DME Lab 1: Data visualisation in R

1 February, 2012

Hello! As the title suggests this lab is devoted to data visualisation in R. R is a powerful statistical software that can be used for data analysis. Among its subtleties we can find a programming language and several graphical facilities for data manipulation and visualisation. It also can be used to perform data mining as it implements the most common tools for classification and clustering.

As with any other computer tool for statistical analysis, it takes time to master R, and naturally this is beyond the scope of the course. However, the intention of these series of labs is that you become aware of the tools that can help you in your final project and -why not?- in your future professional work! In this lab you will learn the basics of this software: you will learn how to load data, manipulate it, and visualise it.

If you are interested in learning more about R, you should take a look at the following links:

- The R-Project website: `http://www.r-project.org/`
- The R-For-Beginners manual: `http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf`
- The R-For-Matlab users page: `http://mathesaurus.sourceforge.net/octave-r.html`
- The R-For-Programmers page: `http://www.johndcook.com/R_language_for_programmers.html`
- And the R-For-DataMining website: `http://www.rdatamining.com/`

Now, let's start with the lab. First of all you should download the data to your computer by following this link:

```
http://www.inf.ed.ac.uk/teaching/courses/dme/2012/labs/prostate.zip
```

We will work with the *Prostate Cancer* dataset presented in the book by Hastie, Tibshirani and Friedman. This dataset consists of 97 observations obtained from medical analysis on patients who were to receive treatment against prostate cancer. The task in this dataset is to predict the outcome of the log of prostate specific antigen (`lpsa`) from another eight measurements including log cancer volume (`lcavol`), that is, we are dealing with a *regression* task.

If you want more information about this study and the features included in the dataset, take a look to the book by Hastie, Tibshirani and Friedman, or follow this link: `http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.info`

After you have downloaded the data, proceed with the following:

1. Open the R environment by typing `R` in the command line of a DICE machine.

2. One command you need to learn straight away is the *help* command, just type `help()` in the console to read its description. This command is helpful when you want to know more about certain function. For example, by typing `help(mean)` you can learn the parameters of the function `mean()` implemented in R.

   A shortcut for the *help* function would be to type `?` followed by the function that you want to know about. For example, `?hist` yields the same result as `help(hist)`. Now, if you don't remember the name of a function in R or you are not sure if it's implemented in it, then you can type `??` followed by a keyword to that function. For example, type `??pca` and this will open a window showing the related functions to principal component analysis.

   Now, type `help.start()`. This will open a window in the web browser showing the online help for R. This might become useful sometime!

3. Ok. Now that you have the data in your computer, you must load it into R. And you will achieve this by using the function `read.table`. You can learn more about this function by typing `help(read.table)`. In summary, you will type something like

   ```
   data<-read.table("prostate.csv",header=TRUE,sep=";")
   ```

   to load the data. With this command we're telling R to load the data into a *data frame* known from now on as `data`, and we're also telling R that the file `prostate.csv` contains a row specying the name of each column, hence the parameter `header=TRUE`. Finally, the parameter `sep=";"` tells R that each column in the file is separated by "`;`".

   R implements functions to read from different kinds of data files. It can read text files, csv files, xls files, spss files and even ARFF files like the ones used in Weka. To know more about this you can type `??read`.

   Wait. The command that you typed above assumes that the data file is located in the working directory of R, which can be found with the `getwd()` function. You can select a different working directory with the `setwd()` function, and avoid entering the full path of the data files.

4. You can take a look at the data loaded into the data frame by typing its name, in this case you would type `data`. As you can see, the data consists of 97 observations and 10 attributes named: *lcavol*, *lweight*, *age*, *lbph*, *svi*, *lcp*, *gleason*, *pgg45*, *lpsa* and *train*. You can get a simple summary of your data by typing `str(data)`, which returns some details about the structure of a given object.

   If you have worked with Matlab before, then you must be familiar with the notion of a *workspace*. The workspace in R consists of all the variables and objects that you use during a session. To know what objects are available in your workspace you can type `ls()`. To remove an object from the workspace you can use the function `rm()`, e.g. `rm(data)`, but don't type this now!

5. Now that the data is in the workspace, we are able to manipulate it as if it was a simple array. If you typed something like `data[1,]`, you'd get the first row from the data. Obviously, the line `data[0,]` would give us the column names. The line `data[1:10,]` would return the first

ten lines of data, whereas `data[,9]` would return all the contents in the ninth column, that is, the target variable `lpsa`. We could have obtained the same results if we typed `data$lpsa`, i.e. referring to the column `lpsa` from the object `data` by using the symbol `$`.

6. As you can see, the data contains a column called `train`, whose value is either true or false. The purpose of this column is to indicate whether the observation was used in the training stage or in the testing stage of the regression task. As we don't care about this distinction, we are going to get rid of this column. To do this we will simply read the contents of the array from the first column to the ninth column and assign it to a different array (or to the same one). In other words, we will type: `data<-data[,1:9]`.

7. Now let's get some statistics about our data. If we want to know the minimum and the maximum values of the attribute `age`, we can write `min(data$age)` and `max(data$age)`, respectively.

   Let's do something before we proceed analysing statistically our data. Type `attach(data)`. This command tells R to make available the column names of the object as if they were any other variable in the workspace. However, if you type `ls()` you won't see them in the workspace. To know why, you could read the description of the function in `help(attach)`, but it doesn't matter for now.

   Ok, now try `min(age)` and `max(age)`.

8. Continue inspecting the statistics of the data by typing `mean(age)`, `median(age)`. In class we saw that the median is a more robust value than the mean. Do you remember why?

   Now, take a look to the standard deviation of this variable by doing `sd(age)`, or its variance by `var(age)`. Remember that the stardard deviation shows how disperse the data is from its average. A low standard deviation indicates that the data points tend to be very close to the mean, whereas high standard deviation indicates that the data points are spread out over a large range of values.

9. Let's take a look to a histogram of this variable and compare it with our numerical results. Type `hist(age)` in the command line. Remember that a histogram shows how frequent a value is for a certain variable. The last command opens a window in which we can see a histogram for `age`. Can you locate the mean and the median? Can you tell which is the value that is more frequent?

   Also remember that a histogram is sensitive to our selection of bins. Do the results from `hist(age, breaks = 20)` look the same as your first plot? Are you able to see the most frequent value of this variable? If not, try this: `dummy<-data.frame(table(age))` and then type `dummy`. This is a very rudimentary version of our histogram. (Don't forget that you can always use `help()` to learn about a function!)

10. Let's focus now on the target variable `lpsa`. You can perform the same statistics as above by just typing `summary(lpsa)`. This function will also give you information regarding the first and third quantiles. Next, we can even try something more sophisticated by doing `bins=seq(-1,6,by=0.25)`, which creates an array of numeric values from -1 to 6 taking steps of one quarter. Right after this type `hist(lpsa,breaks=bins)`. Do you see any outliers? If so, how frequent they are?

11. Next, we will see the frequency of the values in a variable conditioned on the values of another variable. This is known as a *conditional histogram.* This kind of plot is very helpful in classification tasks as it allows us to see what is the distribution of a variable conditioned on the classes of the dataset. In our data we don't have any kind of class, however we will perform a conditional histogram conditioned in a categorical attribute.

    There are two categorical attributes in the dataset: `gleason` and `svi`. If you'd like to see what are the values they take, you could type: `data.frame(table(gleason))`.

    To plot a conditional histogram we will need to load a package into R. The functionality of R is implemented in its packages, these are like the different libraries of programming languages such as C. To see a list of packages in your R distribution type `library()`. And, if you happened to use the `help()` command at some point of this lab, then you should have noticed that the description of any function also includes the information about the package it is included in.

    The function we need is called `histogram` which allows us to do conditional histograms. This function is included in a package named `lattice` to load it just type `library(lattice)`. Now you can use the function we need.

12. The attribute `svi` is binary, so let's see what's the distribution of the target variable `lpsa` conditioned on it. Type: `histogram(~lpsa | svi)`. Don't forget the tilde! To know more about it type `help(tilde)`.

    The last command plots a histogram of `lpsa`, however this histogram is split in two, which is the number of values that the variable `svi` takes. Can you tell which side of the plot corresponds to what value of this binary attribute? Could you tell what value of `svi` is taking the most frequent value of the target variable?

    We could've done this plot using the basic `hist()` command by typing `hist(lpsa[svi==0])`. Do you understand how this command is being interpreted by R?

    Now, plot another conditional histogram but this time conditioning on the categorical attribute `gleason` using the function `histogram()`. Remember that this plot shows the distribution of the target variable conditioned on the values of `gleason`. Do you know what are the values that this variable can take? Do you find anything interesting in this plot? Do you see any correlation between a value of `gleason` and `lpsa`? Could you explain it?

    Finally, can you spot any outliers for any value of the conditioning variable? Is this visualisation helpful?

13. Now, let's consider another kind of visualisation. This time we will make boxplots from our data. Remember that a boxplot provides graphical information of the median, quantiles, minimum and maximum values, and outliers -if any- for a given variable. You can try them right away by typing `boxplot(data)`. This will show a boxplot of each of the attributes in our dataset.

    Let's take a close look to one of them, but first get the *five-number summary* from the target variable `lpsa` by typing `summary(lpsa)`. As mentioned above, this will return the minimum value, the first, second and third quantiles, and the maximum value of this variable. Keep this information at hand.

    Let's do a mixed plot to help us visualize this variable. First, using the `par()` command we will modify the current parameters of the graphic capabilities of R, so by typing `par(mfrow=c(1,2))`, we are telling R to split the graphic interface in a matrix of one row and two columns in which we will put our plots. Now, type again `hist(lpsa,breaks=bins)` and immediately after this `boxplot(lpsa)`. This will open a window with two plots: a histogram and a boxplot of the target variable.

Now you have three different ways to analyse the statistics of this variable. Take a look on the *five-number summary* in both plots. Have you found each of this five values in the boxplot? Have you located the median in both the histogram and the boxplot? How frequent are the quantiles? How disperse are the values from the mean? What's the value of the standard deviation? Have you spot any outliers in any of the plots? Are these plots useful to find outliers? How frequent are they?

You should take some time to explore the rest of the features in the same way.

14. Let's try yet another way to visualise data. We will consider scatter plots of the features in the data. A scatter plot provides a graphical view of the relationship between two variables. You can try a scatterplot of a variable against itselt by typing *plot(lpsa,lpsa)*. Naturally, the relationship of a variable with itself is linear! Moreover, you could perform a correlation test of this to test it. Type `cor(lpsa,lpsa)` and you will get the maximum correlation value that you can get.

    Now try visualising the correlation between another variable and the target variable. For example, find out how correlated is the age with the presence of prostate specific antigen (`lpsa`). Are these variables correlated? What can you say numerically of this correlation? Is it high or low? Would age be enough to predict the outcome of the target variable? Why not?

    Explore the rest of the features against the target variable. Can you find any variable which is highly correlated with the target variable? Which is it? Could it explain the appearance of prostate cancer? Could it predict the outcome of the target variable? Could you foretell what would happen if we get rid of the variables with low correlation?

15. Naturally, you could perform this analysis with different features other than the target variable and discover different dependencies and correlations among them, and by doing this you could reduce the number of attributes in the dataset.

    You could try to build by yourself the *scatter plot matrix* shown in figure (1.1) of the book by Hastie et al. which you can find online, however you could try typing something as simple as `plot(data)` and see the results.

    By the way, you can also inspect the correlation among the variables in the data by simply typing `cor(data)`, that is, setting the whole data as the parameter to the function. Or, in a more presentable way: `round(cor(data), 2)`.

16. Ok. Now let's try to add some color to our plots. It's obvious how color can aid visualisation in almost any sense, but here you will see how by adding color to a plot you -literally- add another dimension to your visualisation.

    Try this: `col<-c("RED","BLUE")`. The contents of this variable are these two *colors*, or at least they will be colors when they are interpreted in the command:

    `plot(lcavol,lpsa,col=col[svi+1])`.

    Have you tried it already? Can you explain what's happening?

Ok. This is what's happening. You already know the scatterplot between two variables, right? Now, in this new plot we are adding a third variable. In this case, we are adding the variable `svi` to our simple scatterplot. Remember that the variable `svi` is binary. Now, pretend that we are visualising these three variables in space, and that the points in color red are the points that lie in the plane $(x, y, 0)$, whereas the blue points lie in the plane $(x, y, 1)$. Do you see how powerful is this plot just by adding colors? But, wait a minute! How did I know that red points correspond to `svi = 0` and blue points to `svi = 1`?

Now, try to do the same with the other categorical feature in the dataset.

17. You've done some nice plots in this lab but you still don't know how to save them. R allows you to save your images in a wide range of formats including pdf, postscript, png, and jpeg among others. Type `?pdf` for more information about this. Also, R allows you to add and format titles, captions, etc. The next sequence of commands will show you how to save a plot in pdf format:

```
> pdf("myplot.pdf")
> col<-c("RED","BLUE")
> plot(lcavol,lpsa,col=col[svi+1])
> title("Scatter plot of lcavol, lpsa and svi")
> dev.off()
```

18. Now take a look at the image that you have just created. Remember that R saves this document in the working directory. If you haven't changed it as mentioned earlier, then you can specify the whole path in which you want to save it.

    Obviously, you can tailor the looks of your plot. To know how take a look to the help page of the functions `pdf()`, `par()` and `plot()`, and also to the online bibliography mentioned at the beginning of this lab.

19. Finally, you close R by typing `q()`. When doing so, it will prompt you if you want to save your workspace. I think you won't need to do this at this moment!

20. Ok. If you have some spare time or if you have finished this lab really fast or simply are curious about machine learning, data mining and artificial intelligence in general, then I invite you to take a look at the following links:

    - Kevin Slavin on how algorithms shape our world: `http://youtu.be/TDaFwnOiKVE`
    - Jeff Hawkins, founder of Palm Computing, on how neuroscience could change the way we do computing : `http://youtu.be/G6CVj5IQkzk`
    - A Wired article about how the Netflix prize was won: `http://www.wired.com/epicenter/2009/09/how-the-netflix-prize-was-won/`

    Enjoy! :-)

JVHU.