36-350: Data Mining

Lab 1
Date: August 30, 2002 **Due: end of lab**

---

1 Introduction

This lab teaches you the R software we will be using in the course and shows you the steps involved in similarity searching.
There are 5 questions. For each one, submit your commands and a response from R demonstrating that they work. Don't forget to check that the response makes sense! And only hand in commands relevant to the question.

2 Starting R

Start by creating a new folder for yourself on the desktop. You can do this with

```
(Right Mouse) -> New -> Folder
```

Then start R from the Start menu:

```
Start -> Programs -> Class software -> R -> R 1.5.1
```

It should immediately give you a command window. The `>` is the prompt for entering commands. Tell R to use your folder, via the menu

```
File -> Change dir...
```

and browse to your folder.

3 Entering Commands

R can be used as a calculator. For example, type `2+3` and then press enter.

```
> 2 + 3
[1] 5
```

A more complex example is

```
> (1 - 3*(4 + 5)^2)/3
[1] -80.66667
```

Arithmetic and order of operations work in the usual way. Multiplication is `*`, division is `/` and exponentiation is `^`. R can also evaluate functions like square root and absolute value:

```
> sqrt(9)
[1] 3

> abs(-5)
[1] 5
```

These functions take one value. Some functions take more than one value, such as `c(2,3)`, or no values at all, such as `plot.new()`. The general format for calling a function is `functionname`(*arg1*, *arg2*, ...), or `functionname()` for a function which takes zero arguments. Typing the function name by itself (with no parentheses) will return the definition of the function. R is case-sensitive, so be sure to use the correct capitalization: `Functionname()` and `functionname()` are different things.
The "[1]" which keeps showing up means that the first item on that line is the first item in the vector of output. This indexing is useful when you want to find a particular observation in a long list of output. In the output below, 1000 is the 36th observation.

```
 [1]    0.73    1.46    2.19    2.92    3.65    4.38    5.11    5.84    6.57
[10]    7.30    8.03    8.76    9.49   10.22   10.95   11.68   12.41   13.14
[19]   13.87   14.60   15.33   16.06   16.79   17.52   18.25   18.98   19.71
[28]   20.44   21.17   21.90   22.63   23.36   24.09   24.82   25.55 1000.00
[37]    1.04    1.30    1.56    1.82    2.08    2.34    2.60    2.86    3.12
[46]    3.38    3.64    3.90    4.16    4.42    4.68    4.94
```

Incomplete commands If you enter an incomplete command (for instance, by forgetting a close-parenthesis), R will give you another prompt, this time a "`+`".

```
> sqrt(16
+
```

At this point, type the close-parenthesis and the command will execute normally.

```
> sqrt(16
+ )
[1] 4
```

Retyping commands If you have already typed a similar command, you can fetch it by typing up-arrow multiple times. You can then edit the command and press enter.

> **Question 1:** Give an R command which computes the Euclidean distance between `(2,3)` and `(4,6)`.

4 Variables

Variables can be created to hold intermediate results. To assign the value `9` to the variable `x`, type:

```
> x <- 9
```

(Read this as "x gets 9".) If you type the variable name, R will return its value:

```
> x
[1] 9
```

Now that `x` has a value, it can be manipulated like any other number.

```
> sqrt(x)
[1] 3

> y <- (5 * (x + 2)) - 3
```

These manipulations do not affect the value of `x`, it is still `9`. To change it, you must assign a new value to `x`.

Text strings Besides numbers, R can also manipulate text strings. You can provide text inside of quotes, such as

```
> y <- "hello"
> y
[1] "hello"
> paste(y,"there")
[1] "hello there"
```

Variable names You can use any name you want, as long as you want, subject to two restrictions. First, the name can only contain letters, numbers, and the period symbol. It can not have blank spaces, underscores, or dashes. If you have two words in a variable name, such as "Water Depth", you could write it as `WaterDepth`, `waterdepth`, or `water.depth`. The second rule is that it shouldn't be the same name as a function, or else you won't be able to use the function anymore. For example, `c`, `t`, and `length` are functions in R, so don't use them for variable names.

5 Vectors

All of the preceding examples were concerned with single numbers, but most statistical analyses involve groups of numbers. R can group numbers into vectors, lists, arrays, and frames. Let's start with vectors. You create a vector of numbers by concatenating them with the `c` function:

```
> observations <- c(2, 4.6, 1, 3.7, 5.9, 4.0, 6.7, 2.8, 1.4, 3.1)
> observations
[1] 2.0 4.6 1.0 3.7 5.9 4.0 6.7 2.8 1.4 3.1
```

Manipulations on vectors work the same way as manipulations on single numbers. Suppose these observations were in inches but need to be converted to centimeters:

```
> 2.54*observations
[1]  5.080 11.684  2.540  9.398 14.986 10.160 17.018  7.112  3.556  7.874
```

Note that the vector `observations` still contains the original data. An arithmetic operation between two vectors will apply element-by-element, such as:

```
> observations*observations
[1]  4.00 21.16  1.00 13.69 34.81 16.00 44.89  7.84  1.96  9.61
```

There are many functions to operate on vectors, e.g. `mean`, `max`, and `sum`:

```
> sum(observations)
[1] 35.2
```

A vector of text strings can be made just as easily as a vector of numbers:

```
> txt <- c("candy","apple","red")
> txt
[1] "candy" "apple" "red"
```

> **Question 2:** Give an R command to compute the Euclidean length of any vector `x`, using the vector operations above. Test it on `x<-c(2,3,4)`.

Selecting from Vectors Subsets of a vector can be selected by using square brackets.

```
> observations[3]
[1] 1
> observations[5:7]
[1] 5.9 4.0 6.7
> observations[c(1,2,7,1)]
[1] 2.0 4.6 6.7 2.0
```

The first command returns the third observation. The second command returns the fifth through seventh observations. The third command returns first, second, seventh and first (again) observations. When a vector has named elements, you can also refer to them by name. For example:

```
> x <- c(2,3,4)
> names(x) <- c("first","second","third")
> x["second"]
second
     3
```

6 Searching a document collection

6.1 Reading a document into R

The rest of this lab requires special files. On the class web page, go to "computer labs" and download the files for lab 1 into your work folder. The file `"lab1.r"` has special functions for manipulating text. Read these functions into your running R application via the command `source("lab1.r")`. If this fails, the file may be been renamed to `lab1.r.txt` when you downloaded it.
The file "politics3.txt" is a sample post in a discussion group. You can read it into R via the command

```
txt <- read.doc("politics3.txt")
```

This command returns a vector of words. It removes the header, removes all punctuation and capitalization, and converts all numbers to the pound sign `#`.

> **Question 3:** According to `read.doc`, what is the 57th word in the document?

6.2 Bag-of-words tabulation

For similarity searching, the vector of words needs to be converted into word counts (the bag-of-words representation). This is done via the function `table`:

```
table(txt)
```

It takes a while to read in and tabulate the full set of documents, so the table is provided for you in `lab1_docs.rda`. This can be loaded via

```
load("lab1_docs.rda")
```

It gives you a table called `doc`, containing 9 posts to `talk.politics.misc` and 9 posts to `talk.religion.misc`. The `dim` function will tell you the size of the table:

```
dim(doc)
```

Discriminating these articles is more difficult than autos/motocycles used in class, because the 'misc' discussions cover a pretty wide range of topics.
When a table has two dimensions, like `doc` does, you select from it by giving two indices, e.g. `doc[2,4]` or `doc[4,"and"]`. An entire row or column can be selected by leaving out the index, e.g. `doc[2,]` or `doc[,"and"]`.

> **Question 4:** According to `doc`, how many times does 'the' occur in `politics3`? Give a command to select the answer from `doc`. (Note that both the rows and columns have names.)

6.3 Document similarity

To begin, it is helpful to remove infrequent words, namely words which occur in only one document. This is done via the function `remove.singletons`.

```
doc <- remove.singletons(doc)
```

The next step is to weight the words by inverse document frequency (IDF):

```
doc <- idf.weight(doc)
```

The function `distances` will compute a matrix of all pairwise Euclidean distances between the data you provide (the matrix is stored in `d`):

```
d <- distances(doc)
```

There are some other useful functions. To divide each document's count vector by its sum:

```
x <- div.by.sum(doc)
```

The result is put into `x` so that the original `doc` can still be used. To instead divide each document's count vector by its Euclidean length:

```
x <- div.by.euc.length(doc)
```

> **Question 5:** (a) Compute a distance matrix between all documents, using IDF but without normalizing for document length. (b) Compute a distance matrix where the weighted word counts have been normalized by the document total. (c) Compute a distance matrix where the weighted word counts have been normalized by the document's Euclidean length. Hand in your code and the three matrices. Because the matrices are big, you only need to show the first five columns.