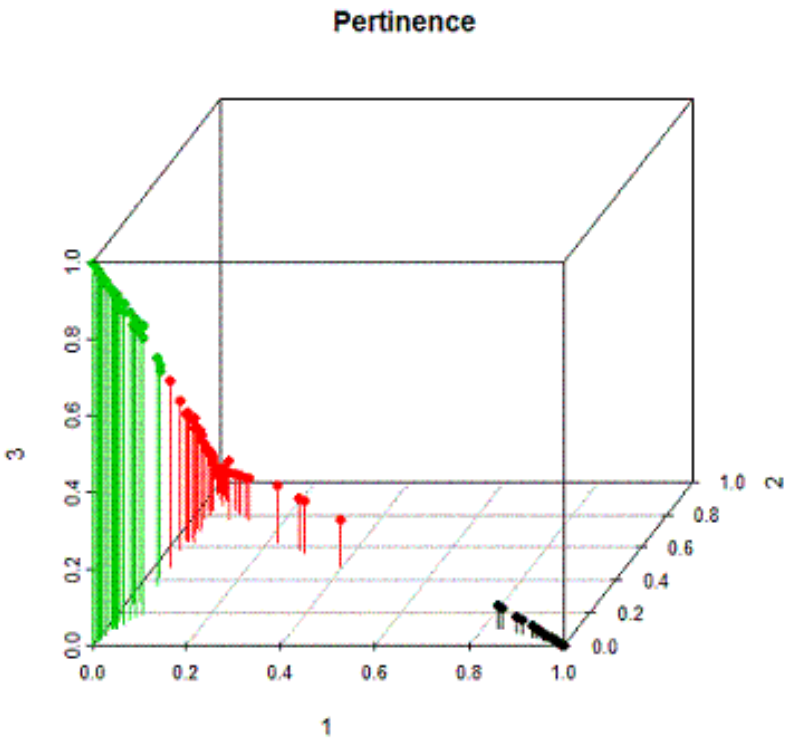
```
 [6,] 0.9304507689 4.703382e-02 2.251542e-02
 [7,] 0.9775132049 1.523242e-02 7.254371e-03
 [8,] 0.9999369153 4.314160e-05 1.994308e-05
 [9,] 0.9225703038 5.279889e-02 2.463081e-02
[10,] 0.9834280681 1.141773e-02 5.154205e-03
[11,] 0.9636309639 2.453957e-02 1.182947e-02
[12,] 0.9914862878 5.851313e-03 2.662399e-03
[13,] 0.9693327053 2.101145e-02 9.655842e-03
[14,] 0.9162524471 5.600693e-02 2.774062e-02
[15,] 0.8773228961 7.968730e-02 4.298980e-02
[16,] 0.8300898328 1.098729e-01 6.003725e-02
[17,] 0.9444844043 3.671434e-02 1.880126e-02
[18,] 0.9964733414 2.388793e-03 1.137865e-03
[19,] 0.8917869903 7.312086e-02 3.509215e-02
[20,] 0.9768481880 1.559135e-02 7.560459e-03
[21,] 0.9638052097 2.505889e-02 1.113590e-02
[22,] 0.9862983266 9.267056e-03 4.434618e-03
[23,] 0.9544955743 3.001379e-02 1.549064e-02
[24,] 0.9879996300 8.348552e-03 3.651818e-03
[25,] 0.9619796590 2.665281e-02 1.136753e-02
[26,] 0.9700883809 2.080884e-02 9.102779e-03
[27,] 0.9978125723 1.505741e-03 6.816871e-04
[28,] 0.9927414381 4.946076e-03 2.312486e-03
[29,] 0.9931235860 4.671305e-03 2.205109e-03
[30,] 0.9770882461 1.581612e-02 7.095630e-03
[31,] 0.9761442368 1.652997e-02 7.325795e-03
[32,] 0.9748518908 1.716740e-02 7.980706e-03
[33,] 0.9320017983 4.510961e-02 2.288859e-02
[34,] 0.8927033428 7.017637e-02 3.712029e-02
[35,] 0.9834280681 1.141773e-02 5.154205e-03
[36,] 0.9833813658 1.121223e-02 5.406405e-03
[37,] 0.9595420958 2.713202e-02 1.332588e-02
[38,] 0.9926144115 4.984365e-03 2.401224e-03
[39,] 0.9331691292 4.525365e-02 2.157722e-02
[40,] 0.9984835370 1.037187e-03 4.792755e-04
[41,] 0.9942973683 3.839814e-03 1.862818e-03
[42,] 0.8379960489 1.102335e-01 5.177041e-02
[43,] 0.9474894758 3.543286e-02 1.707767e-02
[44,] 0.9966468801 2.299980e-03 1.053140e-03
[45,] 0.9422485213 3.983938e-02 1.791210e-02
[46,] 0.9693327053 2.101145e-02 9.655842e-03
[47,] 0.9736866788 1.782324e-02 8.490078e-03
[48,] 0.9713101708 1.953226e-02 9.157566e-03
[49,] 0.9741716274 1.744780e-02 8.380577e-03
[50,] 0.9970704014 1.996528e-03 9.330710e-04
[51,] 0.0396263985 3.645217e-01 5.958519e-01
[52,] 0.0318692598 7.303043e-01 2.378264e-01
```

```
[53,] 0.0257989245 2.759514e-01 6.982496e-01
[54,] 0.0547597033 8.587710e-01 8.646929e-02
[55,] 0.0257236267 7.190557e-01 2.552207e-01
[56,] 0.0044884340 9.781328e-01 1.737878e-02
[57,] 0.0312129593 6.547331e-01 3.140540e-01
[58,] 0.2958341637 5.694232e-01 1.347426e-01
[59,] 0.0303580096 6.398237e-01 3.298183e-01
[60,] 0.0880779511 8.134764e-01 9.844565e-02
[61,] 0.2205273013 6.298451e-01 1.496276e-01
[62,] 0.0105098293 9.591134e-01 3.037677e-02
[63,] 0.0462415004 8.537411e-01 1.000174e-01
[64,] 0.0127683343 8.793406e-01 1.078911e-01
[65,] 0.1097850604 7.908698e-01 9.934511e-02
[66,] 0.0439788476 6.323928e-01 3.236284e-01
[67,] 0.0142403104 9.357246e-01 5.003511e-02
[68,] 0.0107489244 9.647242e-01 2.452687e-02
[69,] 0.0297161608 8.212906e-01 1.489932e-01
[70,] 0.0472514422 8.832597e-01 6.948890e-02
[71,] 0.0244685053 7.865167e-01 1.890147e-01
[72,] 0.0231707557 9.204749e-01 5.635439e-02
[73,] 0.0242412459 6.647915e-01 3.109672e-01
[74,] 0.0115014923 8.931765e-01 9.532196e-02
[75,] 0.0252735484 8.457138e-01 1.290126e-01
[76,] 0.0367090402 7.041271e-01 2.591639e-01
[77,] 0.0295101840 4.167745e-01 5.537153e-01
[78,] 0.0196749600 2.703807e-01 7.099444e-01
[79,] 0.0046165774 9.710517e-01 2.433168e-02
[80,] 0.1233870645 7.695546e-01 1.070584e-01
[81,] 0.0763633248 8.316087e-01 9.202798e-02
[82,] 0.0956781157 8.038125e-01 1.005094e-01
[83,] 0.0317355256 9.149948e-01 5.326970e-02
[84,] 0.0237392176 6.474085e-01 3.288522e-01
[85,] 0.0279975037 8.909775e-01 8.102497e-02
[86,] 0.0346333192 7.957193e-01 1.696474e-01
[87,] 0.0327144512 4.847696e-01 4.825159e-01
[88,] 0.0287006017 8.325287e-01 1.387707e-01
[89,] 0.0265872664 9.220512e-01 5.136156e-02
[90,] 0.0425465997 8.901942e-01 6.725921e-02
[91,] 0.0165454512 9.380639e-01 4.539066e-02
[92,] 0.0126391727 8.984548e-01 8.890606e-02
[93,] 0.0217893662 9.356063e-01 4.260431e-02
[94,] 0.2778346035 5.864740e-01 1.356914e-01
[95,] 0.0130250339 9.574755e-01 2.949948e-02
[96,] 0.0143369500 9.506283e-01 3.503480e-02
[97,] 0.0098869130 9.658287e-01 2.428443e-02
[98,] 0.0128272275 9.306614e-01 5.651133e-02
[99,] 0.3958967535 4.819128e-01 1.221905e-01
```

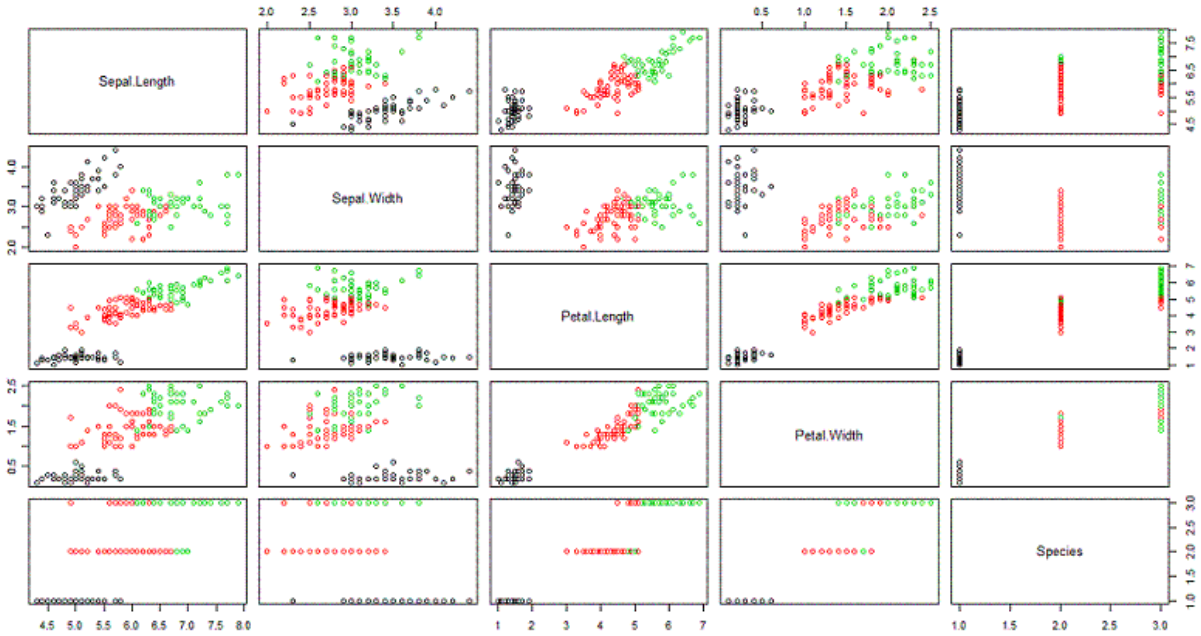
```
[100,] 0.0138782641 9.568112e-01 2.931057e-02
[101,] 0.0168213369 1.202409e-01 8.629378e-01
[102,] 0.0261693250 7.099212e-01 2.639094e-01
[103,] 0.0064283347 4.003438e-02 9.535373e-01
[104,] 0.0121558071 1.363357e-01 8.515085e-01
[105,] 0.0051285341 4.386983e-02 9.510016e-01
[106,] 0.0380603711 1.582822e-01 8.036574e-01
[107,] 0.0796613905 7.682136e-01 1.521250e-01
[108,] 0.0215250742 1.079049e-01 8.705700e-01
[109,] 0.0133896976 1.083710e-01 8.782393e-01
[110,] 0.0222927779 1.077325e-01 8.699748e-01
[111,] 0.0188704621 2.634073e-01 7.177222e-01
[112,] 0.0170114260 2.579486e-01 7.250400e-01
[113,] 0.0012069886 1.088884e-02 9.879042e-01
[114,] 0.0272794217 7.782927e-01 1.944279e-01
[115,] 0.0260263274 7.022101e-01 2.717635e-01
[116,] 0.0143300831 1.808070e-01 8.048629e-01
[117,] 0.0055448160 6.051225e-02 9.339429e-01
[118,] 0.0542553271 1.919346e-01 7.538101e-01
[119,] 0.0522340053 2.006703e-01 7.470957e-01
[120,] 0.0331219678 6.903576e-01 2.765205e-01
[121,] 0.0016396213 1.177291e-02 9.865875e-01
[122,] 0.0232292512 8.358772e-01 1.408936e-01
[123,] 0.0446065451 1.785215e-01 7.768719e-01
[124,] 0.0214638942 6.565434e-01 3.219927e-01
[125,] 0.0033893690 2.584416e-02 9.707665e-01
[126,] 0.0114583443 6.335614e-02 9.251855e-01
[127,] 0.0173352405 7.863541e-01 1.963106e-01
[128,] 0.0207474129 7.187218e-01 2.605308e-01
[129,] 0.0101190052 1.107064e-01 8.791746e-01
[130,] 0.0076951023 4.751066e-02 9.447942e-01
[131,] 0.0203964684 1.059183e-01 8.736853e-01
[132,] 0.0542244082 1.915598e-01 7.542157e-01
[133,] 0.0101190052 1.107064e-01 8.791746e-01
[134,] 0.0209695496 4.545456e-01 5.244849e-01
[135,] 0.0248052462 2.990893e-01 6.761055e-01
[136,] 0.0299588767 1.357829e-01 8.342583e-01
[137,] 0.0163979126 1.472580e-01 8.363440e-01
[138,] 0.0087535054 9.693234e-02 8.943141e-01
[139,] 0.0169168345 8.303003e-01 1.527828e-01
[140,] 0.0037050972 3.198944e-02 9.643055e-01
[141,] 0.0006389323 5.579855e-03 9.937812e-01
[142,] 0.0153630352 1.530446e-01 8.315924e-01
[143,] 0.0261693250 7.099212e-01 2.639094e-01
[144,] 0.0036930153 2.507113e-02 9.712359e-01
[145,] 0.0033893690 2.584416e-02 9.707665e-01
[146,] 0.0106923302 1.279688e-01 8.613389e-01
```

```
[147,] 0.0250155507 5.900274e-01 3.849571e-01
[148,] 0.0138756337 2.012898e-01 7.848346e-01
[149,] 0.0230709595 2.493458e-01 7.275832e-01
[150,] 0.0261065981 6.399365e-01 3.339569e-01

Closest hard clustering:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [46] 1 1 1 1 1 3 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [91] 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 2 2 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 3 3 3
[136] 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Available components:
[1] "centers"     "size"        "cluster"     "membership"  "iter"        "withinerror"
[7] "call"
```

The chart below shows the relevance of each party to each of the clusters generated. Each cluster generated can be identified with a color (green, red or black).



The graphs below show the classification of each instance by comparing each pair of attributes present in the database.



Analysis

Application of Fuzzy C-Means algorithm allowed a homogeneous grouping of classes as expected. Soon, the three generated classes have a very similar amount of instances present. The algorithm presented in addition to the class that was ranked a given instance, the relevance of this instance to that class. This information allows the person responsible for analyzing the results can devote their attention to the proceedings that do not have a high degree of relevance to a particular class.

The instances of the database analyzed that do not have a high degree of relevance, this degree being defined by the user should be analyzed for Chace whether they really belong to the class informed by the algorithm. In the chart presented pertinencia realize that some instances of Class 2, marked in red do not have a high amount of relevance. This possibly indicates that the algorithm may have erred in its classification as the values of the attributes of these instances do not identify with a high degree of certainty these instances.

References

J. C. Dunn (1973): "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters", Journal of Cybernetics 3: 32-57

J. C. Bezdek (1981): "Pattern Recognition with Fuzzy Objective Function Algorithms", Plenum Press, New York Tariq Rashid: “Clustering”

L. A. Zadeh (1965): "Fuzzy sets and systems". In: Fox J, editor. System Theory. Brooklyn, NY: Polytechnic Press, 1965: 29–39.

Iris Data Base in: http://archive.ics.uci.edu/ml/datasets/Iris

The R Project for Statistical Computing in: http://www.r-project.org/

W. Meira; M. Zaki: Fundamentals of Data Mining Algorithms in: http://www.dcc.ufmg.br/miningalgorithms/DokuWiki/doku.php

RockCluster

Introduction

Clustering is a useful technique for grouping data points such that points within a single group/cluster have similar characteristics (or are close to each other), while points in different groups are dissimilar.

We describe the ROCK (RObust Clustering using linKs) clustering algorithm which belongs to the class of agglomerative hierarchical clustering algorithms.

Technique/Algorithm

Algorithm

The steps involved in clustering using ROCK are described in the following figure. After drawing a random sample from the database, a hierarchical clustering algorithm that employs links is applied to the sampled points. Finally, the clusters involving only the sampled points are used to assign the remaining data points on disk to the appropriate clusters. In the following subsections, we first describe the steps performed by ROCK in greater detail.

Clustering Algorithm

ROCK`s hierarchical clustering algorithm is presented in the following figure. It accepts as input the set S of N sampled points to be clustred (that are drawn randomly from the original data set), and the number of desired clusters k. The procedure begins by computing the number of links between pairs of points in Step 1. Initially, each point is separate cluster. For each cluster i, we build a local heap q[i] and maintain the heap during the execution of the algorithm. q[i] contains every cluster j such that link[i,j] is non-zero. The clusters j in q[i] are ordered in the decreasing order of the goodness measure with respect to i, g(i,j).

In addition to the local heaps q[i] for each cluster i, the algorithm also maintains an additional global heap Q that contains all the clusters. Furthermore, the clusters in Q are ordered in the decreasing order of their best goodness measures. Thus, g(j,max(q[j])) is used to order the various clusters j in Q, where max(q[j]), the max element in q[j], is the best cluster to merge with cluster j. At each step, the max cluster j in Q and the max cluster q[j] are the best pair of clusters to be merged.

Computation of Links

For every point, after computing a list of its neighbors, the algorithm considers all pairs of its neighbors. For each pair, the point contributes one link. If the process is repeated for every point and the link count is incremented for each pair of neighbors, then at the end, the link counts for all pairs of points will be tained. If Mi is the size of the neighbor list for point i, then for point i, we have to increase the link count by one in M^2i entries. This, the complexity of the algorithm is the sum of M^2i which is O(N * Mm * Ma), where Ma and Mm are the average and maximum number of the neighbors for a point, respectively. In the worst case, the value of Mm can be n in which case the complexity of the algorithm becomes O(Ma * N^2). In practice, we expect Mm to be reasonably close to Ma and thus, for these cases, the complexity of the algorithm reduces to O(M^2a * n) on average.

Implementation

In order to use the K-Means algorithm in R, one must install "CBA" package. This package includes a function that performs the RockCluster process.

Install CBA Package

```
install.packages("cba")
```

Importing methods and algorithm

```
library("cba")
```

Usage:

```
rockCluster(x, n, beta = 1-theta, theta = 0.5, fun = "dist",
funArgs = list(method="binary"), debug = FALSE)
rockLink(x, beta = 0.5)
```

Arguments:

```
  X: a data matrix; for rockLink an object of class dist.
  n:  the number of desired clusters.
  beta: optional distance threshold.
  theta: neighborhood parameter in the range [0,1).
  fun: distance function to use.
  funArgs: a list of named parameter argu
```

If everything goes ok, an object rockCluster is returned. This object has the following components:

```
x: the data matrix or a subset of it.
cl: a factor of cluster labels.
size: a vector of cluster sizes.
beta: see above.
theta: see above.
rockLink: returns an object of class dist.
```

View

There is one way to show the result from this algorithm. That way would be printing the object RockCluster:

```
print(RockObject)
```

Example

For an example , we will use the algorithm with the "Mushroom" dataset provided by the CBA package:

```
data("Mushroom")
x <- as.dummy(Mushroom[-1])
rc <- rockCluster(x[sample(dim(x)[1],1000),], n=10, theta=0.8)
print(rc)
rp <- predict(rc, x)
table(Mushroom$class, rp$cl)
```

Output - Mushroom

```
Clustering:
computing distances ...
computing links ...
computing clusters ...
rockMerge: terminated with 36 clusters
> print(rc)
 data: x
 beta: 0.2
theta: 0.8
  fun: dist
 args: list(method = "binary")
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
 41 200  25 218 103  21  69  27 153  29  21  42   8   1  11   1   5   5   1   2
 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  1   1   1   1   1   1   1   1   1   2   1   1   1   1   1   1
> rp <- predict(rc, x)
dropping 18 clusters
computing distances ...
computing classes ...
> table(Mushroom$class, rp$cl)

              1    2    3    4    5    6    7    8    9   10   11   12   13
  edible      0 1728    0    0  768  192  512  192    0    0  190  288    0
  poisonous 288    0  192 1728    0    0    0    0 1296  255    0    0   65

              15   17   18   20   30 <NA>
  edible      95   32    0   22   23  166
  poisonous    0    0   36    0    0   56
```

Case Study

In this section, we illustrate a case study with RockCluster:

Scenario

Historically the U.S.A elections are characterized by two major political parties. One called **Republican Party** generally reflecting the American conservatism in the political spectrum and the other called **Democratic Party** known as more "liberal" or "progressive".

The idea is to use an database of the United States Cogressional votes provided by UCI Machine Learning Repository and perform the RockCluster technique to separate **Democrats** from **Republicans**.

Dataset

The Congressional voting dataset was obtained from the UCI Machine Learning Repository. It is the United States Cogressional Voting Records in 1984. Each record corresponds to one Congress man's votes on 16 issues (E.g., education spending, crime). All attributes are boolean with Yes (that is, 1) and No (0) values, and very few contain missing values. A classification label of Republican or Democrat is provided with each data record. The data set contains records for 168 Republicans and 267 Democrats.

Execution

R Code:

```
data(Votes)
x <- as.dummy(Votes[-17])
rc <- rockCluster(x, n=2, theta=0.73, debug=TRUE)
print(rc)
rf <- fitted(rc)
table(Votes$Class, rf$cl)
```

Output

The result of printing the components of the class returned by the function application is shown below:

```
rockMerge: terminated with 63 clusters
> print(rc)
 data: x
 beta: 0.27
theta: 0.73
  fun: dist
 args: list(method = "binary")
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
166   1   1 206   3   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 61  62  63
  1   1   1
> rf <- fitted(rc)
dropping 60 clusters
computing distances ...
computing classes ...
> table(Votes$Class, rf$cl)

               1   4   5 <NA>
  democrat    22 201   3   41
  republican 144   5   0   19
> ## Not run:
> ### large example from paper
>
```

Analysis

As the table illustrates, ROCK and the traditional algorithm, both identify two clusters, one containing a large number of republicans and the other containing a majority of democrats. However, in the cluster for republicans found by the traditional algorithm, around 25% of the members are democrats, while with ROCK, only 12% are

democrats. The improvement in the quality of clustering can be attributed to the usage of links by ROCK.

References

1. Meira Jr., W.; Zaki, M. Fundamentals of Data Mining Algorithms. [1]
2. CBA R package. [1]
3. UCI Machine Learning Repository [2]
4. Sudipto Guha; Rajeev Rastogi; Kyuseok Shim. Rock: A Robust Clustering Algorithm for Categorical Attributes

References

[1] http://cran.r-project.org/web/packages/cba/index.html
[2] http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

Biclust

Introduction

Over the last decade, bicluster methods have become more and more popular in different fields of two way data analysis and a wide variety of algorithms and analysis methods have been published.

Biclustering is an important new technique in two way data analysis. After Cheng and Church (2000) followed the initial bicluster idea of Hartigan (1972) and started to calculate bicluster on microarray data, a wide range of different articles were published dealing with different kinds of algorithms and methods to preprocess and analyze the results of such methods. Comparisons of several bicluster algorithms can be found, e.g., in Madeira and Oliveira (2004) or Prelic et al. (2006).

Why Biclustering?

- Simultaneous clustering of 2 dimensions;
- Large datasets where clustering leads to diffuse results;
- Only parts of the data influence each other;

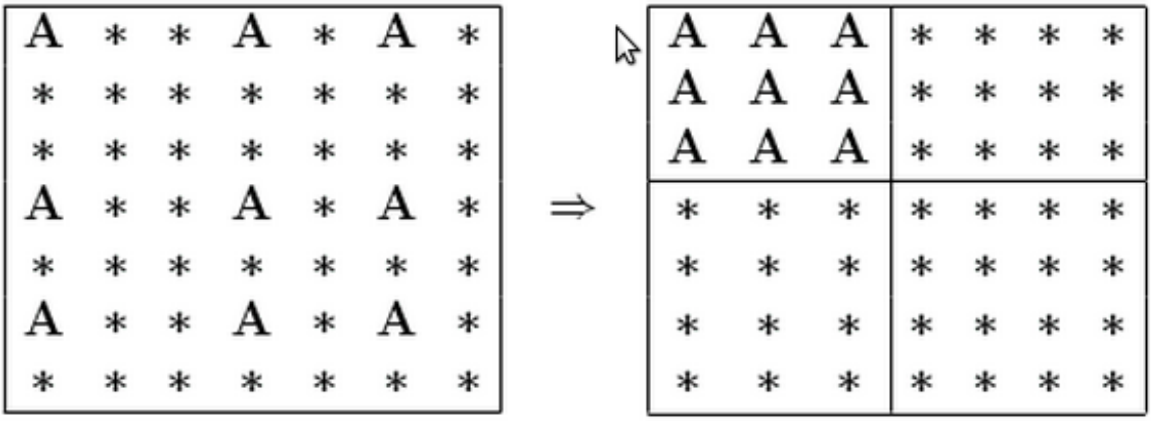
Initial Situation

	c_1	$\dots$	c_i	$\dots$	c_m
r_1	a_{11}	$\dots$	a_{i1}	$\dots$	a_{m1}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
r_j	a_{1j}	$\dots$	a_{ij}	$\dots$	a_{mj}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
r_n	a_{1n}	$\dots$	a_{in}	$\dots$	a_{mn}

Two-Way Dataset:

Goal

Finding subgroups of rows and columns which are as similar as possible to each other and as different as possible to the rest.



Technique/Algorithm

Algorithm

> Sebastian Kaiser and Friedrich Leisch started to implement a comprehensive bicluster toolbox in R (R Development Core Team, 2007).

It provides a growing list of bicluster methods, together with pre-processing and visualization techniques, using S4 classes and methods (Chambers, 1998). The software is open source and freely available from R-Forge at http://R-Forge.R-project.org.

> One of the main design principles of the package is to provide the results as an entity of Biclust-Class, an S4-class containing all information needed for postprocessing of results.

It consists of the four slots Parameters, RowxNumber, NumberxCol and Number. Slot Parameters contains parameters and algorithm used, Number the number of biclusters found. The RowxNumber and NumberxCol slots represents the biclusters that have been found. They are both logical matrices of dimension (rows of data × number of biclusters found) with a TRUE-value in RowxNumber[i,j] if row i is in bicluster j. NumberxCol is the same for the columns, but due to computational reasons, here the rows of the matrix represent the number of biclusters and the columns represent the columns of the data. So by simply calling data [Biclust@RowxNumber[,a] * Biclust@NumberxCol[a,]] the values of the bicluster a can be extracted.

> Objects of class Biclust-class are created using a uniform interface for all bicluster methods by calls of form biclust(x,method=BiclustMethod,...).

This generic function takes as inputs the preprocessed data matrix x, a bicluster algorithm represented as a Biclustmethod-Class and additional arguments. In the following we give a brief description of the five algorithms already implemented in the package, subsection headings correspond to the name of the respective Biclustmethod-Class. The naming scheme is BCxxx where xxx is an abbreviation for the name of the algorithm. Some methods have been chosen because open source code from the original authors is available, others have been newly implemented to make the overall toolbox as comprehensive as possible. Of course, there is always room for improvement, and more methods will be added to the package in the future. See also van Mechelen and Schepers (2006) for a discussion on main directions of bicluster calculation. Algorithms are described in alphabetic order and,

if not stated otherwise, functions were implemented in interpreted S code.

Implementation

Install

In order to use the Biclust algorithm in R, one must install Biclust package and library:

```
install.packages("biclust")
library("biclust")
```

Usage

```
# S4 method for signature 'matrix,BCBimax':
biclust(x, method=BCBimax(), minr=2, minc=2, number=100)
# S4 method for signature 'matrix,BCrepBimax':
biclust(x, method=BCrepBimax(), minr=2, minc=2, number=100, maxc=12)
```

Where the argumets are:

- x - A logical matrix which represents the data.
- method - Here BCBimax, to perform Bimax algorithm.
- minr - Minimum row size of resulting bicluster.
- minc - Minimum column size of resulting bicluster.
- number - Number of Bicluster to be found.
- maxc - Maximum column size of resulting bicluster.

If everything goes OK, an object Biclust is returned.

View

As we can see below, an object Biclust is returned and we can plot it or just get the final object.

```
> test <- matrix(rbinom(400, 50, 0.4), 20, 20)
> res1 <- biclust(test, method=BCCC(), delta=1.5, alpha=1, number=10)
> res1

An object of class Biclust

call:
      biclust(x = test, method = BCCC(), delta = 1.5, alpha = 1, number = 10)

Number of Clusters found:  10

First  5  Cluster sizes:
                   BC 1 BC 2 BC 3 BC 4 BC 5
Number of Rows:    "7"  "4"  "6"  "6"  "5"
Number of Columns: "5"  "8"  "4"  "5"  "6"
```

We'll give some examples of plots in the "Case Study" session.

Case Study

Scenario

As a standard example we ran all the algorithms on the BicatYeast data from Barkow et al. (2006). To do so the data has to be preprocessed and committed to the biclust function

together with the chosen algorithm (here Xmotifs) and parameters.

Datasets

BicatYeast

- Subsample of the Saccharomyces Cerevisiae organism (Yeast).
- Used to present bicluster algorithms by Barkow et al. (2006)
- Microarray data: 419 genes, 80 experiments.

Execution

```
data(BiclustYeast)
x <- discretize(Bicatyeast)
res <- biclust(x, method=BCXmotifs(), alpha=0.05, number=50)
```

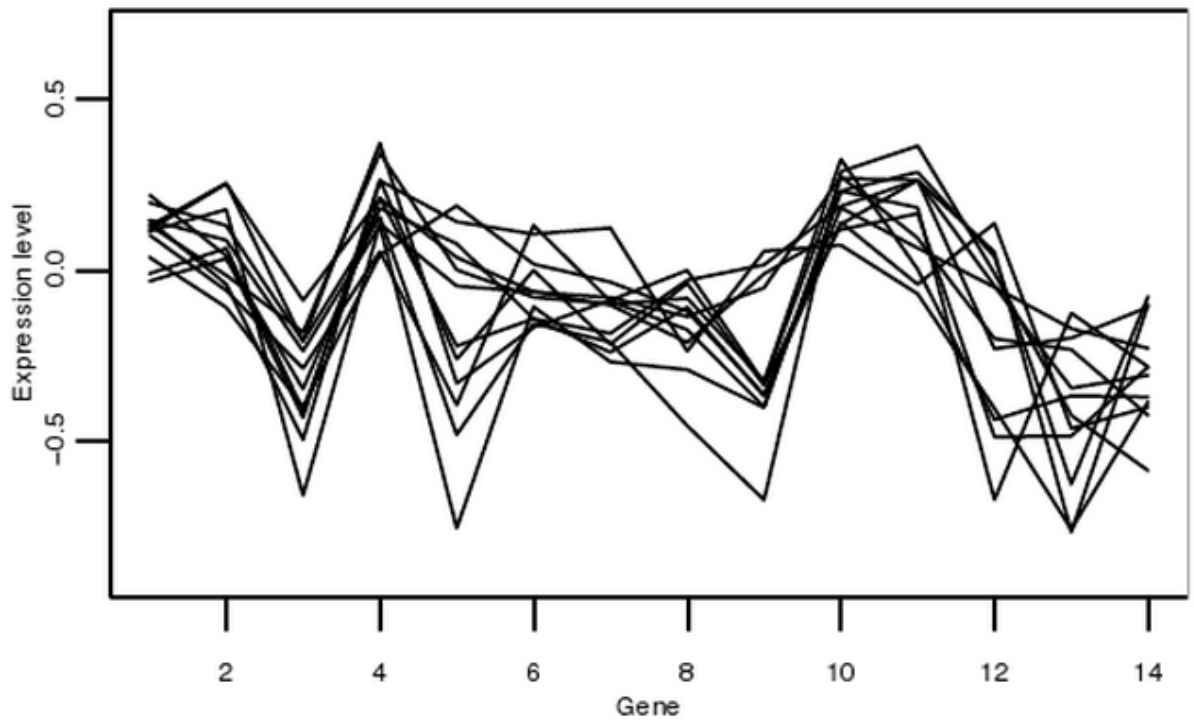
Output

To visualize the result you can simply call any visualization function on the result, for example:

```
> parallelCoordinates( x=BicatYeast, result=res, bicluster=4)
```

Example for parallel coordinates plot: Expression levels of conditions across their genes in the 4th bicluster in the result of

the Xmotifs algorithm.



Bicluster results similarity measure with an adaptation of Jaccard index:

	BCPlaid	BCXmotifs	BCCC	BCSpect.	BCBimax
BCPlaid	1.0000	0.0007	0.0116	0.0000	0.0000
BCXmotifs	0.0007	1.0000	0.1789	0.0935	0.0000
BCCC	0.0116	0.1789	1.0000	0.0898	0.0036
BCSpectral	0.0000	0.0935	0.0898	1.0000	0.0000
BCBimax	0.0000	0.0000	0.0036	0.0000	1.0000

Analysis

Table 1 shows the pairwise Jaccard indices of all bicluster algorithms. The Jaccard index is a measure of similarity between two

cluster results, zero means no concordance, one means that the results are identical. It can be seen that all algorithms find very different sets of biclusters. This can be partly explained by different pre-processing steps which were necessary such that the data conform to the respective assumptions of the algorithms.

Another important aspect is that we selected the first algorithms to implement to get a collection of algorithms which differ from

each other as much as possible. It is now very easy for practitioners to try various bicluster methods in R and choose the one which works best for given data set.

References

1. http://r-forge.r-project.org/projects/biclust/for the newest developments.
2. http://www.statistik.lmu.de/~kaiser/bicluster.html for Papers and Links.
3. BARKOW, S., BLEULER, S., PRELIC, A., ZIMMERMANN, P., and ZITZLER, E. (2006): Bicat: a biclustering analysis toolbox. Bioinformatics, 22,1282–1283.
4. CHAMBERS, J. M. (1998): Programming with data: A guide to the S Language. Chapman & Hall, London.
5. CHENG, Y. and CHURCH, G. M. (2000): Biclustering of expression data. In: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, 1,93–103.
6. GOVEART, G. and NADIF, M. (2003): Clustering with block mixture models. Pattern Recognition, 36, 463–473.
7. VAN MECHELEN, I. and SCHEPERS, J. (2006): A unifying model for biclustering. In: Compstat 2006 - Proceedings in Computational Statistics, 81–88.
8. MURALI, T. and KASIF, S. (2003): Extracting conserved gene expression motifs from gene expression. In: Pacific Symposium on Biocomputing, 8,77–88.
9. PRELIC, A., BLEULER, S., ZIMMERMANN, P., WIL, A., BUHLMANN, P., GRUISSEM, W., HENNING, L., THIELE, L., and ZITZLER, E. (2006): A

systematic comparison and evaluation of biclustering methods for gene expression data. Bioinformatics, 22(9),1122–1129.

1. SANTAMARIA, R., THERON, R., and QUINTALES, L. (2007): A framework to analyze biclustering results on microarray experiments. In: 8th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'07) ,Springer, Berlin, 770–779.
2. TURNER, H., BAILEY, T., and KRZANOWSKI, W. (2005): Improved biclustering of microarray data demonstrated through systematic performance tests. Computational Statistics and Data Analysis, 48,235–254.

Partitioning Around Medoids (PAM)

Introduction

Clustering is a unsupervised machine learning algorithm that groups entities, from a dataset, that have high degree of similarity in a same cluster. Nowadays lots of areas are using this kind of algorithms to separate datasets into groups in a automated way, and still have a good quality result.

The clustering process is not a universal process because that are a lot of kind of groups of datasets, for some of this the kind of metric is a relevant thing, for others the entities that represent each cluster are more interesting. Like datasets groups there are a lot of kind of clustering algorithms each one tries to take advantage of a kind of group of data, so this way each one of them is more suited to a more specific kind of data.

This section will explain a little more about the Partitioning Around Medoids (PAM) Algorithm, showing how the algorithm works, which are its parameters and what they mean, and an example of a dataset, and of how to execute the algorithm, and the result of that execution with the dataset as input.

The Partitioning Around Medoids (PAM) Algorithm

Algorithm

The PAM algorithm was developed by Leonard Kaufman and Peter J. Rousseeuw, and this algorithm is very similar to K-means, mostly because both are partitional algorithms, in other words, both break the datasets into groups, and both works trying to minimize the error, but PAM works with Medoids, that are an entity of the dataset that represent the group in which it is inserted, and K-means works with Centroids, that are artificially created entity that represent its cluster.

The PAM algorithm partitionates a dataset of n objects into a number k of clusters, where both the dataset and the number k is an input of the algorithm. This algorithm works with a matrix of dissimilarity, where its goal is to minimize the overall dissimilarity between the representants of each cluster and its members. The algorithm uses the following model to solve the problem: $F(x) = \text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} d(i,j) z_{ij}$

Subject to:

1. $\Sigma_{i=1}^{n} z_{ij} = 1$, j = 1,2,...,n
2. $z_{ij} \leq y_i$, i, j = 1,2,...,n
3. $\Sigma_{i=1}^{n} y_i = k$, k = number of clusters
4. $y_i , z_{ij} \in \{0,1\}$, i, j = 1,2,...,n

Where F(x) is the main function to minimize, where d(i,j) is the dissimilarity measurement between the entities i and j, and z_{ij} is a variable that ensures that only the dissimilarity between entities from the same cluster will be computed in the main function. The others expressions are constraints that have the following functions: (1.) ensures that every single entity is assigned to one cluster and only one cluster, (2.) ensures that the entity is assigned to its medoid that represent the cluster, (3.) ensures that there is a number exactly equals to k of clusters and (4.) let the decision variables assume just the values of 0 or 1.

The PAM algorithm can work over two kind of input, the first is the matrix representing every entity and the values of its variables, and the second is to work with the dissimilarity matrix directly, in the later the user can provide the dissimilarity directly as an input to the algorithm, instead of the data matrix representing the entities. Either way the algorithm reach an solution to the problem, in a general analysis the algorithm proceed this way:

Build phase:

1. Choose k entities to become the medoids, or in case these entities were provided use the them as the medoids;

2. Calculate the dissimilarity matrix if it was not informed;

3. Assign every entity to its closest medoid;

Swap phase:

4. For each cluster search if any of the entities of the cluster lower the average dissimilarity coefficient, if it does select the entity that lower the most this coefficient as the medoid for this cluster;

5. If at least the medoid from one cluster has changed go to (3), else end the algorithm.

As was said the PAM algorithm works with a matrix of dissimilarity, and to calculate this matrix the algorithm can use two metrics the first one is the euclidean, that are the root sum-of-squares of differences, while the second one is the manhattan distance that are the sum of absolute distances.

Implementation

The pseudocode of PAM algorithm is shown below:

Algorithm 1: PAM Algorithm Input: E = {e1,e2,...en} (dataset to be clustered or matrix of dissimilarity)

k (number of clusters)

metric (kind of metric to use on dissimilarity matrix)

diss (flag indicating that E is the matrix of dissimilarity or not)

Output: M = {m1,m2,...,mk} (vector of clusters medoids)

L = {l(e) | e = 1,2,...,n} (set of cluster labels of E)

foreach m_i € M **do**

$m_i \leftarrow e_j$ € E; (e.g. random selection)

end if diss ≠ true

Dissimilarity ← CalculateDissimilarityMatrix(E, metric);

else

Dissimilarity ← E;

end repeat

foreach e_i € E **do**

$l(e_i)$ ← argminDissimilarity(e_i, Dissimilarity, M);

end

changed ← false;

foreach m_i € M **do**

Mtmp ← SelectBestClusterMedoids(E, Dissimilarity, L);

end

if Mtmp ≠ M

M ← Mtmp;

changed ← true;

end

until changed = true;

In the R programming language, the PAM algorithm is available in the cluster package and can be called by the following command:

```
pam(x, k, diss, metric, medoids, stand, cluster.only, do.swap, keep.diss, keep.data,
trace.lev)
```

Where the parameters are:

x: numerical data matrix representing the dataset entities, or can be the dissimilarity matrix, it depends on the value of the diss parameter. In case x is a data matrix each row is an entity and each column is an variable, and in this case missing values are allowed as long as every pair of entities has at least one case not missing. In case x is a dissimilarity matrix it is not allowed to have missing values.

k: number of clusters that the dataset will be partitioned where 0 < k < n, where n is the number of entities.

diss: logical flag, if it is TRUE x is used as the dissimilarity matrix, if it is FALSE, then x will be considered as a data matrix.

metric: an string specifying each of the two metrics will be used to calculate the dissimilarity matrix, the metric variable can be “euclidean” to use the Euclidean distance, or can be “manhattan” to use the Manhattan distance.

stand: logical flag, if it is TRUE then the measurements in x will be standardized before calculating the dissimilarities. Measurements are standardized for each column, by subtracting the column's mean value and dividing by the variable's mean absolute deviation. If x is a dissimilarity matrix then this parameter is ignored.

cluster.only: logical flag, if it is TRUE, only the clustering will be computed and returned.

do.swap: logical flag, indicates if the swap phase should happen (TRUE) or not (FALSE).

keep.diss: logical flag indicating if the dissimilarities should (TRUE) or not (FALSE) be kept in the result.

keep.data: logical flag indicating if the input data x should (TRUE) or not (FALSE) be kept in the result.

trace.lev: an numeric parameters specifying a trace level for printing diagnostics during the build and swap phase of the algorithm. Default 0 does not print anything.

The PAM algorithm return a pam object that contains the information about the result of the execution of the algorithm.

Visualization

In R there are two ways of seeing the result of the PAM algorithm, the first one is to print the object that the algorithm returns, and the second one is to plot the data from the object creating a graphic of the result. The first way of visualizing the information is a bit more complicated to understand but it gives a more complete and accurate information, but the second way is a lot more easy to understand and let the user have a better view of the information and to add information that wold be relevant for him.

To view the data of the result of the execution of PAM algorithm in a textual way there are two ways one more simple that gives a more summarized information about the object, and another one that gives you a more complete information about it. In the two commands listed below the first one print the information in a summarized way, while the second one print it in a more complete way.

```
print (result)
summary (result)
```

The other way of visualizing the data from the result of the execution of the algorithm is using graphics and that can be done by using the following command:

```
plot (result)
```

Example: To show an example of use of the algorithm and a result from its execution it was used a simple dataset with few entities and few dimension, as it is shown in the table as follows:

Table 1: *Simple dataset*

Object	Attribute x	Attribute y
1	1	1
2	2	3
3	1	2
4	2	2
5	10	4
6	11	5
7	10	6
8	12	5
9	11	6

As we can see the data is separated in two clusters, so we will use an k = 2. The PAM algorithm can be executed as follows:

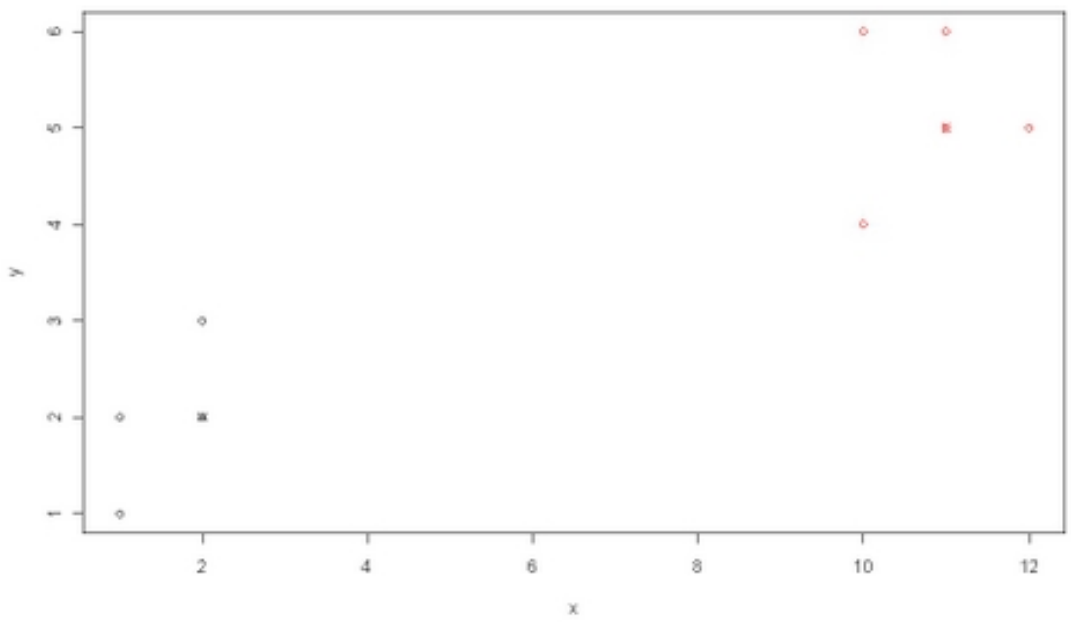
```
#load the table from a file
x <- read.table(“table.txt”)

#execute the pam algorithm with the dataset created for the example
result <- pam(x, 2, FALSE, "euclidean")

#print the results data in the screen
summary(result)

#plot a graphic showing the clusters and the medoids of each cluster
plot(result$data, col = result$clustering)
points(result$medoids, col = 1:2, pch = 4)
```

Printing the result form the execution gives you:

```
Medoids:
 ID  x y
4  4  2 2
6  6 11 5
Clustering vector:
1 2 3 4 5 6 7 8 9
1 1 1 1 2 2 2 2 2
Objective function:
  build     swap
1.255618 0.915849

Numerical information per cluster:
    size max_diss   av_diss diameter separation
[1,]    4 1.414214 0.8535534 2.236068   8.062258
[2,]    5 1.414214 0.9656854 2.236068   8.062258
```

```
Isolated clusters:
 L-clusters: character(0)
 L*-clusters: [1] 1 2

Silhouette plot information:
 cluster neighbor sil_width
3       1        2 0.8898942
4       1        2 0.8788422
1       1        2 0.8549629
2       1        2 0.8297000
6       2        1 0.8790384
9       2        1 0.8631441
8       2        1 0.8425790
7       2        1 0.8232848
5       2        1 0.7747713
Average silhouette width per cluster:
[1] 0.8633498 0.8365635
Average silhouette width of total data set:
[1] 0.8484685

36 dissimilarities, summarized :
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.0000  1.4142  8.3951  6.1559  9.9362 11.7050
Metric :  euclidean
Number of objects : 9

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"  "clusinfo"
   "silinfo"    "diss"       "call"
[10] "data"
```

While plotting gives you:



Case Study

In this section we will see a case study using PAM.

Scenario

In this case study it was used a part of the database iris available in the R package datasets. This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica. Using the data that this dataset provides us, it is natural to think of verifying if the flowers of each one of three species of iris are really similar to the others from the same specie, so in this case study it will be used the length and width of both petal and sepal to cluster the dataset into 3 groups and then verify if the clusters really match with the flowers species.

The dataset that was used into this case study consist of the following columns:

- **Flower:** An id of the flower;
- **Sepal.Length:** A numeric value of the length of the sepal in centimeters;
- **Sepal.Width:** A numeric value of the width of the sepal in centimeters;
- **Petal.Length:** A numeric value of the length of the petal in centimeters;
- **Petal.Width:** A numeric value of the width of the petal in centimeters;
- **Species:** A text identifying the specie of the flower.

Input Data

The input data is a table consisting of 50% (75 entities) of the original iris dataset that have 150 flowers and 5 attributes each. So the dataset used in this case study is represented by the following table:

Table 2: *Sample from iris dataset*

Flower	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
23	4.6	3.6	1.0	0.2	setosa
24	5.1	3.3	1.7	0.5	setosa
25	4.8	3.4	1.9	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
53	6.9	3.1	4.9	1.5	versicolor
54	5.5	2.3	4.0	1.3	versicolor
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
58	4.9	2.4	3.3	1.0	versicolor
59	6.6	2.9	4.6	1.3	versicolor

60	5.2	2.7	3.9	1.4	versicolor
61	5.0	2.0	3.5	1.0	versicolor
62	5.9	3.0	4.2	1.5	versicolor
63	6.0	2.2	4.0	1.0	versicolor
64	6.1	2.9	4.7	1.4	versicolor
65	5.6	2.9	3.6	1.3	versicolor
66	6.7	3.1	4.4	1.4	versicolor
67	5.6	3.0	4.5	1.5	versicolor
68	5.8	2.7	4.1	1.0	versicolor
69	6.2	2.2	4.5	1.5	versicolor
70	5.6	2.5	3.9	1.1	versicolor
71	5.9	3.2	4.8	1.8	versicolor
72	6.1	2.8	4.0	1.3	versicolor
73	6.3	2.5	4.9	1.5	versicolor
74	6.1	2.8	4.7	1.2	versicolor
75	6.4	2.9	4.3	1.3	versicolor
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica
103	7.1	3.0	5.9	2.1	virginica
104	6.3	2.9	5.6	1.8	virginica
105	6.5	3.0	5.8	2.2	virginica
106	7.6	3.0	6.6	2.1	virginica
107	4.9	2.5	4.5	1.7	virginica
108	7.3	2.9	6.3	1.8	virginica
109	6.7	2.5	5.8	1.8	virginica
110	7.2	3.6	6.1	2.5	virginica
111	6.5	3.2	5.1	2.0	virginica
112	6.4	2.7	5.3	1.9	virginica
113	6.8	3.0	5.5	2.1	virginica
114	5.7	2.5	5.0	2.0	virginica
115	5.8	2.8	5.1	2.4	virginica
116	6.4	3.2	5.3	2.3	virginica
117	6.5	3.0	5.5	1.8	virginica
118	7.7	3.8	6.7	2.2	virginica
119	7.7	2.6	6.9	2.3	virginica
120	6.0	2.2	5.0	1.5	virginica
121	6.9	3.2	5.7	2.3	virginica
122	5.6	2.8	4.9	2.0	virginica
123	7.7	2.8	6.7	2.0	virginica

124	6.3	2.7	4.9	1.8	virginica
125	6.7	3.3	5.7	2.1	virginica

Execution

The process was done as follows:

```
#import data
data <- read.table(“sampleiris.txt”)

#execution
result <- pam(data[1:4], 3, FALSE, “euclidean”)

#print results
summary(result)

#plot clusters
plot (data, col = result$clustering)
#add the medoids to the plot
points(result$medoids, col = 1:3, pch = 4)
```

Output

The following data was printed as result of the execution:

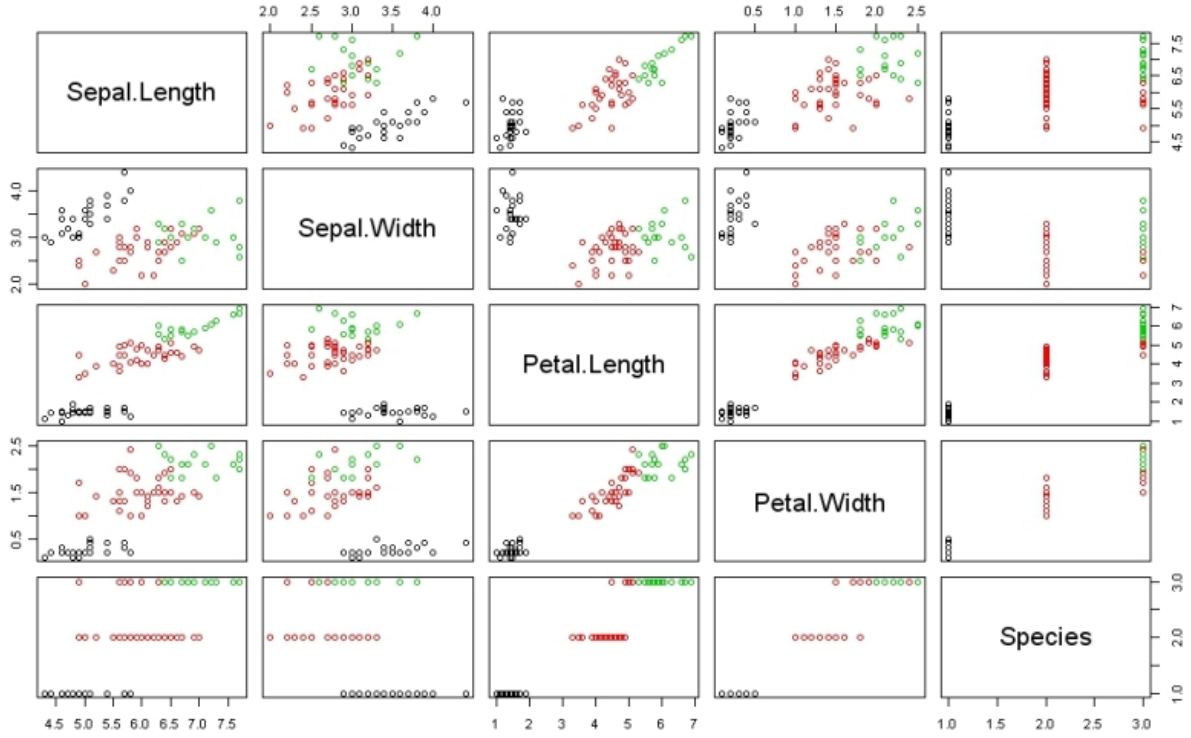
```
Medoids:
    ID Sepal.Length Sepal.Width Petal.Length Petal.Width
8    8          5.0         3.4          1.5         0.2
64  39          6.1         2.9          4.7         1.4
103 53          7.1         3.0          5.9         2.1
Clustering vector:
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22
  23  24  25  51  52  53  54  55  56
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   2   2   2   2   2   2
 57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 101 102 103
 104 105 106 107 108 109 110 111 112
  2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   3   2   3
   3   3   3   2   3   3   3   2   2
113 114 115 116 117 118 119 120 121 122 123 124 125
  3   2   2   3   3   3   3   2   3   2   3   2   3
Objective function:
    build      swap
0.7148339 0.6990539

Numerical information per cluster:
     size max_diss   av_diss diameter separation
[1,]   25 1.236932 0.5137400 2.042058  1.9000000
[2,]   34 1.951922 0.8085343 2.727636  0.3741657
```

```
[3,]   16 1.284523 0.7559609 2.147091  0.3741657

Isolated clusters:
 L-clusters: [1] 1
 L*-clusters: character(0)

Silhouette plot information:
    cluster neighbor   sil_width
1         1        2  0.84941732
5         1        2  0.84830238
8         1        2  0.84812593
18        1        2  0.84784555
12        1        2  0.83221128
22        1        2  0.82890349
20        1        2  0.82456328
3         1        2  0.82337894
7         1        2  0.81910409
10        1        2  0.81662688
11        1        2  0.80769429
2         1        2  0.80592613
13        1        2  0.80278163
4         1        2  0.79810574
23        1        2  0.79482977
24        1        2  0.78999596
17        1        2  0.78539723
21        1        2  0.78454015
25        1        2  0.77452963
6         1        2  0.75995941
9         1        2  0.74605493
14        1        2  0.74277337
19        1        2  0.72082914
15        1        2  0.71581750
16        1        2  0.66155611
68        2        3  0.60036142
56        2        3  0.59753885
62        2        3  0.59698924
72        2        3  0.59691421
70        2        3  0.59514179
54        2        3  0.58507022
67        2        3  0.56989428
60        2        1  0.56350914
63        2        3  0.55592514
75        2        3  0.54720666
74        2        3  0.53971473
64        2        3  0.53757677
69        2        3  0.51098390
65        2        1  0.50762488
```

```
107       2        3  0.48295375
55        2        3  0.46851074
52        2        3  0.46827948
59        2        3  0.44164146
66        2        3  0.42147865
71        2        3  0.41421605
73        2        3  0.41282512
122       2        3  0.40891392
120       2        3  0.40207904
57        2        3  0.39510378
114       2        3  0.37176468
124       2        3  0.34854822
102       2        3  0.33532624
61        2        1  0.32662688
58        2        1  0.20142024
51        2        3  0.19024422
115       2        3  0.16320750
53        2        3  0.11554863
112       2        3 -0.07433144
111       2        3 -0.07748205
103       3        2  0.59622203
106       3        2  0.59241159
108       3        2  0.58027197
110       3        2  0.56716967
123       3        2  0.56182697
121       3        2  0.55568135
119       3        2  0.53242285
118       3        2  0.52551154
125       3        2  0.51206488
105       3        2  0.49243542
101       3        2  0.45749953
113       3        2  0.44409513
109       3        2  0.37181492
117       3        2  0.26375026
116       3        2  0.21777715
104       3        2  0.21412781
Average silhouette width per cluster:
[1] 0.7931708 0.4153331 0.4678177
Average silhouette width of total data set:
[1] 0.5524757

2775 dissimilarities, summarized :
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1000  1.1136  2.5080  2.6329  3.9006  7.0852
Metric :  euclidean
Number of objects : 75
```

```
Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"  "clusinfo"
   "silinfo"    "diss"       "call"
[10] "data"
```

And the following graphic was generated as well:



Analysis

After the execution of the algorithm and the analysis of the data it was possible to tell that the clusters was well grouped and correlated with the species of each flower. In the data there was a total of 75 elements, 25 from *Setosa* species, 25 from *Versicolor* species and 25 from *Virginica* species, and the algorithm clustered the elements from *Setosa* as cluster 1, the ones from *Versicolor* as cluster 2 and the ones from *Virginica* as cluster 3. After verifying the results we find that from 75 elements, 66 were correctly clustered, giving an error margin of 12%—which is a very good result.

References

1. The R Development Core Team, R: A Language and Environment for Statistical Computing.
2. Kaufman, L., Rousseeuw, P. J., Clustering by Means of Medoids.

CLUES

Introduction

Cluster analysis, the organization of patterns into clusters based on similarity (or dissimilarity) measures, is an unsupervised technique widely applied to a broad range of disciplines. It has many applications in data mining, as large data sets need to be partitioned into smaller and homogeneous groups. Clustering techniques have a wide use, such as artificial intelligence, pattern recognition, economics, biology and marketing. Clustering techniques are important, and its importance increases as the amount of data and processing power of computers increases.

clues: Nonparametric Clustering Based on Local Shrinking

The R package clues aims to provide an estimate of the number of clusters and, at the same time, obtain a partition of data set via local shrinking. The shrinking procedure in clues is done by the mean-shift algorithm. It is also influenced by the K-nearest neighbor approach, not using kernel functions. The value K starts with a small number and increases gradually until the measure of strength, CH Index or Silhouette Index, is optimized. A major contribution of the CLUES algorithm is its ability to identify and deal with irregular elements. To help validation of the quality of the number of clusters and the clustering algorithm, five indices are available to support decision making.

Algorithm

CLUES (CLUstEring based on local Shrinking) algorithm has three procedures:

1. Shinking
2. Partition
3. Determination of optimal K

Shrinking

For the shrinking procedure, the data set is calibrated in a way that pushes each data point towards its focal point, the cluster center or mode of the probability density function. The number K is chosen iteratively and, due to the robustness of the median, each data point moves to the element-wise median of the set. This median consists of its K nearest neighbors according to dissimilarity measures, either Euclidean distance or Pearson Correlation.

For this process, a stopping rule needs to be set by the user, beyond which excess iterations will not be significant in terms of accuracy. The mutual gaps in data are apparent after this shrinking procedure.

Partitioning

The partition procedure uses the calibrated data obtained from shrinking. This data is used in place of the original data set. The partitioning starts by picking one arbitrary data point and replacing it by its nearest fellow point, recording their distance. The same move is applied to this fellow point. The selection is without replacement. Once a data point is picked for replacement, it is not picked again in that run. To separate the groups, the summation of the mean distance and 1:5 times the intequartile range is introduced. Each new fellow surpass move creates a new group, with a incremented group index.

Optimal K

Optimal K calculation involves optimizing the strength measure index, either CH Index or Silhouette Index. A factor *f* is introduced to improve the speed of computation. This factor is, by default, 0.05. Users may modify this factor, but must be aware of the large computation time associated with large factors. The choice of K has little effect on clustering result as long as it lies in a neighborhood of the optimal K. The factor 0.05 is chosen to minimize additional computation that will not substantially affect the outcome.

To strengthen the power of the calculation, we initialize K to be fn and set the size of increment K to be fn for the next iterations. We then use the calibrated data form the previous step as the new data.

Implementation

Instalation

```
R > install.packages("clues")
```

Usage

```
library("clues")
```

clues procedure

```
clues(y, n0, alpha, eps, itmax, K2.vec, strengthMethod, strengthIni, disMethod, quiet)
```

Parameters

- y: data matrix which is an R matrix object (for dimension > 1) or vector object (for dimension=1) with rows being observations and columns being variables.
- n0: a guess for the number of clusters. Default value is 5.
- alpha: speed factor. Default set as 0.05.
- eps: a small positive number. A value is regarded as zero if it is less than 'eps'. Default value is 1.0e-4.
- itmax: maximum number of iterations allowed. Default is 20.
- K2.vec: range for the number of nearest neighbors for the second pass of the iteration. Default is n0 (5).
- strengthMethod: specifies the preferred measure of the strength of the clusters (i.e., compactness of the clusters). Two available methods are "sil" (Silhouette index) and "CH" (CH index).
- strengthIni: initial value for the lower bound of the measure of the strength for the clusters. Any negative values will do.
- disMethod: specification of the dissimilarity measure. The available measures are "Euclidean" and "1-corr".
- quiet: logical. Indicates if intermediate results should be output.

Values

This section lists the values that can be viewed when running clues.

- K: number of nearest neighbors can be used to get final clustering.
- size: vector of the number of data points for clusters.
- mem: vector of the cluster membership of data points. The cluster membership takes values: 1, 2, ..., g, where g is the estimated number of clusters.
- g: an estimate of the number of clusters.
- CH: CH index value for the final partition if 'strengthMethod' is "CH".
- avg.s: average of the Silhoutte index value for the final partition if 'strengthMethod' is "sil".
- s: vector of Silhoutte indices for data points if 'strengthMethod' is "sil".
- K.vec: number of nearest neighbors used for each iteration.
- g.vec: number of clusters obtained in each iteration.

- myupdate: logical. Indicates if the partition obtained in the first pass is the same as that obtained in the second pass.
- y.old1: data used for shrinking and clustering.
- y.old2: data returned after shrinking and clustering.
- y: a copy of the data from the input.
- strengthMethod: a copy of the strengthMethod from the input.
- disMethod: a copy of the dissimilarity measure from the input

Example

We will show an example of how to run clues using the Maronna data set. This set has 4 slightly overlapped clusters in the two-dimensional space. Each cluster contains 50 data points. The Maronna data is a simulated data set. The data are drawn from 4 bivariate normal distributions with identity covariance matrix and mean vectors $\mu = \{(0,0), (4,0), (1,6), (5,7)\}$.

```
R > data(Maronna)
R > maronna <- Maronna$maronna
R > res <- clues
R > res <- clues(maronna, quiet = TRUE) # run clues
```

The results are shown below

```
R > summary(res)

Number of data points:
[1] 200

Number of variables:
[1] 2

Number of clusters:
[1] 4

Cluster sizes:
[1] 53 47 50 50

Strength method:
[1] "sil"

avg Silhouette:
[1] 0.5736749

dissimilarity measurement:
[1] "Euclidean"
```

Plotting the results, we can see, in figure 1, the four clusters found by the clues algorithm.

References

1. Meira Jr., W.; Zaki, M. Fundamentals of Data Mining Algorithms. [1]
2. Cluster R package. [29]
3. Wang, X., Qiu, W., Zamar, R. H., CLUES: A non-parametric clustering method based on local shrinking [1]

References

[1] http://www.stat.ubc.ca/~ruben/website/cv/clues.pdf

Self-Organizing Maps (SOM)

Introduction

The Kohonen Self-Organizing Feature Map (SOFM or SOM) is a clustering and data visualization technique based on a neural network viewpoint. As with other types of centroid-based clustering, the goal of SOM is to find a set of centroids (reference or codebook vector in SOM terminology) and to assign each object in the data set to the centroid that provides the best approximation of that object. In neural network terminology, there is one neuron associated with each centroid [1].

As with incremental K-means, data objects are processed one at a time and the closest centroid is updated. Unlike K-means, SOM impose a topographic ordering on the centroids and nearby centroids are also updated. The processing of points continues until some predetermined limit is reached or the centroids are not changing very much. The final output of the SOM technique is a set of centroids that implicitly define clusters. Each cluster consist of the points closest to a particular centroid .

SOM is a clustering technique that enforces neighborhood relationships on the resulting cluster centroids. Because of this, clusters that are neighbors are more related to one another than clusters that are not. Such relationships facilitate the interpretation and visualization of the clustering results. Indeed, this aspect of SOM has been exploited in many areas, such as visualizing Web documents or gene array data.

Algorithm

A distinguishing feature of SOM is that it imposes a topographic(spacial) organization on the centroids (neurons). Figure 1 shows an example of a two-dimensional SOM in which the centroids are represented by nodes that are organized in a rectangular lattice. Each centroid is assigned a pair of coordinates(i,j). Sometimes, such a network is drawn with links between adjacent nodes, but can be misleading because the influence of one centroid on another is via a neighborhood that is defined in terms of coordinates, not links. There are many types of SOM neural networks, but it will be focus on to two-dimensional SOMs with a rectangular or hexagonal organization of the centroids.

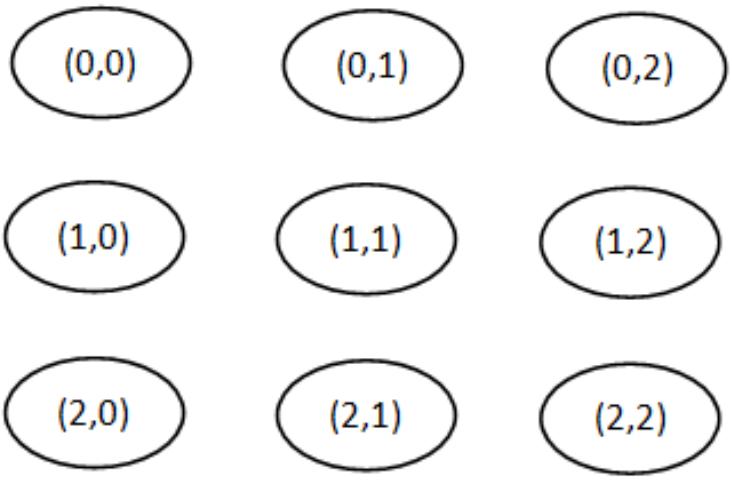


Figure 1: Two-dimensional 3-by-3 rectangular SOM neural network .

Even though SOM is similar to *K-means*, there is a fundamental difference. Centroids used in SOM have a predetermined topographic ordering relationship. During the training process, SOM uses each data point to update the closest centroid and centroids that are nearby in the topographic ordering. In this way, SOM produces an ordered set of centroids for any given data set. In other words, the centroids that are close to each other in the SOM grid are more closely related to each other than to the centroids that are farther away. Because of this constraint, the centroids of a two-dimensional SOM can be viewed as lying on a two-dimensional surface that tries to fit the n-dimensional data as well as possible. The SOM centroids can also be thought of as the result of a nonlinear regression with respect to the data points. At a high level, clustering using the SOM technique consists of the steps described in Algorithm below:

```
1: Initialize the centroids.
2: repeat
3:    Select the next object.
4:    Determine the closest centroid to the object.
5:    Update this centroid and the centroids that are close, i.e., in a specified neighborhood.
6: until The centroids don't change much or a threshold is exceeded.
7: Assign each object to its closest centroid and return the centroids and clusters.
```

Implementation

For R [2] (R Development Core Team 2007), three packages are available from the Comprehensive R Archive Network (CRAN [3]) implementing standard SOMs [4]

- The *kohonen* package implements self-organizing maps as well as some extensions for supervised pattern recognition and data fusion.
- The *som* package provides functions for self-organizing maps.
- The *wccsom* package SOM networks for comparing patterns with peak shifts.

For this discussion the focus is on the *kohonen* package because it gives SOM standards features and order extensions. The R package *kohonen* provides functions for self-organizing maps. It also provides two extensions that allow the use of SOMs for classification and regression tasks as well as data mining tasks. It specifically emphasizes visualisation. The basic functions are: **som** for the usual unsupervised form of self-organizing maps; **xyf** for supervised self-organizing maps and X-Y fused maps, which are useful when additional information in the form of, e.g., a class variable is available for all objects; **bdk**, an alternative formulation called bi-directional Kohonen maps; and finally, from version 2.0.0 on, the generalisation of the **xyf** maps to more than two layers of information, in the

function **supersom**. These functions can be used to define the mapping of the objects in the training set to the units of the map [5].

Several data sets are included in the kohonen package: the wine data from the UCI Machine Learning Repository [6][7], near-infrared spectra from ternary mixtures of ethanol, water and iso-propanol, measured at different temperatures described by Wülfert et al. (1998) [8], and finally a set of microarray data, the yeast data from Spellman et al. (1998)[9]. The wine data set contains information on a set of 177 Italian wine samples from three different grape cultivars; thirteen variables (such as concentrations of alcohol and flavonoids, but also color hue) have been measured. The yeast data are a subset of the original set containing 6178 genes, which are assumed to be related to the yeast cell cycle. The set contains 800 genes for which, using six different synchronization methods, time-dependent expressions have been measured .

The different types of self-organizing maps can be obtained by calling the functions **som**, **xyf**, **bdk**, or **supersom**, with the appropriate data representation as the first argument(s). Several other arguments provide additional parameters, such as the map size, the number of iterations, etcetera. The object that is returned can then be used for inspection, plotting, mapping, and prediction. Below we will show the functions available in the package. Visualization functions will be discussed at Visualization topic .

Function som

Function **som** implement the standard form of self-organizing maps [10].

```
som(data, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01), radius = quantile(nhbrdist, 0.67) * c(1, -1), init,
toroidal = FALSE, n.hood,  keep.data = TRUE)
```

the arguments are:

- **data:** a matrix, with each row representing an object.
- **grid:** a grid for the representatives.
- **rlen:** the number of times the complete data set will be presented to the network.
- **alpha:** learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over rlen updates.
- **radius:** the radius of the neighbourhood, either given as a single number or a vector (start, stop). If it is given as a single number the radius will run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
- **init:** the initial representatives, represented as a matrix. If missing, chosen (without replacement) randomly from 'data'.
- **toroidal:** if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map,the number of rows must be even.
- **n.hood:** the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
- **keep.data:** save data in return object.

return an object of class "kohonen" with components:

- **data:** data matrix, only returned if keep.data == TRUE.
- **grid:** the grid, an object of class "somgrid".
- **codes:** a matrix of code vectors.
- **changes:** vector of mean average deviations from code vectors.
- **unit.classif:** winning units for all data objects, only returned if keep.data == TRUE.
- **distances:** distances of objects to their corresponding winning unit, only returned if keep.data == TRUE.
- **toroidal:** whether a toroidal map is used.

- **method:** the type of som, here "som".

Function xyf

Function **xyf** is a supervised version of self-organizing maps for mapping high-dimensional spectra or patterns to 2D .

```
xyf(data, Y, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01), radius = quantile(nhbrdist, 0.67) * c(1, -1),
xweight = 0.5, contin,  toroidal = FALSE, n.hood, keep.data = TRUE)
```

the arguments are:

- **data:** a matrix, with each row representing an object.
- **Y:** property that is to be modelled. In case of classification, Y is a matrix of zeros, with exactly one '1' in each row indicating the class. For prediction of continuous properties, Y is a vector. A combination is possible, too, but one then should take care of appropriate scaling.
- **grid:** a grid for the representatives.
- **rlen:** the number of times the complete data set will be presented to the network.
- **alpha:** learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over rlen updates.
- **radius:** the radius of the neighbourhood, either given as a single number or a vector start, stop). If it is given as a single number the radius will run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
- **xweight:** the weight given to the X map in the calculation of distances for updating Y. Default is 0.5.
- **contin:** parameter indicating whether Y is continuous or categorical. The default is to check whether all row sums of Y equal 1: in that case contin is FALSE.
- **toroidal:** if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.
- **n.hood:** the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
- **keep.data:** save data in return value.

return an object of class "kohonen" with components:

- **data:** data matrix, only returned if keep.data == TRUE.
- **Y:** Y, only returned if keep.data == TRUE.
- **contin:** parameter indicating whether Y is continuous or categorical.
- **grid:** the grid, an object of class "somgrid".
- **codes:** list of two matrices, containing codebook vectors for X and Y, respectively.
- **changes:** matrix containing two columns of mean average deviations from code vectors. Column 1 contains deviations used for updating Y; column 2 for updating X.
- **toroidal:** whether a toroidal map is used.
- **unit.classif:** winning units for all data objects, only returned if keep.data == TRUE.
- **distances:** distances of objects to their corresponding winning unit, only returned if keep.data == TRUE.
- **method:** the type of som, here "xyf".

Function bdk

Supervised version of self-organising maps for mapping high-dimensional spectra or patterns to 2D: the Bi-Directional Kohonen map .

```
bdk(data, Y, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01), radius = quantile(nhbrdist,0.67)
* c(1, -1), xweight = 0.75, contin, toroidal = FALSE, n.hood, keep.data = TRUE)
```

the arguments are:

- **data:** a matrix, with each row representing an object.
- **Y:** property that is to be modelled. In case of classification, Y is a matrix with exactly one '1' in each row indicating the class, and zeros elsewhere. For prediction of continuous properties, Y is a vector. A combination is possible, too, but one then should take care of appropriate scaling.
- **grid:** a grid for the representatives.
- **rlen:** the number of times the complete data set will be presented to the network.
- **alpha:** learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over rlen updates.
- **radius:** the radius of the neighbourhood, either given as a single number or a vector (start, stop). If it is given as a single number the radius will run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
- **xweight:** the initial weight given to the X map in the calculation of distances for updating Y, and to the Y map for updating X. This will linearly go to 0.5 during training. Defaults to 0.75.
- **contin:** parameter indicating whether Y is continuous or categorical. The default is to check whether all row sums of Y equal 1: in that case contin is FALSE.
- **toroidal:** if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.
- **n.hood:** the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
- **keep.data:** save data in return value.

return an object of class "kohonen" with components:

- **data:** data matrix, only returned if keep.data == TRUE.
- **Y:** Y, only returned if keep.data == TRUE.
- **contin:** parameter indicating whether Y is continuous or categorical.
- **grid:** the grid, an object of class "somgrid".
- **codes:** list of two matrices, containing codebook vectors for X and Y, respectively.
- **changes:** matrix containing two columns of mean average deviations from code vectors. Column 1 contains deviations used for updating Y; column 2 for updating X.
- **toroidal:** whether a toroidal map is used.
- **unit.classif:** winning units for all data objects, only returned if keep.data == TRUE.
- **distances:** distances of objects to their corresponding winning unit, only returned if keep.data== TRUE.
- **method:** the type of som, here "bdk".

Function supersom

An extension of xyf maps to multiple data layers, possibly with different numbers of variables (though equal numbers of objects) .

```
supersom(data, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01), radius = quantile(nhbrdist, 0.67) * c(1, -1), contin,
toroidal = FALSE,  n.hood, whatmap = NULL, weights = 1, maxNA.fraction = .5, keep.data = TRUE)
```

the arguments are:

- **data:** list of data matrices.
- **grid:** a grid for the representatives: see somgrid.
- **rlen:** the number of times the complete data set will be presented to the network.
- **alpha:** learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over rlen updates.
- **radius:** the radius of the neighbourhood, either given as a single number or a vector (start, stop). If it is given as a single number the radius will run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
- **contin:** parameter indicating whether data are continuous or categorical. The default is to check whether all row sums equal 1: in that case contin is FALSE.
- **toroidal:** if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.
- **n.hood:** the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
- **whatmap:** For supersom maps: what layers to use in the mapping.
- **weights:** the weights given to individual layers. Default is 1/n, with n the number of layers.
- **maxNA.fraction:** the maximal fraction of values that may be NA to prevent the row or column to be removed.
- **keep.data:** save data in return value.

return an object of class "kohonen" with components:

- **data:** data matrix, only returned if keep.data == TRUE.
- **contin:** parameter indicating whether elements of data are continuous or categorical.
- **na.rows:** indices of objects (rows) that are removed because at least one of the layers has to many NAs for these objects.
- **unit.classif:** winning units for all data objects, only returned if keep.data == TRUE.
- **distances:** distances of objects to their corresponding winning unit, only returned if keep.data == TRUE.
- **grid:** the grid, an object of class somgrid.
- **codes:** a list of matrices containing codebook vectors.
- **changes:** matrix of mean average deviations from code vectors; every map corresponds with one column.
- **toroidal:** whether a toroidal map is used.
- **n.hood:** the shape of the neighbourhood, either "circular" or "square". The latter is the
- default for rectangular maps, the former for hexagonal maps.
- **weights:** For supersom maps: weights of layers uses in the mapping.
- **whatmap:** For supersom maps: what layers to use in the mapping.
- **method:** type of map, here "supersom".

Function predict.kohonen

Map objects to a trained Kohonen map, and return for each object the property associated with the corresponding winning unit .

```
## S3 method for class 'kohonen':
predict(object, newdata, trainX, trainY, unit.predictions, threshold = 0, whatmap = NULL, weights = 1, ...)
```

the arguments are:

- **object:** Trained network.
- **newdata:** Data matrix for which predictions are to be made. If not given, defaults to the training data (when available).
- **trainX:** Training data for obtaining predictions for unsupervised maps; necessary for som maps trained with the keep.data = FALSE option.
- **trainY:** Values for the dependent variable for the training data; necessary for som and supersom maps.
- **unit.predictions:** Possible override of the predictions for each unit.
- **threshold:** Used in class predictions; see classmat2classvec.
- **whatmap:** For supersom maps: what layers to use in the mapping.
- **weights:** For supersom maps: weights of layers uses in the mapping.
- **...:** Currently not used.

returns a list with components:

- **prediction:** predicted values for the properties of interest. When multiple values are predicted, this element is a list, otherwise a vector or a matrix.
- **unit.classif:** unit numbers to which objects in the data matrix are mapped.
- **unit.predictions:** mean values associated with map units. Again, when multiple properties are predicted, this is a list.

Function classvec2classmat

Convert a classification vector into a matrix or the other way around .

```
classvec2classmat(yvec)
classmat2classvec(ymat, threshold=0)
```

the arguments are:

- **yvec:** class vector. Usually integer values, but other types are also allowed.
- **ymat:** class matrix: every column corresponds to a class.
- **threshold:** only classify into a class if the probability is larger than this threshold.

return:

- **classvec2classmat:** returns the classification matrix, where each column consists of zeros and ones.
- **classmat2classvec:** returns a class vector (integers).

Function check.whatmap

Check the validity of a whatmap argument .

```
check.whatmap(x, whatmap)
```

the arguments are:

- *x:* Either a kohonen object from supersom, or a list of data matrices that can be used as input data for supersom.
- **whatmap:** An indication of a subset of the data; either by naming the elements, or giving indices. If whatmap equals NULL, no selection is performed.

Returns:

- Returns a numerical vector with the indices of the selected layers.

Function map.kohonen

Map a data matrix onto a trained SOM .

```
## S3 method for class 'kohonen':
map(x, newdata, whatmap = NULL, weights, scale.distances = (nmaps > 1), ...)
```

the arguments are:

- **x:** A trained supervised or unsupervised SOM obtained from functions som, xyf or bdk.
- **newdata:** Data matrix, with rows corresponding to objects.
- **whatmap:** For supersom maps: the layers to take into account.
- **weights:** For supersom maps: weights of the layers that are used for mapping.
- **scale.distances:** whether to rescale distances per layer in the case of supersom maps (default): if TRUE the maximal distance of each layer equals one. If the absolute values of the distances per layer should be used, this argument should be set to FALSE. Note that in that case, when mapping the training data, the result returned by map.kohonen will differ from the mapping present in the map.
- **...:** Currently ignored.

return a list with elements:

- **unit.classif:** a vector of units that are closest to the objects in the data matrix.
- **dists:** distances (currently only Euclidean distances) of the objects to the units.
- **whatmap,weights,scale.distances:** Values used for these arguments.

Function unit.distances

Calculate distances between units in a SOM .

```
unit.distances(grid, toroidal)
```

the arguments are:

- **grid:** an object of class somgrid.
- **toroidal:** if true, edges of the map are joined so that the topology is that of a torus.

return:

- Returns a (symmetrical) matrix containing distances. When grid$n.hood equals "circular", Euclidean distances are used; for grid$n.hood is "square" maximum distances. If toroidal equals TRUE, maps are joined at the edges and distances are calculated for the shortest path.

Function tricolor

Function provides colour values for SOM units in such a way that the colour changes smoothly in every direction .

```
tricolor(grid, phis = c(0, 2 * pi/3, 4 * pi/3), offset = 0)
```

the arguments are:

- **grid:** An object of class somgrid, such as the grid element in a kohonen object.
- **phis:** A vector of three rotation angles. Values for red, green and blue are given by the y-coordinate of the units after rotation with these three angles, respectively. The default corresponds to (approximate) red colour of the middle unit in the top row, and pure green and blue colours in the bottom left and right units, respectively. In case of a triangular map, the top unit is pure red.
- **offset:** Defines the minimal value in the RGB colour definition (default is 0). By supplying a value in the range [0, .9], pastel-like colours are provided.

return:

- Returns a matrix with three columns corresponding to red, green and blue. This can be used in the rgb function to provide colours for the units.

View

After the training phase, one can use several plotting functions for the visualisation; the package can show where objects are mapped, has several options for visualizing the codebook vectors of the map units, and provides means to assess the training progress. Summary functions exist for all SOM types. Furthermore, one can easily project new data into the trained map; this provides possibilities for property estimation .

Functions summary and print

Summary and print methods for kohonen objects. The print method shows the dimensions and the topology of the map; if information on the training data is included, the summary method additionally prints information on the size of the data and the mean distance of an object to its closest codebookvector, which is an indication of the quality of the mapping .

```
## S3 method for class 'kohonen':
summary(object, ...)
## S3 method for class 'kohonen':
print(x, ...)
```

the arguments are:

- **x, object:** a kohonen object
- **...:** Not used.

return

```
> print(wine.som)
som map of size 5x4 with a hexagonal topology.
Training data included.
```

Figure 2: Information retorned by function print about wine data

```
> summary(wine.som)
som map of size 5x4 with a hexagonal topology.
Training data included; dimension is 177 by 13
Mean distance to the closest unit in the map: 3.580196
```

Figure 3: Information retorned by function summary about wine data

Function plot.kohonen

Plot self-organising map, obtained from function kohonen. Several types of plots are supported .

```
## S3 method for class 'kohonen':
plot(x, type = c("codes", "changes", "counts", "dist.neighbours", "mapping", "property", "quality"), classif = NULL,
labels = NULL, pchs =  NULL, main = NULL, palette.name = heat.colors, ncolors, bgcol = NULL, zlim = NULL, heatkey = TRUE,
property, contin, whatmap = NULL, codeRendering = NULL, keepMargins = FALSE, heatkeywidth = .2, ...)
```

the arguments are:

- **x:** kohonen object.
- **type:** type of plot.
- **classif:** classification object, as returned by predict.kohonen, or vector of unit numbers. Only needed if type equals "mapping" and "counts".
- **labels:** labels to plot when type equals "mapping".
- **pchs:** symbols to plot when type equals "mapping".
- **main:** title of the plot.
- **palette.name:** colors to use as unit background for "codes", "counts", "prediction", "property", and "quality" plotting types.
- **ncolors:** number of colors to use for the unit backgrounds. Default is 20 for continuous data, and the number of distinct values (if less than 20) for categorical data.
- **bgcol:** optional argument to colour the unit backgrounds for the "mapping" and "codes" plotting type. Defaults to "gray" and "transparent" in both types, respectively.
- **zlim:** optional range for color coding of unit backgrounds.
- **heatkey:** whether or not to generate a heatkey at the left side of the plot in the "property" and "counts" plotting types.
- **property:** values to use with the "property" plotting type.
- **contin:** whether or not the data should be seen as discrete (i.e. classes) or continuous in nature. Only relevant for the colour keys of plots of supervised networks.
- **whatmap:** For supersom maps and a "codes" plot: what maps to show.
- **codeRendering:** How to show the codes. Possible choices: "segments", "stars" and "lines".
- **keepMargins:** if FALSE (the default), restore the original graphical parameters after plotting the kohonen map. If TRUE, one retains the map coordinate system so that one can add symbols to the plot, or map unit numbers using the identify function.
- **Heatkeywidth:** width of the colour key; the default of 0.2 should work in most cases but in some cases, e.g. when plotting multiple figures, it may need to be adjusted.
- **...:** other graphical parameters, e.g. colours of labels, or plotting symbols, in the "mapping" plotting type.

Several different types of plots are supported:

- **"changes":** shows the mean distance to the closest codebook vector during training.
- **"codes":** shows the codebook vectors.
- **"counts":** shows the number of objects mapped to the individual units. Empty units are depicted in gray.
- **"dist.neighbours":** shows the sum of the distances to all immediate neighbours. This kind of visualization is also known as a U-matrix plot. Units near a class boundary can be expected to have higher average distances to their neighbours. Only available for the "som" and "supersom" maps, for the moment.
- **"mapping":** shows where objects are mapped. It needs the "classif" argument, and a "labels" or "pchs" argument.

- **"property":** properties of each unit can be calculated and shown in colour code. It can be used to visualise the similarity of one particular object to all units in the map, to show the mean similarity of all units and the objects mapped to them, etcetera. The parameter property contains the numerical values.
- **"quality":** shows the mean distance of objects mapped to a unit to the codebook vector of that unit. The smaller the distances, the better the objects are represented by the codebook vectors.

return:

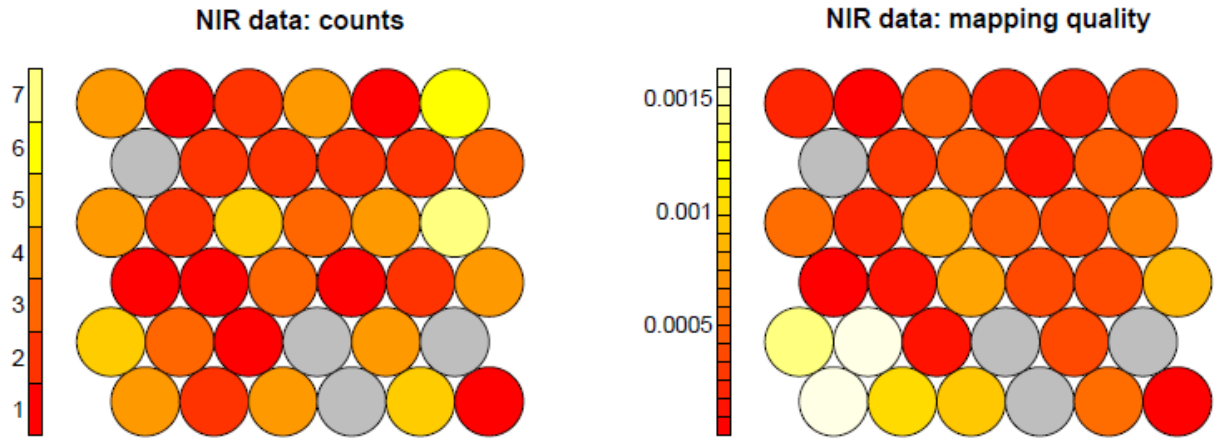


Figure 4: Left Plot the function was called with type "counts". Right Plot the function was called with type "quality" .

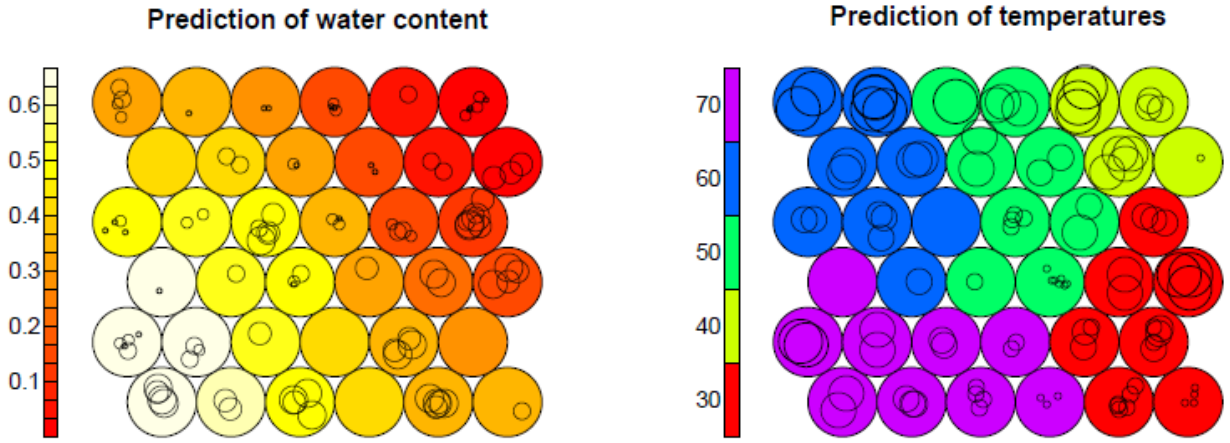


Figure 5: The function Plot was called with type "property" .

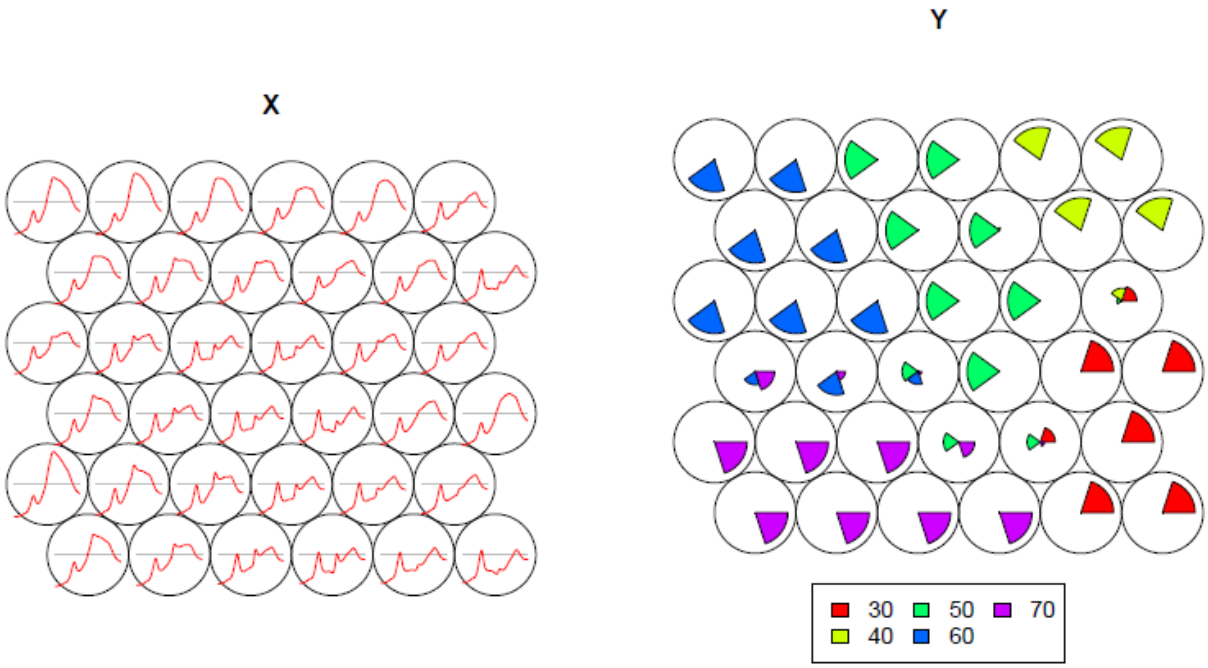


Figure 6: The function Plot was called with type "codes" .

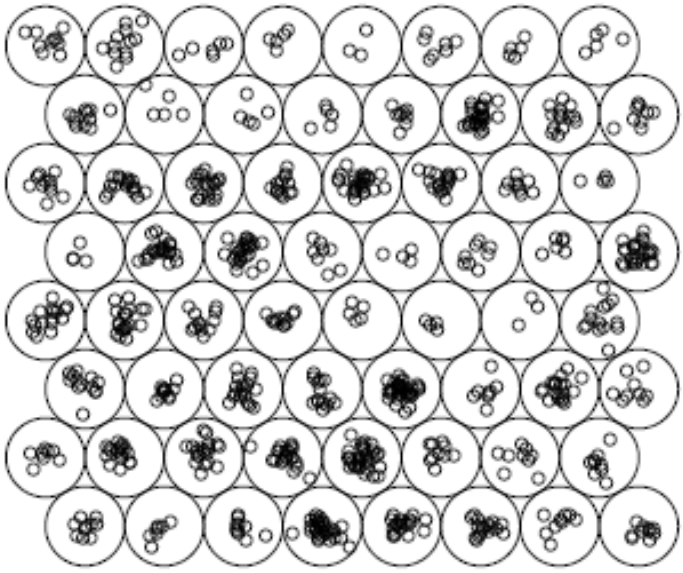


Figure 7: The function Plot was called with type "mapping" .

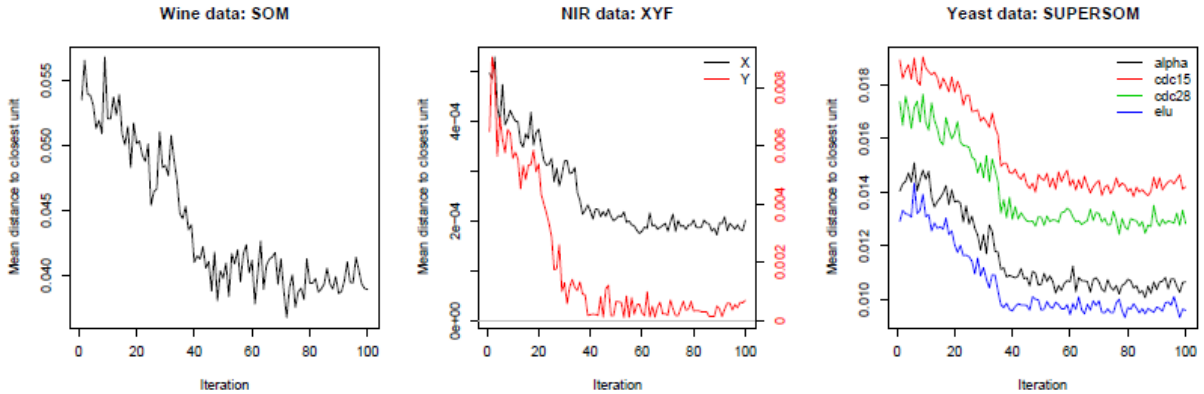


Figure 8: The function Plot was called with type "changes" .

Case Study

In this section, we illustrate a case study using package Kohonen.

Scenario

The standard form of self-organizing maps is implemented in function som. To map the 177-sample wine data set to a map of five-by-four hexagonally oriented units, the som function can be used. First, we load the package (from now on, we assume the package is loaded), and then the data, which are subsequently autoscaled because of the widely different ranges (especially the proline concentration, variable 13, deviates). The fourteenth variable is a class variable and is not used in the mapping; it will be used later for visualisation purposes .

Input data

As the input data we use de dataset wine that are included in the kohonen package. The dataset containing 177 rows and thirteen columns; object vintages contains the class labels. For compatibility with older versions of the package, variable wine.classes is retained, too. These data are the results of chemical analyses of wines grown in the same region in Italy (Piedmont) but derived from three different cultivars: Nebbiolo, Barberas and Grignolino grapes. The wine from the Nebbiolo grape is called Barolo. The data contain the quantities of several constituents found in each of the three types of wines, as well as some spectroscopic variables .

Execution

The following code can be use to create the map from the dataset. Note that first of all, you have to load the package and then load the dataset.

```
> library("kohonen")
Loading required package: class
> data("wines")
> wines.sc <- scale(wines)
> set.seed(7)
> wine.som <- som(data = wines.sc, grid = somgrid(5, 4, "hexagonal"))
> plot(wine.som, main = "Wine data")
```

Output

The result is shown in Figure 9. The codebook vectors are visualized in a segments plot, which is the default plotting type. High alcohol levels, for example, are associated with wine samples projected in the bottom right corner of the map, while color intensity is largest in the bottom left corner .

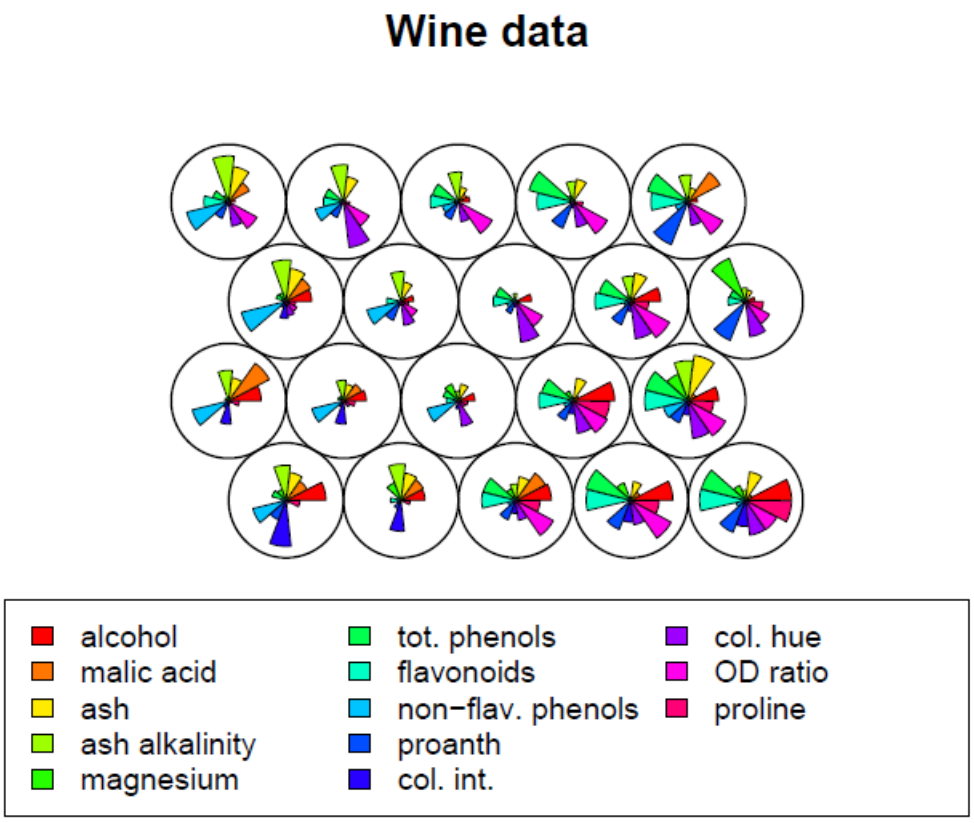


Figure 9: A plot of the codebook vectors of the 5-by-4 mapping of the wine data .

Analysis

The result of the training, the wine.som object, is a list. The most important element is the codes element, which contains the codebook vectors as rows. Another element worth inspecting is changes, a vector indicating the size of the adaptions to the codebook vectors during training. This can be used to assess whether the number of iterations is sufficient .

Extra

An example using the NIR data included in the package is shown below: for every ternary mixture, we have a nearinfrared spectrum, as well as concentrations of the three chemical compounds (summing to 1). Moreover, every sample is measured at five different temperatures. The aim in the example below is to model the water content (the second of the three concentrations). Of the three chemicals, water has the largest effect on the NIR spectra. We start by loading the data and attaching the data frame so that objects spectra, composition and temperature become directly available. Parameter xweight indicates how much importance is given to X; here it is set to 0.5 (X and Y are equally important), also the default value in xyf .

```
> data("nir")
> attach(nir)
> set.seed(13)
> nir.xyf <- xyf(data = spectra, Y = composition[,2], xweight = 0.5, grid = somgrid(6, 6, "hexagonal"))
```

```
> par(mfrow = c(1, 2))
> plot(nir.xyf, type = "counts", main = "NIR data: counts")
> plot(nir.xyf, type = "quality", main = "NIR data: mapping quality")
```

This leads to the output shown in Figure 4. In the left plot, the background color of a unit corresponds to the number of samples mapped to that particular unit; they are reasonably spread out over the map. Four of the units are empty: no samples have been mapped to them. The right plot shows the mean distance of objects, mapped to a particular unit, to the codebook vector of that unit. A good mapping should show small distances everywhere in the map .

References

[1] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley; US ed edition. May 12, 2005.

[2] http://www.r-project.org/

[3] http://CRAN.R-project.org/

[4] Katharine Mullen and Ron Wehrens.*CRAN Task View: Chemometrics and Computational Physics*.2010-11-02. URL (http://cran.r-project.org/web/views/ChemPhys.html).

[5] Ron Wehrens and Lutgarde M. C. Buydens. *Self- and super-organizing maps in r: The kohonen package*. Journal of Statistical Software, 21(5):1-19, 10 2007. URL (http://www.cran.r-project.org/).

[6] http://kdd.ics.uci.edu

[7] Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). Irvine, CA: University of California, School of Information and Computer Science.

[8] Wülfert F, Kok WT, Smilde AK (1998). *Influence of Temperature on Vibration Spectra and Consequences for Multivariate Models*. Analytical Chemistry, 70, 1761–1767.

[9] Spellman PT, Sherlock G, Zhang MQ, Iyer VR, Anders K, Eisen MB, Brown PO, Botstein D, Futcher B (1998). *Comprehensive Identification of Cell Cycle-regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization*. Molecular Biology of the Cell, 9, 3273–3297.

[10] Ron Wehrens and Lutgarde M. C. Buydens. *Self- and super-organizing maps in r: The kohonen package*. Journal of Statistical Software, 21(5):1-19, 10 2007.

Proximus

With the availability of large-scale computing platforms for high-fidelity design and simulations, and instrumentation for gathering scientific as well as business data, increased emphasis is being placed on efficient techniques for analyzing large and extremely high-dimensional data sets. These data sets may comprise discrete attributes, such as those from business processes, information retrieval, and bioinformatics, as well as continuous attributes such as those in scientific simulations, astrophysical measurements, and engineering design.

Analysis of high-dimensional data typically takes the form of extracting correlations between data items, discovering meaningful information in data, clustering data items, and finding efficient representations for clustered data, classification, and event association. Since the volume (and dimensionality) of data is typically large, the emphasis of new algorithms must be on efficiency and scalability to large data sets.

Technique/Algorithm

In this section, we focus on Proximus. The intended area of application is the compression of high-dimensional binary data into representative patterns. For instance, purchase incidence (market basket data) or term-document matrices may be preprocessed by Proximus for later association rule mining. In the next subsection, we give a brief explanation of how the algorithm works.

Algorithm

The Proximus algorithm cluster the rows of a logical matrix. The compression rate of the algorithm can be influenced by the choice of the maximum cluster radius and the minimum cluster size.

The algorithm is of a recursive partitioning type. Specifically, at each step a binary split is attempted using a local rank-one approximation of the current submatrix (row set). That is a specialization of principal components to binary data which represents a matrix as the outer product of two binary vectors. The node expansion stops if a submatrix is pure, i.e., the column (presence set) vector indicates all the rows and the Hamming distances from the row (dominant attribute set) pattern vector, or the size of the row set, are less than or equal the specified threshold. In the case the rank-one approximation does not result in a split but the radius constraint is violated, the matrix is split using a random row and the radius constraint.

The figure shows the recursive structure of proximus, where A represents the original data matrix. Each rectangular internal node is a rank-one approximation and two circular children of these nodes are the matrices that result from partitioning of parent matrix based on this approximation. Leaves of the recursion tree correspond to final decomposition. The overall decomposition is $A \approx XY^T$, where $X = [x_1, x_{01}, x_{00}]^T$ and $Y = [y_1, y_{01}, y_{00}]^T$.

Implementation

Proximus is part of cba package. In this section you find more information about installing and using it on R Environment.

Type the following command in R console to install cba package

```
install.packages("cba")
```

Type the following command in R console to load the package

```
library("cba")
```

The Proximus function, provided by the cba package, might be used as follow

```
proximus(x, max.radius = 2, min.size = 1, min.retry = 10, max.iter = 16, debug = FALSE)
```

where arguments are:

```
x: a logical matrix.
max.radius: the maximum number of bits a member in a row set may deviate from its dominant pattern.
min.size: the minimum split size of a row set.
min.retry: number of retries to split a pure rank-one approximation (translates into a resampling rate).
max.iter: the maximum number of iterations for finding a local rank-one approximation.
debug: optional debugging output.
```

An object of class proximus with the following components:

```
nr: the number of rows of the data matrix.
nc: the number of columns of the data matrix.
a: a list containing the approximations (patterns).
a$x: a vector of row (presence set) indexes.
a$y: a vector of column (dominant attribute set) indexes.
a$n: the number of ones in the approximated submatrix.
a$c: the absolute error reduction by the approximation.
max.radius: see arguments.
min.size: see arguments.
rownames: rownames of the data matrix.
colnames: colnames of the data matrix.
```

View

There is one way to show the result from this algorithm. That way would be typing:

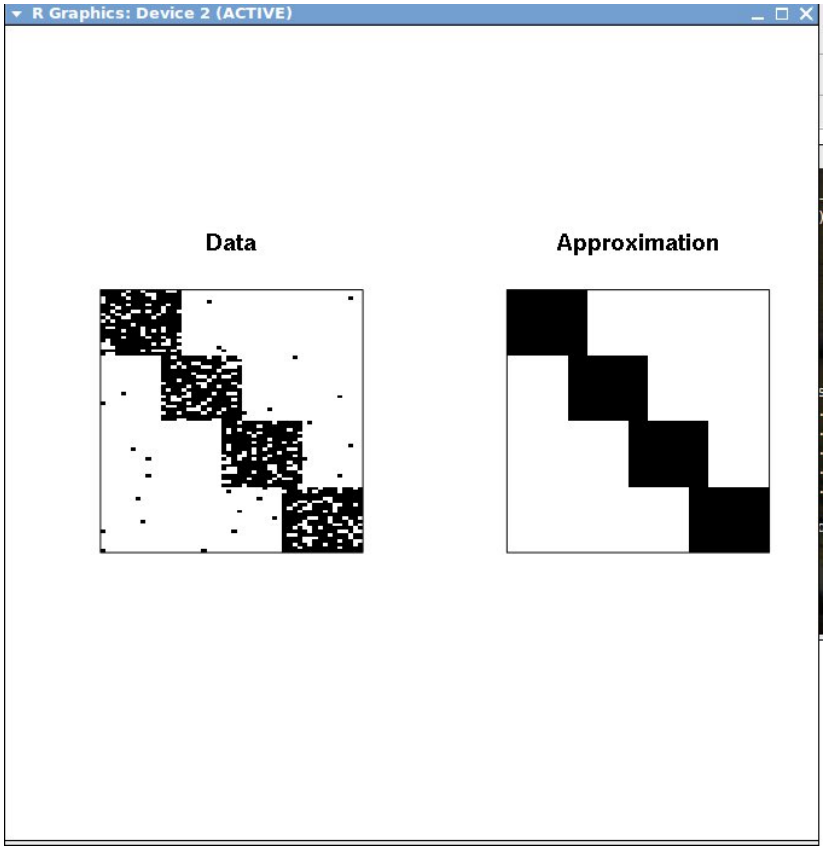
```
summary(ProximusObject)
```

Example

Here we have an example of proximus algorithm processing. The example is quite simple and gives an idea of how it works. Basically, a uniform logical matrix is generated. Then proximus does a rank-4 aproximattion of original logical matrix.

```
x <- rlbmat()
pr <- proximus(x, max.radius=8, debug=TRUE)
op <- par(mfrow=c(1,2), pty="s")
lmplot(x, main="Data")
box()
lmplot(fitted(pr)$x, main="Approximation")
box()
par(op)
```

Output



Case Study

In this section, we illustrate a case study with Proximus:

Scenario

In this section, we use PROXIMUS to cluster terms in hypothetical database to extract semantic information. It means, we want to know what are the main subjects of documents.

Suppose there's a library with some documents and we want to divide these documents in categories. We can describe each document as a set of words that occurs on that document.

Data

We represent the data as a binary term-document matrix by mapping terms to rows and columns to documents, so that a TRUE entry in the matrix indicates the existence of a word in the corresponding document.

The table below shows 14 terms and 10 documents. Clearly, there are two groups of words: those that are related to computers and those that are related to authors. The word love is isolated.

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
intel	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
computer	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
software	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
windows	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE
linux	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
firefox	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
explorer	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
knuth	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
programming	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
shakespeare	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
love	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
book	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
grisham	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
asimov	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE

Execution

R Code:

```
x <- matrix(scan("matriz.txt",what=logical(0),n = 14*10), 14, 10, byrow = TRUE)
pr <- proximus(x, max.radius=8, debug = TRUE)
summary(pr)
```

Output

The result of clustering the matrix of terms and documents is shown below:

```
> pr <- proximus(x, max.radius=8, debug = TRUE)
Non-Zero: 61
Sparsity: 0.44
  0 [14,8,3] 1 >
  1 [8,8,3] 1 * 1
  1 [6,1,0] 1 >
  2 [1,1,0] 1 * 2
  2 [5,5,1] 1 * 3
  1 <
  0 <
> summary(pr)
approximates 14 x 10 matrix
total Error: 0.064
total Fnorm: 3
total  Jsim: 0.96
total Valid: 3
Pattern Summary:
  Size Length Radius Error Fnorm Jsim Valid
1    8      5      3  0.10   2.8 0.80  TRUE
3    5      4      1  0.02   1.0 0.95  TRUE
2    1      3      0  0.00   0.0 1.00  TRUE
>
```

Analysis

As you can see in the output, there's a table with the result of clustering. Clearly, there are three groups of words (each line). The first group is related to computers. The second one is related to authors. The third group represents the isolated word love. The column named size indicates how many words belongs to the same group. Proximus algorithm clustered the words as follows: Group 1 (computers) = {intel, computer, software, linux, windows, Firefox, explorer, programming} Group 2 (authors) = {kuth, shakespeare, grisham, asimov, book} Group 3 (noise) = {love}

References

1. Meira Jr., W.; Zaki, M. Fundamentals of Data Mining Algorithms. [1]
2. CBA R package. [1]
3. Koyuturk; Mehmet and Grama; Ananth. PROXIMUS: a framework for analyzing very high dimensional discrete-attributed dataset.

Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD), pages 147-156, 2003.

CLARA

An obvious way of clustering larger datasets is to try and extend existing methods so that they can cope with a larger number of objects. The focus is on clustering large numbers of objects rather than a small number of objects in high dimensions. Kaufman and Rousseeuw (1990) suggested the CLARA (Clustering for Large Applications) algorithm for tackling large applications. CLARA extends their k-medoids approach for a large number of objects. It works by clustering a sample from the dataset and then assigns all objects in the dataset to these clusters.

Technique To Be Discussed

This work is focused on CLARA, a technique for clustering largers datasets.

Algorithm

Symbols	Definitions
D	Data set to be clustered
n	Number of objects in D
O_i	Object i in D
k	Number of clusters
S	A sample of D
s	Size of S

Table 1: Summary of symbols and definitions

CLARA (CLustering LARge Applications) relies on the sampling approach to handle large data sets. Instead of finding medoids for the entire data set, CLARA draws a small sample from the data set and applies the PAM algorithm to generate an optimal set of medoids for the sample. The quality of resulting medoids is measured by the average dissimilarity between every object in the entire data set D and the medoid of its cluster, defined as the following cost function:

$$Cost(M, D) = \frac{\sum_{i=1}^{n} dissimilarity(O_i, rep(M, O_i))}{n}$$

where M is a set of selected medoids, dissimilarity(Oi, Oj) is the dissimilarity between objects Oi and Oj, and rep(M, Oi) returns a medoid in M which is closest to Oi.

To alleviate sampling bias, CLARA repeats the sampling and clustering process a pre-defined number of times and subsequently selects as the final clustering result the set of medoids with the minimal cost. Assume q to be the number of samplings. The CLARA algorithm is detailed in Figure 1.

Figure 1: Clara Algorithm

Since CLARA adopts a sampling approach, the quality of its clustering results depends greatly on the size of the sample. When the sample size is small, CLARA's efficiency in clustering large data sets comes at the cost of clustering quality.

Implementation

In order to use the CLARA algorithm in R, one must install cluster package. This package includes a function that performs the CLARA process.

Install cluster package

```
install.packages("cluster")
```

Import Contents

```
library("cluster")
```

The CLARA function, provided by the cluster package, might be used as follow:

```
clara(x, k, metric = "euclidean", stand = FALSE, samples = 5, sampsize = min(n, 40 + 2 * k), trace = 0, medoids.x = TRUE, keep.data = medoids.x, rngR = FALSE)
```

where the arguments are:

- **x:** Data matrix or data frame, each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NAs) are allowed.
- **k:** Integer, the number of clusters. It is required that 0 < k < n where n is the number of observations (i.e., n = nrow(x)).
- **metric:** Character string specifying the metric to be used for calculating dissimilarities between observations. The currently available options are "euclidean" and "manhattan". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences.
- **stand:** Logical, indicating if the measurements in x are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation.
- **samples:** Integer, number of samples to be drawn from the dataset.
- **sampsize:** Integer, number of observations in each sample. sampsize should be higher than the number of clusters (k) and at most the number of observations (n = nrow(x)).
- **trace:** Integer indicating a trace level for diagnostic output during the algorithm.
- **medoids.x:** Logical indicating if the medoids should be returned, identically to some rows of the input data x. If FALSE, keep.data must be false as well, and the medoid indices, i.e., row numbers of the medoids will still be returned (i.med component), and the algorithm saves space by needing one copy less of x.
- **keep.data:** Logical indicating if the (scaled if stand is true) data should be kept in the result. Setting this to FALSE saves memory (and hence time), but disables clusplot()ing of the result. Use medoids.x = FALSE to save

even more memory.

- **rngR:** Logical indicating if R's random number generator should be used instead of the primitive clara()-builtin one. If true, this also means that each call to clara() returns a different result – though only slightly different in good situations.

View

There are actually two ways of viewing the result of a CLARA use. Both of them use the object of class *clara* returned by the function application.

The first way is to plot the object, creating a chart that represents the data. Thus, if there are N objects divided into K clusters, the chart must contain N points representing the objects, and those points must be colored in K different colors, each one representing a cluster set. For example, given the object *clarax*, which is a result of the function *clara* application, all one has to do in order to plot the object is:

```
plot(clarax)
```

The second way of viewing the result of a CLARA application is to simply print the components of the object of class *clara*. For example, given the same object *clarax* of the previous example, one could print its components using:

```
print(clarax)
```

Example

Suppose we have 500 objects and each object have two attributes (or features). Our goal is to group these objects into K=2 groups based on their two features. The function CLARA can be used to define the groups as follow:

```
## generate 500 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(200,0,8), rnorm(200,0,8)), cbind(rnorm(300,50,8), rnorm(300,50,8)))
## run clara
clarax <- clara(x, 2)
clarax
clarax$clusinfo

## print components of clarax
print(clarax)

## plot clusters
plot(x, col = clarax$cluster)
## plot centers
points(clarax$centers, col = 1:2, pch = 8)
```

Result of printing components of *clarax*:

```
Call:    clara(x = x, k = 2)
Medoids:
          [,1]       [,2]
[1,]  1.091033 -0.5367556
[2,] 51.044099 51.0638017
Objective function:      9.946085
Clustering vector:       int [1:500] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
Cluster sizes:           200 300
```

```
Best sample:
 [1]   6  45  51  56  67  75  85  90  94  97 110 111 160 170 176 181 201 219 249
[20] 260 264 275 296 304 317 319 337 340 361 362 369 370 374 379 397 398 411 420
[39] 422 424 436 448 465 489

Available components:
 [1] "sample"     "medoids"    "i.med"      "clustering" "objective"
 [6] "clusinfo"   "diss"       "call"       "silinfo"    "data"
```

Result of plotting "clarax"

Figure 2: Result of plotting clarax

Case study

In this section, we illustrate a case study using CLARA.

Scenario

This data set contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

Input Data

A data frame with 50 observations on 4 variables.

[,1] Murder numeric Murder arrests (per 100,000)

[,2] Assault numeric Assault arrests (per 100,000)

[,3] UrbanPop numeric Percent urban population

[,4] Rape numeric Rape arrests (per 100,000)

State	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7
Connecticut	3.3	110	77	11.1
Delaware	5.9	238	72	15.8
Florida	15.4	335	80	31.9
Georgia	17.4	211	60	25.8
Hawaii	5.3	46	83	20.2
Idaho	2.6	120	54	14.2
Illinois	10.4	249	83	24.0
Indiana	7.2	113	65	21.0
Iowa	2.2	56	57	11.3
Kansas	6.0	115	66	18.0

Kentucky	9.7	109	52	16.3
Louisiana	15.4	249	66	22.2
Maine	2.1	83	51	7.8
Maryland	11.3	300	67	27.8
Massachusetts	4.4	149	85	16.3
Michigan	12.1	255	74	35.1
Minnesota	2.7	72	66	14.9
Mississippi	16.1	259	44	17.1
Missouri	9.0	178	70	28.2
Montana	6.0	109	53	16.4
Nebraska	4.3	102	62	16.5
Nevada	12.2	252	81	46.0
New Hampshire	2.1	57	56	9.5
New Jersey	7.4	159	89	18.8
New Mexico	11.4	285	70	32.1
New York	11.1	254	86	26.1
North Carolina	13.0	337	45	16.1
North Dakota	0.8	45	44	7.3
Ohio	7.3	120	75	21.4
Oklahoma	6.6	151	68	20.0
Oregon	4.9	159	67	29.3
Pennsylvania	6.3	106	72	14.9
Rhode Island	3.4	174	87	8.3
South Carolina	14.4	279	48	22.5
South Dakota	3.8	86	45	12.8
Tennessee	13.2	188	59	26.9
Texas	12.7	201	80	25.5
Utah	3.2	120	80	22.9
Vermont	2.2	48	32	11.2
Virginia	8.5	156	63	20.7
Washington	4.0	145	73	26.2
West Virginia	5.7	81	39	9.3
Wisconsin	2.6	53	66	10.8
Wyoming	6.8	161	60	15.6

Table 2: USArrests Database

Execution

The function "clara" was used as follows:

```
## import data
x <- USArrests

## run CLARA
clarax <- clara(x[1:4], 3)

## print components of clarax
print(clarax)

## plot clusters
plot(x, col = clarax$cluster)
## plot centers
points(clarax$centers, col = 1:2, pch = 8)
```

1. plot(Assualt, Murder)

(USArrests) points(254,11.1, pch=16) text(254,11.11, labels ='New York') lines(Assault, (.63168 + (.04191 * Assault)))

Output

The result of printing the components of the class returned by the function application is shown below:

```
Call:    clara(x = x[1:4], k = 3)
Medoids:
         Murder Assault UrbanPop Rape
Michigan   12.1     255       74 35.1
Missouri    9.0     178       70 28.2
Nebraska    4.3     102       62 16.5
Objective function:      29.31019
Clustering vector:       Named int [1:50] 1 1 1 2 1 2 3 1 1 2 3 3 1 3 3 3 3 1 ...
 - attr(*, "names")= chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" "California" "Colorado" "Connecticut" ...
Cluster sizes:           16 14 20
Best sample:
 [1] Alabama        Alaska         Arizona        Arkansas       California
 [6] Colorado       Delaware       Florida        Georgia        Idaho
[11] Illinois       Indiana        Iowa           Kansas         Kentucky
[16] Louisiana      Maine          Maryland       Massachusetts  Michigan
[21] Minnesota      Mississippi    Missouri       Montana        Nebraska
[26] Nevada         New Hampshire  New York       North Carolina North Dakota
[31] Ohio           Oklahoma       Oregon         Pennsylvania   Rhode Island
[36] South Carolina South Dakota   Tennessee      Texas          Utah
[41] Vermont        Virginia       Washington     West Virginia  Wisconsin
[46] Wyoming

Available components:
 [1] "sample"     "medoids"    "i.med"      "clustering" "objective"
```

```
[6] "clusinfo"   "diss"       "call"       "silinfo"    "data"
```

The result of plotting the class returned by the function application it is shown below:

Figure 3: Results of the example

Analysis

The implementation of CLARA generated three clusters, relatively homogeneous, consisting of 16, 14 and 20 countries. Analyzing the cluster means, we can relate each group with each of the three classes of states:

- The cluster formed by Alabama, Alaska, Arizona, California, Delaware, Florida, Illinois, Louisiana, Maryland, Michigan, Mississippi, Nevada, New Mexico, New York, North Carolina, South Carolina has the highest Murder, Assault and Rape arests (per 100,00) and, not least, the largest population.
- The cluster formed by Arkansas, Colorado, Georgia, Massachusetts, Missouri, New Jersey, Oklahoma, Oregon, Rhode Island, Tennessee, Texas, Virginia, Washington, Wyoming has the intermediate Murder, Assault and Rape arests (per 100,00) and, not least, the largest population.
- The cluster formed by Connecticut, Hawaii, Idaho, Indiana, Iowa, Kansas, Kentucky, Maine, Minnesota, Montana, Nebraska, New Hampshire, North Dakota, Ohio, Pennsylvania, South Dakota, Utah, Vermont, West Virginia, Wisconsin has the lowest Murder, Assault and Rape arests (per 100,00) and, not least, the largest population.

Analyzing, based on [3], the states of the two extreme clusters (1,3) it was possible to verify that there is a reason for each country to be in these groups. California, although has a good Human Development Index and Median Personal Earnings rate, has the 3rd biggest Unemployment Rate in the USA, the 2nd is Michigan and the 1st is Nevada, two other states that are also in the cluster one. Connecticut has the highest Human Development Index and is on the cluster three. Wyoming has the best percentage of people with High School Diploma, and is on the cluster two. Others reasons can be verified checking this work together with [3].

References

1. Chih-Ping, Wei, Yen-Hsien, Lee, and Che-Ming, Hsu, Empirical Comparison of Fast Clustering Algorithms for Large Data Sets. [1]
2. The R Development Core Team, R: A Language and Environment for Statistical Computing. [2]
3. American Human Development Project, Mapping the Measure of America [3]
4. Kaufman, L., Rousseeuw, P. J., Clustering by Means of Medoids.

References

[1] http://www.computer.org/comp/proceedings/hicss/2000/0493/02/04932013.pdf
[2] http://cran.r-project.org/doc/manuals/fullrefman.pdf
[3] http://www.measureofamerica.org/maps/

SVM

Introduction

Support Vector Machines (SVMs) are supervised learning methods used for classification and regression tasks that originated from statistical learning theory [1]. As a classification method, SVM is a global classification model that generates non-overlapping partitions and usually employs all attributes. The entity space is partitioned in a single pass, so that flat and linear partitions are generated. SVMs are based on maximum margin linear discriminants, and are similar to probabilistic approaches, but do not consider the dependencies among attributes [2].

Traditional Neural Network approaches have suffered difficulties with generalization, producing models which overfit the data as a consequence of the optimization algorithms used for parameter selection and the statistical measures used to select the best model. SVMs have been gaining popularity due to many attractive features and promising empirical performance. They are based on the Structural Risk Minimization (SRM) principle [3] have shown to be superior to the traditional principle of Empirical Risk Minimization (ERM) employed by conventional Neural Networks. ERM minimizes the error on the training data, while SRM minimizes an upper bound on the expected risk. This gives SRM greater generalization ability, which is the goal in statistical learning [4]. According to [5], SVMs rely on preprocessing the data to represent patterns in a high dimension, typically much higher than the original feature space. Data from two categories can always be separated by a hyperplane when an appropriate nonlinear mapping to a sufficiently high dimension is used.

A classification task usually involves training and test sets which consist of data instances. Each instance in the training set contains one target value (class label) and several attributes (features). The goal of a classifier is to produce a model able to predict target values of data instances in the testing set, for which only the attributes are known. Without loss of generality, the classification problem can be viewed as a two-class problem in which one's objective is to separate the two classes by a function induced from available examples. The goal is to produce a classifier that generalizes well, i.e. that works well on unseen examples. The below picture is an example of a situation in which various linear classifiers can separate the data. However, only one maximizes the distance between itself and the nearest example of each class (i.e. the margin) and for that is called the optimal separating hyperplane. It is intuitively expected that this classifier generalizes better than the other options . The basic idea of SVM classifier uses this approach, i.e. to choose the hyperplane that has the maximum margin.

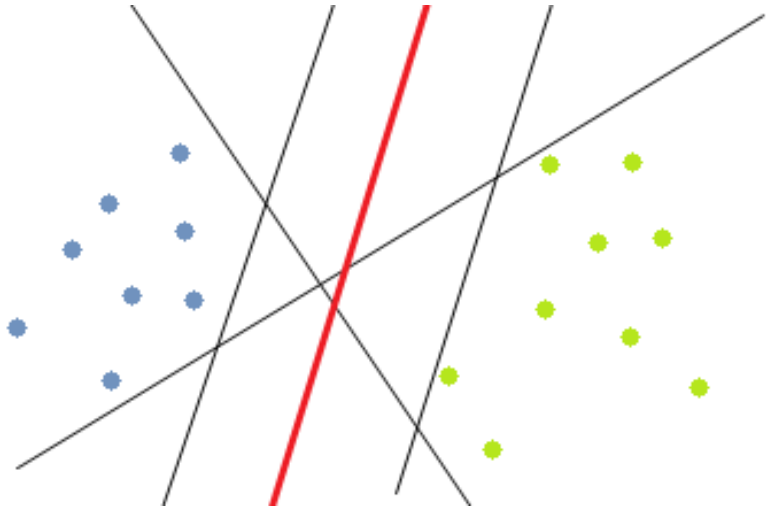


Figure 1: Example of separating hyperplanes

Algorithm

Let *D* be a classification dataset with *n* points in a *d*-dimensional space $D = \{(x_i, y_i)\}$, with $i = 1, 2, ..., n$ and let there be only two class labels such that y_i is either +1 or -1. A hyperplane *h(x)* gives a linear discriminant function in *d* dimensions and splits the original space into two half-spaces:

$$h(x) = w^T x + b = w_1 x_1 + w_2 x_2 + ... + w_d x_d + b\,,$$

where *w* is a *d*-dimensional weight vector and *b* is a scalar bias. Points on the hyperplane have *h(x) = 0*, i.e. the hyperplane is defined by all points for which $w^T x = -b$.

According to , if the dataset is linearly separable, a separating hyperplane can be found such that for all points with label -1, *h(x) < 0* and for all points labeled +1, *h(x) > 0*. In this case, *h(x)* serves as a linear classifier or linear discriminant that predicts the class for any point. Moreover, the weight vector *w* is orthogonal to the hyperplane, therefore giving the direction that is normal to it, whereas the bias *b* fixes the offset of the hyperplane in the *d*-dimensional space.

Given a separating hyperplane *h(x) = 0*, it is possible to calculate the distance between each point x_i and the hyperplane by:

$$\delta_i = \frac{y_i h(x_i)}{||w||}$$

The margin of the linear classifier is defined as the minimum distance of all *n* points to the separating hyperplane.

$$\delta^* = \min_{x_i}\{\frac{y_i h(x_i)}{||w||}\}$$

All points (vectors x^*_i) that achieve this minimum distance are called the support vectors for the linear classifier. In other words, a support vector is a point that lies precisely on the margin of the classifying hyperplane.

In a canonical representation of the hyperplane, for each support vector x^*_i with label y^*_i we have that $y_i^* h(x_i^*) = 1$. Similarly, for any point that is not a support vector, we have that $y_i h(x_i) > 1$, since, by definition, it must be farther from the hyperplane than a support vector. Therefore we have that $y_i h(x_i) \geq 1,\ \forall x_i \in D$.

The fundamental idea behind SVMs is to choose the hyperplane with the maximum margin, i.e. the optimal canonical hyperplane. To do this, one needs to find the weight vector *w* and the bias *b* that yield the maximum margin among all possible separating hyperplanes, that is, the hyperplane that maximizes $\frac{1}{||w||}$. The problem then becomes that of solving a convex minimization problem (notice that instead of maximizing the margin $\frac{1}{||w||}$., one can obtain an equivalent formulation of minimizing $||w||$) with linear constraints, as follows:

Objective Function

$$min\frac{||w||^2}{2}$$

Linear Constraints

$$y_i h(x_i) \geq 1,\ \forall x_i \in D$$

This minimization problem can be solved using the *Lagrange multiplier* method, which introduces a Lagrange multiplier α for each constraint:

$$\alpha_i(y_i h(x) - 1) = 0 \text{ with } \alpha_i \geq 0$$

This method states that $\alpha_i = 0$ for all points that are at a distance larger than $\frac{1}{||w||}$ from the hyperplane, and only for those points that are exactly at the margin, i.e. the support vectors, $\alpha_i > 0$. The weight vector of the classifier is obtained as a linear combination of the support vectors, while the bias is the average of the biases obtained from each support vector .

SVMs can handle linearly non-separable points, where the classes overlap to some extent so that a perfect separation is not possible, by introducing slack variables ε_i for each point x_i in *D*. If $0 \leq \varepsilon_i < 1$, the point is still correctly classified. Otherwise, if $\varepsilon_i > 1$, the point is misclassified. So the goal of the classification becomes that of finding the

hyperplane (*w* and *b*) with the maximum margin that also minimizes the sum of slack variables. A methodology similar to that described above is necessary to find the weight vector *w* and the bias *b*.

SVMs can also solve problems with non-linear decision boundaries. The main idea is to map the original *d*-dimensional space into a *d'*-dimensional space (*d'* > *d*), where the points can possibly be linearly separated. Given the original dataset $D = \{x_i, y_i\}$ with $i = 1,...,n$ and the transformation function Φ, a new dataset is obtained in the transformation space $D_\Phi = \{\Phi(x_i), y_i\}$ with $i = 1,...,n$. After the linear decision surface is found in the *d'*-dimensional space, it is mapped back to the non-linear surface in the original *d*-dimensional space . To obtain *w* and *b*, *Φ(x)* needn't be computed in isolation. The only operation required in the transformed space is the inner product $\Phi(x_i)^T\Phi(x_j)$, which is defined with the kernel function (*K*) between x_i and x_j. Kernels commonly used with SVMs include:

- the polynomial kernel:

  $K(x_i, x_j) = (x_i^T x_j + 1)^q$, where q is the degree of the polynomial
- the gaussian kernel:

  $K(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$, where σ is the spread or standard deviation.
- the gaussian radial basis function (RBF):

  $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma \geq 0$
- the Laplace Radial Basis Function (RBF) kernel:

  $K(x_i, x_j) = e^{-\gamma||x_i - x_j||}, \gamma \geq 0$
- the hyperbolic tangent kernel:

  $K(x_i, x_j) = tanh(x_i^T x_j + offset)$
- the sigmoid kernel:

  $K(x_i, x_j) = tanh(ax_i^T x_j + offset)$
- the Bessel function of the first kind kernel:

  $K(x_i, x_j) = \left(\frac{Bessel_{v+1}^n(\sigma||x_i - x_j||)}{(||x_i - x_j||)^{-n(v+1)}}\right)$
- the ANOVA radial basis kernel:

  $K(x_i, x_j) = \left(\sum_{k=1}^{n} e^{-\sigma(x_i^k - x_j^k)^2}\right)^d$
- the linear splines kernel in one dimension:

  $K(x_i, x_j) = 1 + x_i x_j min(x_i, x_j) - \frac{x_i + x_j}{2} min(x_i, x_j)^2 + \frac{min(x_i, x_j)^3}{3}$

According to , the Gaussian and Laplace RBF and Bessel kernels are general-purpose kernels used when there is no prior knowledge about the data. The linear kernel is useful when dealing with large sparse data vectors as is usually the case in text categorization. The polynomial kernel is popular in image processing, and the sigmoid kernel is mainly used as a proxy for neural networks. The splines and ANOVA RBF kernels typically perform well in regression problems.

Available Implementations in R

R [6] is a language and environment for statistical computing and graphics. There are five packages that implement SVM in R []:

- *e1071* [7]
- *kernlab* [8]
- *klaR* [9]
- *svmpath* [10]
- *shogun* [11]

This documentation will focus on the e1071 package because it is the most intuitive. For information on the others, see the references cited above and the report of .

e1071 package

The e1071 package was the first implementation of SVM in R. The *svm()* function provides an interface to *libsvm* [12], complemented by visualization and tuning functions. *libsvm* is a fast and easy-to-use implementation of the most popular SVM formulation of classification (C and ν), and includes the most common kernels (linear, polynomial, RBF, and sigmoid). Multi-class classification is provided using the one-against-one voting scheme. It also includes the computation of decision and probability values for predictions, shrinking heuristics during the fitting process, class weighting in the classification mode, handling of sparse data, and cross-validation.

The R implementation is based on the S3 class mechanisms. It basically provides a training function with standard and formula interfaces, and a *predict()* method. In addition, a *plot()* method for visualizing data, support vectors, and decision boundaries is provided. Hyperparameter tuning is done using the *tune()* framework, which performs a grid search over specified parameter ranges.

Installing and Starting the e1071 Package

To install e1071 package in R, type

```
install.packages('e1071',dependencies=TRUE)
```

and to start to use the package, type

```
library(e1071)
```

Main Functions in the e1071 Package for Training, Testing, and Visualizing

Some *e1071* package functions are very important in any classification process using SVM in R, and thus will be described here.

The first function is *svm()*, which is used to train a support vector machine. Some import parameters include:

- **data**: an optional data frame containing the variables in the model. If this option is used, the parameters x and y described below, aren't necessary;
- **x**: a data matrix, a vector, or a sparse matrix that represents the instances of the dataset and their respective properties. Rows represent the instances and columns represent the properties;
- **y**: a response vector with one label for each row (instance) of x;
- **type**: sets how svm() will work. The possible values for classification are: C, nu and one (for novelty detection);
- **kernel**: defines the kernel used in training and prediction. The options are: linear, polynomial, radial basis and sigmoid;
- **degree**: parameter needed if the kernel is polynomial (default: 3);

- **gamma**: parameter needed for all types of kernels except linear (default: 1/(data dimension));
- **coef0**: parameter needed for polynomial and sigmoid kernels (default: 0);
- **cost**: cost of constraint violation (default: 1). This is the 'C'-constant of the regularization term in the Lagrange formulation;
- **cross**: specifies the cross-validation. A k > 0 is necessary. In this case, the training data is performed to assess the quality of the model: the accuracy rate for classification;
- **probability**: logical indicating whether the model should allow for probability predictions.

An example of *svm()* usage is given below:

```
library(MASS)
data(cats)
model  <- svm(Sex~., data = cats)
```

The first two commands specify the usage of the cats dataset, which contains 144 instances, 2 numerical attributes for each instance ("Bwt" and "Hwt"), and the class for each instance (attribute "Sex"). The instance class can be "F", for female, or "M", for male. In the third command, the parameter "Sex~." indicates the attribute (column) of the dataset to be used as instance classes.

For information on the parameters of the model and on the number of support vectors, type:

```
print(model)
summary(model)
```

The result of the summary command is shown below:

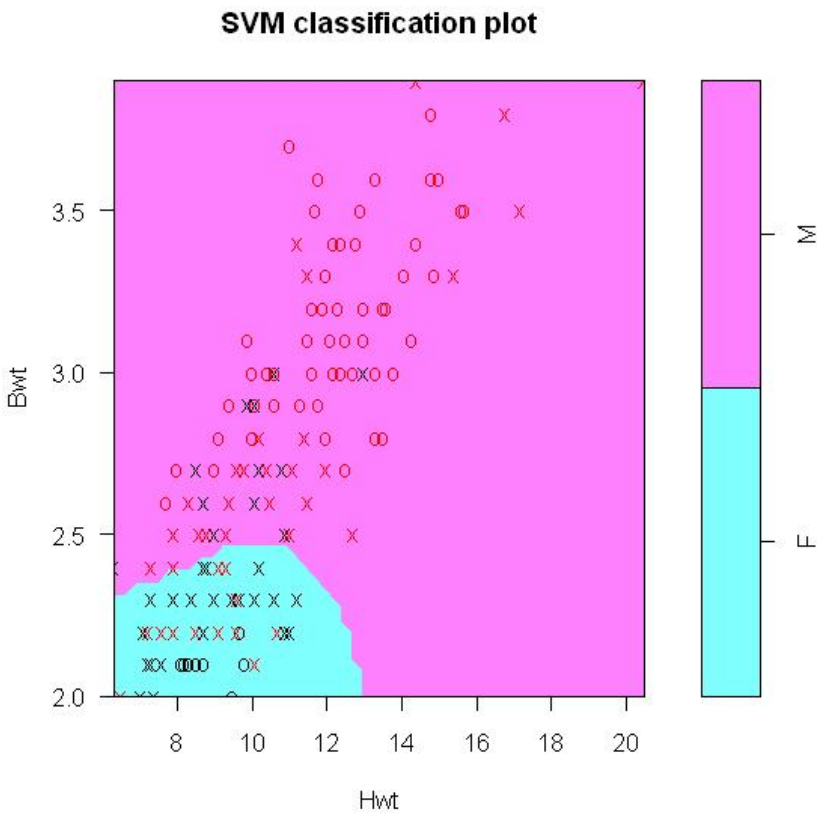
```
Call:
svm(formula = Sex ~ ., data = cats)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  84

 ( 39 45 )

Number of Classes:  2

Levels:
 F M
```

To see the built model with a scatter plot of the input, the *plot()* function can be used. This function optionally draws a filled contour plot of the class regions. The main parameters of this function are listed below:

- **model**: an object of class svm data, which results from the svm() function;
- **data**: the data to visualize. It should be the same data used for building the model in the svm() function;
- **symbolPalette**, **svSymbol**, **dataSymbol**, and **colorPalette**: these parameters control the colors and symbols used to represent support vectors and the other data points.

The following command will produce the below graph, in which support vectors are shown as 'X', true classes are highlighted through symbol color, and predicted class regions are visualized using colored background.

```
plot(model,cats)
```



The *predict()* function predicts values based on a model trained by svm. For a classification problem, it returns a vector of predicted labels. Detailed information about its usage can be obtained with the following command.

```
help(predict.svm)
```

Let us first divide the cats dataset into a train and a test set:

```
index <- 1:nrow(cats)
testindex <- sample(index, trunc(length(index)/3))
testset <- cats[testindex,]
trainset <- cats[-testindex,]
```

Now we run the model again using the train set and predict classes using the test set in order to verify if the model has good generalization.

```
model <- svm(Sex~., data = trainset)
prediction <- predict(model, testset[,-1])
```

The -1 is because the dependent variable, Sex, is in column number 1.

A cross-tabulation of the true versus the predicted values yields (the confusion matrix):

```
tab <- table(pred = prediction, true = testset[,1])
```

If you type tab, you will see the confusion matrix like is shown below:

```
    true
pred  F  M
   F 10  8
   M  6 24
```

With this information, it is possible to compute the sensitivity, the specificity and the precision of the model to the test set.

Model accuracy rates can be computed using the *classAgreement()* function:

```
  classAgreement(tab)
```

The *tune()* function can be used to tune hyperparameters of statistical methods using a grid search over the supplied parameter ranges.

```
  tuned <- tune.svm(Sex~., data = trainset, gamma = 10^(-6:-1), cost = 10^(1:2))
  summary(tuned)
```

These commands will list the best parameters, the best performance, and details of the tested parameter values, as shown below.

```
Parameter tuning of `svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
   0.1  100

- best performance: 0.1566667

- Detailed performance results:
   gamma cost     error dispersion
1  1e-06   10 0.2600000  0.1095195
2  1e-05   10 0.2600000  0.1095195
3  1e-04   10 0.2600000  0.1095195
4  1e-03   10 0.2600000  0.1095195
5  1e-02   10 0.2833333  0.1230890
6  1e-01   10 0.1788889  0.1359264
7  1e-06  100 0.2600000  0.1095195
8  1e-05  100 0.2600000  0.1095195
9  1e-04  100 0.2600000  0.1095195
10 1e-03  100 0.2833333  0.1230890
11 1e-02  100 0.1788889  0.1359264
12 1e-01  100 0.1566667  0.1014909
```

Case Study

In this section we use a dataset to breast cancer diagnostic and apply svm in it. The svm model will be able to discriminate benign and malignant tumors.

The DataSet

The dataset can be downloaded at [13]. In this dataset there are 569 instances and 32 attributes for each instance. The first attribute is the identification of instance, the second is the label for the instance class, which can be M (malignant tumor) or B (benign tumor). The following 30 attributes are real-valued input features that are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. Finally, there are 357 benign instances and 212 malignant instances in dataset.

In order to read the dataset, after downloading it and saving it, type in R:

```
dataset <- read.csv('/home/myprofile/wdbc.data',head=FALSE)
```

'/home/myprofile/' is the path where the dataset was saved.

Preparing the DataSet

Let us now divide at random the dataset in two subsets, one with about 70% of the instances to training, and another with around the remaining 30% of instances to testing:

```
index <- 1:nrow(dataset)

testindex <- sample(index, trunc(length(index)*30/100))

testset <- dataset[testindex,]

trainset <- dataset[-testindex,]
```

Choosing Parameters

Now, we will use the **tune()** function to do a grid search over the supplied parameter ranges (**C** - cost, γ - gamma), using the train set. The range to gamma parameter is between 0.000001 and 0.1. For cost parameter the range is from 0.1 until 10.

It's important to understanding the influence of this two parameters, because the accuracy of an SVM model is largely dependent on the selection them. For example, if **C** is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit. If it is too small, we may have underfitting [14].

Notice that there aren't names for the columns (attributes) in the database . Then, R considers default names for them, as such V1, to the first column, V2 to the second and so on. It's possible to check this typing:

```
names(dataset)
```

Then, as the class label is the second column of the dataset, the first parameter to **tune()** function will be *V2*:

```
tuned <- tune.svm(V2~., data = trainset, gamma = 10^(-6:-1), cost = 10^(-1:1))
```

The results are showed with the following command:

```
summary(tuned)
```

```
Parameter tuning of `svm':
```

```
- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
 0.001   10

- best performance: 0.02006410

- Detailed performance results:
   gamma cost      error dispersion
1  1e-06  0.1 0.36333333 0.05749396
2  1e-05  0.1 0.36333333 0.05749396
3  1e-04  0.1 0.36333333 0.05749396
4  1e-03  0.1 0.30064103 0.06402773
5  1e-02  0.1 0.06256410 0.04283663
6  1e-01  0.1 0.08512821 0.05543939
7  1e-06  1.0 0.36333333 0.05749396
8  1e-05  1.0 0.36333333 0.05749396
9  1e-04  1.0 0.28314103 0.05862576
10 1e-03  1.0 0.05506410 0.04373139
11 1e-02  1.0 0.02756410 0.02188268
12 1e-01  1.0 0.03256410 0.02896982
13 1e-06 10.0 0.36333333 0.05749396
14 1e-05 10.0 0.28314103 0.05862576
15 1e-04 10.0 0.05500000 0.04684490
16 1e-03 10.0 0.02006410 0.01583519
17 1e-02 10.0 0.02256410 0.01845738
18 1e-01 10.0 0.05532051 0.04110686
```

Training The Model

In order to build a svm model to predict breast cancer using **C=10** and **gamma=0.001**, which were the best values according the **tune()** function run before, type:

```
model  <- svm(V2~., data = trainset, kernel="radial", gamma=0.001, cost=10)
```

To see the results of the model, as the number of support vectors is necessary type:

```
summary(model)
```

The result follows:

```
Call:
svm(formula = V2 ~ ., data = trainset, kernel = "radial", gamma = 0.001, cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.001
```

```
Number of Support Vectors:  79

 ( 39 40 )


Number of Classes:  2

Levels:
 B M
```

Testing the Model

Now we run the model again the test set to predict classes.

```
prediction <- predict(model, testset[,-2])
```

The -2 is because the label column to intance classes, V2, is in the second column.

To produce the confusion matrix type:

```
tab <- table(pred = prediction, true = testset[,2])
```

The confusion matrix is:

```
    true
pred   B   M
   B 103   6
   M   0  61
```

This means that there are 103 benign instances in test set and all of them were predicted as benign instances. On the other hand, there are 67 malign instances in test set, 61 were predicted rightly and 6 as benign instances.

Let:

- TP: true positive, i.e. malign instances predicted rightly
- FP: false positive, i.e. benign instances predicted as malign
- TN: true negative, i.e. benign instances predicted rightly
- |N|: total of benign instances
- |P|: total of malign instances

$$sensitivity = \frac{TP}{|P|}$$

$$specificity = \frac{TN}{|N|}$$

$$precision = \frac{TP}{TP + FP}$$

For this problem we have:

$$sensitivity = \frac{61}{61 + 6} = 0.91$$

$$specificity = \frac{103}{103} = 1$$

$$precision = \frac{61}{61 + 0} = 1$$

The classification results are suitable.

References

[1] V. Vapnik, Statistical learning theory. Wiley, New Tork (1998).

[2] M. J. Zaki and W. Meira Jr. Fundamentals of Data Mining Algorithms. Cambridge University Press, 2010.

[3] S. R. Gunn, M. Brown and K. M. Bossley, Network performance assessment for neurofuzzy data modeling. Intelligent Data Analysis, volume 1208 of Lecture Notes in Computer Science (1997), 313.

[4] S. R. Gunn. Support vector machines for classification and regression.Tech. rep., University of Sothampton, UK, 1998.

[5] R. O. Duda, P. E. Hart and D. G. Stork, Pattern Classification. Ed.Wiley-Interscience, 2000.

[6] R Development Core Team (2005). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

[7] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, A. Weingessel. e1071: Misc Functions of the Department of Statistics (e1071). TU Wien, Version 1.5-11, 2005. URL http://CRAN.R-project.org/

[8] A. Karatzoglou, A. Smola, K. Hornik (2009)."kernlab An S4 Package for Kernel Methods in R". URL http://www.jstatsoft.org/v11/i09/.

[9] C. Roever, N. Raabe, K. Luebke, U. Ligges (2005). "klaR – Classification and Visualization." R package, Version 0.4-1. URL http://CRAN.R-project.org/.

[10] T. Hastie. svmpath: The SVM Path algorithm. R package, Version 0.9, 2004. URL http://CRAN.R-project.org/.

[11] S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN Machine Learning Toolbox. Journal of Machine Learning Research, 11:1799-1802, June 2010. URL http://www.shogun-toolbox.org/.

[12] C. Chang and C. Lin (2001). "libsvm: A Library for Support Vector Machines." URL http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[13] http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

[14] E. Alpaydin. Introduction to machine learning. MIT Press, 2004.

penalizedSVM

Introduction

Classifiers are some of the most common data analysis tools. There are a lot of implemented techniques, but we may point SVM (Support Vector Machine) as one of the most powerful, especially in high-dimension data. The most well known SVM algorithm was created by Vladimir Vapnik. [1]

The standard SVM implementation SVM takes a input dataset and, for each given input, predicts which of two possible classes the input set belongs to. That's most common use the algorithm to predict if the input belongs to certain dichotomy, or not. Because of this characteristic, SVM is a called a non-probabilistic binary linear classifier.

On Machine Learning-based algorithms such as SVM, the input data has to be separated on two sets: a training set and a test set. The difference between the training and the test set is that, on the training the examples' classes are known beforehand. The test set contains the examples that should have their classes predicted. Given a set of training examples, an SVM algorithm builds a model that predicts what are the categories of the test set's examples.

Representing the examples in a d-dimension space, the model built by the algorithm is a hyper-plan that separates the examples belonging to the dichotomy from the ones that aren't. New examples are then mapped into that space so we can predict the categories they belong, based on the side of the gap it falls on.

The technique described here is a variation of the standard SVM using penalty functions. The technique is implemented on the R-package called penalized SVM, that has smoothly clipped absolute deviation (SCAD), 'L1-norm', 'Elastic Net' ('L1-norm' and'L2-norm') and 'Elastic SCAD' (SCAD and 'L2-norm') as available penalties.

Technique/Algorithm

Algorithm

Given a training dataset $(x_1, y_1), ..., (x_n, y_n)$, where x_i is a d-tuple of the d input parameters i, and $y_i \in -1, 1$ where $y_1 = 1$ means i belongs to the dichotomy, and $y_i = -1$, the opposite. The SVM divides the space by a linear boundary:

$$f(x) = \sum_{j=1}^{d} w_j h_j + b$$

where $w = (w_1, ..., w_d)$ are the coefficients of the hyper-plan and b denotes the intercept of the hyperplane. The output for an example from the test set x_test would be $y_test = sign[f(x_test)]$. So, the example belongs to the dichotomy if $f(x_test) > 0$.

Finding the best model to classify new examples is, as a matter of fact, similar to the problem of finding the optimal hyperplane. The quality of a model built by the algorithm is measured by the margin of its hyperplane. The margin is the distance between the hyperplanes wx+b = 1 and wx+b = -1,being so that the closest points to wx+b = 0, in each side of that hyperplane are crossed by the other two hyperplanes. The best hyperplanes are those with the biggest margins.

The problem of finding the optimal hyperplane with maximal margin is solved by convex optimization. Maximizing the margin can be achieved by solving:

$$\min_{b,w} \sum [1 - y_i f(x_i)]_+ + \text{pen}_\lambda(w)$$

The term

$$pen_\lambda(w) = \lambda \left\| w \right\|_2^2$$

for SVM implementation has the form of L2 norm ('ridge penalty'). This penalty causes the reduction of the coefficients, but not always to values greater than zero.

The quality of a model can also be measured by the error computed for the predictions on the training set. A low prediction is required, but reducing it to zero may cause a problem know as over fitting. This means that the model built classifies very well the examples on training set, but is not appropriate to classify the ones on the test set. The predictions made in this case, would usually be wrong for the new examples.

Not only the low prediction defines a model's quality. It is possible to improve it further by identifying covariates that plays important roles on discrimination and assessing their contribution to the classifier. This can be achieved by applying a feature selection method.

Feature selection methods are divided into two classes: filter and wrapper methods[2]. Filter methods drop irrelevant features before the model is built by the algorithm. Wrapper methods increases the prediction power by providing the selection within the optimization procedure.

The R package 'penalizedSVM' provides some feature selection methods. One of them is the wrapper feature selection SCAD (Smoothly Clipped Absolute Deviation). SCAD is a non-convex penalty function first proposed by Fan [3] and discussed in [4].On [5] SVM is combined with SCAD for feature selection. The penalization term for SCAD SVM has the form:

$$pen_\lambda(w) = \sum_{j=1}^{d} p_\lambda(w_j)$$

where the SCAD penalty function for each coefficient wj is defined as:

$$p_\lambda(w_j) = \begin{cases} \lambda |w_j| & \text{if } |w_j| \leq \lambda \\ -\frac{|w_j|^2 - 2a\alpha|w_j| + \lambda^2}{2(a-1)} & \text{if } |w_j| \leq a\lambda \\ \frac{a+1/\lambda^2}{2} & \text{if } |w_j| > a\lambda \end{cases}$$

with tuning parameters a > 2 (in the package, a = 3.7) and λ > 0. pλ (w) corresponds to a quadratic spline function with knots at λ and aλ.

For small coefficients, the SCAD has the same behavior as the L1. For large coefficients, however, the SCAD applies a constant penalty, in contrast to the L1 penalty, which increases linearly as the coefficient increases. This absolute maximum of the SCAD penalty, which is independent from the input data, decreases the possible bias for estimating large coefficients.

Implementation

The package described in this section is *penalizedSVM*. This package provides feature selection SVM using penalty functions. The smoothly clipped absolute deviation (SCAD), 'L1-norm', 'Elastic Net' ('L1-norm' and 'L2-norm') and 'Elastic SCAD'(SCAD and 'L2-norm') penalties are available. The tuning parameters can be found using either a fixed grid or a interval search.

This package has several dependencies. The packages that need to be installed and theirs descriptions are listed below:

- **e1071** Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support

vector machines, shortest path computation, bagged clustering, naive Bayes classifier, ...

- **MASS**Functions and datasets to support Venables and Ripley, 'Modern Applied Statistics with S' (4th

edition).

- **corpcor**This package implements a James-Stein-type shrinkage estimator for the covariance matrix, with

separate shrinkage for variances and correlations. The details of the method are explained in Sch\"afer and Strimmer (2005) and Opgen-Rhein and Strimmer (2007).

- **stadmod** Various statistical modeling functions including growth curve comparisons, limiting dilution

analysis, mixed linear models, heteroscedastic regression, Tweedie family generalized linear models, the inverse-Gaussian distribution and Gauss quadrature.

- **tgp** Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian processes

with jumps to the limiting linear model (LLM).

- **mlepg** Maximum likelihood Gaussian process modeling for univariate and multi-dimensional outputs with

diagnostic plots. Contact the maintainer for a package version that implements sensitivity analysis functionality.

- **lhs** This package provides a number of methods for creating and augmenting Latin Hypercube Samples

The major function on penalizedSVM is svm.fs. Its use can be described as follow:

```
## Default S3 method:
svm.fs(
x,
y,
fs.method = c("scad", "1norm", "scad+L2", "DrHSVM"),
### tuning parameter settings
# chose the search method for tuning lambda1,2: 'interval' or 'discrete'
grid.search=c("interval","discrete"),
#fixed grid for lambda1, lambda2
lambda1.set=NULL,
```

```
lambda2.set=NULL,
# define range for lambda1,2 for interval search
bounds=NULL,
# parms.coding="none" or "log2"
parms.coding= c("log2","none"),
# internal parameter for DIRECT
maxevals=500,
### valuidation settings
# fot nested validation, 'cross.outer'-fold cv
#cross.outer= 0,
# method for the inner validation: cross validation, gacv
inner.val.method = c("cv", "gacv"),
# 'cross.inner'-fold cv
cross.inner= 5,
# show plots in Direct?
show= c("none", "final"),
### other settings
# internal parameter for svm
calc.class.weights=FALSE,
class.weights=NULL,
#seed
seed=123,
# max iterations for the feature selection svm method
maxIter=700,
# verbose?
verbose=TRUE,
...)
```

where the arguments are:

- *x:* input matrix with genes in columns and samples in rows!
- *y:* numerical vector of class labels, -1 , 1
- *fs.method:* feature selection method. Availible 'scad', '1norm' for 1-norm, "DrHSVM" for Elastic Net and "scad+L2" for Elastic SCAD
- *grid.search:* chose the search method for tuning lambda1,2: 'interval' or 'discrete', default: 'interval'
- *lambda1.set:* for fixed grid search: fixed grid for lambda1, default: NULL
- *lambda2.set:* for fixed grid search: fixed grid for lambda2, default: NULL
- *bounds:* for interval grid search: fixed grid for lambda2, default: NULL
- *parms.coding:* for interval grid search: parms.coding: none or log2 , default: log2
- *maxevals:* the maximum number of DIRECT function evaluations, default: 500.
- *cross.outer:* fold of outer cross validation, default is 0, no cv.
- *calc.class.weights:* calculate class.weights for SVM, default: FALSE
- *class.weights:* a named vector of weights for the different classes, used for asymmetric class

sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.

- *inner.val.method:* method for the inner validation: cross validation, gacv , default cv
- *cross.inner:* 'cross.inner'-fold cv, default: 5
- *show:* for interval search: show plots of DIRECT algorithm: none, final iteration, all iterations. Default: none
- *seed:* seed

- *maxIter:* maximal iteration, default: 700
- *verbose:* verbose?, default: TRUE
- *... additional:* argument(s)

View

To visualize the result of the algorithm, you can use the function show, as exampled above.

Case Study

In this section, we illustrate a case study with penalizedSVM:

Scenario

The objective is to tell if a picture belongs to the same category of another one selected previously. For that purpose, we use the difference between the values of those characteristics. The attributes for the data used as input represents the difference from the value of the characteristic in the picture selected and the ones we want to classify.

Dataset

The training set and the test set were generated using the commands shown below:

```
> train<-sim.data(n = 200, ng = 100, nsg = 10, corr=FALSE, seed=seed )
> print(str(train))
List of 3
 $ x   : num [1:100, 1:200] -0.5605 2.1988 -0.0736 1.074 0.3563 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:100] "pos1" "pos2" "pos3" "pos4" ...
  .. ..$ : chr [1:200] "1" "2" "3" "4" ...
 $ y   : Named num [1:200] 1 -1 1 1 1 -1 -1 -1 1 -1 ...
  ..- attr(*, "names")= chr [1:200] "1" "2" "3" "4" ...
 $ seed: num 123
NULL
> test<-sim.data(n =20, ng = 100, nsg = 10, corr=FALSE, seed=seed+1 )
> print(str(test))
List of 3
 $ x   : num [1:100, 1:20] -1.3851 -1.1036 -0.2677 0.2836 -0.0951 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:100] "pos1" "pos2" "pos3" "pos4" ...
  .. ..$ : chr [1:20] "1" "2" "3" "4" ...
 $ y   : Named num [1:20] -1 1 -1 1 1 1 1 1 1 -1 ...
  ..- attr(*, "names")= chr [1:20] "1" "2" "3" "4" ...
 $ seed: num 124
NULL
```

Execution

To build the model, the following command were used:

```
> bounds=t(data.frame(log2lambda1=c(-10, 10)))
>colnames(bounds)<-c("lower", "upper")
# computation intensive; for demostration reasons only for the first 100 features
# and only for 10 Iterations maxIter=10, default maxIter=700
>system.time( scad<- svm.fs(t(train$x)[,1:100], y=train$y, fs.method="scad", bounds=bounds,
+ cross.outer= 0, grid.search = "interval", maxIter = 10,
+ inner.val.method = "cv", cross.inner= 5, maxevals=500,
+ seed=seed, parms.coding = "log2", show="none", verbose=FALSE ) )
```

Output

To see the model created:

```
> print(str(scad$model))
List of 11
 $ w       : Named num [1:23] 0.625 0.616 0.353 0.258 0.959 ...
  ..- attr(*, "names")= chr [1:23] "pos1" "pos2" "pos3" "pos4" ...
 $ b       : num -0.115
 $ xind    : int [1:23] 1 2 3 4 5 6 7 8 9 10 ...
 $ index   : int [1:83] 3 4 9 14 17 18 22 35 37 40 ...
 $ fitted  : num [1:200] 2.6 1.24 0.65 1 1.15 ...
 $ type    : num 0
 $ lambda1 : num 0.126
 $ lambda2 : NULL
 $ iter    : num 10
 $ q.val   : num 0.195
 $ fit.info:List of 13
  ..$ fmin       : num 0.195
  ..$ xmin       : Named num -2.99
  .. ..- attr(*, "names")= chr "log2lambda1"
  ..$ iter       : num 26
  ..$ neval      : num 46
  ..$ maxevals   : num 500
  ..$ seed       : num 123
  ..$ bounds     : num [1, 1:2] -10 10
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr "log2lambda1"
  .. .. ..$ : chr [1:2] "lower" "upper"
  ..$ Q.func     : chr ".calc.scad"
  ..$ points.fmin:'data.frame':      1 obs. of  2 variables:
  .. ..$ log2lambda1: num -2.99
  .. ..$ f          : num 0.195
  ..$ Xtrain     : num [1:46, 1] -7.52 -2.26 -1.34 9.99 9.03 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr "log2lambda1"
```

```
  ..$ Ytrain     : num [1:46] 3.65e-01 3.20e-01 4.60e-01 1.00e+16 1.00e+16 ...
  ..$ gp.seed    : num [1:25] 123 124 125 126 127 128 129 130 131 132 ...
  ..$ model.list :List of 1
  .. ..$ model:List of 10
  .. .. ..$ w               : Named num [1:23] 0.625 0.616 0.353 0.258 0.959 ...
  .. .. .. ..- attr(*, "names")= chr [1:23] "pos1" "pos2" "pos3" "pos4" ...
  .. .. ..$ b               : num -0.115
  .. .. ..$ xind            : int [1:23] 1 2 3 4 5 6 7 8 9 10 ...
  .. .. ..$ index           : int [1:83] 3 4 9 14 17 18 22 35 37 40 ...
  .. .. ..$ fitted          : num [1:200] 2.6 1.24 0.65 1 1.15 ...
  .. .. ..$ type            : num 0
  .. .. ..$ lambda1         : num 0.126
  .. .. ..$ iter            : num 10
  .. .. ..$ q.val           : num 0.195
  .. .. ..$ inner.val.method: chr "cv"
NULL
```

To predict a class for the examples on the test set:

```
>(scad.5cv.test<-predict.penSVM(scad, t(test$x)[,1:100], newdata.labels=test$y) )
$pred.class
 [1] -1 1  -1 -1 -1 1  1  1  1  -1 -1 -1 1  1  1  -1 1  1  -1 -1
Levels: -1 1

$fitted
         [,1]
1  -2.5344366
2   2.3440943
3  -1.3972349
4  -0.3613470
5  -2.1187284
6   1.1287477
7   2.5584662
8   1.9155333
9   1.5543941
10 -0.7128084
11 -1.6944994
12 -0.2943272
13  1.8497781
14  2.7800572
15  0.8927699
16 -0.1289518
17  2.4560094
18  0.8756835
19 -2.2114729
20 -1.7342811

$tab
```

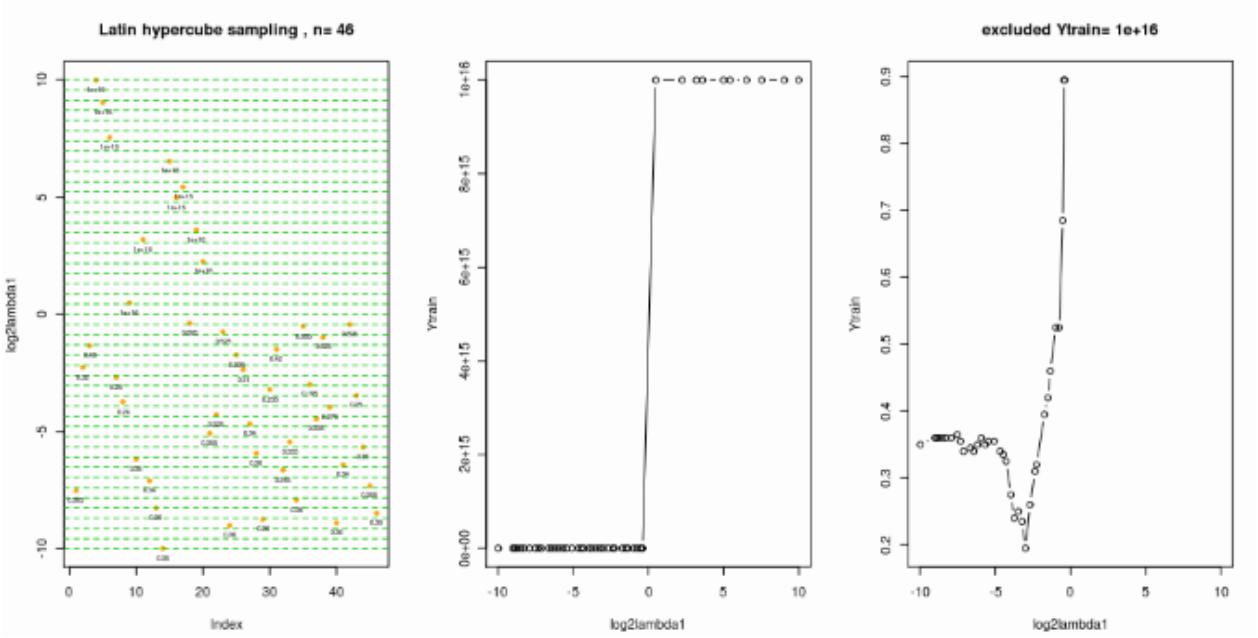
```
          newdata.labels
pred.class -1 1
        -1  6 4
        1   1 9

$error
[1] 0.25

$sensitivity
[1] 0.6923077

$specificity
[1] 0.8571429

> test<-sim.data(n = 20, ng = 100, nsg = 10, corr=FALSE, seed=seed+1 )
> print(str(test))
List of 3
 $ x   : num [1:100, 1:20] -1.3851 -1.1036 -0.2677 0.2836 -0.0951 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:100] "pos1" "pos2" "pos3" "pos4" ...
  .. ..$ : chr [1:20] "1" "2" "3" "4" ...
 $ y   : Named num [1:20] -1 1 -1 1 1 1 1 1 1 -1 ...
  ..- attr(*, "names")= chr [1:20] "1" "2" "3" "4" ...
 $ seed: num 124
NULL
> (scad.5cv.test<-predict.penSVM(scad, t(test$x)[,1:100], newdata.labels=test$y))
$pred.class
 [1] -1 1  -1 -1 -1 1  1  1  1  -1 -1 -1 1  1  1  -1 1  1  -1 -1
Levels: -1 1

$fitted
         [,1]
1  -2.5344366
2   2.3440943
3  -1.3972349
4  -0.3613470
5  -2.1187284
6   1.1287477
7   2.5584662
8   1.9155333
9   1.5543941
10 -0.7128084
11 -1.6944994
12 -0.2943272
13  1.8497781
14  2.7800572
15  0.8927699
```

```
16 -0.1289518
17  2.4560094
18  0.8756835
19 -2.2114729
20 -1.7342811

$tab
          newdata.labels
pred.class -1 1
        -1  6 4
        1   1 9

$error
[1] 0.25

$sensitivity
[1] 0.6923077

$specificity
[1] 0.8571429
```

Analysis

To analyse the results, the follow commands can be used:

```
> print(paste("minimal 5-fold cv error:", scad$model$fit.info$fmin,
+ "by log2(lambda1)=", scad$model$fit.info$xmin))
[1] "minimal 5-fold cv error: 0.195 by log2(lambda1)= -2.990937219120 59"
> print(" all lambdas with the same minimum? ")
[1] " all lambdas with the same minimum? "
> print(scad$model$fit.info$ points.fmin)
log2lambda1     f
36   -2.990937 0.195
> print(paste(scad$model$fit.info$neval, "visited points"))
[1] "46 visited points"
#Plot the results
>.plot.EPSGO.parms (scad$model$fit.info$Xtrain, scad$model$fit.info$Ytrain,
+ bound=bounds, Ytrain.exclude=10^16, plot.name=NULL )
```



References

[1] Corinna Cortes and V. Vapnik, Support-Vector Networks, Machine Learning, 20, 1995.http://www.springerlink.com/content/k238jx04hm87j80g/

[2] Blum A and Langley P, Selection of relevant features and examples in machine learning. Artif. Intell. 1997;97:245-271.

[3] Fan J.,Comments on 'Wavelets in Statistics: A Review' by A. Antoniadis. and J. Italian Stat.Assoc. 1997;6:131-138.

[4] Fan J.,Comments on 'Wavelets in Statistics: A Review' by A. Antoniadis. and J. Italian Stat.Assoc. 1997;6:131-138.

[5] Zhang HH, et al,Gene selection using support vector machines with nonconvex penalty.Bioinformatics 2006;22:88-95.

kNN

Introduction

This chapter introduces the k-Nearest Neighbors (kNN) algorithm for classification. kNN, originally proposed by Fix and Hodges [1] is a very simple 'instance-based' learning algorithm. Despite its simplicity, it can offer very good performance on some problems. We present a high level overview of the algorithm, explaining the relevant parameters and implementation choices, followed by a step by step case study.

k-Nearest Neighbors

The kNN algorithm, like other instance-based algorithms, is unusual from a classification perspective in its lack of explicit model training. While a training dataset is required, it is used solely to populate a sample of the search space with instances whose class is known. No actual model or learning is performed during this phase; for this reason, these algorithms are also known as lazy learning algorithms. Different distance metrics can be used, depending on the nature of the data. Euclidean distance is typical for continuous variables, but other metrics can be used for categorical data. Specialized metrics are often useful for specific problems, such as text classification. When an instance whose class is unknown is presented for evaluation, the algorithm computes its k closest neighbors, and the class is assigned by voting among those neighbors. To prevent ties, one typically uses an odd choice of k for binary classification. For multiple classes, one can use plurality voting or majority voting. The latter can sometimes result in no class being assigned to an instance, while the former can result in classifications being made with very low support from the neighborhood. One can also weight each neighbor by an inverse function of its distance to the instance being classified. The main advantages of kNN for classification are:

- Very simple implementation.
- Robust with regard to the search space; for instance, classes don't have to be linearly separable.
- Classifier can be updated online at very little cost as new instances with known classes are presented.
- Few parameters to tune: distance metric and k.

The main disadvantages of the algorithm are:

- Expensive testing of each instance, as we need to compute its distance to all known instances. Specialized algorithms and heuristics exist for specific problems and distance functions, which can mitigate this issue. This is problematic for datasets with a large number of attributes. When the number of instances is much larger than the number of attributes, a R-tree or a kd-tree can be used to store instances, allowing for fast exact neighbor identification.
- Sensitiveness to noisy or irrelevant attributes, which can result in less meaningful distance numbers. Scaling and/or feature selection are typically used in combination with kNN to mitigate this issue.
- Sensitiveness to very unbalanced datasets, where most entities belong to one or a few classes, and infrequent classes are therefore often dominated in most neighborhoods. This can be alleviated through balanced sampling of the more popular classes in the training stage, possibly coupled with ensembles.

Algorithm Description

The training phase for kNN consists of simply storing all known instances and their class labels. A tabular representation can be used, or a specialized structure such as a kd-tree. If we want to tune the value of 'k' and/or perform feature selection, n-fold cross-validation can be used on the training dataset. The testing phase for a new instance 't', given a known set 'I' is as follows:

1. Compute the distance between 't' and each instance in 'I'
2. Sort the distances in increasing numerical order and pick the first 'k' elements
3. Compute and return the most frequent class in the 'k' nearest neighbors, optionally weighting each instance's class by the inverse of its distance to 't'

Available Implementations

There are at least three implementations of kNN classification for R, all available on CRAN [2]:

- knn [3]
- kknn [4]
- RWeka [5], which is a bridge to the popular WEKA [6] machine and datamining toolkit, and provides a kNN implementation as well as dozens of algorithms for classification, clustering, regression, and data engineering.

We choose RWeka for this tutorial, as it provides a lot more than simply kNN classification, and the sample session shown below can be used to generate other classifiers with little modification.

Installing and Running RWeka

RWeka is a CRAN package, so it can be installed from within R:

```
> install.packages("RWeka", dependencies = TRUE)
```

Most R graphical user interfaces also provide package management through their UIs. Once installed, RWeka can be loaded in as a library:

```
> library(RWeka)
```

It comes with several well-known datasets, which can be loaded in as ARFF files (Weka's default file format). We now load a sample dataset, the famous Iris dataset [7], and learn a kNN classifier for it, using default parameters:

```
> iris <- read.arff(system.file("arff", "iris.arff", package = "RWeka"))
> classifier <- IBk(class ~., data = iris)
> summary(classifier)

=== Summary ===

Correctly Classified Instances         150               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0085
Root mean squared error                  0.0091
Relative absolute error                  1.9219 %
Root relative squared error              1.9335 %
Total Number of Instances              150

=== Confusion Matrix ===
```

```
  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 50  0 |  b = Iris-versicolor
  0  0 50 |  c = Iris-virginica
```

While in the above session we have only used default parameters, RWeka allows us to customize the KNN classifier in several ways, aside from setting the value of k:

- We can weight neighbors by the inverse of their distance to the target instance, or by 1 - distance.
- We can use leave-one-out cross-validation to choose the optimal value for k in the training data.
- We can use a sliding window of training instances, so once the classifier knows about W instances, older instances are dropped as new ones are added.
- We can customize the way the algorithm stores the known instances, which allows us to use kd-trees and similar data structures for faster neighbor search in large datasets.

Case Study

We will now perform a more detailed exploration of the Iris dataset, using cross-validation for real test statistics, and also performing some parameter tuning.

Dataset

The Iris dataset contains 150 instances, corresponding to three equally-frequent species of iris plant (Iris setosa, Iris versicolour, and Iris virginica). An Iris versicolor is shown below, courtesy of Wikimedia Commons.

Iris versicolor

Each instance contains four attributes:sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. The next picture shows each attribute plotted against the others, with the different classes in color.

Execution and Results

We will generate a kNN classifier, but we'll let RWeka automatically find the best value for k, between 1 and 20. We'll also use 10-fold cross validation to evaluate our classifier:

```
> classifier <- IBk(class ~ ., data = iris,
                    control = Weka_control(K = 20, X = TRUE))
> evaluate_Weka_classifier(classifier, numFolds = 10)
=== 10 Fold Cross Validation ===

=== Summary ===
```

```
Correctly Classified Instances         142               94.6667 %
Incorrectly Classified Instances         8                5.3333 %
Kappa statistic                          0.92
Mean absolute error                      0.041
Root mean squared error                  0.1414
Relative absolute error                  9.2339 %
Root relative squared error             29.9987 %
Total Number of Instances              150

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 46  4 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica

> classifier
IB1 instance-based classifier
using 6 nearest neighbour(s) for classification
```

As cross-validation generates random partitions of the dataset, your results may vary. Typically, however, the model will make fewer than 10 mistakes.

Analysis

This simple case study shows that a kNN classifier makes few mistakes in a dataset that, although simple, is not linearly separable, as shown in the scatterplots and by a look at the confusion matrix, where all misclassifications are between Iris Versicolor and Iris Virginica instances. The case study also shows how RWeka makes it trivially easy to learn classifiers (and predictors, and clustering models as well), and to experiment with their parameters. In this classic dataset, a kNN classifier makes very few mistakes.

References

1. ^ Fix, E., Hodges, J.L. (1951); Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas.
2. ^ Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update. SIGKDD Explorations, 11(1).
3. ^ Fisher,R.A. (1936); The use of multiple measurements in taxonomic problems. Annual Eugenics, 7, Part II, 179-188.

References

[1] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/kNN#endnote_knn
[2] http://cran.r-project.org
[3] http://stat.ethz.ch/R-manual/R-patched/library/class/html/knn.html
[4] http://cran.r-project.org/web/packages/kknn/
[5] http://cran.r-project.org/web/packages/RWeka/
[6] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/kNN#endnote_Weka
[7] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/kNN#endnote_Fisher

Outliers

Introduction

Outlier detection is one of the most important tasks in data analysis. In this approach, an expert can explore a set of associative rules in order to find how much the interestingness measure of these rules are away from their average values in different subsets of the database. The threshold which divides abnormal and non-abnormal data numerically is often the basis for important decisions. Most of the methods for univariate outlier detection are based on (robust) estimation of location and scatter or on quantiles of the data. A major disadvantage is that these rules are independent from the sample size. The dependence from the sample size is desirable to allow the threshold to be fitted according to the sample size. Moreover, outliers are identified even for “clean” data, or at least no distinction is made between outliers and extremes of a distribution.

Algorithm

Discovery methods for interesting exception rules can be divided into two approaches from the viewpoint of background knowledge:

1. In a directed approach, a method is first provided with background knowledge typically in the form of rules, then the method obtains exception rules each of which deviates from these rules;
2. In an undirected approach, on the other hand, no background knowledge is provided.

The problem can be summarized as finding a set of rule pairs each of which consists of an exception rule associated with a strong rule. Suppose a strong rule is represented by *if Y then x"*, where $Y = y_1 \wedge y_2 \wedge ... \wedge y_n$ is a conjunction of atoms and x is a single atom. Let $Z = z_1 \wedge z_2 \wedge ... \wedge z_n$ be a conjunction of atoms and x' be a single atom which has the same attribute but a value different to the atom x, then the exception rule is represented by *if Y and Z then x"*. For instance, consider the rule "using a seat belt is risky for a child", which represents exceptions to the well-known fact "using a seat belt is safe".

Package Installation

- Step1: If you already have the R package installed in your system jump to Step2. To install the R package you can use your system apt-get capabilities, just typing the following command:

```
$ sudo apt-get install r-base-core
```

If your system does not have apt-get capabilities, don't give up! You can download the package by visiting the Outlier Package [1] website.

- Step2: It is necessary to install the mvoutlier package. In order to install the mvoutlier package you first need to run R. It can be done by the command:

```
$ R
```

Then type the following command in the R environment:

```
> install.packages("mvoutlier")
```

The installation is done.

Now it is necessary to load the package:

```
> library(mvoutlier)
```

Visualization

In order to show how we can visualize the results of the mvoutlier package, we will use a practical example. The data set and its use in mvoutlier are described below.

Swiss Fertility and Socioeconomic Indicators (1888) Data

Swiss [2] is a database that contains standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888.

Format

A data frame with 47 observations on 6 variables, each of which is in percent.

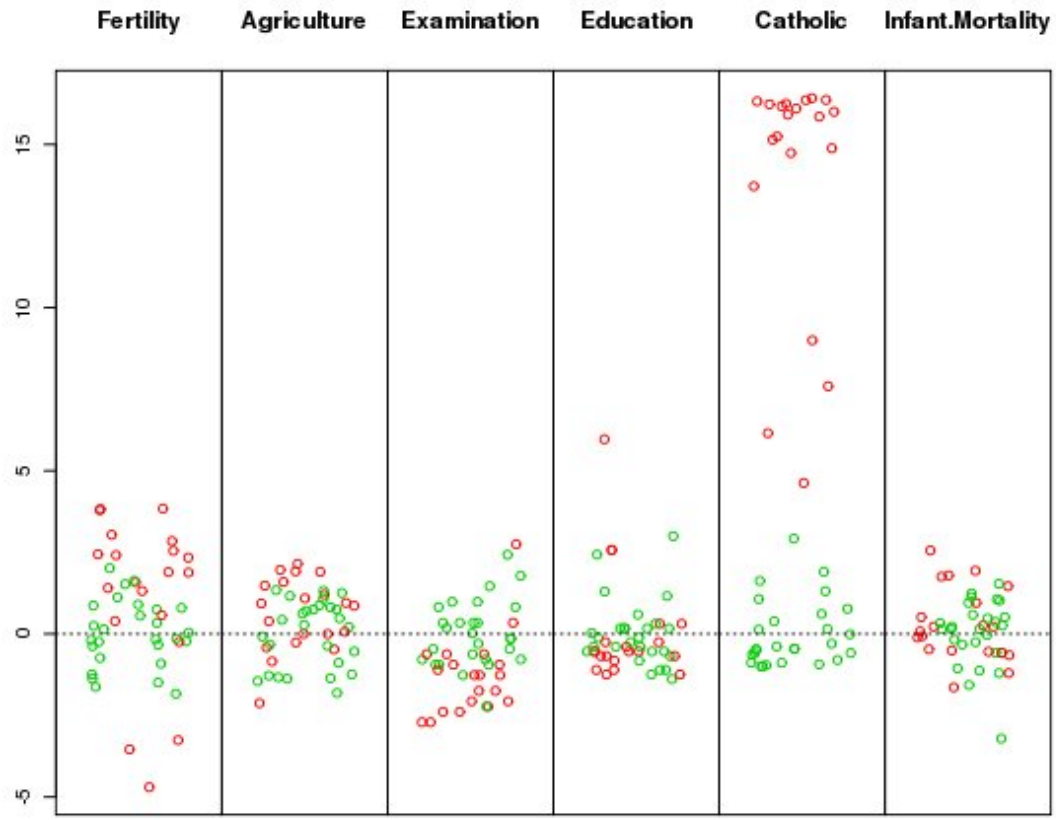
Fertility	Ig, 'common standardized fertility measure'
Agriculture	% of males involved in agriculture as occupation
Examination	% draftees receiving highest mark on army examination
Education	% education beyond primary school for draftees.
Catholic	% "catholic" (as opposed to "protestant").
Infant.Mortality	live births who live less than 1 year.

All variables but "Fertility" give proportions of the population.

To find the outliers of this dataset just type these two commands below:

```
> data(swiss)
> uni.plot(swiss)
```

The above commands will generate the following figure:

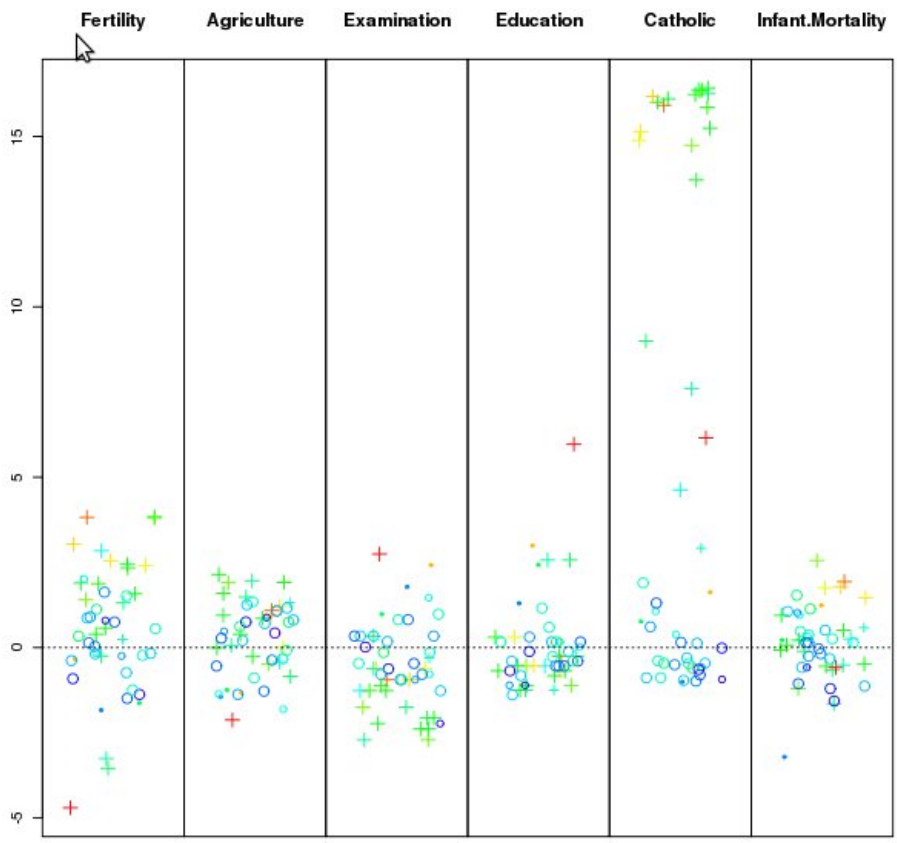


The exceptions are found by analyzing the correlation among each feature represented by the columns. For example, the red points next to the value 0 would not be outliers if they were analyzed separately, but as the correlation is considered the points are outliers. Also it is important to point out that the outliers from a column are the same in the others.

The y-axis represents the robust Mahalanobis distance based on the mcd estimator. The zero point indicates the statistical average of the values.

A more detailed output can be reached with the following commands:

```
> data(swiss)
> uni.plot(swiss, symb=TRUE)
```



In the first picture there are only two colors and no special symbols are used. Outliers are marked red.

In the second picture we have set the argument symbol TRUE. In this case, different symbols (cross means big value, circle means little value) according to the robust Mahalanobis [3] distance based on the mcd estimator and different colors (red means big value, blue means little value) according to the euclidean distances of the observations are used.

Besides highlight the outliers in the figure, a table is generated to identify which elements correspond to the outliers highlighted. In this table, the elements marked as TRUE are the outliers. An example of this table is shown in the Section #Case_Study .

Case Study

Scenario

Suppose you want to buy an antique car, because you're a famous collector. You have a list with many characteristics of each car.

A car that stands out would be a good idea, but a car that "stands out" can be very good or very bad. So which car to buy?

Input Data

Description

The dataset that we are going to use in this case study, called mtcars [4], was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Format

A data frame with 32 observations on 11 variables.

mpg	Miles/(US) gallon
cyl	Number of cylinders
disp	Displacement (cu.in.)
hp	Gross horsepower
drat	Rear axle ratio
wt	Weight (lb/1000)
qsec	1/4 mile time
vs	V/S
am	Transmission (0 = automatic, 1 = manual)
gear	Number of forward gears
carb	Number of carburetors

As a collector, you are only interested in three characteristics: *mpg*, *qseq* and *hp*. So a filter need to be done in the dataset. Moreover, you are only interested in the first 15 cars of the list, because you already have the others :)

Filtered Dataset

```
> cars
                   mpg  qsec  hp
Mazda RX4         21.0 16.46 110
Mazda RX4 Wag     21.0 17.02 110
Datsun 710        22.8 18.61  93
Hornet 4 Drive    21.4 19.44 110
Hornet Sportabout 18.7 17.02 175
Valiant           18.1 20.22 105
Duster 360        14.3 15.84 245
Merc 240D         24.4 20.00  62
Merc 230          22.8 22.90  95
Merc 280          19.2 18.30 123
Merc 280C         17.8 18.90 123
Merc 450SE        16.4 17.40 180
Merc 450SL        17.3 17.60 180
Merc 450SLC       15.2 18.00 180
Cadillac Fleetwood 10.4 17.98 205
```

Execution

Loading the dataset:

```
> data(mtcars)
```

Filtering the dataset:

```
> cars = mtcars[1:15, c("mpg", "qsec", "hp")]
```

To see what we have filtered:

```
> cars
```

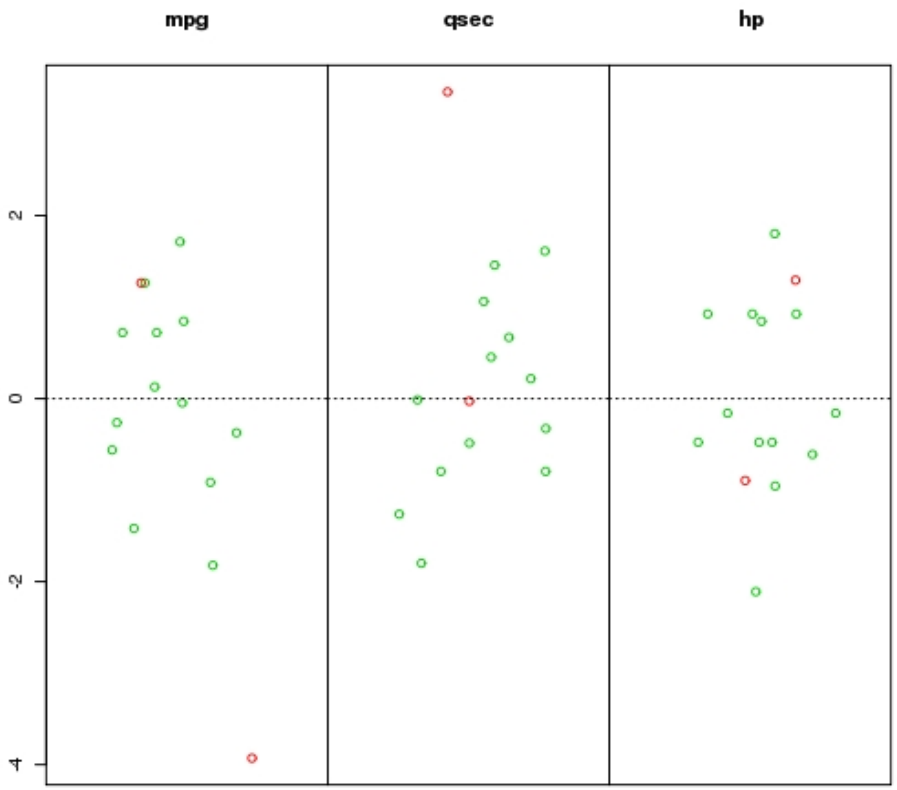
Finding the stands out cars:

```
> uni.plot(log(cars))
```

The "log" in the above command is used to put y-axis in logarithmic scale.

Results

mvoutlier Graphic:



mvoutlier Tables:

```
$outliers
        Mazda RX4      Mazda RX4 Wag         Datsun 710     Hornet 4 Drive
            FALSE              FALSE              FALSE              FALSE
 Hornet Sportabout            Valiant         Duster 360          Merc 240D
            FALSE              FALSE              FALSE              FALSE
         Merc 230          Merc 280          Merc 280C         Merc 450SE
             TRUE              FALSE              FALSE              FALSE
       Merc 450SL       Merc 450SLC Cadillac Fleetwood
            FALSE              FALSE               TRUE

$md
        Mazda RX4      Mazda RX4 Wag         Datsun 710     Hornet 4 Drive
        2.2027769          1.5647467          0.8632522          1.4611238
 Hornet Sportabout            Valiant         Duster 360          Merc 240D
        1.1628023          1.2830063          2.1006165          1.3942974
         Merc 230          Merc 280          Merc 280C         Merc 450SE
        4.3446714          0.1270212          0.7114143          0.8058947
       Merc 450SL       Merc 450SLC Cadillac Fleetwood
        0.8559053          1.1592932          4.7235206
```

Analysis

The first figure shows us that we have two outliers, i.e., two cars that stands out. In the figure, these cars are represented by the red points. The outliers in a column are the same in the others.

It is worth noting that exceptions are found by analyzing the correlation among each feature of the car. For example, the red dot next to the value 0 in column qsec would not be an outlier if it were analyzed separately, but as the correlation is considered the point is an outlier.

To find out the cars that stands out we analyze the second figure. Clearly we can see the outliers marked as TRUE. So the outliers are Merc 230 and Cadillac Fleetwood. Now the collector life is easier. It only remains to identify what kind of "stand out" the collector is interested. Looking the #Filtered Dataset, we can note that the car Merc 230 is very economical, but slower and less powerful and we can see too that the Cadillac Fleetwood is very fast and powerful, but consume too much fuel. As a good collector, he will likely choose the Cadillac Fleetwood, because it was probably the most desired car.

Congratulations!!! You have just bought this amazing car!!!

References

1. ^ P. Filzmoser. A multivariate outlier detection method. In S. Aivazian, P. Filzmoser, and Yu. Kharin, editors, Proceedings of the Seventh International Conference on Computer Data Analysis and Modeling, volume 1, pp. 18-22, Belarusian State University, Minsk, 2004.
2. ^ GONÇALVES, E. C. ; ALBUQUERQUE, C. V. N. ; PLASTINO, A. . Mineração de Exceções Aplicada aos Sistemas para Detecção de Intrusoes. Revista Eletrônica de Sistemas de Informação, v. 8, p. 1-9, 2006.
3. ^ Outlier Package [1]
4. ^ Outlier Manual [5]
5. ^ Swiss [2]
6. ^ Mahalanobis Distance [3]
7. ^ mtcars dataset [4]

References

[1] http://cran.r-project.org/web/packages/mvoutlier/index.html
[2] http://stat.ethz.ch/R-manual/R-patched/library/datasets/html/swiss.html
[3] http://en.wikipedia.org/wiki/Mahalanobis_distance
[4] http://stat.ethz.ch/R-manual/R-patched/library/datasets/html/mtcars.html
[5] http://cran.r-project.org/web/packages/mvoutlier/mvoutlier.pdf

Decision Trees

Introduction

The philosophy of operation of any algorithm based on decision trees is quite simple. In fact, although sometimes containing important differences in the way to do this or that step, any algorithm of this category is based on the strategy of divide and conquer. In general, this philosophy is based on the successive division of the problem into several subproblems with a smaller number of dimensions, until a solution for each of the simpler problems can be found. Based on this principle, the classifiers based on decision trees try to find ways to divide the universe into successively more subgroups (creating nodes containing the respective tests) until each addressing only one class or until one of the classes shows a clear majority do not justifying further divisions, generating in this situation a leaf containing the class majority. Obviously, the classification is only to follow the path dictated by the successive test placed along the tree until it found a leaf containing the class to assign to the new example.

Although the basic philosophy of all the classifiers based on decision trees is identical, there are many possibilities for its construction. Among all the key points in the selection of an algorithm to build decision trees some of them should be highlighted for their importance:

- Criteria for the choice of feature to be used in each node
- How to calculate the partition of the set of examples
- When you decide that a node is a leaf
- What is the criterion to select the class to assign to each leaf

Some important advantages can be pointed to the decision trees, including:

- Can be applied to any type of data
- The final structure of the classifier is quite simple and can be stored and handled in a graceful manner
- Handles very efficiently conditional information, subdividing the space into sub-spaces that are handled individually
- Reveal normally robust and insensitive to misclassification in the training set

- The resulting trees are usually quite understandable and can be easily used to obtain a better understanding of the phenomenon in question. This is perhaps the most important of all the advantages listed

Algorithm

The basic algorithm for decision tree is the greedy algorithm that constructs decision trees in a top-down recursive divide-and-conquer manner. We usually employ greedy strategies because they are efficient and easy to implement, but they usually lead to sub-optimal models. A bottom-up approach could also be used. The top-down decision tree algorithm is given in Algorithm 1. It is a recursive divide-and-conquer algorithm. It takes a subset of data D as input and evaluate all possible splits (Lines 4 to 11). The best split decision (Line 12), i.e. the split with the highest information gain, is chosen to partition the data in two subsets (divide-and-conquer) and the method is called recursively (Lines 14 and 15). The algorithm stops when the stop conditions are met (Line 1 to 3).

Stopping Criteria

A number of stopping conditions can be used to stop the recursive process. The algorithm stops when any one of the conditions is true:

- All the samples belong to the same class, i.e. have the same label since the sample is already "pure"
- Stop if most of the points are already of the same class. This is a generalization of the first approach, with some error threshold
- There are no remaining attributes on which the samples may be further partitioned
- There are no samples for the branch test attribute

Attribute Selection

We now need an objective criteria for judging how good a split is. The information gain measure is used to select the test attribute at each node in the tree. The attribute with the highest information gain (or greatest entropy reduction) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions.

Entropy, in general, measures the amount of disorder or uncertainty in a system. In the classification setting, higher entropy (i.e., more disorder) corresponds to a sample that has a mixed collection of labels. Lower entropy corresponds to a case where we have mostly pure partitions. In information theory, the entropy of a sample *D* is defined as follows:

$$H(D) = -\sum_{i=1}^{k} P(c_i|D) \log_2 P(c_i|D)$$

where $P(ci|D)$is the probability of a data point in *D* being labeled with class c_i, and *k* is the number of classes. $P(ci|D)$can be estimated directly from the data as follows:

$$P(c_i|D) = \frac{|\{x_j \in D | x_j \text{ has label } y_j = c_i\}|}{|D|}$$

We can also define the weighted entropy of a decision/split as follows:

$$H(D_L, D_R) = \frac{|D_L|}{|D|} H(D_L) + \frac{|D_R|}{|D|} H(D_R)$$

where *D* has been partitioned into D_L and D_R due to some split decision. Finally, we can define the information gain for a given split as:

$$Gain(D, D_L, D_R) = H(D) - H(D_L, D_R)$$

In other words, *Gain* is the expected reduction in entropy caused by knowing the value of an attribute.

Rpart

The rpart package found in the R tool can be used for classification by decision trees and can also be used to generate regression trees. Recursive partitioning is a fundamental tool in data mining. It helps us explore the stucture of a set of data, while developing easy to visualize decision rules for predicting a categorical (classification tree) or continuous (regression tree) outcome. The rpart programs build classification or regression models of a very general structure using a two stage procedure; the resulting models can be represented as binary trees. The tree is built by the following process: first the single variable is found which best splits the data into two groups ('best' will be defined later). The data is separated, and then this process is applied separately to each sub-group, and so on recursively until the subgroups either reach a minimum size (5 for this data) or until no improvement can be made. The resultant model is, with certainty, too complex, and the question arises as it does with all stepwise procedures of when to stop. The second stage of the procedure consists of using cross-validation to trim back the full tree.

Grow the Tree

To grow a tree, use

```
rpart (formula, data=, method=, control=)
```

where:

formula	is in the format: outcome ~ predictor1+predictor2+predictor3+ect.
data=	specifies the dataframe
method=	"class" for a classification tree "anova" for a regression tree
control=	optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001) requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted.

Visualization and Examples

printcp

The command printcp displays the cp table for fitted rpart object. Prints a table of optimal prunings based on a complexity parameter.

Usage

```
printcp(object, digits=getOption("digits") - 2)
```

where object is an rpart object and digits is the number of digits of numbers to print.

Example:

```
fit <- rpart(Price ~ HP, car.test.frame)
printcp(fit)
```

Output

```
Regression tree:
rpart(formula = Price ~ HP, data = car.test.frame)
Variables actually used in tree construction:
[1] HP
Root node error: 983551497/60 = 16392525
n= 60
```

```
    CP nsplit rel error xerror xstd
    1 0.41417 0 1.00000 1.03808 0.21528
    2 0.12304 1 0.58583 0.71817 0.15575
    3 0.01000 2 0.46279 0.62650 0.11675
```

plotcp

The command plotcp gives a visual representation of the cross-validation results in an rpart object. The set of possible cost-complexity prunings of a tree from a nested set. For the geometric means of the intervals of values of cp for which a pruning is optimal, a cross-validation has (usually) been done in the initial construction by rpart. The cptable in the fit contains the mean and standard deviation of the errors in the cross-validated prediction against each of the geometric means, and these are plotted by this function. A good choice of cp for pruning is often the leftmost value for which the mean lies below the horizontal line.

Usage

```
    plotcp(object, minline = TRUE, lty = 3, col = 1, upper = c("size", "splits", "none"), args)
```

Where object is an rpart object, minline is whether a horizontal line is drawn 1SE above the minimum of the curve, lty is the line type for this line, col is the colour for this line upper what is plotted on the top axis: the size of the tree (the number of leaves), the number of splits or nothing and args are the arguments to be passed to or from other methods.

Example:

```
    fit <- rpart(Kyphosis ~ Age + Number + Start, method="class", data=kyphosis)
    plotcp(fit)
```

rsq.rpart

Plot approximate r-squared versus the number of splits and relative error for different splits versus the number of splits (two plots).

Usage

```
    rsq.rpart(object)
```

Where object is an rpart object.

Example:

```
    fit <- rpart(Mileage ~ Weight, car.test.frame)
    rsq.rpart(fit)
```

print

Prints an rpart object.

Usage

```
    print(object, minlength=0, spaces=2, cp, digits= getOption("digits"), args)
```

where object is an rpart object, minlength controls the abbreviation of labels, spaces is the number of spaces to indent nodes of increasing depth, digits is the number of digits of numbers to print, cp prune all nodes with a complexity less than cp from the printout and args are the arguments to be passed to or from other methods.

Example:

```
fit <- rpart(Kyphosis ~ Age + Number + Start, method="class", data=kyphosis)
print(fit)
```

```
n= 81
node), split, n, loss, yval, (yprob)
* denotes terminal node
1) root 81 17 absent (0.7901235 0.2098765)
2) Start>=8.5 62 6 absent (0.9032258 0.0967742)
4) Start>=14.5 29 0 absent (1.0000000 0.0000000) *
5) Start< 14.5 33 6 absent (0.8181818 0.1818182)
10) Age< 55 12 0 absent (1.0000000 0.0000000) *
11) Age>=55 21 6 absent (0.7142857 0.2857143)
22) Age>=111 14 2 absent (0.8571429 0.1428571) *
23) Age< 111 7 3 present (0.4285714 0.5714286) *
3) Start< 8.5 19 8 present (0.4210526 0.5789474) *
```

summary

Returns a detailed listing of a fitted rpart object.

Usage

```
summary(object, cp=0, digits=getOption("digits"), file, args)
```

Where object is an rpart object, digits is the number of significant digits to be used in the result, cp trim nodes with a complexity of less than cp from the listing, file write the output to a given file name and args are the arguments to be passed to or from other methods.

Example:

```
fit <- rpart(Mileage ~ Weight, car.test.frame)
summary(fit)
```

```
Call:
rpart(formula = Mileage ~ Weight, data = car.test.frame)
n= 60
CP nsplit rel error xerror xstd
1 0.59534912 0 1.0000000 1.0294527 0.17907324
2 0.13452819 1 0.4046509 0.6261647 0.10545991
3 0.01282843 2 0.2701227 0.4746041 0.08567822
4 0.01000000 3 0.2572943 0.4884699 0.08551818
Node number 1: 60 observations, complexity param=0.5953491
mean=24.58333, MSE=22.57639
left son=2 (45 obs) right son=3 (15 obs)
Primary splits:
Weight < 2567.5 to the right, improve=0.5953491, (0 missing)
Node number 2: 45 observations, complexity param=0.1345282
mean=22.46667, MSE=8.026667
left son=4 (22 obs) right son=5 (23 obs)
Primary splits:
Weight < 3087.5 to the right, improve=0.5045118, (0 missing)
Node number 3: 15 observations
```

```
mean=30.93333, MSE=12.46222
Node number 4: 22 observations
mean=20.40909, MSE=2.78719
Node number 5: 23 observations, complexity param=0.01282843
mean=24.43478, MSE=5.115312
left son=10 (15 obs) right son=11 (8 obs)
Primary splits:
Weight < 2747.5 to the right, improve=0.1476996, (0 missing)
Node number 10: 15 observations
mean=23.8, MSE=4.026667
Node number 11: 8 observations
mean=25.625, MSE=4.984375
```

plot (and text)

Plots an rpart object on the current graphics device as a decision tree. The function text label the decision tree plot.

Usage

```
plot(object, uniform=FALSE, branch=1, compress=FALSE, nspace, margin=0, minbranch=.3, args)
```

Where object is an rpart object, uniform if TRUE uniform vertical spacing of the nodes is used, branch controls the shape of the branches from parent to child node, compress if FALSE, the leaf nodes will be at the horizontal plot coordinates of 1:nleaves, if TRUE, the routine attempts a more compact arrangement of the tree, nspace is the amount of extra space between a node with children and a leaf, margin is an extra percentage of white space to leave around the borders of the tree, minbranch set the minimum length for a branch to minbranch times the average branch length and args are the arguments to be passed to or from other methods.

Example:

```
fit <- rpart(Price ~ Mileage + Type + Country, cu.summary)
plot(fit, compress=TRUE)
text(fit, use.n=TRUE)
```

post

Create a PostScript presentation plot of an rpart object.

Usage

```
plot(object, uniform=FALSE, branch=1, compress=FALSE, nspace, margin=0, minbranch=.3, args)
```

Object is an rpart object, uniform if TRUE uniform vertical spacing of the nodes is used, branch controls the shape of the branches from parent to child node, compress if FALSE, the leaf nodes will be at the horizontal plot coordinates of 1:nleaves, if TRUE, the routine attempts a more compact arrangement of the tree, nspace is the amount of extra space between a node with children and a leaf, margin is an extra percentage of white space to leave around the borders of the tree, minbranch set the minimum length for a branch to minbranch times the average branch length and args are the arguments to be passed to or from other methods.

Example:

```
fit <- rpart(Mileage ~ Weight, car.test.frame)
post(fit, file = "", title="Classification Tree for Wikibook")
post(fit, file = "c:/output.ps", title = " Classification Tree for Wikibook")
```

Case Study

Scenario and Input data

Consider the relational database in the table below, whose schema is composed of attributes Play, Outlook, Temperature, Humidity and Windy. A decision tree allows predicting the values of the attribute Play, given that we know the values for attributes like Outlook, Humidity and Windy.

weather	Temperature	Humidity	Wind	Golf play					
fine	hot	high	none	no					
fine	hot	high	few	no	cloud	hot	high	none	yes
rain	warm	high	none	yes					
rain	cold	mediam	none	yes					
rain	cold	mediam	few	no					
cloud	cold	mediam	few	yes					
fine	warm	heigh	none	no					
fine	cold	mediam	none	yes					
rain	warm	mediam	none	yes					
fine	warm	mediam	few	yes					
cloud	warm	high	few	yes					
cloud	hot	mediam	none	yes					
rain	warm	high	few	no					

Importing data into R is simple. From a comma delimited text (CSV) file whose the first row contains variable names we can use the command below:

```
play_base <- read.table("path_to_the_file/play.csv", header=TRUE, sep=",")
```

We can use the command *print(play_base)* or just *play_base* to see the loaded table and the command "summary(play_base)" to see a detailed listing of the rpart object:

```
Play    Outlook      Temperature   Humidity   Windy
no :3   overcast:2   cool:5        high :4    false:7
yes:7   rainy :4     hot :2        normal:6   true :3
        sunny :4     mild:3
```

Execution and Output

After we have loaded the data we need to build the decision tree. The "Play" attribute is the outcome that will the predicted. We can use the command below:

```
fit <- rpart(Play ~ Outlook + Temperature + Humidity + Wind, method="class", data=play_base,
control=rpart.control(minsplit=1))
```

We can use the command *summary(fit)* to see a detailed listing of the loaded decision tree and the command *print(fit)* to see the decision tree:

```
 1) root 10 3 yes (0.3000000 0.7000000)
   2) Temperature= mild 3 1 no (0.6666667 0.3333333)
     4) Outlook= sunny 2 0 no (1.0000000 0.0000000) *
```

```
        5) Outlook= overcast 1 0 yes (0.0000000 1.0000000) *
      3) Temperature= cool, hot 7 1 yes (0.1428571 0.8571429)
        6) Windy= true 1 0 no (1.0000000 0.0000000) *
        7) Windy= false 6 0 yes (0.0000000 1.0000000) *
```

The commands below plots an rpart object on the current graphics device as a decision tree:

```
plot(fit, uniform=TRUE, main="Decision Tree - Play?")
text(fit, use.n=TRUE, all=TRUE, cex=.8)
```

Analysis

The building of a decision tree starts with a description of a problem which should specify the variables, actions and logical sequence for a decision-making. In a decision tree, a process leads to one or more conditions that can be brought to an action or other conditions, until all conditions determine a particular action, once built you can have a graphical view of decision-making.

The decision tree generated to solve the problem, the sequence of steps described determines and the weather conditions, verify if it is a good choice to play or not to play. For instance, in the sequence of conditions *(temperature = mild) -> (Outlook = overcast) -> play = yes*, whereas in the sequence *(temperature = cold) -> (Windy = true) -> play = no*. This shows that a decision tree is a great tool for making decisions. Thus, this method of classification may become an excellent tool for obtaining information, which often organizations do not know they have, and which are extremely important to the tactical and management level.

References

Jiawei Han. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, 2001.

Mohammed J. Zaki and Wagner Meira Jr.. Fundamentals of Data Mining Algorithms. Cambridge University Press, 2010.

Terry M. Therneau, Elizabeth J. Atkinson and Mayo Foundation. An Introduction to Recursive Partitioning Using the RPART Routines, 1997.

R Language Definition - [1]

An Introduction to R - [2]

Quick-R - [3]

Terry M Therneau and Beth Atkinson. Package ‘rpart’, 2009.

References

[1] http://cran.r-project.org/doc/manuals/R-lang.html

[2] http://cran.r-project.org/doc/manuals/R-intro.html

[3] http://www.statmethods.net/

Naïve Bayes

Introduction

This chapter introduces the Naïve Bayes algorithm for classification. Naïve Bayes (NB) based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. It is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods.

Naïve Bayes

Naive Bayes classifiers can handle an arbitrary number of independent variables whether continuous or categorical. Given a set of variables, X = { $x_1, x_2, x_3..., x_d$ }, we want to construct the posterior probability for the event C_j among a set of possible outcomes C = { $c_1, c_2, c_3..., c_n$ }. In a more familiar language, X is the predictors and C is the set of categorical levels present in the dependent variable. Using Bayes' rule:

$$p(C|x_1, \ldots, x_d) = \frac{p(C)\ p(x_1, \ldots, x_d|C)}{p(x_1, \ldots, x_d)}.$$

where $p(C_j|x_1, \ldots, x_d)$ is the posterior probability of class membership, i.e., the probability that X belongs to C_j.

In practice we are only interested in the numerator of that fraction, since the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability:

$$p(C, x_1, \ldots, x_d) = p(C)\ p(x_1|C)\ p(x_2|C, F_1)\ p(x_3|C, x_1, x_2)\ \ldots p(x_n|C, x_1, x_2, x_3, \ldots, F_{d-1}).$$

The "naive" conditional independence assumptions come into play: assume that each feature x_i is conditionally statistical independent of every other feature x_j for $j \neq i$. This means that

$$p(x_i|C, x_j) = p(x_i|C)$$

for $i \neq j$, and so the joint model can be expressed as

$$p(C, x_1, \ldots, x_d) = p(C)\ p(x_1|C)\ p(x_2|C)\ p(x_3|C)\ \cdots$$
$$= p(C) \prod_{i=1}^{d} p(x_i|C).$$

This means that under the above independence assumptions, the conditional distribution over the class variable C can be expressed like this:

$$p(C|x_1, \ldots, x_d) = \frac{1}{Z} p(C) \prod_{i=1}^{d} p(x_i|C)$$

where Z (the evidence) is a scaling factor dependent only on $x_1, \ldots, x_d$, i.e., a constant if the values of the feature variables are known.

Finally, we can label a new case F with a class level C_j that achieves the highest posterior probability:

$$\text{classify}(F_1, \ldots, F_d) = \underset{c}{\text{argmax}}\ p(C = c) \prod_{i=1}^{d} p(x_i = F_i|C = c).$$

Available Implementations

There are at least two R implementations of Naïve Bayes classification available on CRAN [2]:

- e1071 [1]
- klaR [2]

Installing and Running the Naïve Bayes Classifier

E1071 is a CRAN package, so it can be installed from within R:

```
> install.packages('e1071', dependencies = TRUE)
```

Once installed, e1071 can be loaded in as a library:

```
> library(class)
> library(e1071)
```

It comes with several well-known datasets, which can be loaded in as ARFF files (Weka's default file format). We now load a sample dataset, the famous Iris dataset [3] and learn a Naïve Bayes classifier for it, using default parameters. First, let us take a look at the Iris dataset.

Dataset

The Iris dataset contains 150 instances, corresponding to three equally-frequent species of iris plant (Iris setosa, Iris versicolour, and Iris virginica). An Iris versicolor is shown below, courtesy of Wikimedia Commons.

Iris versicolor

Each instance contains four attributes:sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. The next picture shows each attribute plotted against the others, with the different classes in color.

```
> pairs(iris[1:4], main = "Iris Data (red=setosa,green=versicolor,blue=virginica)",
      pch = 21, bg = c("red", "green3", "blue")[unclass(iris$Species)])
```

Execution and Results

First of all, we need to specify which base we are going to use:

```
> data(iris)
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
```

```
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

After that, we are ready to create a Naïve Bayes model to the dataset using the first 4 columns to predict the fifth. (Factor the target column by so: dataset$col <- factor(dataset$col))

```
> classifier<-naiveBayes(iris[,1:4], iris[,5])
> table(predict(classifier, iris[,-5]), iris[,5])

             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          3        47
```

Analysis

This simple case study shows that a Naïve Bayes classifier makes few mistakes in a dataset that, although simple, is not linearly separable, as shown in the scatterplots and by a look at the confusion matrix, where all misclassifications are between Iris Versicolor and Iris Virginica instances.

References

1. ^ Fisher,R.A. (1936); The use of multiple measurements in taxonomic problems. Annual Eugenics, 7, Part II, 179-188.

References

[1] http://cran.r-project.org/web/packages/e1071/index.html
[2] http://cran.r-project.org/web/packages/klaR/index.html
[3] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/Na%C3%AFve_Bayes#endnote_Fisher

adaboost

Boosting is one of the most important developments in classification methodology. Boosting works by sequentially applying a classification algorithm to reweighted versions of the training data and then taking a weighted majority vote of the sequence of classifiers thus produced. For many classification algorithms, this simple strategy results in dramatic improvements in performance. This seemingly mysterious phenomenon can be understood in terms of well-known statistical principles, namely additive modeling and maximum likelihood. For the two-class problem, boosting can be viewed as an approximation to additive modeling on the logistic scale using maximum Bernoulli likelihood as a criterion.

Technique/Algorithm

Algorithm

While boosting has evolved somewhat over the years, we describe the most commonly used version of the AdaBoost procedure (Freund and Schapire - 1996) which we call Discrete AdaBoost. This is essentially the same as AdaBoost.M1 for binary data in Freund and Schapire. Here is a concise description of AdaBoost in the two-class classification setting.We have training data $(x_1, y_1), ..., (x_n, y_n)$ with x_i a vector valued feature and $y_i = -1$ or 1. We define $F(x) = \sum_{1}^{M} c_m f_m$ where each $f_m(x)$ is a classifier producing values plus or minus 1 and c_m are constants; the corresponding prediction is sign $(F(x))$. The AdaBoost trains the classifiers f_m(x) on weighted versions of the training sample, giving higher weight to cases that are currently misclassified. This is done for a sequence of weighted samples, and then the final classifier is defined to be a linear combination of the classifiers from each stage.

Implementation

Adaboost is part of ada package. In this section you find more information about installing and using it on R Environment.

Type the following commands in R console to install and load the ada package:

```
install.packages("ada")
library("rpart")
library("ada")
```

The function used to execute the algorithm adaboost is:

```
ada(x, y,test.x,test.y=NULL, loss=c("exponential","logistic"),
type=c("discrete", "real", "gentle"), iter=50, nu=0.1,
bag.frac=model.coef=TRUE, bag.shift=FALSE, max.iter=20, delta=10^(-10),
 verbose=...,na.action=na.rpart)
```

The arguments are:

```
x: matrix of descriptors.


Y: vector of responses. ‘y’ may have only two unique values.


test.x: testing matrix of discriptors (optional)
```

```
test.y: vector of testing responses (optional)

loss: loss="exponential", "ada","e" or any variation corresponds to the default boosting
under exponential loss. loss="logistic","l2","l" provides boosting under logistic
loss.

type: type of boosting algorithm to perform. “discrete”performs discrete Boosting
(default). “real”performs Real Boost. “gentle”performs Gentle Boost.

Iter: number of boosting iterations to perform. Default = 50.

Nu: shrinkage parameter for boosting, default taken as 1.

bag.frac: sampling fraction for samples taken out-of-bag. This allows one to use random
permutation which improves performance.

model.coef: flag to use stageweights in boosting. If FALSE then the procedure corresponds
to epsilon-boosting.

bag.shift: flag to determine whether the stageweights should go to one as nu goes to zero.
This only makes since if bag.frac is small. The rationale behind this parameter
is discussed in (Culp et al., 2006).

max.iter: number of iterations to perform in the newton step to determine the coeficient.

delta: tolarence for convergence of the newton step to determine the coeficient.

Verbose: print the number of iterations necessary for convergence of a coeficient.

Formula: a symbolic description of the model to be fit.

data: an optional data frame containing the variables in the model.

Subset: an optional vector specifying a subset of observations to be used in the fitting
process.

na.action: a function that indicates how to process ‘NA’values. Default=na.rpart.

...: arguments passed to rpart.control. For stumps, use rpart.control(maxdepth=1,cp=-
1,minsplit=0,xval=0). maxdepth controls the depth of trees, and cp
controls the complexity of trees. The priors should also be fixed through the
parms argument as discussed in the second reference.
```

Type the following command to show the result from this algorithm:

```
summary(AdaObject)
varplot(VariableImportanceObject)
```

When using usage 'ada(x,y)': x data can take the form data.frame or as.matrix. y data can take form data.frame, as.factor, as.matrix, as.array, or as.table. missing values must be removed from the data prior to execution.

When using usage 'ada(y~.)': data must be in a data frame. Response can have factor or numeric values. missing values can be present in the descriptor data, whenever na.action is set to any option other than na.pass.

After the model is fit, 'ada' prints a summary of the function call, the method used for boosting, the number of iterations, the final confusion matrix (observed classification vs predicted classification; labels for classes are same as in response), the error for the training set, and testing, training , and kappa estimates of the appropriate number of iterations.

A summary of this information can also be obtained with the command 'print(x)'. Corresponding functions (Use help with summary.ada, predict.ada, . . . varplot for additional information on these commands): summary : function to print a summary of the original function call, method used for boosting, number of iterations, final confusion matrix, accuracy, and kappa statistic (a measure of agreement between the observed classification and predicted classification). 'summary' can be used for training, testing, or validation data.

predict : function to predict the response for any data set (train, test, or validation)

plot : function to plot performance of the algorithm across boosting iterations. Default plot is iteration number (x-axis) versus prediction error (y-axis) for the data set used to build the model. Function can also simultaneously produce an error plot for an external test set and a kappa plot for training and test sets.

pairs : function to produce pairwise plots of descriptors. Descriptors are arranged by decreasing frequency of selection by boosting (upper left = most frequently chosen). The color of the marker in the plot represents class membership; the Size of the marker represents predicted class probability. The larger the marker, the higher the probability of classification.

varplot : plot of variables ordered by the variable importance measure (based on improvement).

addtest : add a testing data set to the ada object, therefore the testing errors only have to be computed once.

update : add more trees to the ada object.

Case Study

Scenario

A data set that contains information about compounds used in drug discovery. Specifically, this data set consists of 5631 compounds on which an in-house solubility screen (ability of a compound to dissolve in a water/solvent mixture) was performed. Based on this screen, compounds were categorized as either insoluble (n=3493) or soluble (n=2138). Then, for each compound, 72 continuous, noisy structural descriptors were computed. Of these descriptors, one contained missing values for approximately 14% (n=787) of the observations. The objective of the analysis is to model the relationship between the structural descriptors and the solubility class. The data will be called soldat.

Data

Input format:

```
x1 a numeric vector
x2 a numeric vector
x3 a numeric vector
x4 a numeric vector
x5 a numeric vector
x6 a numeric vector
```

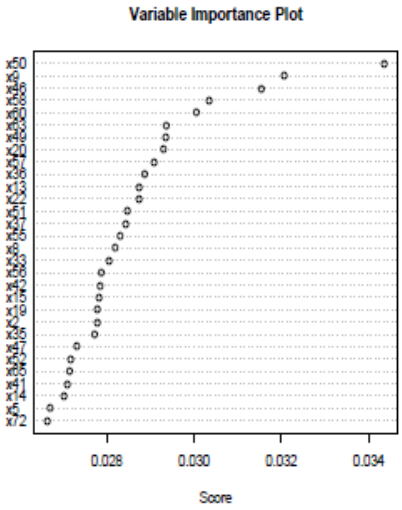
```
x7 a numeric vector
x8 a numeric vector
x9 a numeric vector
x10 a numeric vector
x11 a numeric vector
x12 a numeric vector
x13 a numeric vector
x14 a numeric vector
x15 a numeric vector
x16 a numeric vector
x17 a numeric vector
x18 a numeric vector
x19 a numeric vector
x20 a numeric vector
.
.
.
x72 a numeric vector with missing data
y a numeric vector
```

Execution

```
data("soldat")

n <- nrow(soldat)

set.seed(100)

ind <- sample(1:n)

trainval <- ceiling(n * .5)

testval <- ceiling(n * .3)

train <- soldat[ind[1:trainval],]

test <- soldat[ind[(trainval + 1):(trainval + testval)],]

valid <- soldat[ind[(trainval + testval + 1):n],]


control <- rpart.control(cp = -1, maxdepth = 14,maxcompete = 1,xval = 0)

gen1 <- ada(y~., data = train, test.x = test[,-73], test.y = test[,73], type = "gentle", control = control, iter = 70)

gen1 <- addtest(gen1, valid[,-73], valid[,73])

summary(gen1)

varplot(gen1)
```

Output

```
Loss: exponential Method: gentle Iteration: 70
Training Results
Accuracy: 0.987 Kappa: 0.972
Testing Results
Accuracy: 0.765 Kappa: 0.487
```



Analysis

Testing accuracy rates are printed in the order they are entered so the accuracy on the testing set is 0.765 and on the validation set 0.781. For this type of early drug discovery data, the Gentle AdaBoost algorithm performs adequately with test set accuracy of 76.5% (kappa is aproximately 0.5). In order to enhance our understanding regarding the relationship between descriptors and the response, the varplot function was employed.

References

1. Meira Jr., W.; Zaki, M. Fundamentals of Data Mining Algorithms. [1]
2. CBA R package. [1]
3. ADDITIVE LOGISTIC REGRESSION: A STATISTICAL VIEW OF BOOSTING, by Jerome Friedman, Trevor Hastie and Robert Tibshirani

References

[1] http://cran.r-project.org/web/packages/ada/index.html

JRip

Synopsis

This class implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed by William W. Cohen as an optimized version of IREP. It is based in association rules with reduced error pruning (REP), a very common and effective technique found in decision tree algorithms. In REP for rules algorithms, the training data is split into a growing set and a pruning set. First, an initial rule set is formed that over ts the growing set, using some heuristic method. This overlarge rule set is then repeatedly simplified by applying one of a set of pruning operators typical pruning operators would be to delete any single condition or any single rule. At each stage of simplification, the pruning operator chosen is the one that yields the greatest reduction of error on the pruning set. Simplification ends when applying any pruning operator would increase error on the pruning set.

The algorithm is briefly described as follows: Initialize RS = { }, and for each class from the less prevalent one to the more frequent one, DO:

1. Building stage:

Repeat 1.1 and 1.2 until the description length (DL) of the rule set and examples is 64 bits greater than the smallest DL met so far, or there are no positive examples, or the error rate >= 50%.

1.1. Grow phase:

Grow one rule by greedily adding antecedents (or conditions) to the rule until the rule is perfect (i.e. 100% accurate). The procedure tries every possible value of each attribute and selects the condition with highest information gain: p(log(p/t)-log(P/T)).

1.2. Prune phase:

Incrementally prune each rule and allow the pruning of any final sequences of the antecedents;The pruning metric is (p-n)/(p+n) – but it's actually 2p/(p+n) -1, so in this implementation we simply use p/(p+n) (actually (p+1)/(p+n+2), thus if p+n is 0, it's 0.5).

2. Optimization stage:

after generating the initial rule set {Ri}, generate and prune two variants of each rule Ri from randomized data using procedure 1.1 and 1.2. But one variant is generated from an empty rule while the other is generated by greedily adding antecedents to the original rule. Moreover, the pruning metric used here is (TP+TN)/(P+N).Then the smallest possible DL for each variant and the original rule is computed. The variant with the minimal DL is selected as the final representative of Ri in the rule set. After all the rules in {Ri} have been examined and if there are still residual positives, more rules are generated based on the residual positives using Building Stage again. 3. Delete the rules from the rule set that would increase the DL of the whole rule set if it were in it. and add resultant rule set to RS. ENDDO Note that there seem to be 2 bugs in the original ripper program that would affect the rule set size and accuracy slightly. This implementation avoids these bugs and thus is a little bit different from Cohen's original implementation. Even after fixing the bugs, since the order of classes with the same frequency is not defined in ripper, there still seems to be some trivial difference between this implementation and the original ripper, especially for audiology data in UCI repository, where there are lots of classes of few instances. Details please see:

1. William W. Cohen: Fast Effective Rule Induction. In: Twelfth International Conference on Machine Learning, 115-123, 1995.

Installation

The caret package can be installed by using the following command on R's command-line:

```
install.packages("caret", dependencies = TRUE)
```

The above command shall recursivelly download and install all packages that caret depend to along with fpc itself.

Example

The example in this section will illustrate the carets's JRip usage on the IRIS database:

```
>library(caret)
>library(RWeka)
>data(iris)
>TrainData <- iris[,1:4]
>TrainClasses <- iris[,5]
>jripFit <- train(TrainData, TrainClasses,method = "JRip")
```

Study Case

Dataset

The Iris dataset contains 150 instances, corresponding to three equally-frequent species of iris plant (Iris setosa, Iris versicolour, and Iris virginica). An Iris versicolor is shown below, courtesy of Wikimedia Commons.

Iris versicolor

Each instance contains four attributes:sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. The next picture shows each attribute plotted against the others, with the different classes in color.

Execution and Results

First of all, we need to specify which base we are going to use:

```
> data(iris)
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

```
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

After that, we are ready to create a Naïve Bayes model to the dataset using the first 4 columns to predict the fifth.

```
>data(iris)
>varIndex <- 1:numSamples
>
>TrainData <- iris[,1:4]
>TrainClasses <- iris[,5]
>jripFit <- train(TrainData, TrainClasses,method =
"JRip",preProcess = c("center", "scale"),tuneLength = 10,trControl =
trainControl(method = "cv"))
```

Output

```
Loading required package: class

Attaching package: 'class'

The following object(s) are masked from 'package:reshape':

   condense

Fitting: NumOpt=1
Fitting: NumOpt=2
Fitting: NumOpt=3
Fitting: NumOpt=4
Fitting: NumOpt=5
Fitting: NumOpt=6
Fitting: NumOpt=7
Fitting: NumOpt=8
Fitting: NumOpt=9
Fitting: NumOpt=10
Aggregating results
Selecting tuning parameters
Fitting model on full training set
```

Result

```
> jripFit

Call:
train.default(x = TrainData, y = TrainClasses, method = "JRip",
    preProcess = c("center", "scale"), trControl = trainControl(method = "cv"),
    tuneLength = 10)

150 samples
```

```
  4 predictors

Pre-processing: centered, scaled
Resampling: Cross-Validation (10 fold)

Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...

Resampling results across tuning parameters:

  NumOpt  Accuracy  Kappa  Accuracy SD  Kappa SD  Selected
  1       0.953     0.93   0.045        0.0675
  2       0.953     0.93   0.045        0.0675    *
  3       0.933     0.9    0.0444       0.0667
  4       0.94      0.91   0.0584       0.0876
  5       0.94      0.91   0.0584       0.0876
  6       0.94      0.91   0.0584       0.0876
  7       0.94      0.91   0.0584       0.0876
  8       0.94      0.91   0.0584       0.0876
  9       0.94      0.91   0.0584       0.0876
  10      0.94      0.91   0.0584       0.0876

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was NumOpt = 2.
```

The caret package runned the training tunning the NumOpt JRip parameter from 1 to 10 and chouse the best performance wich is NumOpt=2 with a 95.3% accuracy. If some other algorithm was chosen, other algorithm parameter would be tunned.

If we plot the results we have a plot of the parameter choosing accuracy:

```
>plot(jripFit)
```

JRip accuracy vs. NumOpt parameter.

References

1. William W. Cohen: Fast Effective Rule Induction. In: Twelfth International Conference on Machine Learning, 115-123, 1995.

RWeka

Description

An R interface to Weka (Version 3.7.2). Weka is a collection of machine learning algorithms for data mining tasks written in Java, containing tools for data pre-processing, classification, regression, clustering, association rules, and visualization. Package RWeka contains the interface code, the Weka jar is in a separate package RWekajars.

Requirements

R (>= 2.6.0)

RWekajars (>= 3.7.2)

rJava (>= 0.6-3)

Java (>= 5.0)

General API

1. Evaluate Weka Classifier
2. Predict Weka Classifier
3. Predict Weka Clusterer
4. Weka associators
5. Weka classifiers
6. Weka classifiers functions
7. Weka classifier lazy
8. Weka classifier meta
9. Weka classifier rules
10. Weka classifier trees
11. Weka clusterers
12. Weka control
13. Weka converters
14. Weka filters
15. Weka interfaces
16. Weka stemmers
17. Weka tokenizers

IO API

1. Dot
2. Read.arff
3. Write.arff

Package API

1. WOW
2. WPM

gausspred

Description

This software is used to predict the discrete response based on selected high dimensional features, such as gene expression data. The data are modeled with Bayesian Gaussian models. When a large number of features are available, one may like to select only a subset of features to use, typically those features strongly correlated with the response in training cases. Such a feature selection procedure is however invalid since the relationship between the response and the features will appear stronger. This package provides a way to avoid this bias and yields well-calibrated prediction for the test cases when one uses F-statistic to select features.

Requirements

R (>= 2.8.1)

General API

1. assess prediction
2. data gau
3. train pred gau

optimsimplex

Description

Provides a building block for optimization algorithms based on a simplex. The optimsimplex package may be used in the following optimization methods: the simplex method of Spendley et al., the method of Nelder and Mead, Box's algorithm for constrained optimization, the multi-dimensional search by Torczon, etc...

Requirements

optimbase

General API

CCMtools

Description

This package proposes a clustering method called “Correlation Clustering Model” (CCM) based on mixture of canonical correlation analysis (CCA). It also provides some tools for cluster analysis.

General API

1. CCM
2. CWGLI
3. DI
4. Info.Criterion
5. learn.and.project.clusters
6. Percent.bad.and.false.classif.per.cluster
7. WGP

FactoMineR

FactoMineR is a R [2] package dedicated to multivariate data analysis. The main features of this package is the possibility to take into account different types of variables (quantitative or categorical), different types of structure on the data (a partition on the variables, a hierarchy on the variables, a partition on the individuals) and finally supplementary information (supplementary individuals and variables). Moreover, the dimensions issued from the different exploratory data analyses can be automatically described by quantitative and/or categorical variables. Numerous graphics are also available with various options. Finally, a graphical user interface is implemented within the Rcmdr environment in order to propose a user friendly package.

Methods

The methods implemented in this package are conceptually similar with respect to its main goal, for example, merge and simplify the data by reducing the dimensionality of the data set. These methods are used depending on what data are available and if the variables are quantitative (Numerous) or qualitative (categorical or nominal).

Several methods are implemented, the most classical (PCA, Correspondence Analysis, Multiple Correspondence Analysis, Multiple Factor Analysis) as well as some advanced methods (Hierarchical Multiple Factor Analysis, Mixed Data Analysis, Dual Multiple Factor Analysis).

For the classical ones we have the following situation-use solutions:

- Principal component analysis (PCA) when individuals are described by quantitative variables;
- □ Correspondence analysis (CA) when individuals are described by two categorical variables that leads to a contingency table;
- □ Multiple correspondence analysis (MCA) when individuals are described by categorical variables.

And for the advanced methods:

- MFA (Multiple Factorial Analysis), for which the variables of a same group may be numerical or categorical.
- HMFA (Hierarchical Multiple Factorial Analysis), an extension of MFA for which variables are structured according to a hierarchy.
- GPA (Generalized Procustean Analysis), for which variables must be continuous.

Let X be the data table of interest. In order to reduce the dimensionality, X is transformed to a new coordinate system by an orthogonal linear transformation. Let F_s (resp. G_s) denotes the vector of the coordinates of the rows (resp. columns) on the axis of rank s. Those two vectors are related by the so called “transition formulae”. In the case of PCA, they can be written:

$$F_s(i) = \frac{1}{\lambda_s} \sum_k x_{ik} m_k G_s(k)$$

$$G_s(i) = \frac{1}{\lambda_s} \sum_k x_{ik} p_k F_s(k)$$

where F_s (i) denotes the coordinate of the individual i on the axis s, G_s (k) the coordinate of the variable k on the axis s, λ_s the eigenvalue associated with the axis s, m_k the weight associated to the variable k, p_i the weight associated to the individual i, x_{ik} the general term of the data table (row i, column k).

The transition formulae lay the foundation of our point of view and consequently set the graphical outputs at the roots of our practice. From these formulae it is crucial to analyze the scatter plots of the individuals and of the variables conjointly: an individual is at the same side as the variables for which it takes high values, and at the opposite side of the variables for which it takes low values.

Supplementary elements

Another important feature of the transition formulae is that they can be applied to supplementary individuals and/or variables in order to add supplementary information on the scatter plots for a better understanding of the data. In the PCA framework, let i' be a new individual, its coordinate on the axis of rank s can be easily obtained as followed:

$$F_s(i') = \frac{1}{\lambda_s} \sum_k x_{i'k} m_k G_s(k)$$

In the same manner, it is also easy to calculate the coordinate of a supplementary variable when the former is quantitative; in this case the supplementary variable lies in the scatter plot of the variables. When the variable is categorical, its modalities are represented by the way of a “mean individual” per modality. For each modality, the values associated with each “mean individual” are the means of each variable over the individuals endowed with this modality; in this case the supplementary variable lies in the scatter plot of the individuals.

Implementation

Installation

Load FactoMineR in your R session by writing the following line code:

```
library(FactoMineR)
```

to Download the graphical interface of FactoMineR in your R session write the following line code (you have to be connected to internet):

```
source("http://factominer.free.fr/install-facto.r")
```

Loading and using

Load FactoMineR for each new R session by typing the following line code:

```
library(FactoMineR)
```

Or load FactoMineR and its GUI for each new R session by typing the following line code:

```
library(Rcmdr)
```

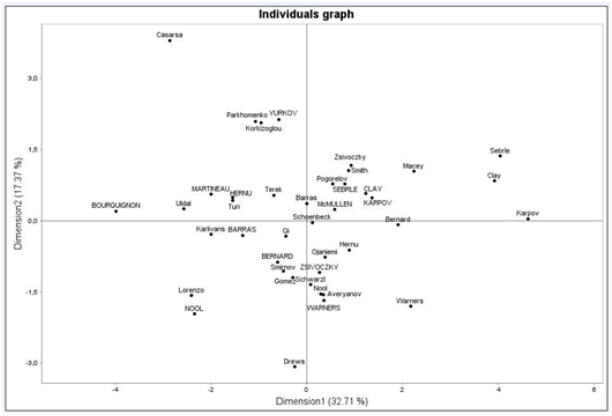
Functions Reference

A complete implementation reference of all fifty FactoMineR functions, with description, usage, arguments and values, can be foud here [1]

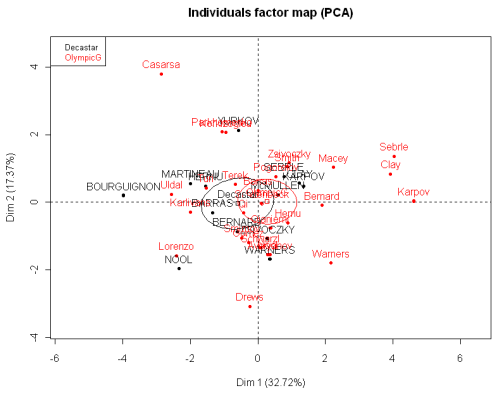
Visualization

With the function plot, you can draw graphs and results. Usage:

```
R> plot(<method>,<what variable to color individuals from>)
```

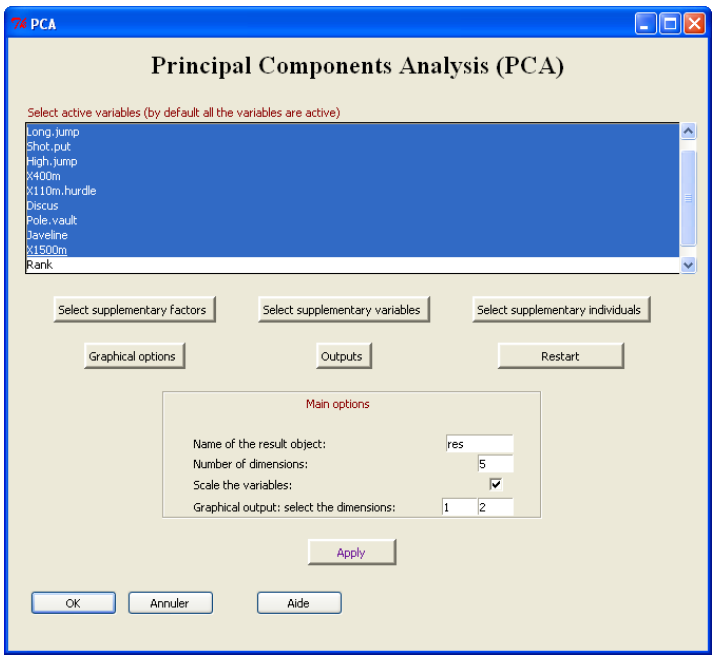


Decathlon data - available with the package documentation) : supplementary variables are in blue



Individuals graph (Decathlon data - - available with the package documentation): individuals are colored from the athletics meeting

With the graphical interface of FactoMineR, you can perform easily and intuitive tasks. As an example, we have the interface for the PCA function: The main window allows to choose the active variables (by default all the variables are active and the PCA can be performed). Several buttons allow to choose the supplementary quantitative or categorical variables, the supplementary individuals, the outputs to be displayed or the graphs to be plotted.



Individuals graph (Decathlon data - - available with the package documentation): individuals are colored from the athletics meeting

Example

As an example, we use here a data set issued from a questionnaire about French women's work in 1974. You can load the data set here [2].

Presentation of the data

1724 women have answered several questions about women's work among which:

```
* What do you think the perfect family is ?
      o Both husband and wife work
      o Husband works more than wife
      o Only husband works
* Which activity is the best for a mother when children go to school?
      o Stay at home
      o Part-time work
      o Full-time work
* What do you think of the following sentence: women who do not work feel cut off from the world?
      o Totally agree
      o Quite agree
      o Quite disagree
      o Totally disagree
```

The data set is two contingency tables which cross the answers of the first question with the two others. To each crossing, the value given is the number of women who gave both answers.

	stay.at.home	part-time.work	full-time.work	housewives.cut.from.world.totally.agree	housewives.cut.from.world.quite.agree	housewives.cut.from.world.quite.disagree	housewives.cut.from.world.totally.disagree
both.man.and.woman.work	13	142	106	107	75	40	39
man.morks.more	30	408	117	192	175	100	88
only.man.works	241	573	94	140	215	254	299

To load the package and the data set, write the following line code:

```
library(FactoMineR)

women_work=read.table("http://factominer.free.fr/classical-methods/datasets/women_work.txt", header=TRUE, row.names=1, sep="\t")
```

Objectives

The objectives of CA are quite the same as PCA's: to get a typology of rows and columns and to study the link between these two typologies. However, the concept of similarity between rows or columns is different. Here, similarity between two rows or two columns is completely symmetric. Two rows (resp. columns) will be close to each other if they associate with the columns (resp. rows) in the same way.

We are looking for the rows (resp. columns) whose distribution is the most different from the population's. The ones which look the most or the less alike. Each group of rows (resp. columns) is characterized by the columns (resp. rows) to which it is too much or to little associated.

CA

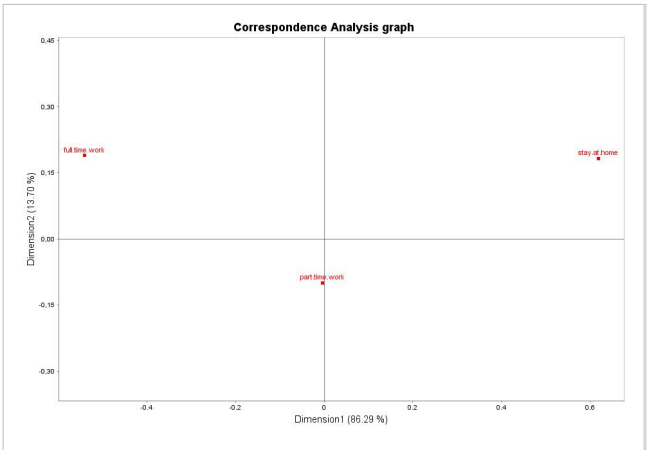
We are going to use the first three columns (corresponding to the answers to the second question) as active variables and the four last ones (corresponding to the third question) as supplementary variables.

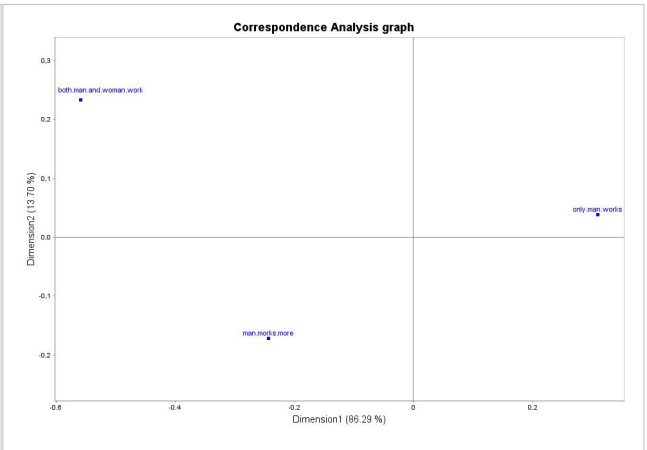
Active rows and columns only

To see the scatterplots of rows and columns separately, type:

```
res.ca.rows = CA(women_work[,1:3], invisible="col")
res.ca.col = CA(women_work[,1:3], invisible="row")
```

```
#women_work: the data set used
#invisible: elements we do not want to be plotted
```





On the scatterplot of the columns, we can see that the first axis opposes "Stay at home" and "Full-time work", which means it opposes two women's profiles. Women who answered "Stay at home" answered "Only husband works" more often than the population and "Both husband and wife work" less often than the population. In the same way, women who answered "Full-time work" answered "Only husband works" less often than the population and "Both husband and wife work" more often than the population. The first axis orders the categories of the second question from the less to the most in favour of women's work.

We can make the same interpretation for the first axis of the row's scatterplot. The categories are sorted from the less ("Only husband works") to the most ("Both husband and wife work") in favour of women's work.

To have the representation of both rows and columns, type:

```
res.ca = CA(women_work[,1:3])
#women_work: the data set used
```

"Stay at home" is much associated with "Only husband works" and little associated to the two other categories.

"Both husband and wife work" is associated with "Full-time work" and opposed to "Stay at home".

Addition of supplementary columns

We now add the columns corresponding to the third question as supplementary variables. Type:

```
res.ca = CA(women_work, col.sup=4:ncol(women_work))
#women_work: the data set used
#col.sup: vector of the indexes of the supplementary columns
```

"Totally agree" and "Quite agree" for "Women who do not work feel cut off from the world" are close to categories in favour of women's work. "Quite disagree" and "Totally "disagree" are close to categories opposed to women's work.

References

1. FactoMineR offial website [3]
2. FactoMineR: An R Package for Multivariate Analysis [4]
3. Comprehensive R Archive Network [2]

References

[1] http://cran.r-project.org/web/packages/FactoMineR/FactoMineR.pdf
[2] http://factominer.free.fr/classical-methods/datasets/women_work.txt
[3] http://factominer.free.fr/index.html
[4] http://factominer.free.fr/docs/article_FactoMineR.pdf

nnet

This chapter introduces the Feed-Forward Neural Network package for prediction and classification data. An artificial neural network (ANN), usually called "neural network" (NN), is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data. In this chapter will explore the use of feed-forward neural network through the package NNET[1][2] created by Ridley.

Feed-Forward Neural Network

A **feedforward neural network** is an artificial neural network where connections between the units do *not* form a directed cycle. This is different from recurrent neural networks.

The feedforward neural network was the first and arguably simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

ADALINE

ADALINE stands for **Ada**ptive **Lin**ear **E**lement. It was developed by Professor Bernard Widrow and his graduate student Ted Hoff at Stanford University in 1960. It is based on the McCulloch-Pitts model and consists of a weight, a bias and a summation function.

Operation: $y_i = wx_i + b$

Its adaptation is defined through a cost function (error metric) of the residual $e = d_i - (b + wx_i)$where d_i is the desired input. With the MSE error metric $E = \frac{1}{2N}\sum_i^N e_i^2$the adapted weight and bias become: $b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{N(\sum_i (x_i - \bar{x})^2)}$and $w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2}$

The Adaline has practical applications in the controls area. A single neuron with tap delayed inputs (the number of inputs is bounded by the lowest frequency present and the Nyquist rate) can be used to determine the higher order transfer function of a physical system via the bi-linear z-transform. This is done as the Adaline is, functionally, an adaptive FIR filter. Like the single-layer perceptron, ADALINE has a counterpart in statistical modelling, in this

case least squares regression.

There is an extension of the Adaline, called the Multiple Adaline (MADALINE) that consists of two or more adalines serially connected.

NNET Package

The package NNET created by Ripley provides methods for using feed-forward neural networks with a single hidden layer, and for multinomial log-linear models. Specifically, this chapter of the book will be portrayed NNET method. Below is briefly described the method and parameters used.

The implementation of NNET for Feed-Forward Neural Network for R is available on CRAN [2] and already is embbed in Environment R:

- nnet [3]

Description

Fit single-hidden-layer neural network, possibly with skip-layer connections.

```
//Usage
nnet(x, ...)

//S3 method for class 'formula':
nnet(formula, data, weights, ...,
subset, na.action, contrasts = NULL)

//Default S3 method:
nnet(x, y, weights, size, Wts, mask,
linout = FALSE, entropy = FALSE, softmax = FALSE,
censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Arguments

- **formula** A formula of the form class ~ x1 + x2 + ...
- **x** matrix or data frame of x values for examples.
- **y** matrix or data frame of target values for examples.
- **weights** (case) weights for each example – if missing defaults to 1.
- **size** number of units in the hidden layer. Can be zero if there are skip-layer units.
- **data** Data frame from which variables specified in formula are preferentially to be taken.
- **subset** An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
- **na.action** A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
- **contrasts** a list of contrasts to be used for some or all of the factors appearing as variables in the model formula.
- **Wts** initial parameter vector. If missing chosen at random.
- **mask** logical vector indicating which parameters should be optimized (default all).

- **linout** switch for linear output units. Default logistic output units.
- **entropy** switch for entropy (= maximum conditional likelihood) fitting. Default by leastsquares.
- **softmax** switch for softmax (log-linear model) and maximum conditional likelihood fitting. *linout*, *entropy*, *softmax* and *censored* are mutually exclusive.
- **censored** A variant on softmax, in which non-zero targets mean possible classes. Thus for *softmax* a row of (0, 1, 1) means one example each of classes 2 and 3, but for censored it means one example whose class is only known to be 2 or 3.
- **skip** switch to add skip-layer connections from input to output.
- **rang** Initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(|x|) is about 1.
- **decay** parameter for weight decay. Default 0.
- **maxit** maximum number of iterations. Default 100.
- **Hess** If true, the Hessian of the measure of fit at the best set of weights found is returned as component Hessian.
- **trace** switch for tracing optimization. Default TRUE.
- **MaxNWts** The maximum allowable number of weights. There is no intrinsic limit in the code, but increasing MaxNWts will probably allow fits that are very slow and time-consuming.
- **abstol** Stop if the fit criterion falls below abstol, indicating an essentially perfect fit.
- **reltol** Stop if the optimizer is unable to reduce the fit criterion by a factor of at least 1 - reltol.
- **...** arguments passed to or from other methods.

Details

If the response in *formula* is a factor, an appropriate classification network is constructed; this has one output and entropy fit if the number of levels is two, and a number of outputs equal to the number of classes and a softmax output stage for more levels. If the response is not a factor, it is passed on unchanged to *nnet.default*.

Optimization is done via the *BFGS* method of optim.

Value

object of class *nnet* or *nnet.formula*. Mostly internal structure, but has components

- **wts** the best set of weights found.
- **value** value of fitting criterion plus weight decay term.
- **fitted.values** the fitted values for the training data.
- **residuals** the residuals for the training data.
- **convergence** 1 if the maximum number of iterations was reached, otherwise 0.

```
'''Utilizing Example'''

//use half the iris data
library(“nnet”)

ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])
targets <- class.ind( c(rep("s", 50), rep("c", 50), rep("v", 50)) )
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))

ir1 <- nnet(ir[samp,], targets[samp,], size = 2, rang = 0.1,
```

```
decay = 5e-4, maxit = 200)

test.cl <- function(true, pred) {
   true <- max.col(true)
   cres <- max.col(pred)
   table(true, cres)
}

test.cl(targets[-samp,], predict(ir1, ir[-samp,]))

// or
library(“nnet”)

ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
species = factor(c(rep("s",50), rep("c", 50), rep("v", 50))))

ir.nn2 <- nnet(species ~ ., data = ird, subset = samp, size = 2, rang = 0.1,
decay = 5e-4, maxit = 200)

table(ird$species[-samp], predict(ir.nn2, ird[-samp,], type = "class"))
```

Study Case

The case study is designed to illustrate just one among many possible applications of the package NNET.

Scenario

Accurate diagnosis may avoid complications for patients. Desiring to establish whether a patient has breast cancer, the analysis of several factors help determine an accurate diagnosis. Thus, from the collection of data from many patients seek to infer the diagnosis of patients with satisfactory accuracy.

Data Details

The data used in this case study are from a database of the UCI [13][4][5][6][7]. The database consists of 10 variables (9 input and 1 output) having 569 instances of records of patients diagnosed.

Input

- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- Marginal Adhesion
- Single Epithelial Cell Size
- Bare Nuclei
- Bland Chromatin
- Normal Nucleoli
- Mitoses

Output (Class)

- Diagnostic: Benign or Malignant

Execution and Results

The implementation using the package is pretty simple. Below is the part of the training and test data.

```
trainingInput <- read.table("trainingInput.data", sep=", ", header=TRUE)
trainingOutput <- read.table("trainingOutput.data", sep=",", header=TRUE)

library("nnet")
neuralNetworkModel <- nnet(trainingInput, trainingOutput, size = 19, rang = 0.1, decay = 5e-4, maxit = 2000)

neuralNetworkTest <- function(true, pred) {
      true <- max.col(true)
      cres <- max.col(pred)
      table(true, cres)
}

neuralNetworkTest(trainingOutput, predict(neuralNetworkModel,
trainingInput))
```

As a result of the training function *neuralNetoworkModel <- nnet (...)* has the steps of iteration and approximation of operations in accordance with the parameters set for *rang*, *decay* and *maxit*.

```
  # weights:  230
  initial  value 146.391298
  iter  10 value 14.225442
  iter  20 value 0.478782
  iter  30 value 0.149068
  iter  40 value 0.140717
  iter  50 value 0.131745
  iter  60 value 0.124368
  iter  70 value 0.116663
  ...
  iter 740 value 0.086414
  iter 750 value 0.086414
  final  value 0.086414
  converged
```

After training, there is confusion over the matrix (*function neuralNetworkTest*) the results obtained. From the matrix it is possible to apply various metrics for information as accuracy, error, among others.

```
      cres
  true  1   2
    1  180  0
    2   0  120
```

From this matrix, we find that the test was very successful, being that *True Positive = (1,1), False Positive = (1,2), False Negative = (2,1) and True Negative (2,2).*

Analysis

From this case study, the neural net model obtained from data sets of patients are sufficient to provide a reliable diagnosis. But for this, the patient data must respect the reality of the model represented by the network, otherwise the ANN present a misdiagnosis. Thus, it is noted that the package NNET is convenient to use, making it accessible to various distinct audiences, which may make use of it without needing a thorough knowledge on the subject.

Reference

1. ^ B. D. Ripley: "Pattern Recognition and Neural Networks", Cambridge, 1996.
2. ^ W. N. Venables and B. D. Ripley: "Modern Applied Statistics with S.", Fourth edition, Springer, 2002.
3. ^ O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.
4. ^ William H. Wolberg and O.L. Mangasarian: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193-9196.
5. ^ O. L. Mangasarian, R. Setiono, and W.H. Wolberg: "Pattern recognition via linear programming: Theory and application to medical diagnosis", in: "Large-scale numerical optimization", Thomas F. Coleman and Yuying Li, editors, SIAM Publications, Philadelphia 1990, pp 22-30.
6. ^ K. P. Bennett & O. L. Mangasarian: "Robust linear programming discrimination of two linearly inseparable sets", Optimization Methods and Software 1, 1992, 23-34 (Gordon & Breach Science Publishers).

References

[1] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet#endnote_Ripley96
[2] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet#endnote_Ripley02
[3] http://cran.r-project.org/web/packages/nnet/
[4] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet#endnote_Wolberg90a
[5] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet#endnote_Wolberg90b
[6] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet#endnote_Wolberg90c
[7] http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet#endnote_Mangasarian92

Article Sources and Contributors

Data Mining Algorithms In R *Source*: http://en.wikibooks.org/w/index.php?oldid=2579992 *Contributors*: Adrignola, Dcljr, Liam987, Milanez, PAC2, QuiteUnusual, Skorpio11, Tteixeira, Yanchangzhao, 6 anonymous edits

Dimensionality Reduction *Source*: http://en.wikibooks.org/w/index.php?oldid=1663410 *Contributors*: Adrignola, Odon, Tteixeira

Frequent Pattern Mining *Source*: http://en.wikibooks.org/w/index.php?oldid=1980737 *Contributors*: Adrignola, Ferre, Reifortes, Smarzaro, 1 anonymous edits

Sequence Mining *Source*: http://en.wikibooks.org/w/index.php?oldid=1685191 *Contributors*: Adrignola, Tteixeira, 1 anonymous edits

Clustering *Source*: http://en.wikibooks.org/w/index.php?oldid=1980741 *Contributors*: Adrignola, Augustofarnese, Brmonteiro, Cnosense, EduardoAlves, Emanuelvianna, Guisousa, Henriquerocha, Lgsd, Milanez, Rapha, Tcp, Torres, 8 anonymous edits

Classification *Source*: http://en.wikibooks.org/w/index.php?oldid=2000700 *Contributors*: Adrignola, Andrehora, Matheuscs, Rguidolini, Tteixeira, 3 anonymous edits

R Packages *Source*: http://en.wikibooks.org/w/index.php?oldid=1980787 *Contributors*: Dmztheone, Rapha, Torres, 15 anonymous edits

Principal Component Analysis *Source*: http://en.wikibooks.org/w/index.php?oldid=2643956 *Contributors*: Adrignola, Hmossri, Neil Smithline, QuiteUnusual, Tteixeira, Vitorcoliveira, 8 anonymous edits

Singular Value Decomposition *Source*: http://en.wikibooks.org/w/index.php?oldid=2644914 *Contributors*: Adrignola, Dcljr, Digitalbro, Michelgbr, Milanez, Neil Smithline, 6 anonymous edits

Feature Selection *Source*: http://en.wikibooks.org/w/index.php?oldid=2601175 *Contributors*: Adrignola, Avicennasis, Odon, 6 anonymous edits

The Eclat Algorithm *Source*: http://en.wikibooks.org/w/index.php?oldid=2619177 *Contributors*: Avicennasis, Denny73, Recent Runes, Smarzaro, 7 anonymous edits

arulesNBMiner *Source*: http://en.wikibooks.org/w/index.php?oldid=2611948 *Contributors*: Avicennasis, Bobmath, DM 00001, Ferre

The Apriori Algorithm *Source*: http://en.wikibooks.org/w/index.php?oldid=2064531 *Contributors*: Aciel, Alex.Szatmary, Altenmann, Applemeister, Avicennasis, BagpipingScotsman, Bask, Beefyt, Cobi, CommonsDelinker, DanMS, Davidgothberg, Ddddan, DeXXus, Desouky, Ederfmartins, Enochlau, Exa, Gigiiity goo, Kbh3rd, Kdau, Kotsiantis, Listener, Male1979, Martin Hinks, Maximus Rex, Michael Hardy, Pearle, Pgan002, Phreed, Piojo, QuiteUnusual, Redskin9, SB Johnny, Simeon, Sonett72, SueHay, Thepbac, Timo Honkasalo, VSEPR, 52 anonymous edits

The FP-Growth Algorithm *Source*: http://en.wikibooks.org/w/index.php?oldid=2646294 *Contributors*: Adrignola, Denny73, J36miles, QuiteUnusual, Reifortes, 10 anonymous edits

SPADE *Source*: http://en.wikibooks.org/w/index.php?oldid=2422186 *Contributors*: Adrignola, Avicennasis, Hagalmir, Osvaldojr, Tteixeira, 6 anonymous edits

DEGSeq *Source*: http://en.wikibooks.org/w/index.php?oldid=2064542 *Contributors*: Avicennasis, 4 anonymous edits

K-Means *Source*: http://en.wikibooks.org/w/index.php?oldid=2579441 *Contributors*: Adrignola, Avicennasis, Lucmir, Matheusv, Panic2k4, Paul2520, 11 anonymous edits

Hybrid Hierarchical Clustering *Source*: http://en.wikibooks.org/w/index.php?oldid=2679402 *Contributors*: Henriquerocha, Panic2k4, Xania, 7 anonymous edits

Expectation Maximization (EM) *Source*: http://en.wikibooks.org/w/index.php?oldid=2680774 *Contributors*: Adrignola, Avicennasis, Chriteixeira, CommonsDelinker, Emanuelvianna, Hagindaz, Panic2k4, QuiteUnusual, Sesanker, 9 anonymous edits

Dissimilarity Matrix Calculation *Source*: http://en.wikibooks.org/w/index.php?oldid=2246125 *Contributors*: Adrignola, Guisousa, Igor.rafael, Panic2k4, 1 anonymous edits

Hierarchical Clustering *Source*: http://en.wikibooks.org/w/index.php?oldid=2535596 *Contributors*: Avicennasis, Brmonteiro, Jichi, Jomegat, Smarzaro, 5 anonymous edits

Density-Based Clustering *Source*: http://en.wikibooks.org/w/index.php?oldid=2665418 *Contributors*: Adrignola, Avicennasis, Duilio, Mohamed cherif dani, QuiteUnusual, Xania, 77 anonymous edits

K-Cores *Source*: http://en.wikibooks.org/w/index.php?oldid=2676532 *Contributors*: Adrignola, Augustofarnese, Avicennasis, Sinaide.nb, 2 anonymous edits

Fuzzy Clustering - Fuzzy C-means *Source*: http://en.wikibooks.org/w/index.php?oldid=2623028 *Contributors*: Adrignola, Avicennasis, Bobmath, Carlajmachado, CommonsDelinker, J36miles, Rfalexandre, 15 anonymous edits

RockCluster *Source*: http://en.wikibooks.org/w/index.php?oldid=2609005 *Contributors*: Arthurvogel, Rapha, Rapha2, 7 anonymous edits

Biclust *Source*: http://en.wikibooks.org/w/index.php?oldid=2124297 *Contributors*: Fnielsen, Liptorres, Torres

Partitioning Around Medoids (PAM) *Source*: http://en.wikibooks.org/w/index.php?oldid=2665960 *Contributors*: Avicennasis, GongYi, Hgrandrade, J36miles, 5 anonymous edits

CLUES *Source*: http://en.wikibooks.org/w/index.php?oldid=2501611 *Contributors*: Avicennasis, Tcp, 1 anonymous edits

Self-Organizing Maps (SOM) *Source*: http://en.wikibooks.org/w/index.php?oldid=2262166 *Contributors*: Adrignola, Felipejfj, 22 anonymous edits

Proximus *Source*: http://en.wikibooks.org/w/index.php?oldid=2674317 *Contributors*: Adrignola, Avicennasis, CommonsDelinker, Lgsd, 3 anonymous edits

CLARA *Source*: http://en.wikibooks.org/w/index.php?oldid=2537246 *Contributors*: Adrignola, Avicennasis, Cnosense, CommonsDelinker, QuiteUnusual, 8 anonymous edits

SVM *Source*: http://en.wikibooks.org/w/index.php?oldid=2496635 *Contributors*: Adrignola, Avicennasis, Elisaboari, Gabrielgmendonca, Mirlaine, 11 anonymous edits

penalizedSVM *Source*: http://en.wikibooks.org/w/index.php?oldid=2480994 *Contributors*: Avicennasis, Bender2k14, Jahobr, Lvsm, 3 anonymous edits

kNN *Source*: http://en.wikibooks.org/w/index.php?oldid=2581016 *Contributors*: Adrignola, Ansa211, Pennachin, Tteixeira, Whym, 9 anonymous edits

Outliers *Source*: http://en.wikibooks.org/w/index.php?oldid=2064515 *Contributors*: Adrignola, Avicennasis, Rguidolini, 1 anonymous edits

Decision Trees *Source*: http://en.wikibooks.org/w/index.php?oldid=2627450 *Contributors*: Adrignola, Andrechalom, Andrehora, Luoli2000, Tcarnus, Ziltonjr, 6 anonymous edits

Naïve Bayes *Source*: http://en.wikibooks.org/w/index.php?oldid=2654643 *Contributors*: Luoli2000, Palotti, 2 anonymous edits

adaboost *Source*: http://en.wikibooks.org/w/index.php?oldid=2655175 *Contributors*: Adrignola, Avicennasis, Glauberpmm, 8 anonymous edits

JRip *Source*: http://en.wikibooks.org/w/index.php?oldid=2281845 *Contributors*: Adrignola, Avicennasis, Matheuscs, 2 anonymous edits

RWeka *Source*: http://en.wikibooks.org/w/index.php?oldid=1977625 *Contributors*: Adrignola, 19 anonymous edits

gausspred *Source*: http://en.wikibooks.org/w/index.php?oldid=1977643 *Contributors*: 1 anonymous edits

optimsimplex *Source*: http://en.wikibooks.org/w/index.php?oldid=1979780 *Contributors*: 2 anonymous edits

CCMtools *Source*: http://en.wikibooks.org/w/index.php?oldid=1980287 *Contributors*: 1 anonymous edits

FactoMineR *Source*: http://en.wikibooks.org/w/index.php?oldid=2064533 *Contributors*: Adrignola, Avicennasis, Gureis, 3 anonymous edits

nnet *Source*: http://en.wikibooks.org/w/index.php?oldid=2604815 *Contributors*: Adrignola, Avicennasis, Dmztheone, 1 anonymous edits

Image Sources, Licenses and Contributors

File:Pca_plot.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Pca_plot.jpg *License*: Creative Commons Attribution-Sharealike 3.0,2.5,2.0,1.0 *Contributors*: Adrignola, Hmossri

File:Tux_grey.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Tux_grey.png *License*: Creative Commons Attribution 3.0 *Contributors*: Michelgbr

File:Tux_1.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Tux_1.png *License*: Creative Commons Attribution 3.0 *Contributors*: Michelgbr

File:Tux_35.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Tux_35.png *License*: Creative Commons Attribution 3.0 *Contributors*: Michelgbr

File:ex-visualization1.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Ex-visualization1.jpg *License*: unknown *Contributors*: Smarzaro

File:FPG_FIG_01.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_01.jpg *License*: GNU General Public License *Contributors*: Reinaldo

File:FPG_FIG_02A.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_02A.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG_FIG_02B.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_02B.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG_FIG_02C.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_02C.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG_FIG_02D.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_02D.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG_FIG_02E.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_02E.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG_FIG_02F.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_02F.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG FIG 03.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_03.jpg *License*: GNU General Public License *Contributors*: BrightRaven, Reifortes

File:FPG_FIG_04.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_04.jpg *License*: GNU General Public License *Contributors*: Michael Barera, Reifortes

File:FPG_FIG_05.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:FPG_FIG_05.jpg *License*: GNU General Public License *Contributors*: Marek Mazurkiewicz, Reifortes

File:Algorithm_kmeans.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Algorithm_kmeans.png *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: Lucas Cunha

File:kmeans_plotting.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Kmeans_plotting.png *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: Lucas Cunha

File:kmeans_plotting2.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Kmeans_plotting2.png *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: Lucas Cunha

Image:agglomerative_clustering_dendogram.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Agglomerative_clustering_dendogram.png *License*: Public Domain *Contributors*: Henriquerocha

Image:hybrid_clustering_case_study_dendogram.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Hybrid_clustering_case_study_dendogram.png *License*: Public Domain *Contributors*: Henriquerocha

File:basic bic.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Basic_bic.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:basic cluster.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Basic_cluster.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:basic uncertainty.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Basic_uncertainty.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:basic density.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Basic_density.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:points random.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Points_random.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:gaussian points.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Gaussian_points.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:gaussian bic.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Gaussian_bic.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:gaussian cluster.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Gaussian_cluster.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:default bic.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Default_bic.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:customize bic.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Customize_bic.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:cluster customize.pdf *Source*: http://en.wikibooks.org/w/index.php?title=File:Cluster_customize.pdf *License*: GNU Free Documentation License *Contributors*: Emanuelvianna

File:Hierarchical_Clustering_Italy.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Hierarchical_Clustering_Italy.jpg *License*: Public Domain *Contributors*: Original uploader was Smarzaro at en.wikibooks

File:Dendograma.JPG *Source*: http://en.wikibooks.org/w/index.php?title=File:Dendograma.JPG *License*: unknown *Contributors*: Smarzaro

Image:Center of a galaxy.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Center_of_a_galaxy.PNG *License*: Public Domain *Contributors*: T. Joseph W. Lazio. Original uploader was Matheusv at en.wikibooks

Image:Pre Processed Astronomical Image.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Pre_Processed_Astronomical_Image.PNG *License*: Public Domain *Contributors*: Duílio Campos Sasdelli. Original uploader was Matheusv at en.wikibooks

Image:ClusteringResults.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:ClusteringResults.PNG *License*: Public Domain *Contributors*: Duílio Campos Sasdelli. Original uploader was Matheusv at en.wikibooks

File:GraficoPertinencia.gif *Source*: http://en.wikibooks.org/w/index.php?title=File:GraficoPertinencia.gif *License*: GNU General Public License *Contributors*: Rafael Alexandre

File:GraficoComparativo.gif *Source*: http://en.wikibooks.org/w/index.php?title=File:GraficoComparativo.gif *License*: GNU General Public License *Contributors*: Rafael Alexandre

File:rock1.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Rock1.png *License*: GNU General Public License *Contributors*: Sudipto Guha

File:rock2.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Rock2.png *License*: GNU General Public License *Contributors*: Sudipto Guha

File:rock3.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Rock3.png *License*: GNU General Public License *Contributors*: Sudipto Guha

File:rock4.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Rock4.png *License*: GNU General Public License *Contributors*: rock

File:rock5.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Rock5.png *License*: GNU General Public License *Contributors*: me

File:rock6.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Rock6.png *License*: GNU General Public License *Contributors*: Rapha2, TFCforever

File:Siduiee.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Siduiee.png *License*: GNU General Public License *Contributors*: Funfood, Liptorres

File:Saoidjfi99.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Saoidjfi99.png *License*: GNU General Public License *Contributors*: Funfood, Liptorres

File:Ajosiu99.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Ajosiu99.png *License*: GNU General Public License *Contributors*: Hedwig in Washington, Liptorres

File:Aksiud80.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Aksiud80.png *License*: GNU General Public License *Contributors*: Liptorres, Michael Barera

File:Hgrandrade example.jpeg *Source*: http://en.wikibooks.org/w/index.php?title=File:Hgrandrade_example.jpeg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Hgrandrade

File:HgrandradeIris.jpeg *Source*: http://en.wikibooks.org/w/index.php?title=File:HgrandradeIris.jpeg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Hgrandrade

File:som neural network.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Som_neural_network.png *License*: Free Art License *Contributors*: Pang - Ning Tan, Michael Steinbach and Vipin Kumar

File:print wine.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Print_wine.png *License*: GNU General Public License *Contributors*: The R Fondaution for Statistical Group

File:summary wine.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Summary_wine.png *License*: unknown *Contributors*: The R Foundation for Statistical Compinting

File:Plot NIR data.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Plot_NIR_data.png *License*: GNU General Public License *Contributors*: The R Foundation for Statistical Computing

File:Plot predict water.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Plot_predict_water.png *License*: GNU General Public License *Contributors*: The R Foundation for Statistical Computing

File:Plot Codes.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Plot_Codes.png *License*: GNU General Public License *Contributors*: The R Foundation for Statistical Computing

File:Plot Mapping All.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Plot_Mapping_All.png *License*: GNU General Public License *Contributors*: The R Foundation for Statistical Computing

File:Plot Training Process.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Plot_Training_Process.png *License*: GNU General Public License *Contributors*: The R Foundation for Statistical Computing

File:Plot wine data.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Plot_wine_data.png *License*: GNU General Public License *Contributors*: The R Foundation for Statistical Computing

File:Proximus2.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Proximus2.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Lgsd

File:Proximus3.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Proximus3.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Lgsd

File:proximus4.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Proximus4.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Lgsd

File:SVM Example of Hyperplanes.png *Source*: http://en.wikibooks.org/w/index.php?title=File:SVM_Example_of_Hyperplanes.png *License*: Public Domain *Contributors*: Elisaboari

File:SvmPlot.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:SvmPlot.jpg *License*: Public Domain *Contributors*: Mirlaine Aparecida Crepalde/Elisa Boari

File:penalizedSVM6.png *Source*: http://en.wikibooks.org/w/index.php?title=File:PenalizedSVM6.png *License*: Public Domain *Contributors*: Lvsm

Image:Iris_versicolor_3.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Iris_versicolor_3.jpg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Dbenbenn, Dlanglois, Quadell

Image:Exemplo1_1.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Exemplo1_1.jpg *License*: Public Domain *Contributors*: Rickson, Wallace

Image:Rguidolini_exemplo1_2_r_outliers.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Rguidolini_exemplo1_2_r_outliers.jpg *License*: Public Domain *Contributors*: Rickson, Wallace

Image:outliers_car_data.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Outliers_car_data.jpg *License*: Public Domain *Contributors*: Adrignola, Rguidolini, 2 anonymous edits

Image:Cars.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Cars.jpg *License*: Public Domain *Contributors*: Adrignola, Rguidolini, 2 anonymous edits

Image:outliers_info.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Outliers_info.jpg *License*: Public Domain *Contributors*: Adrignola, Rguidolini, 2 anonymous edits

Image:Cadillac_Fleetwood_2.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Cadillac_Fleetwood_2.jpg *License*: Public Domain *Contributors*: Adrignola, Rguidolini, 7 anonymous edits

File:VariableImportancePlot.png *Source*: http://en.wikibooks.org/w/index.php?title=File:VariableImportancePlot.png *License*: Public Domain *Contributors*: Glauberpmm

File:Example.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Example.jpg *License*: Public Domain *Contributors*: bdk

File:Principal-correspondence-analysis-ind-small.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Principal-correspondence-analysis-ind-small.png *License*: GNU General Public License *Contributors*: Gureis, McZusatz, Michael Barera

File:Coord-ellipses.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Coord-ellipses.PNG *License*: GNU General Public License *Contributors*: Gureis, Hedwig in Washington, McZusatz

File:Principal-component-analysis-activar.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Principal-component-analysis-activar.PNG *License*: GNU General Public License *Contributors*: Gureis, Michael Barera

File:Women work.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Women_work.PNG *License*: GNU General Public License *Contributors*: Gureis, Michael Barera

File:Correspondence-analysis-columns.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Correspondence-analysis-columns.PNG *License*: GNU General Public License *Contributors*: factominer.free.fr

File:Correspondence-analysis-rows.PNG *Source*: http://en.wikibooks.org/w/index.php?title=File:Correspondence-analysis-rows.PNG *License*: GNU General Public License *Contributors*: factominer.free.fr

License