Wintersemester 2015/2016, AST

Cluster Analysis Übung

Assignment due date: 16.11.2015
Exercises on 2.11.2015 and 9.11.2015

1 Goals

For this Übung, you should:

- familiarise 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" system:

"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, `http://www.r-project.org/`

First of all we create a directory to work in. The name of the directory is just a suggestion, feel free to choose any name you like. Type in the following commands from a shell prompt:

```
mkdir ast-cluster
cd ast-cluster
```

We start R by running the following command from a shell prompt:

```
R
```

You should now see the R command prompt. Remember that you can save yourself a lot of typing by using the up and down arrows of the keyboard to cycle through previously entered commands and the tab key to automatically complete commands (just like you can at the shell prompt).

To store a value in a variable, we use the assignment operator "<-":

```
> answer <- 42
```

You may also type "=" instead of "<-".

We can inspect the value of the variable `answer` by simply typing its name at the command prompt:

```
> answer
[1] 42
```

The first part of the output ("[1]") indicates that this is the first row. This is of course irrelevant for simple numbers or vectors but will become useful later on when we deal with matrices. Use the function `ls()` to get a list of all variables in the current session. When you type in the name of a function without the parentheses and arguments, the function definition will be printed. Try this out by entering `ls`, but don't worry if you don't understand the code.

To create a vector, we will use the "c" function (an abbreviation of combine):

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

The contents of vectors can be accessed by using the vector name with the index in square brackets. IMPORTANT: in contrast to many programming languages, the index of the first element of a vector is "1". Accessing index "0" of any variable gives you its type.

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

Matrices are constructed in a similar manner. The matrix constructor accepts a data vector and optional specifications, such as the number of rows or columns. The matrix is filled columnwise, unless the parameter `byrow` is set to `TRUE`. Please beware that if there are not enough entries in the data vector to fill the matrix, the entries of the data vector will be reused instead of raising an error message.

```
> 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.

2.1 Storing your R commands in a script file

Often it is more convenient to be able to store the R commands you want executed in a script file and run them from there. You can create the script file for R with your favourite text editor (if you don't have a favourite text editor you can use the program `kate`). The following command (*to be executed from the shell prompt, not the R prompt*) would run the R commands stored in the file `aufgabe1.r`:

```
Rscript aufgabe1.r
```

This command will not work yet unless you have created a file called `aufgabe1.r`. As a reminder, if you want to store the output of R that was printed to the screen in the file `out-aufgabe1.txt` you would use:

```
Rscript aufgabe1.r > out-aufgabe1.txt
```

Please note that simply typing in the name of a variable at the interactive prompt will print its value, but to do the same in a script you will have to actually call the print function on the variable. For example, to print the variable `x` you would call `print(x)`.

It is also possible to read in R commands stored in a file from the interactive prompt with the function `source`, i.e. running `source('foo.r')` will read and execute everything stored in the file `foo.r` as if it had been typed into the command prompt.

HINT: During this exercise, you will have to type in longer function definitions. It is advisable to store these function definitions in a text file so you can edit them more easily. For the homework assignment, you will have to submit your code so that it is runnable with `Rscript`.

2.2 Getting help

Every function in R is thoroughly documented. You can access the documentation to a command with:

```
> help(plot)
```

This can be abbreviated to:

```
> ?plot
```

You can search inside the documentation with:

```
> ??plot
```

3 Exercises

Open the data file and assign the data to a variable (storing it as a matrix):

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

Check the dimensionality of your data:

```
> dim(cluster.data)
[1] 100 2
```

This tells you that your data is a matrix with 100 rows and 2 columns. You can also plot your data with the `plot` command (see figure 1 for the result):

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

To store the plot to the file `myplot.png` use the following commands:

```
> png('myplot.png')
> plot(cluster.data)
> dev.off()
```

The final `dev.off()` command finalises the plot and writes out the file. The next plot will again be shown on your graphical display.

You will be using two different clustering methods, k-means and hierarchical clustering. This can be done in R with the help of the `kmeans` and `hclust` functions. If you are unsure about the correct usage of these functions or simply curious, you can read their help documentation:

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

3.1 K-means clustering

Do several runs with k-means clustering, and use a different number of clusters as parameter for each run (replace `<number of clusters>` with the number of clusters in the following command):

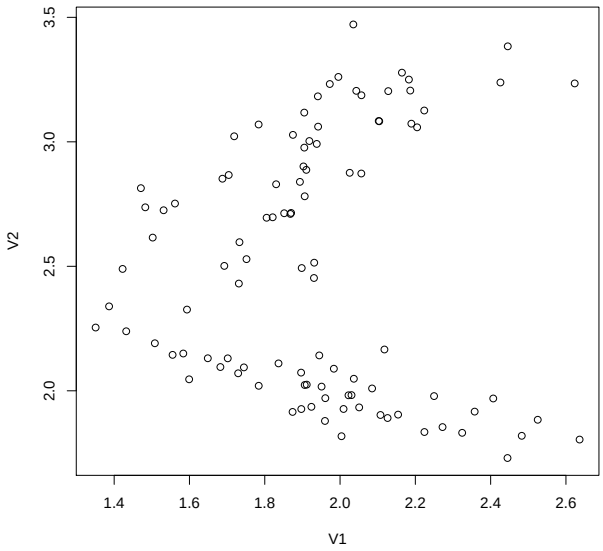

Figure 1: Plot of the test data set

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

You can visualise them with the following commands (don't close the plot window before running the second command):

```
> 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 by entering `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 colour (the `$` symbol selects this substructure by name).

Repeat the clustering twice with the same number of clusters, but save both results into 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)`):

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

The diagonal entries denote how often a data point was assigned to the same cluster in both runs, while the off-diagonal entries denote how often the cluster index didn't match. This table

is therefore called a confusion matrix. Unfortunately, since the cluster centres are randomly initialised, the cluster numbering may be different even if the clustering is identical. If we want to calculate the rate of mismatches for a given k-means pairing, we will have to first find a correct matching of clusters and then count the number of mismatches. Note that the function may be split over two pages in this handout.

```
cluster.mismatchrate <- function(x)
{
    miss <- 0                # initialise
    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)      # normalise
    miss                     # return result
}
```

The "#" symbol marks a comment that extends to the end of the line. This short function is adequate for well-behaved data sets and is the one we will use in the following. Play around with the different commands used in the function until you feel comfortable using them.

We use the function by passing it the previously stored table `tab` like so:

```
> cluster.mismatchrate(tab)
```

3.2 Hierarchical clustering

The hierarchical clustering algorithm in R needs a precomputed distance matrix, which contains the pairwise distances between data points. Fortunately, a function to generate a distance matrix is already implemented in R.

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

The resulting data structure is a lower triangular matrix, where the entry at (i,j) contains the distance between data points i and j. We can now run the hierarchical clustering algorithm:

```
> 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 initialisation. This output can be plotted as a *dendrogram* by running

`plot(cluster.hclust)`. 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 last 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: ast_uebung[at]zbh[dot]uni-hamburg[dot]de (please include your full name). Attach one file containing the written answers to all the questions in the assignment (only .txt or .pdf formats please), and one file per question with source code. Your source code should be runnable with `Rscript`.

1. Write a batch function to run and compare 500 pairs of k-means clusterings 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 clusterings. Is there a systematic difference between the clustering results? Explain your observations.

3. What is the linkage method used in the hierarchical clustering and how does it work? Does the clustering change if you use a different linkage method? Start by reading the R help information on the `hclust` method.

4. You may add up to 5 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, i.e. can you add the points in a way that will completely change the clustering? Where did you add them and why? Do they affect the k-means clustering?