WS2008 AST

Cluster Analysis Übung

13.11.2008 due date: 27.11.2008

1 Goals

For this Übung, you should:

- familiarize yourself with some of the common functions of the statistics package "R"
- employ and understand some of the clustering methods
- and submit a very brief report (questions in section 4)

2 Using the "R" package:

"R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files." – R project page

First of all, start the R application:

```
/usr/local/zbh/bin/R
```

This should present you with a command prompt. Invoking any denominator will print out information relevant to that name. In case of variables, output type and associated values will be displayed, whereas in case of functions it will print the source code (if available).

The `ls()` command lists all variables that exist in the current session. Note that functions and constants defined by libraries are not listed. `help(foo)` will bring up more detailed information about the function or data type `foo`. Quit the help screen with the standard UNIX pager keybinding "q".

It is not necessary to declare or explicitly associate a type with a variable, instead you simply assign a value (which may be a complex construct) to a name. The assignment operator is "`<-`".

To assign data types other than numbers and strings, you will need to use constructors. The frequently used data type vector is created with the "c" command(combine). Its argument is a comma-separated list of the vectors contents. E.g.:

```
> foo <- c(1, 2, 3)
> foo
[1] 1 2 3
```

The first output is always the row number of the element. This is of course irrelevant for vectors, but allows for consistent treatment of matrices with only one row. The contents of vectors can be accessed by using the vector name with the index in square brackets. NOTE: Contrary to many programming languages, the starting index for the elements is "1". Index "0" denotes the type of the vector.

```
> foo[0]
numeric(0)
> foo[1]
[1] 1
```

Matrices are constructed in a similar manner. The matrix constructor will accept a data vector and optional specifications, especially the number of rows or columns. The matrix is filled columnwise, unless the parameter `byrow` is set to `TRUE`. If there are not enough data points to fill the matrix, the first data points are reused, so beware.

```
> bar <- matrix(c(1,2,3,4,5,6), nrow=3)
> bar
      [,1]  [,2]
 [1,]   1     4
 [2,]   2     5
 [3,]   3     6
```

As with vectors, you can access matrix elements using square brackets, but unlike C arrays, all indices are placed as a comma separated list in the first (and only) pair of square brackets. If an index is negative, the row or column corresponding to that (positive) index will be omitted.

```
> bar[3,2]
[1] 6
```

```
> bar[-3,2]
[1] 4 5
```

Simply leaving out a parameter selects the entire column or row:

```
> bar[,2]
[1] 4 5 6
```

To concatenate matrices, you can use the `rbind` and `cbind` commands, which will try to append matrices and vectors row and columnwise, respectively.

```
> rbind(bar,c(3.5,6.5))

       [,1]  [,2]
  [1,]   1     4
  [2,]   2     5
  [3,]   3     6
  [4,]  3.5   6.5
```

```
> cbind(bar, c(7,8,9))

       [,1]  [,2]  [,3]
  [1,]   1     4     7
  [2,]   2     5     8
  [3,]   3     6     9
```

As with the matrix constructor, if a vector does not contain enough elements, the first elements will be replicated at the missing positions. This does not hold true for matrices, though. Concatenating incompatible matrices will produce an error message.

3 Exercises

Open the data file and assign the data to a variable:

```
> cluster.datapath <- file.path('/home/reuter/uebung_clustering/sampledata')
> cluster.data <- as.matrix(read.table(cluster.datapath, sep=','))
```

Check the dimensionality of your data:

```
> dim(cluster.data)
```

This should show you a row stating 100 entries of dimensionality 2.

```
[1] 100 2
```

You can also plot your data with the `plot` command:

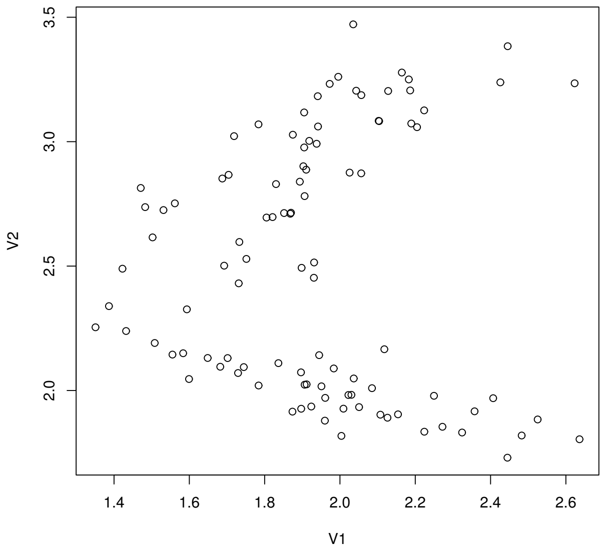

Figure 1: Plot of the test data set

```
plot(cluster.data)
```

(See figure 1)

You will be using two different clustering methods, k-means and hierarchical clustering.
If you are unsure about the workings, or simply curious, read the help documentation for the kmeans and hclust functions.

```
> help(kmeans)
> help(hclust)
```

3.1 K-means

Do several runs with k-means clustering, and use a different number of clusters as parameter:

```
> cluster.kmeans <- kmeans(cluster.data,<number of clusters>)
```

You can visualize them with:

```
> plot( cluster.data, col = cluster.kmeans$cluster)
> points(cluster.kmeans$centers, col = 1:6, pch=7, lwd=3)
```

The contents of the data structure can be inspected with (`> cluster.kmeans`). As you can see, the cluster subset is a vector of integers, each denoting the cluster membership. This cluster membership is used in the plot command to select the color (the $ symbol selects this substructure by name).

Repeat the clustering twice, but save both results in different variables, `cluster.kmeans.1` and `cluster.kmeans.2`. To compare these two, create a table which denotes how often each element is matched in both vectors. (also read `> help(table)`):

```
> table(cluster.kmeans.1$cluster, cluster.kmeans.2$cluster)
```

The diagonal entries denote how often a data point was assigned to the same cluster in both runs, while the non-diagonal entries denote how often the cluster index didn't match. Unfortunately, since the cluster centers are randomly initialized, even if the clustering is identical, the cluster numbering may be different. If we want to calculate the hit rate for a given k-means-pairing we will have to first find a correct matching of clusters and then count the number of mismatches.

```
cluster.mismatchrate <- function(x)
{
    miss <- 0  # initialize
    for(i in 1:nrow(x)) # look at all rows
    {
        maxindex = which.max(as.vector(x[i,])) # find matching clusters
        miss <- miss+sum(x[i,-maxindex]) # sum over row excluding match
    }
    miss <- miss/sum(x) # normalize
    miss # output
}
```

This quick makeshift function will take a confusion table as described above and calculate the rate of mismatches (obviously, the function will break down for ugly datasets where clusters overlap only partially, in which case a much more sophisticated matching algorithm must be used). Play around with the different commands used in the function until you feel comfortable using them.

3.2 Hierarchical clustering

Next, consider the hierarchical clustering algorithm included in R. The hierarchical clustering operates on a precomputed set of distance information, so we will have to provide that first. Fortunately, a simple method to generate a distance matrix is implemented in R.

```
> cluster.dist <- dist(cluster.data,method='euclidean')
```

The resulting data structure is a triangular matrix with the bottom half (i,j) entries containing the distance between data points i and j. On this data structure, the hierarchical clustering algorithm can be run.

```
cluster.hclust <- hclust(cluster.dist)
```

The output is a list of the order in which adjacent clusters are joined, with each data point being its own cluster at initialization. This output can be plotted (`> plot(cluster.hclust)`) as a *dendrogram*. To compare this to the k-means clustering, the output has to be reduced to a number of clusters. This can be achieved by using the `cutree` method, which produces the n latest clusters to be joined.

```
cluster.hcut <- cutree(cluster.hclust,<number of clusters>)
```

The `cluster.hcut` vector contains the same type of information as the previously used `cluster.kmeans$cluster`, and can be compared with the same methods.

4 Assignment

Please submit a brief report for the following questions via email: reuter[at]zbh[dot]uni-hamburg[dot]de (Please include your full name. A notification will be sent back to you after receiving your assignment.)

1. Write a batch function to run and compare 500 pairs of k-means clustering and calculate the average mismatch rate. Where and why do they differ(if they do)? Is this consistent with your expectations? Elaborate.
2. Compare 500 pairs of k-means and hierarchical clustering. Is there a systematic difference between the clustering results? Explain your observations.
3. You may add up to 15 data points to the data set (do not use outlandishly large values, stay within the general area of the existing data set). Can you add them in a way that breaks the hclust method? Where did you add them and why? Do they affect the k-means clustering?