R for Data Science
(manual)

Andreas Wichert

1 Introduction

- http://www.r-project.org   -> official Manuals, Packages
- **Programming language** for statistical data analysis, inference and visualization
- Ports for Unix, Windows and MacOSX
  Highly extensible through user-defined functions Multivariate Statistics, Machine Learning, Natural Language Processing, Bioinformatics
- **R is free**

Command Line Interface

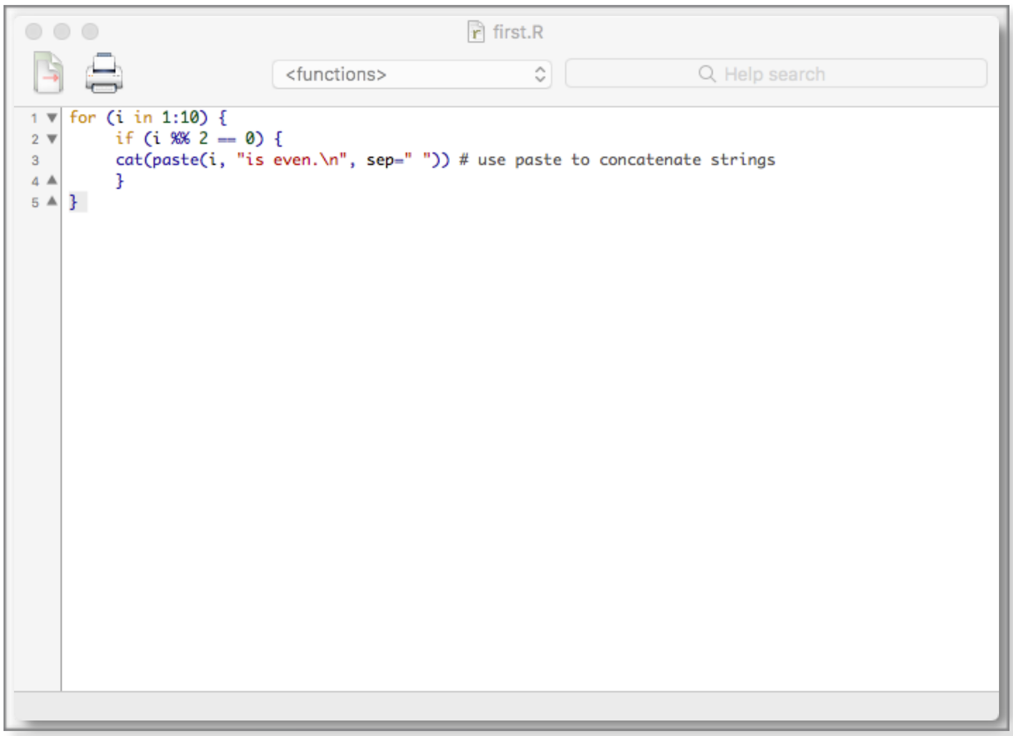
>*help()*
to get help about sum,
>help(sum)
or
>?sum

figure out your current directory, type
>*getwd()*
In Windows
> *setwd("C:\\Datasets")*

- All computations and statistics are performed with commands
- These commands are called "functions" in R
- Commands are separated either by a semicolon ; or newline
- An expression command is evaluated and (normally) printed
- An assignment command evaluates an expression and passes the value to a variable but the result is not printed

Running code

- Go to “File” in the top menu and click on “New script.”
- This opens up a new window that you can save as a .R file.
- To execute the code highlight the lines you wish to run, and press Ctrl-R on a PC or Command-Enter on a Mac.
- If you want to run an entire script go to “Edit,” and click “Run all.



```
for (i in 1:10) {
     if (i %% 2 == 0) {
     cat(paste(i, "is even.\n", sep=" ")) # use paste to concatenate strings
     }
}
```

```
> for (i in 1:10) {
+      if (i %% 2 == 0) {
+      cat(paste(i, "is even.\n", sep=" ")) # use paste to concatenate strings
+      }
+ }
2 is even.
4 is even.
6 is even.
8 is even.
10 is even.
>
```

Installing and loading packages

- Functions in R are grouped into packages, a number of which are automatically loaded when you start R.
- These include “base,” “utils,” “graphics,” and “stats.”
- Some of them are preinstalled but we have to load them with the function library()
- Go to Packages and load
- You may need to download additional packages to obtain other useful functions.
- For example, an important classification method called Support Vector Machines is contained in a package called
- “e1071.”
- To install this package, click “Packages” in the top menu, then “Install package(s)...” When asked to select a CRAN mirror, choose a location close to you, such as “France (ON).” Finally select “e1071.”
- To load the package, type library(e1071) at the command prompt.
- Note that you need to install a package only once, but that if you want to use it, you need to load it each time you start R.

- Help on the contents of the packages is available
- > library(help = foreign)
- Help on installed packages is also available by help.start()

Used Packages

- ada
- arules
- **class**
- e1071
- **apart**
- foreign
- **MASS**
- **kernlab**
- neuralnet
- **randomForest**
- xlsReadWrite

Sample Session of Basic Functions

```
>1+1
[1] 2

> res = 1 + 5
>res
[1] 6

> 1:20
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20


> powers.of.2 = 2^(1:16)
> powers.of.2
 [1]     2     4     8    16    32    64   128   256   512  1024  2048  4096  8192 16384 32768 65536

> class(powers.of.2)
[1] "numeric"
> ls()
[1] "powers.of.2" "res"
> rm(powers.of.2)
```

Vesctors

```
> vect = c(1, 2, 99, 6, 8, 9)
> is(vest)
[1] "numeric" "vector"
> vect[2]
[1] 2
> vect[2:3]
[1] 299
> length(vect)
[1] 6
> sum(vect)
[1] 125


> v <- 1:5
> v
[1] 1 2 3 4 5
> v <- c(1,2,3,4,5)
> v
[1] 1 2 3 4 5
> v <- seq(from=1,to=5,by=1)
> v
[1] 1 2 3 4 5
```

```
> v[3]
[1] 3

> v1 <- c(1,2,3,4,5)
> v2 <- c(6,7,8,9,10)
> v3 <- c(11,12,13,14,15)
> v4 <- c(16,17,18,19,20)
>
> cbind(v1,v2,v3,v4)
     v1 v2 v3 v4
[1,]  1  6 11 16
[2,]  2  7 12 17
[3,]  3  8 13 18
[4,]  4  9 14 19
[5,]  5 10 15 20
>
> rbind(v1,v2,v3,v4)
   [,1] [,2] [,3] [,4] [,5]
v1    1    2    3    4    5
v2    6    7    8    9   10
v3   11   12   13   14   15
v4   16   17   18   19   20
>
```

```
> vect3 = c("austria", "spain", "france", "uk", "belgium", "poland")

> is(vect3)
[1] "character"
[2] "vector"
[3] "data.frameRowLabels"
[4] "SuperClassMethod"
```

Matrices

```
> mymat <- matrix(1:10, 2, 5)
> mymat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

```
> mymat <- matrix(1:10, 5, 5)
> mymat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6    1    6    1
[2,]    2    7    2    7    2
[3,]    3    8    3    8    3
[4,]    4    9    4    9    4
```

```
[5,]    5   10    5   10    5


> v <- seq(from=1,to=20,by=1)
> matrix(v, nrow=4, ncol=5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
>

> matrix20 <- matrix(v, nrow=4, ncol=5, byrow=TRUE)
> colnames(matrix20) <- c("Col1","Col2","Col3","Col4","Col5")
> rownames(matrix20) <- c("Row1","Row2","Row3","Row4")
> matrix20
     Col1 Col2 Col3 Col4 Col5
Row1    1    2    3    4    5
Row2    6    7    8    9   10
Row3   11   12   13   14   15
Row4   16   17   18   19   20

> matrix20[,"Col2"]
Row1 Row2 Row3 Row4
   2    7   12   17

> matrix20["Row3","Col1"]
[1] 11
> matrix20[3,1]
[1] 11
> dim(matrix20)
[1] 4 5
> ncol(matrix20)
[1] 5
```

Basic Mathematical Operations

```
#Vectors

> a=c(1,2,3)
> a
[1] 1 2 3
> b=c(5,6,7)
> b
[1] 5 6 7

> a+b
[1]  6  8 10

#Multiplication Element by Element
> a*b
[1]  5 12 21
```

```
#Scalar
> a %*% b
     [,1]
[1,]   38


#Multiplication
> t(b)
     [,1] [,2] [,3]
[1,]    5    6    7

> a %*% t(b)
     [,1] [,2] [,3]
[1,]    5    6    7
[2,]   10   12   14
[3,]   15   18   21

#Density Matrix
a %*% t(a)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9



> A=matrix(c(2, 4, 3, 1, 5, 7),2,3)
>
> A
     [,1] [,2] [,3]
[1,]    2    3    5
[2,]    4    1    7
> A=matrix(c(2, 4, 3, 1, 5, 7),3,2)
> A
     [,1] [,2]
[1,]    2    1
[2,]    4    5
[3,]    3    7

#Transpose
> A=t(matrix(c(2, 4, 3, 1, 5, 7),3,2))
>
> A
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
>

> b=c(5, 6, 7)
[1] 5 6 7

#Matrix Vector multiplication
> z=A %*% b

> z
     [,1]
```

```
[1,]   55
[2,]   84

#Matrix multiplication
> A %*% t(A)
     [,1] [,2]
[1,]   29   43
[2,]   43   75

#Functions
 pi
[1] 3.141593
> cos(pi)
[1] -1


> exp(1)
[1] 2.718282

> sqrt(-1)
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced

> sqrt(as.complex(-1))
[1] 0+1i
```

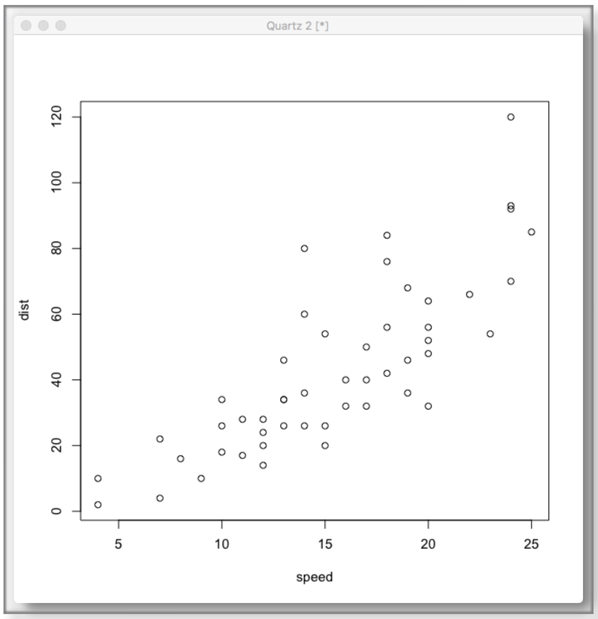
- R stores objects in workspace that is kept in memory When quiting R ask you if you want to save that workspace
- The workspace containing all objects you work on can then be restored next time you work with R along with a history of the used commands.

Datasets

data()

https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html

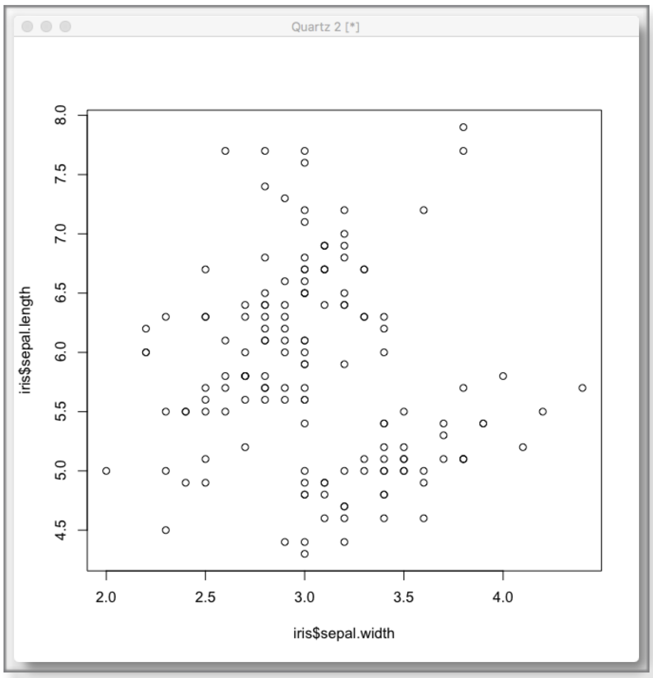
```
>  data(cars)
> cars
  speed dist
1     4    2
2     4   10
....
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  

 plot(cars)
```



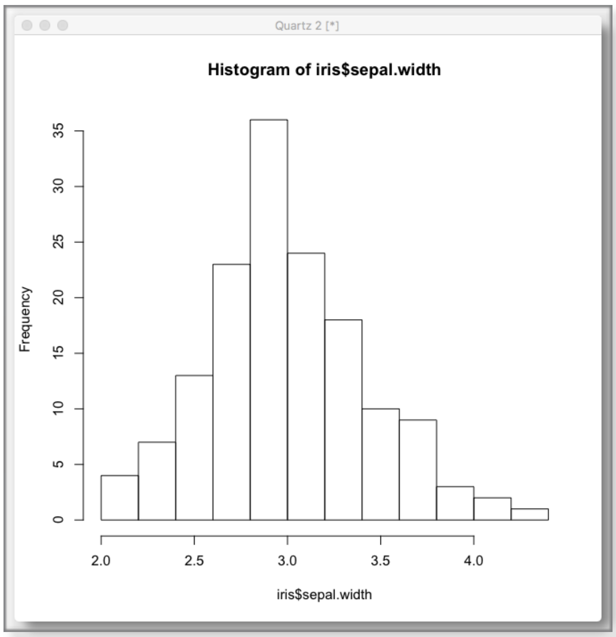
Descriptive Statistics

```
colMeans(cars)
speed  dist 
15.40 42.98 

> median(cars$speed)
[1] 15
```

```
> quantile(cars$speed)
  0%  25%  50%  75% 100% 
   4   12   15   19   25 

> var(cars$speed)
[1] 27.95918

> sd(cars$speed)
[1] 5.287644

> cor(cars)
          speed      dist
speed 1.0000000 0.8068949
dist  0.8068949 1.0000000

> cor(cars$speed,cars$dist)
[1] 0.8068949

> cov(cars)
          speed     dist
speed  27.95918 109.9469
dist  109.94694 664.0608




> data(iris)
> iris
    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
….

> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 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                  

 names(iris) <- tolower(names(iris))

> names(iris)
[1] "sepal.length" "sepal.width"  "petal.length" "petal.width"  "species"     
> plot(iris$sepal.width, iris$sepal.length)
```

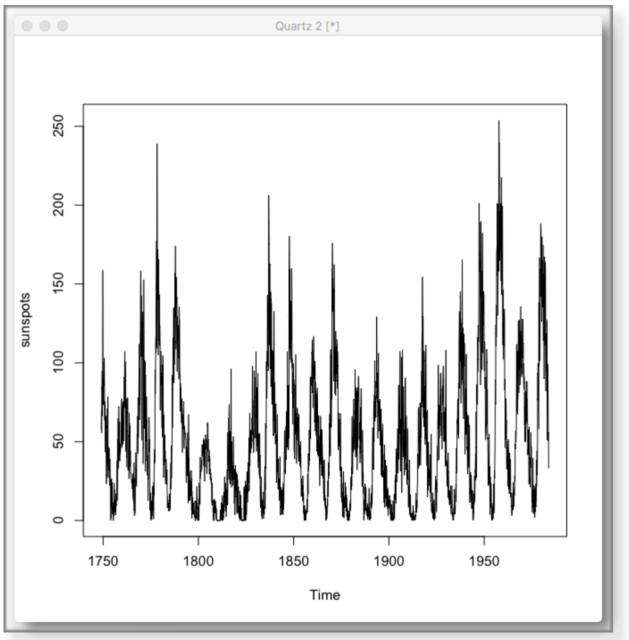


```
>hist(iris$sepal.width)
```



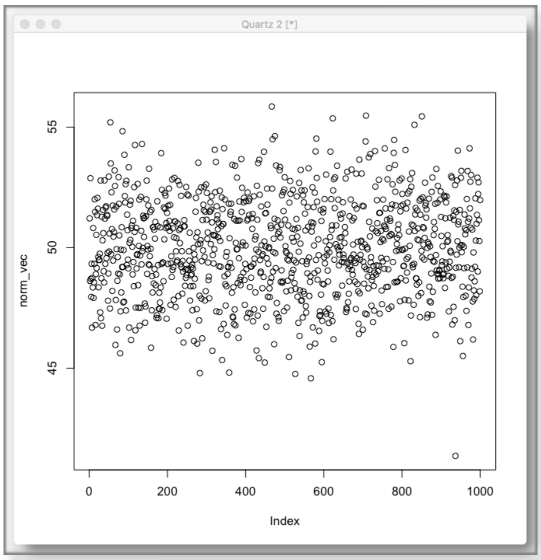
```
> data(sunspots)
> summary(sunspots)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   15.70   42.00   51.27   74.92  253.80
> sunspots
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1749  58.0  62.6  70.0  55.7  85.0  83.5  94.8  66.3  75.9  75.5 158.6  85.2
1750  73.3  75.9  89.2  88.3  90.0 100.0  85.4 103.0  91.2  65.7  63.3  75.4
1751  70.0  43.5  45.3  56.4  60.7  50.7  66.3  59.8  23.5  23.2  28.5  44.0
```

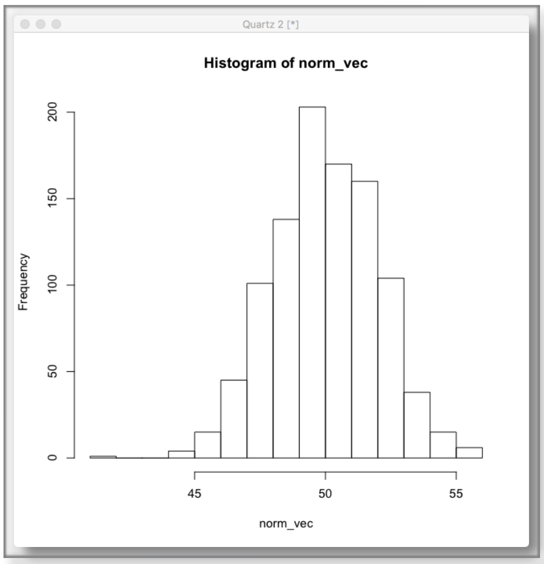
plot(sunspots)



Sampling

```
>norm_vec <- rnorm(n=1000, mean=50, sd=2)
>plot(norm_vec)
> hist(norm_vec)
```





T-Test

```
> xi=c(14, 17.5, 17, 17.5, 15.4)
> xi
[1] 14.0 17.5 17.0 17.5 15.4
> yi=c(17, 20.7, 21.6, 20.9,17.2)
> yi
[1] 17.0 20.7 21.6 20.9 17.2

> t.test(xi,yi,mu=0,paired = TRUE)

        Paired t-test

data:  xi and yi
t = -7.1554, df = 4, p-value = 0.002019
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.441664 -1.958336
sample estimates:
mean of the differences
                   -3.2
```

Linear regression

- In R, use the lm function to generate these models.
- Functions have a "formula" as one of their as one of their arguments
- Suppose you have a response variable y and independent variables x1, x2, and x3.
- To express that y depends linearly on x1, x2, and x3, you would use the formula *y ~ x1 + x2 + x3,* where *y, x1, x2, and x3* are also column names in your data matrix.

```
> x_i=c(40.5, 38.6, 37.9, 36.2, 35.1, 34.6)
> y_i=c(104.5, 102, 100, 97.5, 95.5, 94)
> data=as.data.frame(cbind(y_i,x_i) )
> data
   y_i  x_i
1 104.5 40.5
2 102.0 38.6
3 100.0 37.9
4  97.5 36.2
5  95.5 35.1
6  94.0 34.6
> lm_model <- lm(y_i ~ x_i, data)
>
> lm_model

Call:
lm(formula = y_i ~ x_i, data = data)

Coefficients:
(Intercept)        x_i
     33.53        1.76

newdata = data.frame(x_i=37)
predict(lm_model, newdata, interval="predict")
      fit      lwr      upr
1 98.65264 97.37838 99.92691

> x_i=c(37, 35)
> new_data3=as.data.frame(cbind(x_i))
 predict(lm_model, new_data3, interval="predict")
       fit      lwr      upr
1 98.65264 97.37838 99.92691
2 95.13235 93.76316 96.50153
```

Prediction

- For most of the following algorithms (as well as linear regression), we would in practice first generate the model using training data, and then predict values for test data.
- To make predictions, we use the predict function.
- The first argument is the variable in which you saved the model, and the second argument is a matrix or data frame of test data.

```
>help(predict.lm)
```

save(lm_model, file="LM")
load(file="LM")

```
>library(MASS)
>data(hills)

>hills
                          dist climb    time
Greenmantle                2.5   650  16.083
Carnethy                  6.0  2500  48.350
Craig Dunain              6.0   900  33.650
Ben Rha                   7.5   800  45.600
Ben Lomond                8.0  3070  62.267
Goatfell                  8.0  2866  73.217
Bens of Jura              16.0  7500 204.617
Cairnpapple               6.0   800  36.367
Scolty                    5.0   800  29.750
…

> mhill <- lm(time ~ dist, data = hills)
> mhill

Call:
lm(formula = time ~ dist, data = hills)

Coefficients:
(Intercept)        dist
     -4.841       8.330
```

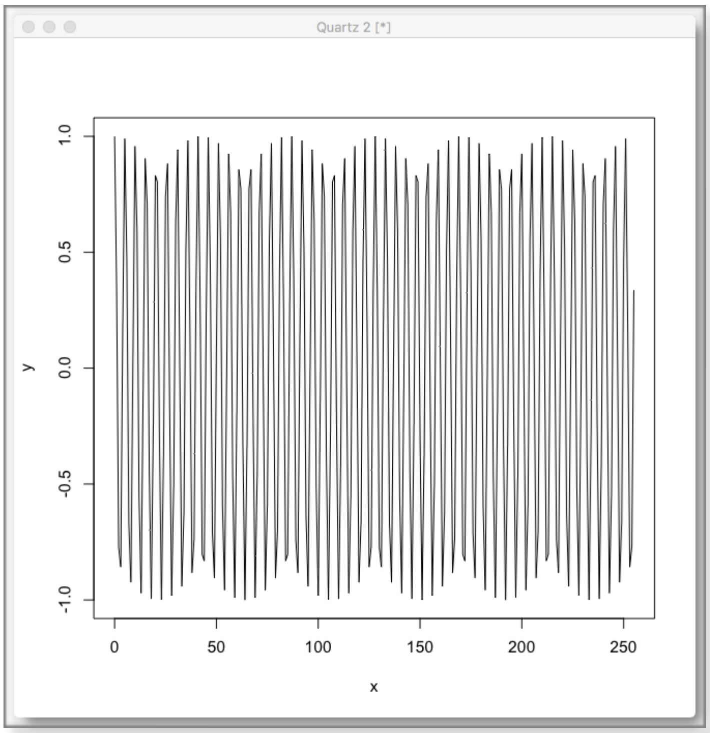
```
save(mhill, file=“M_hill”)
load(file=“M_hill")
```

2 Transform Functions

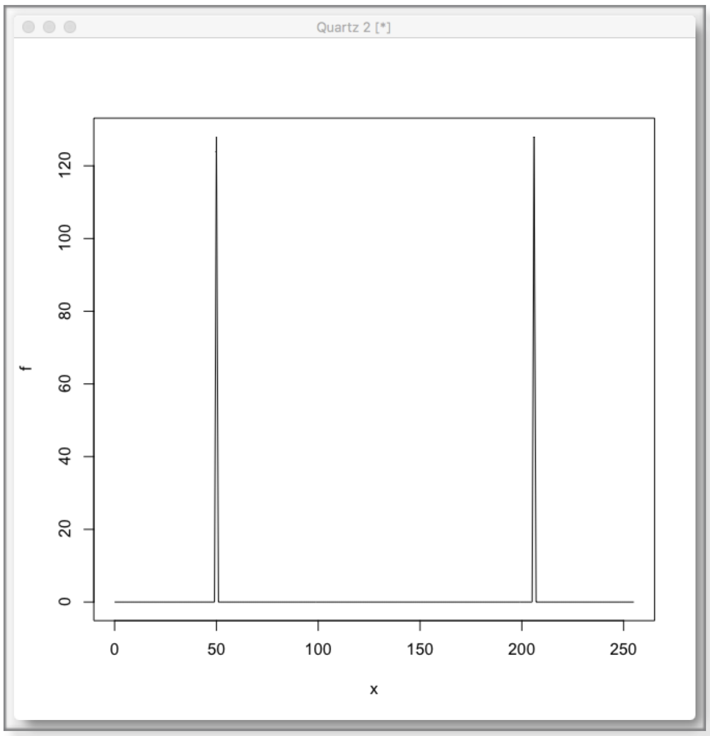
Preprocessing

DFT

```
> x <- 0:255
> y <- cos(50*x*2*pi/256)
> plot(x,y,type="l")
```

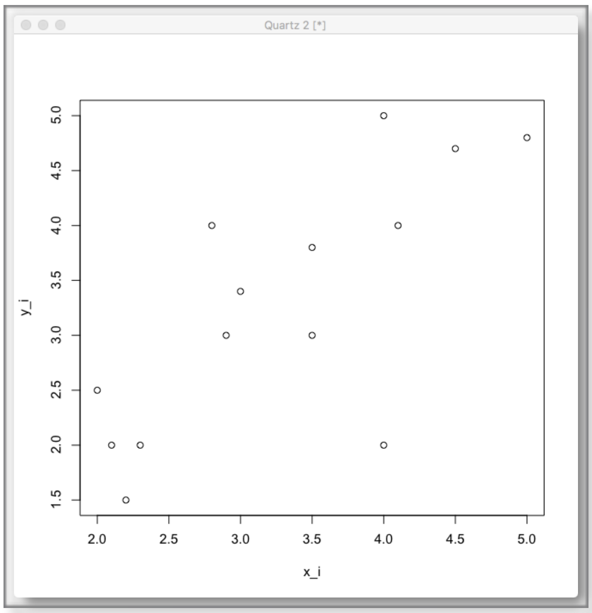


```
>  f=abs(fft(y))
>  plot(x,f,type="l")
```



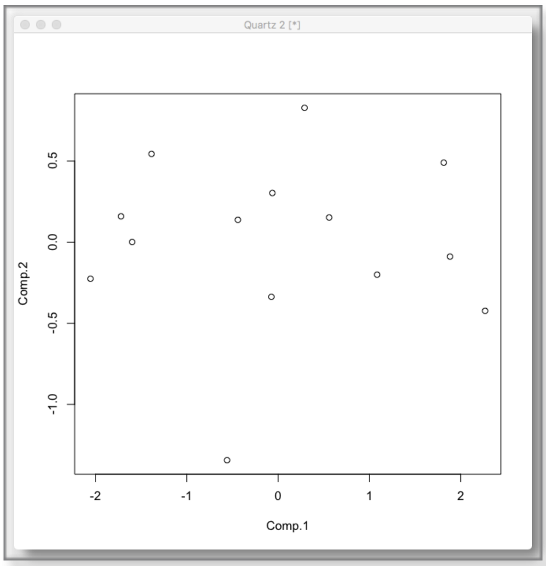
PCA

```
> x_i=c(2.1, 2.3, 2.9, 4.1, 5, 2, 2.2, 4, 4, 2.8, 3, 3.5, 4.5, 3.5)
> y_i=c(2, 2, 3, 4, 4.8, 2.5, 1.5, 5, 2, 4, 3.4, 3.8, 4.7, 3)
> data=as.data.frame(cbind(x_i,y_i) )
> data
  x_i y_i
1  2.1 2.0
2  2.3 2.0
3  2.9 3.0
4  4.1 4.0
5  5.0 4.8
6  2.0 2.5
7  2.2 1.5
8  4.0 5.0
9  4.0 2.0
10 2.8 4.0
11 3.0 3.4
12 3.5 3.8
13 4.5 4.7
14 3.5 3.0
>plot(data)
```



```
> U=cov(data)
> U
          x_i       y_i
x_i 0.9125824 0.8245604
y_i 0.8245604 1.3424725

> eigen(U)
eigen() decomposition
$values
[1] 1.9796432 0.2754117   #variance! (std^2)

$vectors
          [,1]       [,2]
[1,] 0.6114537 -0.7912802
[2,] 0.7912802  0.6114537


> prcomp(data)
Standard deviations (1, .., p=2):
[1] 1.4069980 0.5247968

Rotation (n x k) = (2 x 2):
          PC1        PC2
x_i 0.6114537 -0.7912802
y_i 0.7912802  0.6114537
```

```
> my.pca=princomp(x = data)

 my.pca
Call:
princomp(x = data)

Standard deviations:
   Comp.1    Comp.2
1.3558172 0.5057069


> loadings(my.pca)

Loadings:
    Comp.1 Comp.2
x_i  0.611 -0.791
y_i  0.791  0.611

               Comp.1 Comp.2
SS loadings       1.0    1.0
Proportion Var    0.5    0.5
Cumulative Var    0.5    1.0
>
> new.data<-predict(my.pca)
> new.data
          Comp.1       Comp.2
 [1,] -1.72104614  0.159527987
 [2,] -1.59875539  0.001271951
 [3,] -0.44060296  0.137957581
 [4,]  1.08442170 -0.200124898
 [5,]  2.26775421 -0.423114071
 [6,] -1.38655142  0.544382875
 [7,] -2.05554085 -0.225326900
 [8,]  1.81455651  0.490456858
 [9,] -0.55928403 -1.343904358
[10,]  0.28953184  0.828539337
[11,] -0.06294552  0.303411058
[12,]  0.55929343  0.152352463
[13,]  1.88289933 -0.088619354
[14,] -0.07373072 -0.336810528


> plot(new.data)
```



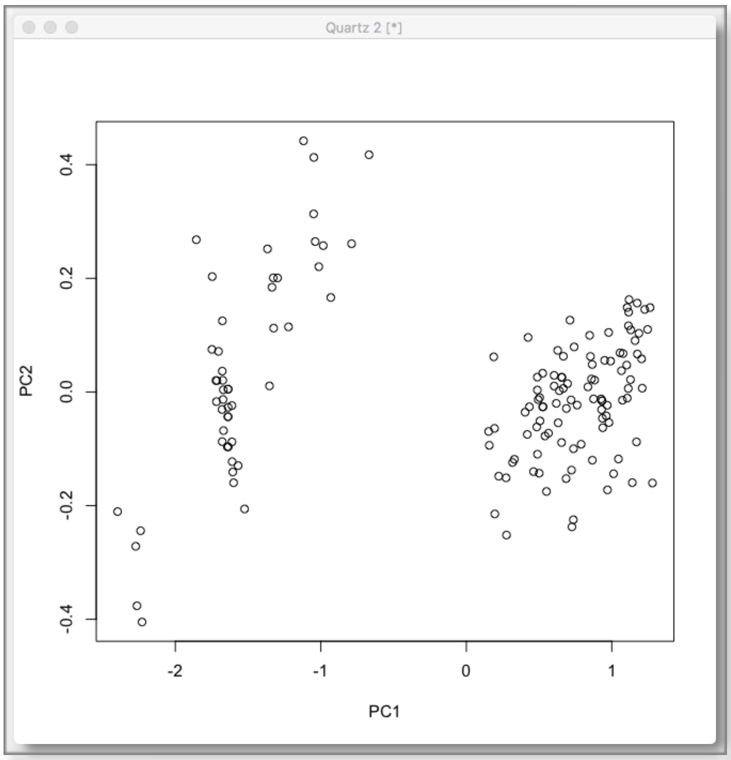
prcomp:
The calculation is done by a singular value decomposition of the (centered and possibly scaled) data matrix, not by using eigen on the covariance matrix. This is generally the preferred method for numerical accuracy.
princomp :
The calculation is done using eigen on the correlation or covariance matrix, as determined by cor. This is done for compatibility with the S-PLUS result. A preferred method of calculation is to use svd on x, as is done in prcomp."

So, prcomp is preferred, although in practice you are unlikely to see much difference (for example, if you run the examples on the help pages you should get identical results).

Example

```
>data(iris)
>head(iris, 3)
  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


>iris_data=log(iris[, -5])   #logarithmic scale

> eigen(cov(iris_data))
eigen() decomposition
$values
[1] 1.314598414 0.019141453 0.018294581 0.002895452

$vectors
            [,1]          [,2]       [,3]        [,4]
[1,]  0.10090019 -0.0008537483 -0.4891583  0.86633858
[2,] -0.05759298  0.5745110809 -0.7140592 -0.39590340
[3,]  0.50527032 -0.6870939247 -0.4269180 -0.30057416
[4,]  0.85510473  0.4447900940  0.2618865  0.04871476


> ir.pca <- prcomp(iris_data)
> ir.pca
Standard deviations (1, .., p=4):
[1] 1.14655938 0.13835264 0.13525746 0.05380941

Rotation (n x k) = (4 x 4):
                    PC1           PC2        PC3         PC4
Sepal.Length  0.10090019 -0.0008537483 -0.4891583  0.86633858
Sepal.Width  -0.05759298  0.5745110809 -0.7140592 -0.39590340
Petal.Length  0.50527032 -0.6870939247 -0.4269180 -0.30057416
Petal.Width   0.85510473  0.4447900940  0.2618865  0.04871476
```

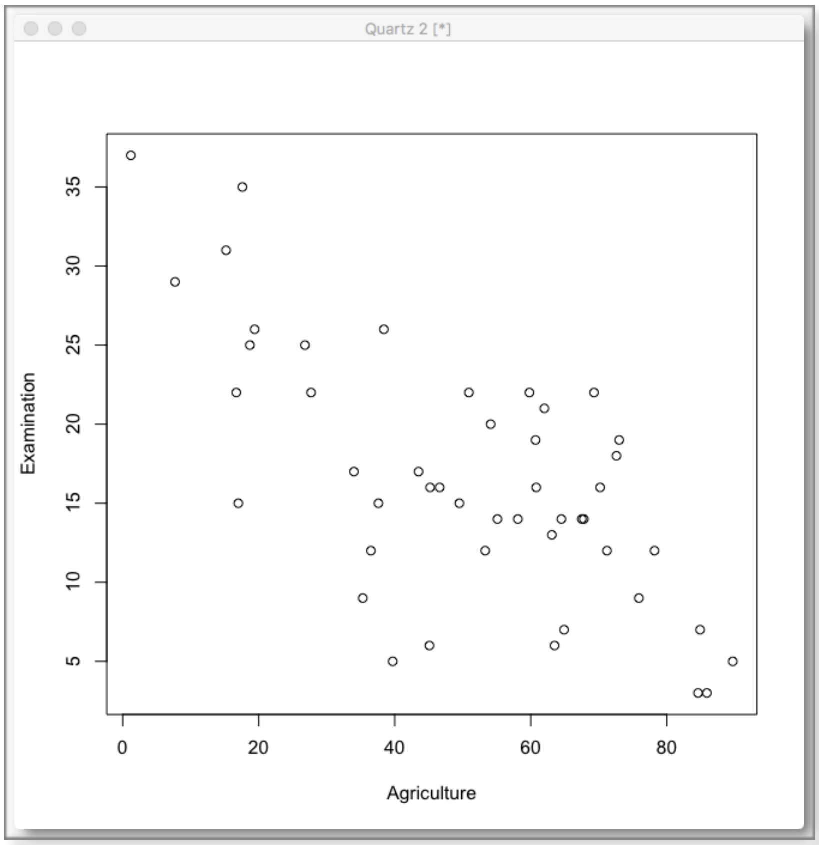
```
>pred.iris=predict(ir.pca)
> d=pred.iris[,1:2]
>plot(d)
```



Clustering

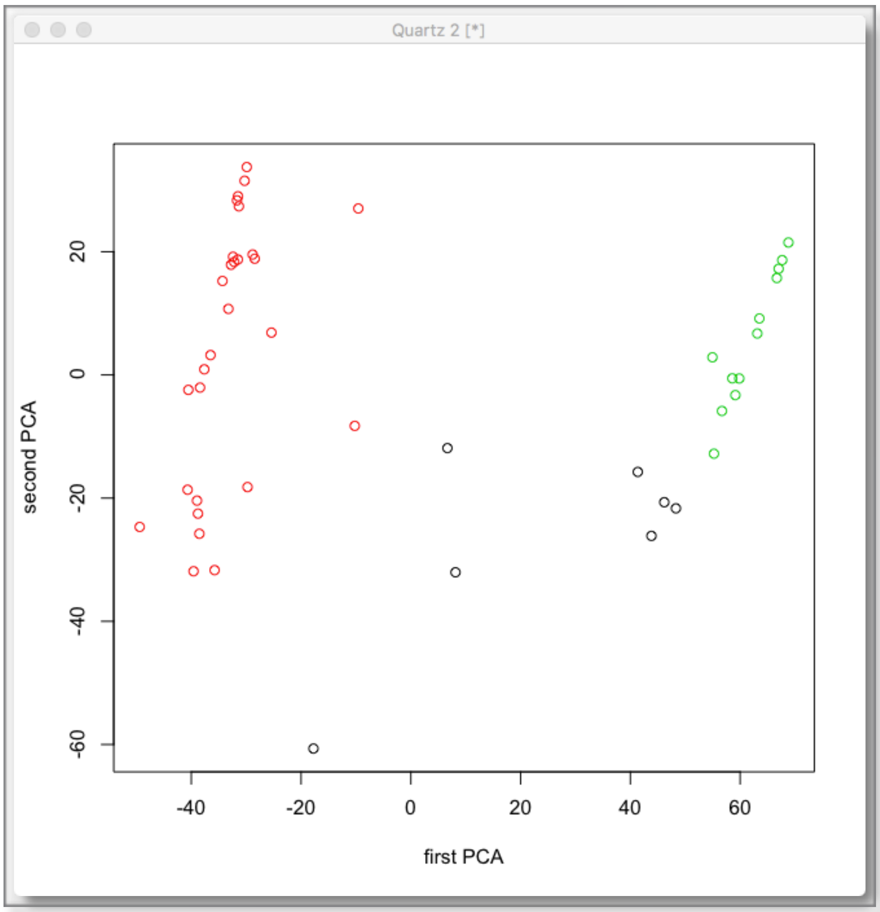
```
> clust <- kmeans(iris[, -5], centers = 4)

> clust$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.901613    2.748387     4.393548    1.433871
2     6.850000    3.073684     5.742105    2.071053
3     5.006000    3.428000     1.462000    0.246000
> clust
K-means clustering with 4 clusters of sizes 40, 32, 28, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     6.252500    2.855000     4.815000    1.625000
2     6.912500    3.100000     5.846875    2.131250
3     5.532143    2.635714     3.960714    1.228571
4     5.006000    3.428000     1.462000    0.246000

Clustering vector:
  [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
1 1 1 3 1 3 1 3 1 3 3 3 3 1 3 1 3 3 1 3 1 3 1 1 1 1 1 1 1 3 3 3
 [83] 3 1 3 1 1 1 3 3 3 1 3 3 3 3 3 1 3 3 2 1 2 2 2 2 3 2 2 2 1 1 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2
2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 1 2 1

Within cluster sum of squares by cluster:
[1] 13.624750 18.703437  9.749286 15.151000
 (between_SS / total_SS =  91.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
"iter"         "ifault"

> clust$size
[1] 40 32 28 50

> clust$iter
[1] 2
```

```
> data(swiss)
> swiss.x <- as.matrix(swiss[, -1])

> head(swiss.x, 3)
                  Agriculture Examination Education Catholic Infant.Mortality
Courtelary              17.0          15       12         9.96          22.2
Delemont                45.1           6        9        84.84            22.2
Franches-Mnt            39.7           5        5        93.40            20.2
>  plot(swiss.x)
```



```
>
> km <- kmeans(swiss.x, 3)
> km
K-means clustering with 3 clusters of sizes 16, 12, 19

Cluster means:
  Agriculture Examination Education Catholic Infant.Mortality
1    65.51875     9.43750  6.625000  96.1500        20.77500
2    20.92500    24.58333 21.666667  23.1775          19.31667
3    56.92632    17.31579  7.894737   6.1700        19.63684

Clustering vector:
  Courtelary     Delemont Franches-Mnt      Moutier   Neuveville   Porrentruy        Broye        Glane
Gruyere       Sarine      Veveyse        Aigle      Aubonne
           2            1            1            2            3            1            1            1            1            1            1
3            3
```

```
  Avenches     Cossonay    Echallens     Grandson     Lausanne     La Vallee       Lavaux       Morges
Moudon         Nyone          Orbe          Oron        Payerne
         3            3            3            3            2            2            3            3            3            3            3
3            3
Paysd'enhaut         Rolle         Vevey       Yverdon       Conthey    Entremont       Herens      Martigwy
Monthey   St Maurice        Sierre          Sion        Boudry
         3            3            2            3            1            1            1            1            1            1            1
1            3
La Chauxdfnd      Le Locle     Neuchatel    Val de Ruz ValdeTravers V. De Geneve  Rive Droite  Rive
Gauche
         2            2            2            3            2            2            2            2

> swiss.pca <- princomp(swiss.x)
> swiss.px <- predict(swiss.pca)
>  dimnames(km$centers)[[2]] <- dimnames(swiss.x)[[2]]    #Names of the first raw
> dimnames(km$centers)[[2]]
[1] "Agriculture"      "Examination"      "Education"       "Catholic"        "Infant.Mortality"
```



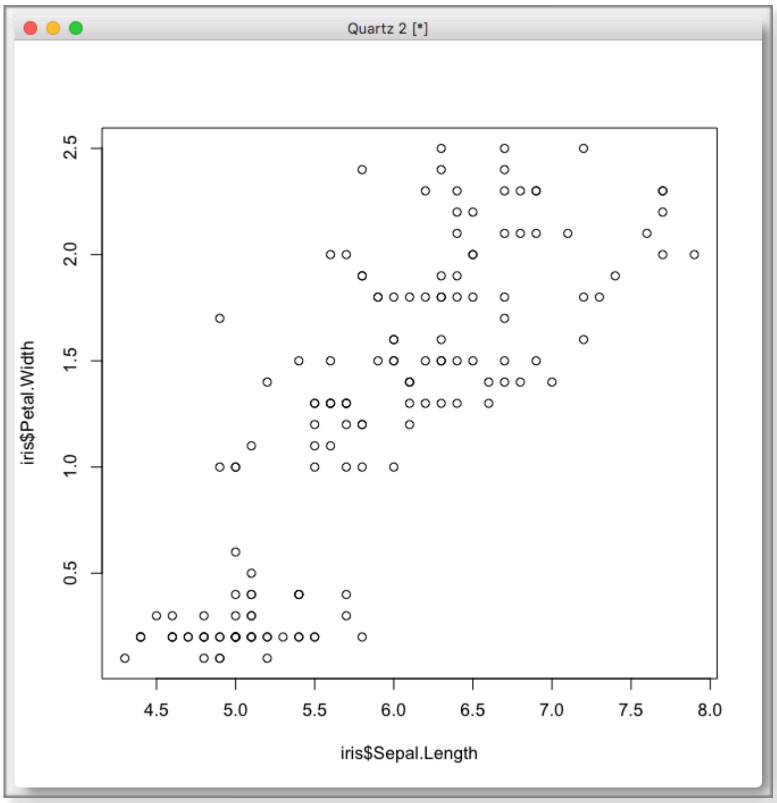
clustering validity is based on relative criteria

Attempts to identify "compact and well separated clusters"

-Dunn index, a cluster validity index for k- means clustering proposed in Dunn (1974)
-The Davies-Bouldin (DB) index (1979)

Install package clv
France Lyon 1

```
>library(clv)
>data(iris)

> iris
   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
....

> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 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


>plot(iris$Sepal.Length,iris$Petal.Width)
```



```
>iris.data <- iris[,1:4]


>km.mod <- kmeans(iris.data,5) # create five clusters


K-means clustering with 5 clusters of sizes 19, 8, 23, 38, 62

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     4.678947    3.084211     1.378947    0.200000
2     5.512500    4.000000     1.475000    0.275000
3     5.100000    3.513043     1.526087    0.273913
4     6.850000    3.073684     5.742105    2.071053
5     5.901613    2.748387     4.393548    1.433871

Clustering vector:
  [1] 3 1 1 1 3 2 1 3 1 1 2 3 1 1 2 2 2 3 2 3 3 3 1 3 3 1 3 3 3 1 1 3 2 2 1 1 3 3 1 3 3 1 1 3 3 1 3 1 3 3
5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5
 [83] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 4 4 4 4 5 4 4 4 4 4 4 5 5 4 4 4 4 5 4 5 4 5 4 4 5 5 4 4 4 4
4 5 4 4 4 4 5 4 4 4 5 4 4 4 5 4 4 5

Within cluster sum of squares by cluster:
[1]  2.488421  0.958750  2.094783 23.879474 39.820968
 (between_SS / total_SS =  89.8 %)

Available components:
```

```
[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
"iter"         "ifault"
```

```
>v.pred <- as.integer(km.mod$cluster) # get cluster ids associated to given data objects
```

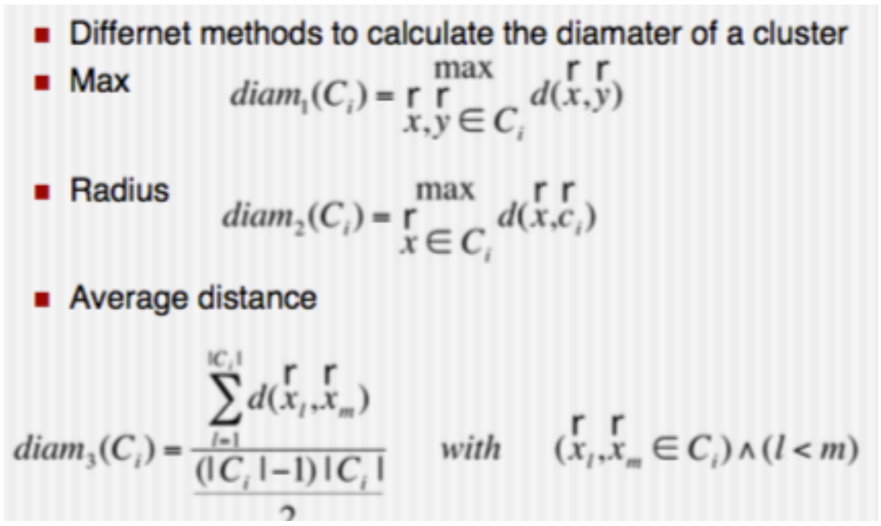
```
> v.pred
  [1] 3 1 1 1 3 2 1 3 1 1 2 3 1 1 2 2 2 3 2 3 3 3 1 3 3 1 3 3 3 1 1 3 2 2
 1 1 3 3 1 3 3 1 1 3 3 1 3 1 3 3 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 4 5 5 5 5
 [83] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 4 4 4 4 5 4 4 4 4 4 4 5 5 4
 4 4 4 5 4 5 4 5 4 4 5 5 4 4 4 4 4 5 4 4 4 4 5 4 4 4 5 4 4 4 5 4 4 5
```

```
> cls.scatt1 <- cls.scatt.data(iris.data, v.pred)
>
> cls.scatt1
$intracls.complete
         c1        c2        c3       c4       c5
[1,] 1.341641 0.7874008 0.9273618 2.418677 2.677686

$intracls.average
          c1        c2        c3       c4       c5
[1,] 0.4700329 0.5087326 0.4090316 1.022906 1.033869

$intracls.centroid
          c1        c2        c3        c4        c5
[1,] 0.3196929 0.3387864 0.2817023 0.7198385 0.7381524

$intercls.single
          c1        c2        c3        c4        c5
c1 0.0000000 0.7071068 0.2236068 4.0249224 1.7406895
c2 0.7071068 0.0000000 0.1000000 3.6891733 2.0566964
c3 0.2236068 0.1000000 0.0000000 3.7336309 1.6401219
c4 4.0249224 3.6891733 3.7336309 0.0000000 0.2645751
c5 1.7406895 2.0566964 1.6401219 0.2645751 0.0000000

$intercls.complete
         c1       c2       c3       c4       c5
c1 0.000000 2.428992 1.726268 7.085196 4.871345
c2 2.428992 0.000000 1.417745 6.519202 4.635731
c3 1.726268 1.417745 0.000000 6.627971 4.519956
c4 7.085196 6.519202 6.627971 0.000000 4.839421
c5 4.871345 4.635731 4.519956 4.839421 0.000000

$intercls.average
          c1        c2        c3       c4       c5
c1 0.0000000 1.3035969 0.7321721 5.255304 3.549028
c2 1.3035969 0.0000000 0.7466826 4.949310 3.474315
c3 0.7321721 0.7466826 0.0000000 4.962004 3.347606
```

```
c4 5.2553036 4.9493095 4.9620040 0.000000 1.950426
c5 3.5490283 3.4743155 3.3476064 1.950426 0.000000

$intercls.centroid
          c1        c2        c3       c4       c5
c1 0.0000000 1.2443197 0.6231342 5.220303 3.495418
c2 1.2443197 0.0000000 0.6402296 4.907235 3.402771
c3 0.6231342 0.6402296 0.0000000 4.925450 3.285593
c4 5.2203032 4.9072354 4.9254500 0.000000 1.797182
c5 3.4954178 3.4027710 3.2855928 1.797182 0.000000

$intercls.ave_to_cent
          c1        c2        c3       c4       c5
c1 0.0000000 1.2739025 0.6769235 5.240597 3.529076
c2 1.2739025 0.0000000 0.6917364 4.935953 3.456556
c3 0.6769235 0.6917364 0.0000000 4.946487 3.325952
c4 5.2405970 4.9359529 4.9464871 0.000000 1.867927
c5 3.5290756 3.4565560 3.3259522 1.867927 0.000000

$intercls.hausdorff
          c1       c2        c3       c4       c5
c1 0.0000000 1.679286 1.1269428 4.813523 2.503997
c2 1.4352700 0.000000 0.8306624 4.172529 2.605763
c3 0.8062258 0.781025 0.0000000 4.296510 2.204541
c4 6.3206012 6.039868 6.0671245 0.000000 2.395830
c5 4.2154478 4.114608 3.9661064 2.677686 0.000000

$cluster.center
       [,1]     [,2]     [,3]     [,4]
c1 4.678947 3.084211 1.378947 0.200000
c2 5.512500 4.000000 1.475000 0.275000
c3 5.100000 3.513043 1.526087 0.273913
c4 6.850000 3.073684 5.742105 2.071053
c5 5.901613 2.748387 4.393548 1.433871

$cluster.size
[1] 19  8 23 38 62

attr(,"class")
[1] "cls.list"
```

```
>  cls.scatt2 <- cls.scatt.data(iris.data, v.pred, dist="manhattan")
>       # the same using dissimilarity matrix
>     iris.diss.mx <- as.matrix(daisy(iris.data))
>     cls.scatt4 <- cls.scatt.diss.mx(iris.diss.mx, v.pred)
```

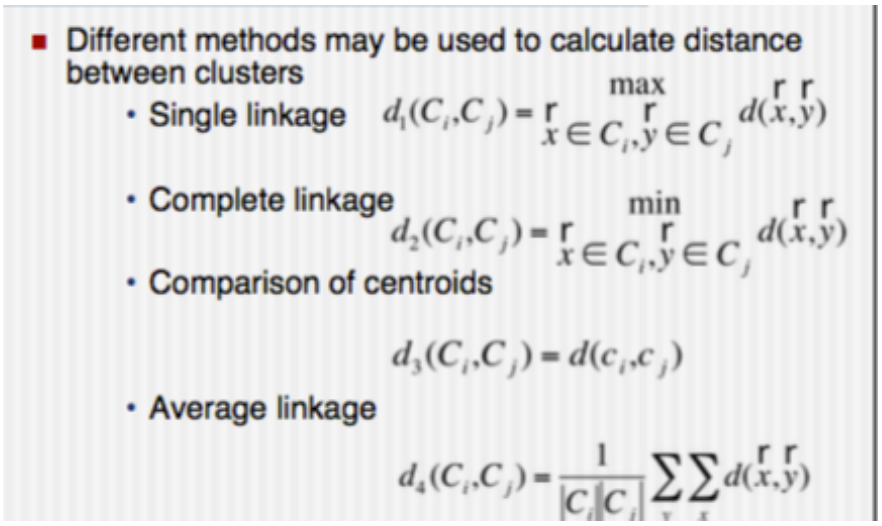
```
> intraclust = c("complete","average","centroid")
```



- Differnet methods to calculate the diamater of a cluster
- Max $diam_1(C_i) = \max_{\vec{x},\vec{y} \in C_i} d(\vec{x},\vec{y})$
- Radius $diam_2(C_i) = \max_{\vec{x} \in C_i} d(\vec{x},\vec{c}_i)$
- Average distance

$$diam_3(C_i) = \frac{\sum_{l=1}^{|C_i|} d(\vec{x}_l,\vec{x}_m)}{\frac{(|C_i|-1)|C_i|}{2}} \quad with \quad (\vec{x}_l,\vec{x}_m \in C_i) \wedge (l < m)$$

```
> interclust = c("single", "complete", "average","centroid", "aveToCent", "hausdorff")
```



- Different methods may be used to calculate distance between clusters
  - Single linkage $d_1(C_i,C_j) = \max_{\vec{x} \in C_i, \vec{y} \in C_j} d(\vec{x},\vec{y})$
  - Complete linkage $d_2(C_i,C_j) = \min_{\vec{x} \in C_i, \vec{y} \in C_j} d(\vec{x},\vec{y})$
  - Comparison of centroids $d_3(C_i,C_j) = d(c_i,c_j)$
  - Average linkage $d_4(C_i,C_j) = \frac{1}{|C_i||C_j|} \sum \sum d(\vec{x},\vec{y})$

```
> dunn1 <- clv.Dunn(cls.scatt1, intraclust, interclust)

> dunn1
           comp        ave       cent
sin   0.03734568 0.09672404 0.1354734
comp  0.52946646 1.37129994 1.9206667
ave   0.27343468 0.70818644 0.9918983
cent  0.23271373 0.60272056 0.8441810
aveto 0.25280171 0.65474774 0.9170512
haus  0.29167912 0.75543890 1.0580810
```

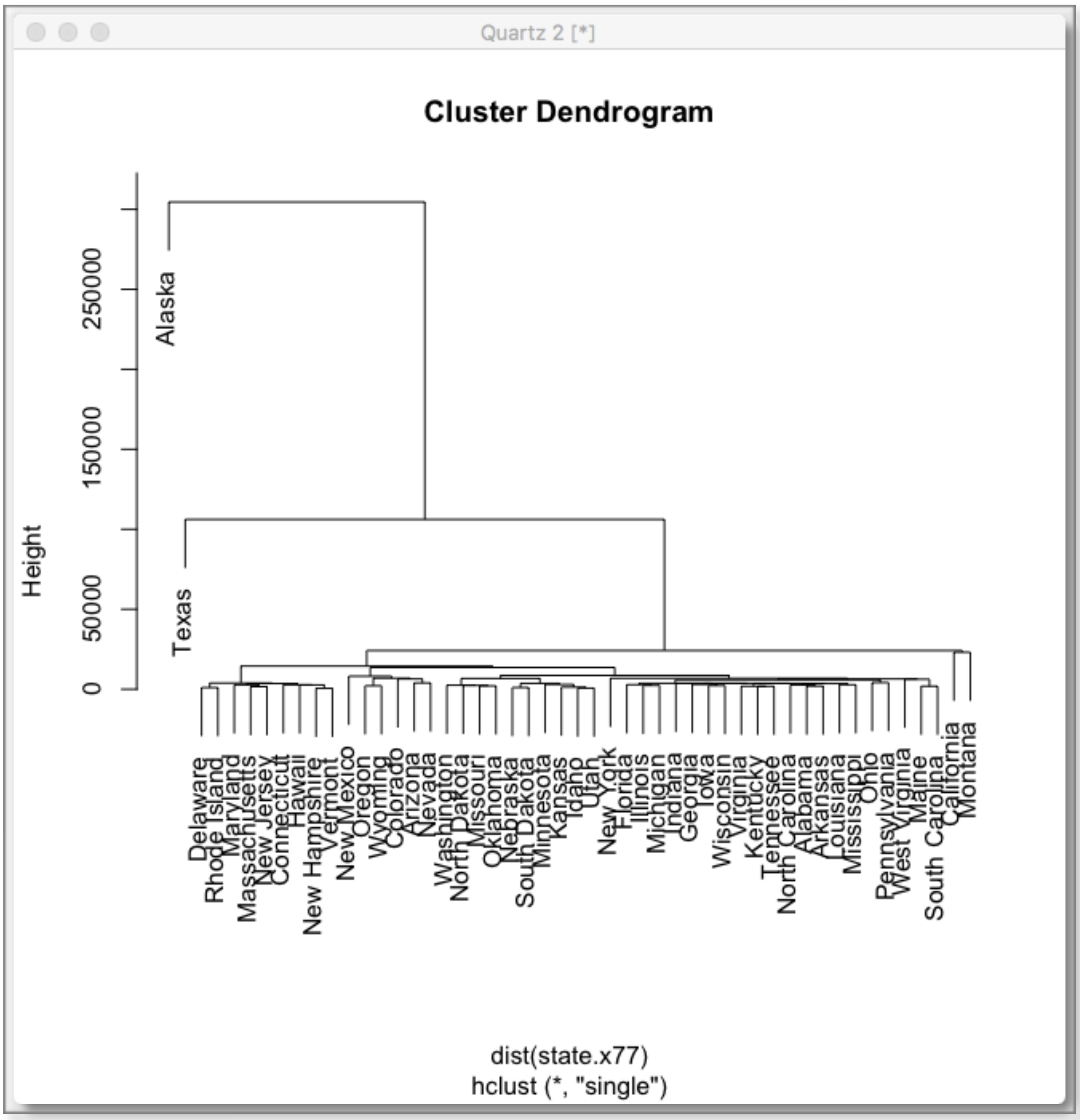
```
> davies1 <- clv.Davies.Bouldin(cls.scatt1, intraclust, interclust)
```

```
>  davies1
          comp       ave      cent
sin   16.593485 7.5668678 5.2241338
```

```
comp   1.188897 0.5307838 0.3652489
ave    2.744081 1.1535843 0.7956842
cent   3.126482 1.3133177 0.9051950
aveto  2.927899 1.2308652 0.8487012
haus   2.326155 0.9953464 0.6847682



> intraclust = c("complete")
> interclust = c("single")

> dunn2 <- clv.Dunn(cls.scatt1, intraclust, interclust)
> davies2 <- clv.Davies.Bouldin(cls.scatt1, intraclust, interclust)

> dunn2
          comp
sin 0.03734568


> davies2
        comp
sin 16.59348
>
```

Tree clustering

```
>data=data(state)
> help(state)
> head(state.x77,5)
          Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
Alabama          3615   3624       2.1    69.05   15.1    41.3    20  50708
Alaska            365   6315       1.5    69.31   11.3    66.7   152 566432
Arizona          2212   4530       1.8    70.55    7.8    58.1    15 113417
Arkansas         2110   3378       1.9    70.66   10.1    39.9    65  51945
California      21198   5114       1.1    71.71   10.3    62.6    20 156361

> h <- hclust(dist(state.x77), method = "single")
>help(hclust)
> plot(h)
```



Kohonen Map

```
> data(crabs)
> help(crabs)
```

Description
The crabs data frame has 200 rows and 8 columns, describing 5 morphological measurements on 50 crabs each of two colour forms and both sexes, of the species Leptograpsus variegatus collected at Fremantle, W. Australia.
….
species - "B" or "O" for blue or orange.
….

```
> head(crabs,5)
  sp sex index  FL  RW   CL   CW  BD
1  B   M     1 8.1 6.7 16.1 19.0 7.0
2  B   M     2 8.8 7.7 18.1 20.8 7.4
3  B   M     3 9.2 7.8 19.0 22.4 7.7
4  B   M     4 9.6 7.9 20.1 23.1 8.2
5  B   M     5 9.8 8.0 20.3 23.0 8.2
```

We will map this **5 dimensional** data on a **2 dimensional** Self Organizing Map (Kohonen Map)

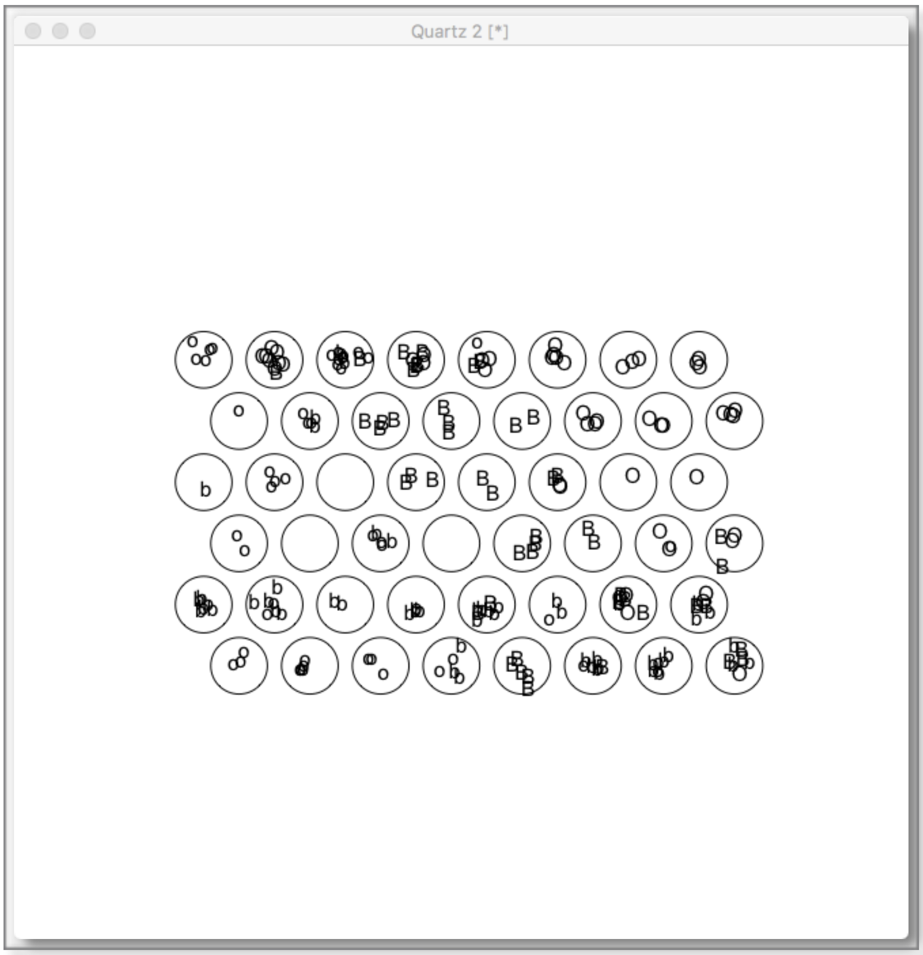
We will annotate B species B male and b for female
We will annotate O species O male and o for female

Execute the file by copying it into script

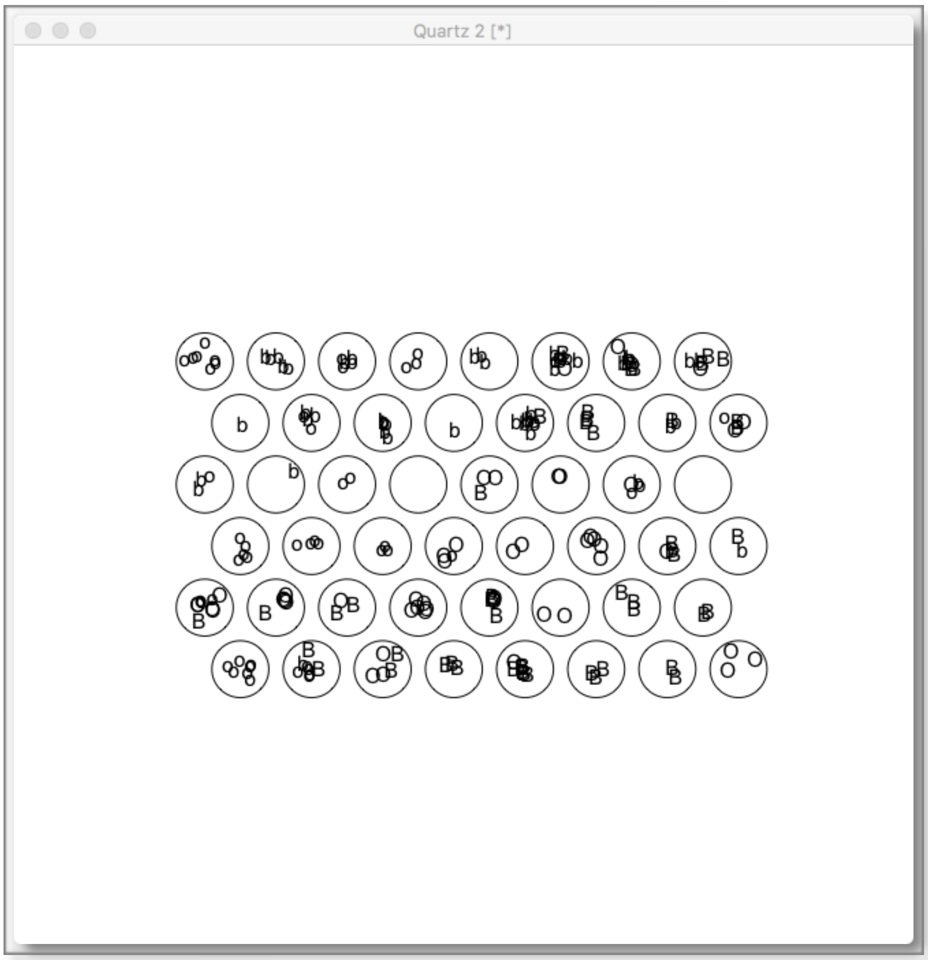
- Go to “File” in the top menu and click on “New script.”
- This opens up a new window that you can save as a .R file.
- To execute the code highlight the lines you wish to run, and press Ctrl-R on a PC or Command-Enter on a Mac.

```
library(class)
library(MASS)

require(graphics)
    data(crabs, package = "MASS")
    lcrabs <- log(crabs[, 4:8])
    crabs.grp <- factor(c("B", "b", "O", "o")[rep(1:4, rep(50,4))])
    gr <- somgrid(topo = "hexagonal")
    crabs.som <- batchSOM(lcrabs, gr, c(4, 4, 2, 2, 1, 1, 1, 0, 0))
    plot(crabs.som)
    bins <- as.numeric(knn1(crabs.som$code, lcrabs, 0:47))
    plot(crabs.som$grid, type = "n")
    symbols(crabs.som$grid$pts[, 1], crabs.som$grid$pts[, 2],
            circles = rep(0.4, 48), inches = FALSE, add = TRUE)
    text(crabs.som$grid$pts[bins, ] + rnorm(400, 0, 0.1),
         as.character(crabs.grp))
```

1*st* run



2*th* run



Each run will give you a different mapping. Do you know why?

3 Supervised Machine Learning

Supervised Machine Learning

Training and Test set

```
> x <- 1:12
> sample(x)
 [1]  2  5  4  7  9 11  6  8 12  3 10  1
> sample(x, replace = TRUE)
 [1]  4  9  8  4 11 10  4  3  2  4  5  9
> index=sample(x,6)
> index
[1] 12  4  9  8  3  6
 x[index]
[1] 12  4  9  8  3  6
> x[-index]
[1]  1  2  5  7 10 11

> data(iris)
> dim(iris)
[1] 150   5
```

KNN

```
>library(knn)
>help(knn)
```

Description
k-nearest neighbour classification for test set from training set. For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

```
data(iris)
library(class)
iidx <- sample(1:dim(iris)[1], 70)
iristrain <- iris[-iidx, ]
iristest <- iris[iidx, ]


irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],1)
table(irismodel,iristest[,5])
```

Experiments

```
> irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],1)
> table(irismodel,iristest[,5])

irismodel    setosa versicolor virginica
  setosa        23         0        0
  versicolor     0        22        1
  virginica      0         1       23


> irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],5)
> table(irismodel,iristest[,5])

irismodel    setosa versicolor virginica
  setosa        24         0        0
  versicolor     0        25        1
  virginica      0         5       15

> irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],19)
> table(irismodel,iristest[,5])

irismodel    setosa versicolor virginica
  setosa        26         0        0
  versicolor     0        21        2
  virginica      0         2       19


> irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],4)
> table(irismodel,iristest[,5])

irismodel    setosa versicolor virginica
  setosa        18         0        0
  versicolor     0        26        3
  virginica      0         0       23
```

LVQ

>data(iris)
>ibrary(class)

>help(lvqinit)

Description

Construct an initial codebook for LVQ methods.

Usage
lvqinit(x, cl, size, prior, k = 5)

Arguments

x:	a matrix or data frame of training examples, n by p.
cl:	the classifications for the training examples. A vector or factor of length n.
size:	*the size of the codebook. Defaults to **min(round(0.4*ng*(ng-1 + p/2),0), n)** where ng is the number of classes.*
prior:	*Probabilities to represent classes in the codebook. **Default proportions in the training set.***
k:	*k used for k-NN test of correct classification. **Default is 5**.*

```
> lvqinit(iristrain[, -5], iristrain[, 5])
$x
   Sepal.Length Sepal.Width Petal.Length Petal.Width
        17          5.4        3.9          1.3        0.4
        49          5.3        3.7          1.5        0.2
        64          6.1        2.9          4.7        1.4
        58          4.9        2.4          3.3        1.0
        113          6.8        3.0          5.5        2.1
        101          6.3        3.3          6.0        2.5
$cl
[1] setosa     setosa     versicolor versicolor virginica  virginica
Levels: setosa versicolor virginica

>cd <- lvqinit(iristrain[, -5], iristrain[, 5], 10)
> cd
$x
   Sepal.Length Sepal.Width Petal.Length Petal.Width
        34          5.5        4.2          1.4        0.2
        28          5.2        3.5          1.5        0.2
        30          4.7        3.2          1.6        0.2
        4          4.6        3.1          1.5        0.2
        53          6.9        3.1          4.9        1.5
        99          5.1        2.5          3.0        1.1
        71          5.9        3.2          4.8        1.8
        136          7.7        3.0          6.1        2.3
        129          6.4        2.8          5.6        2.1
        135          6.1        2.6          5.6        1.4
$cl
 [1] setosa     setosa     setosa     setosa     versicolor versicolor versicolor virginica  virginica
virginica
Levels: setosa versicolor virginica
```

```
> help(lvq1)
```

Description
Moves examples in a codebook to better represent the training set.

Usage
lvq1(x, cl, codebk, niter = 100 * nrow(codebk$x), alpha = 0.03)

```
> cd0 <- lvq1(iristrain[, -5], iristrain[, 5],cd)
> lvqtest(cd0, iristest[, -5])
 [1] versicolor versicolor setosa     virginica  versicolor setosa     setosa     virginica  setosa
versicolor virginica  versicolor versicolor virginica  virginica
[16] virginica  versicolor virginica  virginica  setosa     versicolor virginica  virginica  versicolor
versicolor setosa     versicolor setosa     virginica  versicolor
[31] virginica  versicolor virginica  versicolor virginica  versicolor setosa     setosa     virginica
versicolor versicolor virginica  setosa     versicolor virginica
[46] setosa     versicolor versicolor versicolor versicolor setosa     virginica  setosa     versicolor
virginica  setosa     setosa     virginica  virginica  versicolor
[61] versicolor virginica  virginica  versicolor setosa     versicolor setosa     versicolor versicolor
setosa
Levels: setosa versicolor virginica
> table(lvqtest(cd0, iristest[, -5]),iristest[, 5])

            setosa versicolor virginica
  setosa        18          0         0
  versicolor     0         26         3
  virginica      0          0        23
```

Back to KNN

Is k=4 the best value? Be careful with the sample!

```
> iidx <- sample(1:dim(iris)[1], 70)
> iristrain <- iris[-iidx, ]
> iristest <- iris[iidx, ]
> irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],4)
> table(irismodel,iristest[,5])

irismodel    setosa versicolor virginica
  setosa         22          0         0
  versicolor      0         19         0
  virginica       0          3        26


> irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],4,prob=TRUE)

irismodel    setosa versicolor virginica
  setosa         22          0         0
  versicolor      0         19         0
  virginica       0          3        26


> irismodel
 [1] versicolor virginica  virginica  virginica  virginica  versicolor virginica  setosa     versicolor setosa
setosa     versicolor setosa     virginica  versicolor
[16] virginica  versicolor setosa     virginica  virginica  virginica  virginica  setosa     virginica
virginica  virginica  setosa     setosa     setosa     versicolor
[31] virginica  setosa     versicolor setosa     virginica  setosa     virginica  virginica  setosa     setosa
setosa     virginica  versicolor versicolor setosa
[46] versicolor virginica  setosa     versicolor virginica  virginica  setosa     versicolor versicolor
versicolor virginica  virginica  versicolor setosa     virginica
[61] setosa     versicolor virginica  versicolor virginica  versicolor setosa     virginica  setosa
virginica
attr(,"prob")
 [1] 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.75 1.00 1.00 1.00 1.00 0.75
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.75 1.00 1.00 0.50
[34] 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.75 1.00
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.50 1.00
[67] 1.00 1.00 1.00 1.00
Levels: setosa versicolor virginica
```

To repeat the same sample you can set the seed of R random number generator.

Execute the file by copying it into script

- Go to “File” in the top menu and click on “New script.”
- This opens up a new window that you can save as a .R file.
- To execute the code highlight the lines you wish to run, and press Ctrl-R on a PC or Command-Enter on a Mac.
-

```
data(iris)
library(class)
set.seed(42)
iidx <- sample(1:dim(iris)[1], 70)
iristrain <- iris[-iidx, ]
iristest <- iris[iidx, ]
irismodel <- knn(iristrain[, -5],iristest[, -5], iristrain[,5],4)
table(irismodel,iristest[,5])
```

When you repeat the program it will give always the values (at my computer)

```
irismodel    setosa versicolor virginica
  setosa        21        0        0
  versicolor     0       21        3
  virginica      0        1       24
```

Or you can build the data set by hand, for example

```
>iris3
>help(iris3)
```

iris3 gives the same data arranged as a 3-dimensional array of size 50 by 4 by 3, as represented by S-PLUS. The first dimension gives the case number within the species subsample, the second the measurements with names Sepal L., Sepal W., Petal L., and Petal W., and the third the species.

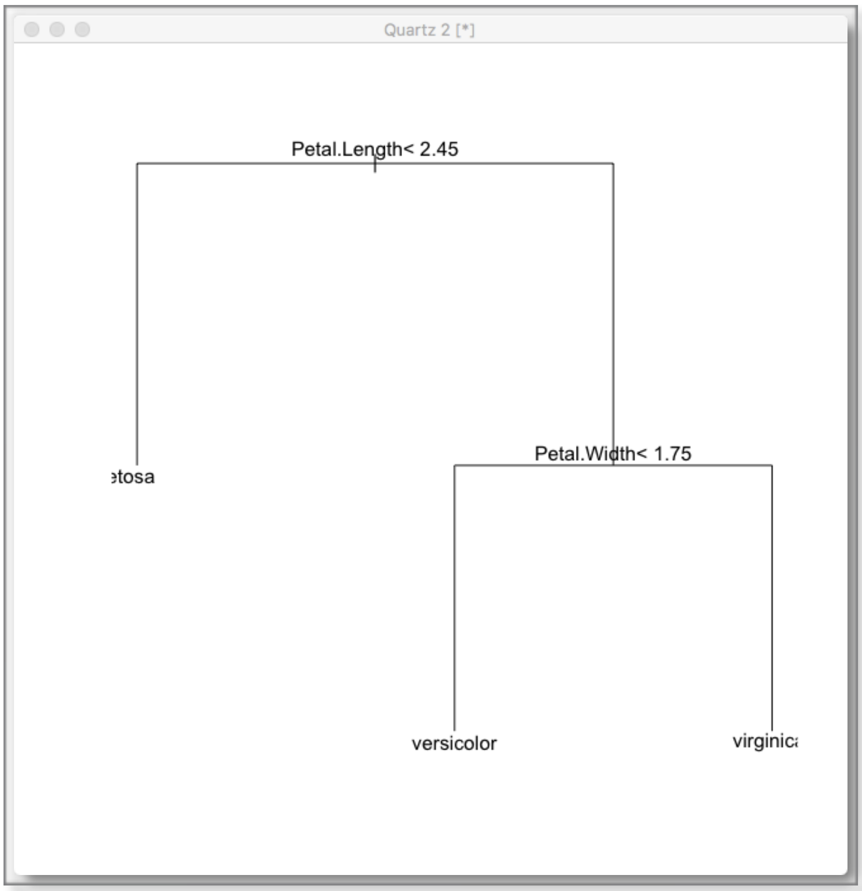
```
>train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
>test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
>cl <- factor(c(rep("s",25), rep("c",25), rep(“v",25)))
>cl
 [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c c c c c c c c c c c c c c c c c v v v v
v v v v v v v v v v v v v v v v v v v v v
Levels: c s v
> irismodel2=knn(train, test, cl, k = 3, prob=TRUE)
> table(irismodel2,cl)
          cl
irismodel2       c  s  v
                 c 23  0  4
                 s  0 25  0
                 v  2  0 21
```

```
>irismodel2
 [1] s s s s s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c c v c c c c c c c c c c c c c c c c c c v c c v
v v v v c v v v v c v v v v v v v v v v v
attr(,"prob")
 [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
[17] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
1.0000000 1.0000000 1.0000000 0.6666667 1.0000000 1.0000000 1.0000000 1.0000000
[33] 1.0000000 0.6666667 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
[49] 1.0000000 1.0000000 1.0000000 0.6666667 0.7500000 1.0000000 1.0000000 1.0000000
1.0000000 1.0000000 0.5000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
[65] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
1.0000000 1.0000000 0.6666667
Levels: c s v
```

Decision Tree

CART is implemented in the rpart package. Again using the formula, the command is
> cart_model <- rpart(y ~ x1 + x2, data=as.data.frame(cbind(y,x1,x2)), method="class") You can use plot.rpart and text.rpart to plot the decision tree.

```
library(rpart)
data(iris)
tree <- rpart(Species ~ ., data = iris)
plot(tree)
text(tree, digits = 3)
```



> help(rpart)

```
> tree
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
  2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
    6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259) *
    7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *
>
```

4 Advanced Supervised Machine Learning

SPAM data set

```
> library(kernlab)
> library(class)
> data(spam)
> help(spam)

> dim(spam)
[1] 4601   58

> set.seed(42)    #seed
> idx <- sample(1:dim(spam)[1], 300)
> spamtrain <- spam[-idx, ] #without this index
> spamtest <- spam[idx, ]

> dim(spamtest)
[1] 300  58
> dim(spamtrain)
[1] 4301   58
```

KNN and LVQ

```
> model <- knn(spamtrain[, -58],spamtest[, -58], spamtrain[,58],3)
>
> table(model,spamtest[,58])

model     nonspam spam
  nonspam       157   41
  spam              23   79
>
> cd <- lvqinit(spamtrain[, -58], spamtrain[, 58])
> cd0 <- olvq1(spamtrain[, -58], spamtrain[, 58],cd)
>
>
> table(lvqtest(cd0, spamtest[, -58]),spamtest[, 58])

                    nonspam spam
  nonspam               135   42
  spam                  45   78
>
```

Neural Network

```
>library(nnet)
>help(nnet)

> library(nnet)
> spam.nn2 <- nnet(type ~ ., data = spamtrain,  size = 2, rang = 0.1,   decay = 5e-4, maxit = 200)
# weights:  119
initial  value 2975.213619
iter  10 value 2105.128003
iter  20 value 1743.602215
iter  30 value 1458.226132
iter  40 value 1149.992092
iter  50 value 906.485129
iter  60 value 803.474196
iter  70 value 728.951006
iter  80 value 658.836717
iter  90 value 626.213186
iter 100 value 612.225881
iter 110 value 598.130847
iter 120 value 592.476345
iter 130 value 587.483021
iter 140 value 582.427904
iter 150 value 571.585729
iter 160 value 563.947090
iter 170 value 561.429112
iter 180 value 561.242177
iter 190 value 560.732887
iter 200 value 560.203116
final  value 560.203116
stopped after 200 iterations
>
> mailnn2 <- predict(spam.nn2, spamtest[,-58], type = "class")
>
> table(mailnn2, spamtest[, 58])

mailnn2   nonspam spam
  nonspam     171   12
  spam          9  108
>
```

```
> library(nnet)
> spam.nn2 <- nnet(type ~ ., data = spamtrain,  size = 10, rang = 0.1,
+               decay = 5e-4, maxit = 200)
# weights:  591
initial  value 2980.072169
iter  10 value 2157.539056
iter  20 value 1525.642020
iter  30 value 1114.731714
iter  40 value 812.092316
iter  50 value 723.865081
iter  60 value 663.558473
iter  70 value 611.796996
iter  80 value 556.845362
iter  90 value 501.314102
iter 100 value 465.580285
iter 110 value 445.245627
iter 120 value 422.522105
iter 130 value 398.848431
iter 140 value 390.878415
iter 150 value 389.238111
iter 160 value 386.039358
iter 170 value 383.360574
iter 180 value 380.878172
iter 190 value 377.606889
iter 200 value 373.510834
final  value 373.510834
stopped after 200 iterations
>
> mailnn2 <- predict(spam.nn2, spamtest[,-58], type = "class")
>
> table(mailnn2, spamtest[, 58])

mailnn2   nonspam spam
  nonspam     174   14
  spam          6  106
>
```

SVM

```
>library(kernlab)
>help(ksvm)

> filter <- ksvm(type ~ ., data = spamtrain, kernel = "rbfdot", kpar = list(sigma = 0.05),C = 5, cross
= 3)
> mailtype <- predict(filter, spamtest[,-58])
> table(mailtype, spamtest[, 58])

mailtype  nonspam spam
  nonspam     175   16
  spam          5  104
```

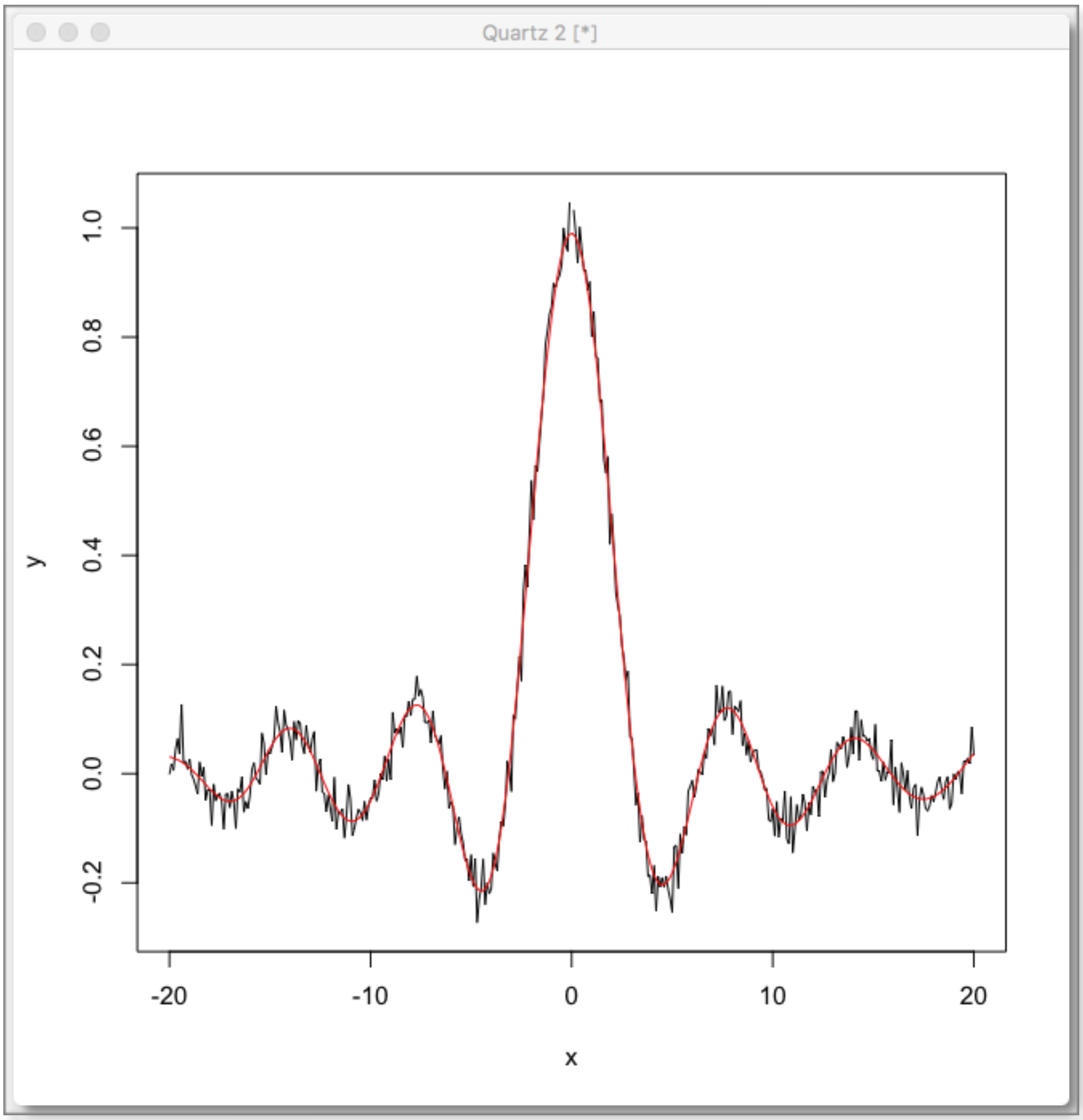
Approximation

Execute the file by copying it into script

- Go to “File” in the top menu and click on “New script.”
- This opens up a new window that you can save as a .R file.
- To execute the code highlight the lines you wish to run, and press Ctrl-R on a PC or Command-Enter on a Mac.

```
x <- seq(-20, 20, 0.1)
 y <- sin(x)/x + rnorm(401, sd = 0.03)
 plot(x, y, type = "l")
regm <- ksvm(x, y, epsilon = 0.02,    kpar = list(sigma = 16), cross = 3)
lines(x, predict(regm, x), col = "red")
rgm
```

```
Support Vector Machine object of class "ksvm"

SV type: eps-svr  (regression)
 parameter : epsilon = 0.02  cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  16

Number of Support Vectors : 344

Objective Function Value : -36.9118
Training error : 0.012147
Cross validation error : 0.000963
```


Quartz 2 [*]
y
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
x
-20
-10
0
10
20

Deep Neural Networks and R

- **install the package deepnet**

```
>library(deepnet)


#Create artificial data set, two dimension

>Var1 <- c(rnorm(50, 1, 0.5), rnorm(50, -0.6, 0.2))

>Var2 <- c(rnorm(50, -0.8, 0.2), rnorm(50, 2, 1))

>x <- matrix(c(Var1, Var2), nrow = 100, ncol = 2)

>head(x)

       [,1]       [,2]
[1,] 1.0574337 -0.7836990
[2,] 1.2360664 -0.5238815
[3,] 1.2537867 -0.8634439
[4,] 0.9294656 -1.1632775
[5,] 1.5729758 -0.6683861
[6,] 1.4750238 -0.7982185
>

#Create artificial binary data set, half of its elements is one, the other one zero


>y <- c(rep(1, 50), rep(0, 50))

>y

 [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 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [83] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

#train the network. It tries to clasify the values one an zero

 > nn <- nn.train(x, y, hidden = c(5))

#the architecture of the network

>nn$size

 2 5 1
```

Input dimension is two. Five hidden neurons in one layer, output dimension is one

```
#create test data set where the output is still y

> test_Var1 <- c(rnorm(50, 1, 0.5), rnorm(50, -0.6, 0.2))
 >test_Var2 <- c(rnorm(50, -0.8, 0.2), rnorm(50, 2, 1))
```

```
>test_x <- matrix(c(test_Var1, test_Var2), nrow = 100, ncol = 2)


>yy <- nn.predict(nn, test_x)

>yy

…..

>err <- nn.test(nn, test_x, y)
>print(err)

>0.25

#network did not leann

#train the network with more epochs

>nn2 <- nn.train(x, y, numepochs=10000, hidden = c(5))
>test_x2 <- matrix(c(test_Var1, test_Var2), nrow = 100, ncol = 2)
>yy2 <- nn.predict(nn2, test_x)
 > err <- nn.test(nn2, test_x, y)
>print(err)

0

#this time the network with the same architecture learned
```

Now we do the experiment with the iris data set.

```
>data(iris)
```

We divide the dataset into train and test as before. Sample for test has 70 entries the other one 80 since there are 150 entries.

```
>set.seed(42)
> iidx <- sample(1:dim(iris)[1], 70)
> iristrain <- iris[-iidx, ]
> iristest <- iris[iidx, ]
```

Represent the train input as matrix, as required by the package

```
>x <-
matrix(c(iristrain$Sepal.Length,iristrain$Sepal.Width,iristrain$Petal.Length,iristrain$Petal.Width),nro
w = 80, ncol = 4)


> head(x)
     [,1] [,2] [,3] [,4]
[1,]  4.9  3.0  1.4  0.2
[2,]  4.7  3.2  1.3  0.2
[3,]  5.4  3.9  1.7  0.4
[4,]  4.6  3.4  1.4  0.3
```

```
[5,]  5.0  3.4  1.5  0.2
[6,]  4.4  2.9  1.4  0.2
```

Represent the train output as a matrix, one of n coding, as required by the package

```
>v <- 1:80
>t=table(v,iristrain$Species)

>y <-matrix(t,nrow = 80, ncol = 3)
```

One of n coding, one of three coding

```
\>y
      [,1] [,2] [,3]
 [1,]    1    0    0
 [2,]    1    0    0
 [3,]    1    0    0
 [4,]    1    0    0
 [5,]    1    0    0
 [6,]    1    0    0
 [7,]    1    0    0
 [8,]    1    0    0
 [9,]    1    0    0
[10,]    1    0    0
[11,]    1    0    0
[12,]    1    0    0
[13,]    1    0    0
[14,]    1    0    0
[15,]    1    0    0
[16,]    1    0    0
[17,]    1    0    0
[18,]    1    0    0
[19,]    1    0    0
[20,]    1    0    0
[21,]    1    0    0
[22,]    1    0    0
[23,]    1    0    0
[24,]    1    0    0
[25,]    1    0    0
[26,]    1    0    0
[27,]    1    0    0
[28,]    1    0    0
[29,]    1    0    0
[30,]    0    1    0
[31,]    0    1    0
[32,]    0    1    0
[33,]    0    1    0
[34,]    0    1    0
[35,]    0    1    0
[36,]    0    1    0
[37,]    0    1    0
[38,]    0    1    0
[39,]    0    1    0
[40,]    0    1    0
```

```
[41,]   0   1   0
[42,]   0   1   0
[43,]   0   1   0
[44,]   0   1   0
[45,]   0   1   0
[46,]   0   1   0
[47,]   0   1   0
[48,]   0   1   0
[49,]   0   1   0
[50,]   0   1   0
[51,]   0   1   0
[52,]   0   1   0
[53,]   0   1   0
[54,]   0   1   0
[55,]   0   1   0
[56,]   0   1   0
[57,]   0   1   0
[58,]   0   0   1
[59,]   0   0   1
[60,]   0   0   1
[61,]   0   0   1
[62,]   0   0   1
[63,]   0   0   1
[64,]   0   0   1
[65,]   0   0   1
[66,]   0   0   1
[67,]   0   0   1
[68,]   0   0   1
[69,]   0   0   1
[70,]   0   0   1
[71,]   0   0   1
[72,]   0   0   1
[73,]   0   0   1
[74,]   0   0   1
[75,]   0   0   1
[76,]   0   0   1
[77,]   0   0   1
[78,]   0   0   1
[79,]   0   0   1
[80,]   0   0   1
>
```

```
>nniris <- nn.train(x,y, hidden = c(5,5))
```

```
>nniris$size
[1] 4 5 5 3
```

Input dimension is four. Two hidden layers, five hidden neurons in each layer, output dimension is three.

test set

```
>x_test <-
matrix(c(iristest$Sepal.Length,iristest$Sepal.Width,iristest$Petal.Length,iristest$Petal.Width),nrow
= 70, ncol = 4)
```

```
> head(x_test)
     [,1] [,2] [,3] [,4]
[1,]  6.4  3.1  5.5  1.8
[2,]  6.9  3.1  5.4  2.1
[3,]  4.4  3.2  1.3  0.2
[4,]  7.7  2.8  6.7  2.0
[5,]  5.0  2.3  3.3  1.0
[6,]  6.6  3.0  4.4  1.4
```

```
>v_test <- 1:70
>t_test=table(v_test,iristest$Species)

>y_test <-matrix(t_test,nrow = 70, ncol = 3)
```

One of n coding, one of three coding

```
> y_test
      [,1] [,2] [,3]
 [1,]    0    0    1
 [2,]    0    0    1
 [3,]    1    0    0
 [4,]    0    0    1
 [5,]    0    1    0
 [6,]    0    1    0
 [7,]    0    0    1
 [8,]    1    0    0
 [9,]    0    0    1
[10,]    0    1    0
[11,]    0    1    0
[12,]    0    0    1
[13,]    0    0    1
[14,]    1    0    0
[15,]    0    1    0
[16,]    0    0    1
[17,]    0    0    1
[18,]    1    0    0
[19,]    0    0    1
[20,]    0    1    0
[21,]    0    0    1
[22,]    1    0    0
[23,]    0    0    1
[24,]    0    0    1
[25,]    1    0    0
[26,]    0    0    1
[27,]    1    0    0
[28,]    0    0    1
[29,]    0    1    0
[30,]    0    0    1
[31,]    0    1    0
[32,]    0    1    0
[33,]    1    0    0
[34,]    0    1    0
[35,]    1    0    0
```

```
[36,]   0   1   0
[37,]   0   0   1
[38,]   1   0   0
[39,]   0   0   1
[40,]   0   1   0
[41,]   1   0   0
[42,]   1   0   0
[43,]   1   0   0
[44,]   0   0   1
[45,]   0   0   1
[46,]   0   0   1
[47,]   0   1   0
[48,]   0   1   0
[49,]   0   0   1
[50,]   0   0   1
[51,]   1   0   0
[52,]   0   0   1
[53,]   1   0   0
[54,]   0   1   0
[55,]   1   0   0
[56,]   0   1   0
[57,]   0   1   0
[58,]   0   0   1
[59,]   1   0   0
[60,]   1   0   0
[61,]   0   1   0
[62,]   0   1   0
[63,]   0   1   0
[64,]   1   0   0
[65,]   0   0   1
[66,]   1   0   0
[67,]   1   0   0
[68,]   0   1   0
[69,]   0   1   0
[70,]   0   0   1
>
>


>nniris <- nn.train(x,y, hidden = c(5,5))

>yy <- nn.predict(nniris, x_test)

>yy

…

> err <- nn.test(nniris, x_test, y_test)
>print(err)

[1] 0.5


#Second try :-)

>nniris <- nn.train(x,y, numepochs=10000,hidden = c(5,5))
```

```
>yy <- nn.predict(nniris, x_test)

>yy

>yy
            [,1]        [,2]         [,3]
 [1,] 0.0006877613 0.095885377 9.192647e-01
 [2,] 0.0001885404 0.001656604 9.993384e-01
 [3,] 0.9883058793 0.015507435 3.492028e-05
 [4,] 0.0001606711 0.001124474 9.996028e-01
 [5,] 0.0179159735 0.991404193 9.621886e-04
 [6,] 0.0123183058 0.993562215 1.142341e-03
 [7,] 0.0004031639 0.016377601 9.895921e-01
 [8,] 0.9886069074 0.015240143 3.452447e-05
 [9,] 0.0001611213 0.001130053 9.995999e-01
[10,] 0.0127389561 0.993411706 1.123660e-03
[11,] 0.0167997806 0.991839040 9.904991e-04
[12,] 0.0001562720 0.001058102 9.996347e-01
[13,] 0.0001622738 0.001148812 9.995909e-01
[14,] 0.9878950982 0.015877571 3.544933e-05
[15,] 0.0142080983 0.992878517 1.065821e-03
[16,] 0.0006475765 0.078449821 9.359939e-01
[17,] 0.0008197358 0.166211683 8.475741e-01
[18,] 0.9888149657 0.015057385 3.424672e-05
[19,] 0.0001568832 0.001067148 9.996304e-01
[20,] 0.0125766574 0.993476065 1.129341e-03
[21,] 0.0001868614 0.001622029 9.993562e-01
[22,] 0.9885018530 0.015332967 3.466342e-05
[23,] 0.0040562603 0.978866960 1.030650e-02
[24,] 0.0001610466 0.001129387 9.996003e-01
[25,] 0.9886229127 0.015226114 3.450320e-05
[26,] 0.0001680783 0.001244871 9.995441e-01
[27,] 0.9886133146 0.015234581 3.451594e-05
[28,] 0.0001953937 0.001820261 9.992542e-01
[29,] 0.0083675809 0.994412477 1.519456e-03
[30,] 0.0001954119 0.001819712 9.992544e-01
[31,] 0.0143046682 0.992832815 1.063315e-03
[32,] 0.0131069748 0.993280153 1.107878e-03
[33,] 0.9877793373 0.015982353 3.559637e-05
[34,] 0.0143114691 0.992833387 1.062904e-03
[35,] 0.9885541186 0.015286826 3.459437e-05
[36,] 0.0157640989 0.992277524 1.016912e-03
[37,] 0.0001638030 0.001172807 9.995792e-01
[38,] 0.9872406044 0.016477021 3.626812e-05
[39,] 0.0013179807 0.518906380 4.497262e-01
[40,] 0.0173669822 0.991653539 9.739836e-04
[41,] 0.9850081347 0.018604138 3.886625e-05
[42,] 0.9881850079 0.015615778 3.507724e-05
[43,] 0.9886031323 0.015243524 3.452946e-05
[44,] 0.0001597311 0.001109290 9.996100e-01
[45,] 0.0039527129 0.977292632 1.126871e-02
```

Looks better, the probability value is indicated

```
>err <- nn.test(nniris, x_test, y_test)
```

```
>print(err)

[1] 0.07142857
```

Now we use three hidden layers. Will it get better?

```
>nniris <- nn.train(x,y, numepochs=10000,,hidden = c(5,5,5))

>yy <- nn.predict(nniris, x_test)

>err <- nn.test(nniris, x_test, y_test)
>print(err)

 0.01428571
```

Overfitting

Now four hidden layers. Maybe it is too much?

```
>nniris <- nn.train(x,y, numepochs=10000,hidden_dropout = 0.000,hidden = c(5,5,5,5))

>yy <- nn.predict(nniris, x_test)

>err <- nn.test(nniris, x_test, y_test)
>print(err)
 0.5
```

This is bad!!! :-(

Regularization- dropout

- The term "dropout" refers to dropping out units (mostly in hidden layer) in a neural network. This is mostly done randomly.
- Considering, millions of parameters to be learned, regularization becomes an imperative requisite to prevent overfitting.
- Dropout is trivial to implement and generally results into faster learning.
- A default value of 0.5 is a good choice for less complex model a dropout of 0.2 is a good choice.

```
>nniris <- nn.train(x,y, numepochs=10000,hidden_dropout = 0.2,hidden = c(5,5))

yy <- nn.predict(nniris, x_test)
>
>err <- nn.test(nniris, x_test, y_test)
>print(err)
0.04285714
```

```
>nniris <- nn.train(x,y, numepochs=10000,hidden_dropout = 0.1,hidden = c(5,5))
```

```
>yy <- nn.predict(nniris, x_test)

>err <- nn.test(nniris, x_test, y_test)
>print(err)
0.07142857
```

Random Forest

- RF is an ensemble classifier that consist of many decision trees. Decisions are taken using a majority vote from the trees.
- Number of training cases be N, and the number of variables in the classifier be M.
- m is the number of input variables to be used to determine the decision at a node of the tree; m should be much less than M.
- Sample a training set for this tree by choosing N times with replacement from all N available training cases (i.e. take a bootstrap sample).
- For each node of the tree, randomly choose m variables on which to base the decision at that node.
- Calculate the best split based on these m variables in the training set.

```
> library(randomForest)
randomForest 4.6-12
Type rfNews() to see new features/changes/bug fixes.
> filter <- randomForest(type ~ .,    data = spam)
> filter

Call:
 randomForest(formula = type ~ ., data = spam)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 7

        OOB estimate of  error rate: 4.43%
Confusion matrix:
        nonspam spam class.error
nonspam    2716   72  0.02582496
spam        132 1681  0.07280750
>
```

```
set.seed(42)    #seed
idx <- sample(1:dim(spam)[1], 300)
spamtrain <- spam[-idx, ] #without this index
spamtest <- spam[idx, ]
```

```
library(randomForest)
filter <- randomForest(type ~ .,    data = spamtrain)
mailtype <- predict(filter, spamtest[,-58])
table(mailtype, spamtest[, 58])
```

```
mailtype  nonspam spam
  nonspam     178   13
  spam          2  107
```

Cross Validation in R

```
#Randomly shuffle the data

data(iris)
library(class)

yourData<-iris[sample(nrow(iris)),]


yourData<-yourData[sample(nrow(yourData)),]

#Create 4 equally size folds
folds <- cut(seq(1,nrow(yourData)),breaks=4,labels=FALSE)

truepositive <- 1:4

#Perform 4 fold cross validation
for(i in 1:4){
   #Segement your data by fold using the which() function
   testIndexes <- which(folds==i,arr.ind=TRUE)
   testData <- yourData[testIndexes, ]
   trainData <- yourData[-testIndexes, ]
   irismodel <- knn(trainData[, -5],testData[, -5], trainData[,5],1)
   results <-table(irismodel,testData[,5])
   print(results)

   truepositive[i]=results[1]+results[5]++results[9]

   cat("TP=",truepositive[i])

   #Use the test and train data partitions however you desire...
}

print("results:")
m=mean(truepositive)
s=sd(truepositive)

cat("Mean=",m)
cat("Stdev=",s)
```

Unbalanced Dataset and R

A dataset is said to be unbalanced when the class of interest (minority class) is much rarer than normal behaviour (majority class). The cost of missing a minority class is typically much higher that missing a majority class. Most learning systems are not prepared to cope with unbalanced data and several techniques have been proposed to rebalance the classes.

Generally, these methods aim to modify an imbalanced data into balanced distribution using some mechanism.

install the packages

install.packages("data.table", dependencies=TRUE)
install.packages("parallelMap", dependencies=TRUE)

further on install:

- **foreach**
- **IdoParallel**
- **ParamHelpers**
- **BBmisc**
- **checkmate**
- **backports**
- **mlr**

- **RANN**
- **FNN**

- **unbalanced**

Undersampling

This method works with majority class. It **reduces** the number of observations from majority class to make the data set balanced.

ubUnder Under-sampling

```
> library(unbalanced)
 >data(ubIonosphere)
```

```
>head(ubIonosphere, 3)
```

```
       V3      V4      V5      V6      V7      V8      V9     V10     V11     V12     V13     V14     V15     V16     V17     V18     V19     V20     V21
1 0.99539 -0.05889 0.85243  0.02306  0.83398 -0.37708 1.00000  0.03760 0.85243 -0.17755 0.59755 -0.44945  0.60536 -0.38223 0.84356 -0.38542
0.58212 -0.32192  0.56971
2 1.00000 -0.18829 0.93035 -0.36156 -0.10868 -0.93597 1.00000 -0.04549 0.50874 -0.67743 0.34432 -0.69707 -0.51685 -0.97515 0.05499 -0.62237
0.33109 -1.00000 -0.13151
3 1.00000 -0.03365 1.00000  0.00485  1.00000 -0.12062 0.88965  0.01198 0.73082  0.05346 0.85443  0.00827  0.54591  0.00299 0.83775 -0.13644
0.75535 -0.08540  0.70887
      V22      V23      V24      V25      V26      V27      V28      V29      V30      V31      V32      V33      V34 Class
1 -0.29674  0.36946 -0.47357  0.56811 -0.51171  0.41078 -0.46168  0.21266 -0.34090  0.42267 -0.54487  0.18641 -0.45300     0
2 -0.45300 -0.18056 -0.35734 -0.20332 -0.26569 -0.20468 -0.18401 -0.19040 -0.11593 -0.16626 -0.06288 -0.13738 -0.02447     1
3 -0.27502  0.43385 -0.12062  0.57528 -0.40220  0.58984 -0.22145  0.43100 -0.17365  0.60436 -0.24180  0.56045 -0.38238     0
```

```
>summary(ubIonosphere)
```

```
 V3               V4                V5               V6               V7               V8                V9                V10               V11
 Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.0000   Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.00000   Min.   :-1.00000   Min.   :-1.00000
 1st Qu.: 0.4721   1st Qu.:-0.06474   1st Qu.: 0.4127   1st Qu.:-0.0248   1st Qu.: 0.2113   1st Qu.:-0.05484   1st Qu.: 0.08711   1st Qu.:-0.04807   1st Qu.:
0.02112
 Median : 0.8711   Median : 0.01631   Median : 0.8092   Median : 0.0228   Median : 0.7287   Median : 0.01471   Median : 0.68421   Median : 0.01829
Median : 0.66798
 Mean   : 0.6413   Mean   : 0.04437   Mean   : 0.6011   Mean   : 0.1159   Mean   : 0.5501   Mean   : 0.11936   Mean   : 0.51185   Mean   : 0.18135   Mean   :
0.47618
 3rd Qu.: 1.0000   3rd Qu.: 0.19418   3rd Qu.: 1.0000   3rd Qu.: 0.3347   3rd Qu.: 0.9692   3rd Qu.: 0.44567   3rd Qu.: 0.95324   3rd Qu.: 0.53419   3rd Qu.:
0.95790
 Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000   Max.   :
1.00000
     V12               V13              V14               V15              V16               V17              V18                V19              V20
 Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.000000   Min.   :-1.0000   Min.   :-1.00000
 1st Qu.:-0.06527   1st Qu.: 0.0000   1st Qu.:-0.07372   1st Qu.: 0.0000   1st Qu.:-0.08170   1st Qu.: 0.0000   1st Qu.:-0.225690   1st Qu.: 0.0000   1st
Qu.:-0.23467
 Median : 0.02825   Median : 0.6441   Median : 0.03027   Median : 0.6019   Median : 0.00000   Median : 0.5909   Median : 0.000000   Median : 0.5762
Median : 0.00000
 Mean   : 0.15504   Mean   : 0.4008   Mean   : 0.09341   Mean   : 0.3442   Mean   : 0.07113   Mean   : 0.3819   Mean   :-0.003617   Mean   : 0.3594
Mean   :-0.02402
 3rd Qu.: 0.48237   3rd Qu.: 0.9555   3rd Qu.: 0.37486   3rd Qu.: 0.9193   3rd Qu.: 0.30897   3rd Qu.: 0.9357   3rd Qu.: 0.195285   3rd Qu.: 0.8993   3rd
Qu.: 0.13437
 Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.000000   Max.   : 1.0000   Max.   :
1.00000
     V21              V22                V23              V24               V25              V26               V27              V28               V29
 Min.   :-1.0000   Min.   :-1.000000   Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000
 1st Qu.: 0.0000   1st Qu.:-0.243870   1st Qu.: 0.0000   1st Qu.:-0.36689   1st Qu.: 0.0000   1st Qu.:-0.33239   1st Qu.: 0.2864   1st Qu.:-0.44316   1st Qu.:
0.0000
 Median : 0.4991   Median : 0.000000   Median : 0.5318   Median : 0.00000   Median : 0.5539   Median :-0.01505   Median : 0.7082   Median :-0.01769
Median : 0.4966
 Mean   : 0.3367   Mean   : 0.008296   Mean   : 0.3625   Mean   :-0.05741   Mean   : 0.3961   Mean   :-0.07119   Mean   : 0.5416   Mean   :-0.06954   Mean
: 0.3784
 3rd Qu.: 0.8949   3rd Qu.: 0.188760   3rd Qu.: 0.9112   3rd Qu.: 0.16463   3rd Qu.: 0.9052   3rd Qu.: 0.15676   3rd Qu.: 0.9999   3rd Qu.: 0.15354   3rd
Qu.: 0.8835
 Max.   : 1.0000   Max.   : 1.000000   Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000   Max.   :
1.0000
     V30               V31              V32                V33              V34             Class
 Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.000000   Min.   :-1.0000   Min.   :-1.00000   0:225
 1st Qu.:-0.23689   1st Qu.: 0.0000   1st Qu.:-0.242595   1st Qu.: 0.0000   1st Qu.:-0.16535   1:126
 Median : 0.00000   Median : 0.4428   Median : 0.000000   Median : 0.4096   Median : 0.00000
 Mean   :-0.02791   Mean   : 0.3525   Mean   :-0.003794   Mean   : 0.3494   Mean   : 0.01448
 3rd Qu.: 0.15407   3rd Qu.: 0.8576   3rd Qu.: 0.200120   3rd Qu.: 0.8138   3rd Qu.: 0.17166
 Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.000000   Max.   : 1.0000   Max.   : 1.00000
```

```
 >n<-ncol(ubIonosphere)

>n
>33

>output<-ubIonosphere$Class

output
  [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
 [83] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
[165] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
[247] 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[329] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
Levels: 0 1
>

 > input<-ubIonosphere[ ,-n]     #Without the Class

> data<-ubUnder(X=input, Y= output, perc = 40,  method = "percPos")

> newData<-cbind(data$X, data$Y)

>summary(newData)

>…. 0:189
>….1:126

> data<-ubUnder(X=input, Y= output)

> newData<-cbind(data$X, data$Y)

>summary(newData)

>…. 0:126
>….1:126
```

Oversampling

This method works with minority class. It r**eplicates** the observations from minority class to balance the data.

ubOver Over-sampling

```
> library(unbalanced)
>data(ubIonosphere)
>n<-ncol(ubIonosphere)
>output<-ubIonosphere$Class
>input<-ubIonosphere[ ,-n]
 >data<-ubOver(X=input, Y= output)
 >newData<-cbind(data$X, data$Y)
```

Data Import Export

Read.table creates a data frame from the values and tries to guess the type of each variable

```
> help(read.table)
> mydata <- read.table("file.csv", sep = ",")
```

```
> library(xlsReadWrite)
> data <- read.xls("sampledata.xls")
```

Save data by write.table() or save()

```
write.table(mydata, file = "mydata.csv",quote = FALSE)
> save(mydata, file = "mydata.rda")
> load(file = "mydata.rda")
```

Additional Methods

Logistic Regression

```
help(glm)
glm_mod <-glm(y ~ x1+x2, family=binomial(link="logit"), data=as.data.frame(cbind(y,x1,x2))
```

Apriori

```
library(arules)
help(apriori)
```

Na¨ıve Bayes

```
library(e1071 )
nB_model <- naiveBayes(y ~ x1 + x2, data=as.data.frame(cbind(y,x1,x2)))
```

AdaBoost

```
library(rpart)
library(ada)

boost_model <- ada(x=X, y=labels)
```

Bibliography

- Machine Learning in R, Alexandros Karatzoglou, Telefonica Research Barcelona, Spain
- https://alexiskz.wordpress.com

- Allison Chang, R for Machine Learning. MIT OpenCourseWare, Prediction: Machine Learning and Statistics
- https://ocw.mit.edu/

- http://www.r-project.org

- A. Wichert. Intelligent Big Multimedia Databases, World Scientific, 2015

- [L3] Decision Support Systems
- http://web.ist.utl.pt/andreas.wichert/