Data Mining and Machine Learning

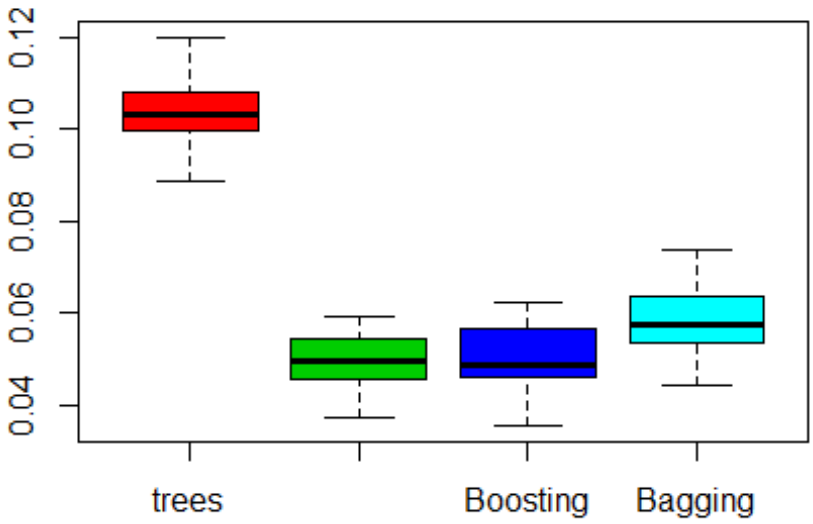
Sailee Rumao

December 20, 2018

```
###Task:

Comparing the methods using Decision Trees, Bagging, Boosting, Random Forest
on spam dataset, prostrate dataset, breast cancer dataset and DNA dataset.

library(adabag)

## Loading required package: rpart

## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

## Loading required package: foreach

## Loading required package: doParallel

## Loading required package: iterators

## Loading required package: parallel

library(ada)
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

library(mlbench)
library(ipred)

##
## Attaching package: 'ipred'
```

```
## The following object is masked from 'package:adabag':
##
##     bagging


#SPAM dataset
spam<-read.csv(file.choose(),header = T,sep=",")
xy.spam <- spam


p   <- ncol(xy.spam)-1
pos.spam <- 1+p
y.spam   <- xy.spam[,1]
n.spam   <- nrow(xy.spam)
colnames(xy.spam)[1] <- 'y'



#  Set the total number of replications

R <- 40

#  Initialize the test error vector

err.spam <- matrix(0, ncol=4, nrow=R)

for(r in 1:R)
{

  id.F.spam    <- which(xy.spam$y == 'spam')
  n.F.spam     <- length(id.F.spam)
  id.F.tr.spam <- sample(sample(sample(id.F.spam)))[1:round(0.75*n.F.spam)]
  id.F.te.spam <- setdiff(id.F.spam, id.F.tr.spam)

  id.S.spam <- which(xy.spam$y == 'nonspam')
  n.S.spam  <- length(id.S.spam)
  id.S.tr.spam <- sample(sample(sample(id.S.spam)))[1:round(0.75*n.S.spam)]
  id.S.te.spam <- setdiff(id.S.spam, id.S.tr.spam)

  xy.tr.spam <- xy.spam[c(id.F.tr.spam,id.S.tr.spam), ]
  xy.te.spam <- xy.spam[c(id.F.te.spam,id.S.te.spam), ]
  ntr.spam <- nrow(xy.tr.spam)
  nte.spam <- n.spam - ntr.spam

  tree.xy.spam <- rpart(y~., data=xy.tr.spam)
  yhat.tree.spam <- predict(tree.xy.spam, xy.te.spam[,-1], type="class")
  err.tree.spam <- 1-sum(diag(table(xy.te.spam$y, yhat.tree.spam)))/nte.spam
```

```
  err.spam[r,1] <- err.tree.spam


  forest.xy.spam <- randomForest(y~., data=xy.tr.spam)
  yhat.forest.spam <- predict(forest.xy.spam, xy.te.spam[,-1], type='class')
  err.forest.spam <- 1-sum(diag(table(xy.te.spam$y,
yhat.forest.spam)))/nte.spam

  err.spam[r,2] <- err.forest.spam


  boost.xy.spam <- ada(y~., data=xy.tr.spam)
  yhat.boost.spam <- predict(boost.xy.spam, xy.te.spam[,-1])
  err.boost.spam <- 1-sum(diag(table(xy.te.spam$y,
yhat.boost.spam)))/nte.spam

  err.spam[r,3] <- err.boost.spam

  bagging.xy.spam <- bagging(y~ ., data=xy.tr.spam, nbagg= 50, coob=FALSE)
  yhat.bagging.te.spam <- predict(bagging.xy.spam, xy.te.spam[,-1])
  err.spam[r,4] <- 1-sum(diag(table(xy.te.spam$y,
yhat.bagging.te.spam)))/nte.spam


  if (r%%40==0)  cat('\n', round(100*r/R,0),'% completed\n')
}

##
##  100 % completed

cols.spam <- c(2,3,4,5)
metodos.spam <- c('trees','Random Forest','Boosting','Bagging')

#Boxplots
boxplot(err.spam, col=cols.spam, names=metodos.spam,title= "Boxplots:Spam
data")
```




```
#Calculating average test errors
avg.err <- round(apply(err.spam, 2, mean),4)
avg.err.spam <- avg.err
avg.acc <- 1-avg.err.spam


#Computing gain in accuracy and reduction in error relative to single tree.

gain_rf     <- round(100*((avg.acc[1]-avg.acc[2])/avg.acc[1]),2)
decrease_rf <- round(100*((avg.err[1]-avg.err[2])/avg.err[1]),2)
cat('\n The gain in accuracy for Random forest is: ',  gain_rf ,'%\n')

##
##  The gain in accuracy for Random forest is:  -6.13 %

cat('\n The reduction in error for Random forest is: ',  decrease_rf ,'%\n')

##
##  The reduction in error for Random forest is:  52.64 %

gain_boost     <- round(100*((avg.acc[1]-avg.acc[3])/avg.acc[1]),2)
decrease_boost <- round(100*((avg.err[1]-avg.err[3])/avg.err[1]),2)
cat('\n The gain in accuracy for Random forest is: ',  gain_boost ,'%\n')

##
##  The gain in accuracy for Random forest is:  -6.03 %
```

```r
cat('\n The reduction in error for Random forest is: ',  decrease_boost
,'%\n')
```

```
##
##  The reduction in error for Random forest is:  51.77 %
```

```r
gain_bag     <- round(100*((avg.acc[1]-avg.acc[4])/avg.acc[1]),2)
decrease_bag <- round(100*((avg.err[1]-avg.err[4])/avg.err[1]),2)
cat('\n The gain in accuracy for Random forest is: ',  gain_bag ,'%\n')
```

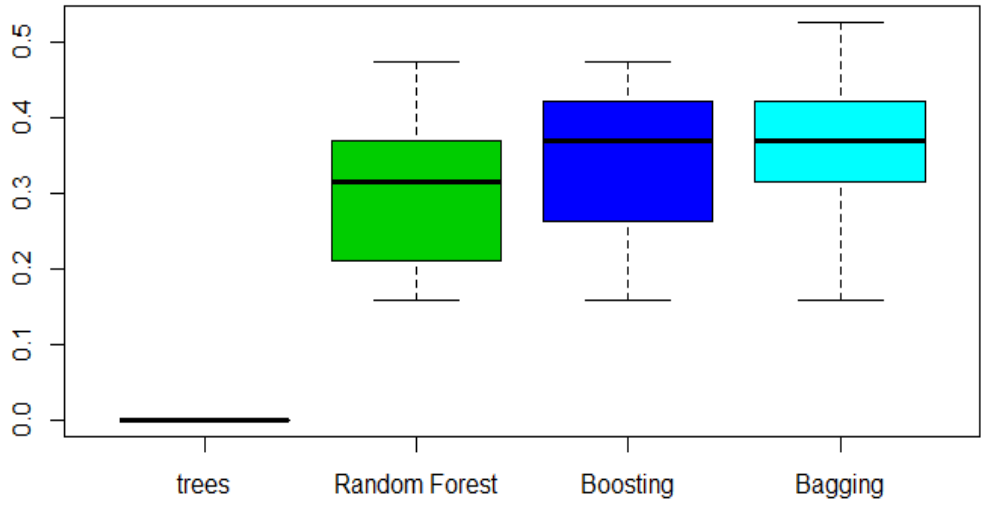
```
##
##  The gain in accuracy for Random forest is:  -5.1 %
```

```r
cat('\n The reduction in error for Random forest is: ',  decrease_bag ,'%\n')
```

```
##
##  The reduction in error for Random forest is:  43.82 %
```

```r
#Prostate dataset


prostate <-read.csv(file.choose(),header = T,sep=",")
xy.pc <- prostate


p.pc   <- ncol(xy.pc)-1
pos.pc <- 1+p.pc
y   <- xy.pc[,1]
n.pc  <- nrow(xy.pc)
colnames(xy.pc)[1] <- 'y'



#  Set the total number of replications

R <- 40

#  Initialize the test error vector

err.pc <- matrix(0, ncol=4, nrow=R)

for(r in 1:R)
{

  id.F.pc   <- which(xy.pc$y == 0)
  n.F.pc     <- length(id.F.pc)
  id.F.tr.pc <- sample(sample(sample(id.F.pc)))[1:round(0.75*n.F.pc)]
  id.F.te.pc <- setdiff(id.F.pc, id.F.tr.pc)

  id.S.pc <- which(xy.pc$y == 1)
  n.S.pc  <- length(id.S.pc)
  id.S.tr.pc <- sample(sample(sample(id.S.pc)))[1:round(0.75*n.S.pc)]
  id.S.te.pc <- setdiff(id.S.pc, id.S.tr.pc)

  xy.tr.pc <- xy.pc[c(id.F.tr.pc,id.S.tr.pc), ]
  xy.te.pc <- xy.pc[c(id.F.te.pc,id.S.te.pc), ]
  ntr.pc <- nrow(xy.tr.pc)
  nte.pc <- n.pc - ntr.pc

  tree.xy.pc <- rpart(y~., data=xy.tr.pc)
  yhat.tree.pc <- predict(tree.xy.pc, xy.te.pc[,-1], type="class")
  err.tree.pc <- 1-sum(diag(table(xy.te.pc$y, yhat.tree.pc)))/nte.pc

  err.pc[r,1] <- err.tree.pc
```

```
  forest.xy.pc <- randomForest(y~., data=xy.tr.pc)
  yhat.forest.pc <- predict(forest.xy.pc, xy.te.pc[,-1], type='class')
  err.forest.pc <- 1-sum(diag(table(xy.te.pc$y, yhat.forest.pc)))/nte.pc

  err.pc[r,2] <- err.forest.pc


  boost.xy.pc<- ada(y~., data=xy.tr.pc)
  yhat.boost.pc <- predict(boost.xy.pc, xy.te.pc[,-1])
  err.boost.pc <- 1-sum(diag(table(xy.te.pc$y, yhat.boost.pc)))/nte.pc

  err.pc[r,3] <- err.boost.pc

  bagging.xy.pc <- bagging(y~ ., data=xy.tr.pc, nbagg= 50, coob=FALSE)
  yhat.bagging.te.pc <- predict(bagging.xy.pc, xy.te.pc[,-1])
  err.pc[r,4] <- 1-sum(diag(table(xy.te.pc$y, yhat.bagging.te.pc)))/nte.pc


  if (r%%40==0)  cat('\n', round(100*r/R,0),'% completed\n')
}

##
##  100 % completed

cols.pc <- c(2,3,4,5)
metodos.pc <- c('trees','Random Forest','Boosting','Bagging')

#Boxplots
boxplot(err.pc, col=cols.pc, names=metodos.pc,title= "Boxplots:prostate
data")
```




```
#Calculating average test errors
avg.err <- round(apply(err.pc, 2, mean),4)
avg.err.pc <- avg.err
avg.acc.pc <- 1-avg.err.pc


#Computing gain in accuracy and reduction in error relative to single tree.

gain_rf.pc     <- round(100*((avg.acc.pc[1]-avg.acc.pc[2])/avg.acc.pc[1]),2)
decrease_rf.pc <- round(100*((avg.err.pc[1]-avg.err.pc[2])/avg.err.pc[1]),2)
cat('\n The gain in accuracy for Random forest is: ',  gain_rf.pc ,'%\n')
```

```
##
##  The gain in accuracy for Random forest is:   30.13 %
```

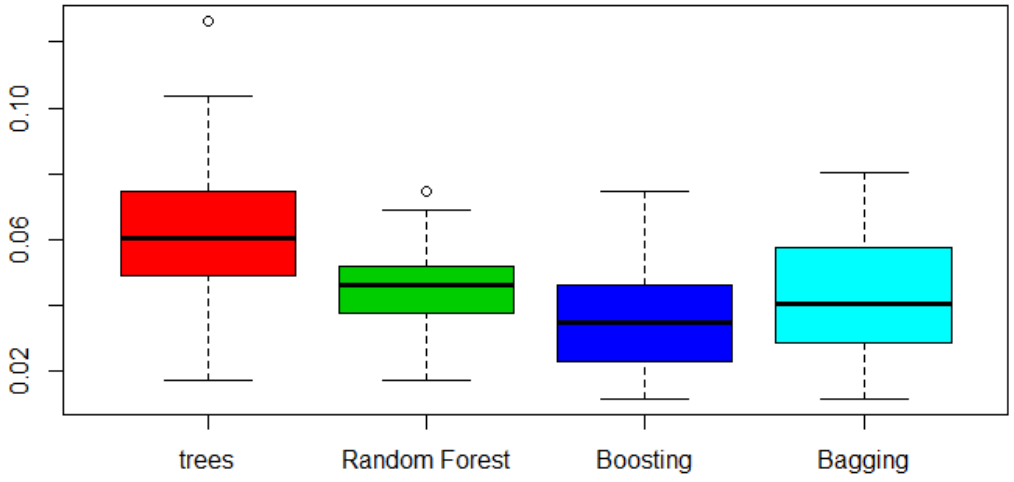
```
cat('\n The reduction in error for Random forest is: ',  decrease_rf.pc
,'%\n')


##  The reduction in error for Random forest is:  -Inf %



gain_boost.pc     <- round(100*((avg.acc.pc[1]-
avg.acc.pc[3])/avg.acc.pc[1]),2)
decrease_boost.pc <- round(100*((avg.err.pc[1]-
```

```
avg.err.pc[3])/avg.err.pc[1]),2)
cat('\n The gain in accuracy for boosting is: ',  gain_boost.pc ,'%\n')

##
##  The gain in accuracy for boosting is:  32.76 %

cat('\n The reduction in error for boosting is: ',  decrease_boost.pc ,'%\n')

##
##  The reduction in error for boosting is:  -Inf %

gain_bag.pc     <- round(100*((avg.acc.pc[1]-avg.acc.pc[4])/avg.acc.pc[1]),2)
decrease_bag.pc <- round(100*((avg.err.pc[1]-avg.err.pc[4])/avg.err.pc[1]),2)
cat('\n The gain in accuracy for bagging is: ',  gain_bag.pc ,'%\n')

##
##  The gain in accuracy for bagging is:  34.47 %

cat('\n The reduction in error for bagging is: ',  decrease_bag.pc ,'%\n')

##
##  The reduction in error for bagging is:  -Inf %
```

```
#BreastCancer dataset


data("BreastCancer")
BreastCancer<- BreastCancer[,-1]
BreatCancer<-na.omit(BreastCancer)

xy.bc<-BreastCancer
```

```r
p   <- ncol(xy.bc)-1
pos.bc <- 1+p
y.bc   <- xy.bc[,pos]
n.bc   <- nrow(xy.bc)
colnames(xy.bc)[pos] <- 'y'



#  Set the total number of replications

R <- 40

#  Initialize the test error vector

err.bc <- matrix(0, ncol=4, nrow=R)

for(r in 1:R)
{

  id.F.bc    <- which(xy.bc$y == 'benign')
  n.F.bc     <- length(id.F.bc)
  id.F.tr.bc <- sample(sample(sample(id.F.bc)))[1:round(0.75*n.F.bc)]
  id.F.te.bc <- setdiff(id.F.bc, id.F.tr.bc)

  id.S.bc <- which(xy.bc$y == 'malignant')
  n.S.bc  <- length(id.S.bc)
  id.S.tr.bc <- sample(sample(sample(id.S.bc)))[1:round(0.75*n.S.bc)]
  id.S.te.bc <- setdiff(id.S.bc, id.S.tr.bc)

  xy.tr.bc <- xy.bc[c(id.F.tr.bc,id.S.tr.bc), ]
  xy.te.bc <- xy.bc[c(id.F.te.bc,id.S.te.bc), ]
  ntr.bc <- nrow(xy.tr.bc)
  nte.bc <- n.bc - ntr.bc

  tree.xy.bc <- rpart(y~., data=xy.tr.bc)
  yhat.tree.bc <- predict(tree.xy.bc, xy.te.bc[,-pos], type="class")
  err.tree.bc <- 1-sum(diag(table(xy.te.bc$y, yhat.tree.bc)))/nte.bc

  err.bc[r,1] <- err.tree.bc


  forest.xy.bc <- randomForest(y~., data=xy.tr.bc)
  yhat.forest.bc <- predict(forest.xy.bc, xy.te.bc[,-pos], type='class')
  err.forest.bc <- 1-sum(diag(table(xy.te.bc$y, yhat.forest.bc)))/nte.bc

  err.bc[r,2] <- err.forest.bc
```

```
  boost.xy.bc <- ada(y~., data=xy.tr.bc)
  yhat.boost.bc <- predict(boost.xy.bc, xy.te.bc[,-pos])
  err.boost.bc <- 1-sum(diag(table(xy.te.bc$y, yhat.boost.bc)))/nte.bc

  err.bc[r,3] <- err.boost.bc

  bagging.xy.bc <- bagging(y~ ., data=xy.tr.bc, nbagg= 50, coob=FALSE)
  yhat.bagging.te.bc <- predict(bagging.xy.bc, xy.te.bc[,-pos])
  err.bc[r,4] <- 1-sum(diag(table(xy.te.bc$y, yhat.bagging.te.bc)))/nte.bc


  if (r%%40==0)  cat('\n', round(100*r/R,0),'% completed\n')
}

##
##  100 % completed

cols.bc <- c(2,3,4,5)
metodos.bc <- c('trees','Random Forest','Boosting','Bagging')

#Boxplots
boxplot(err.bc, col=cols.bc, names=metodos.bc,title= "Boxplots:Spam data")
```

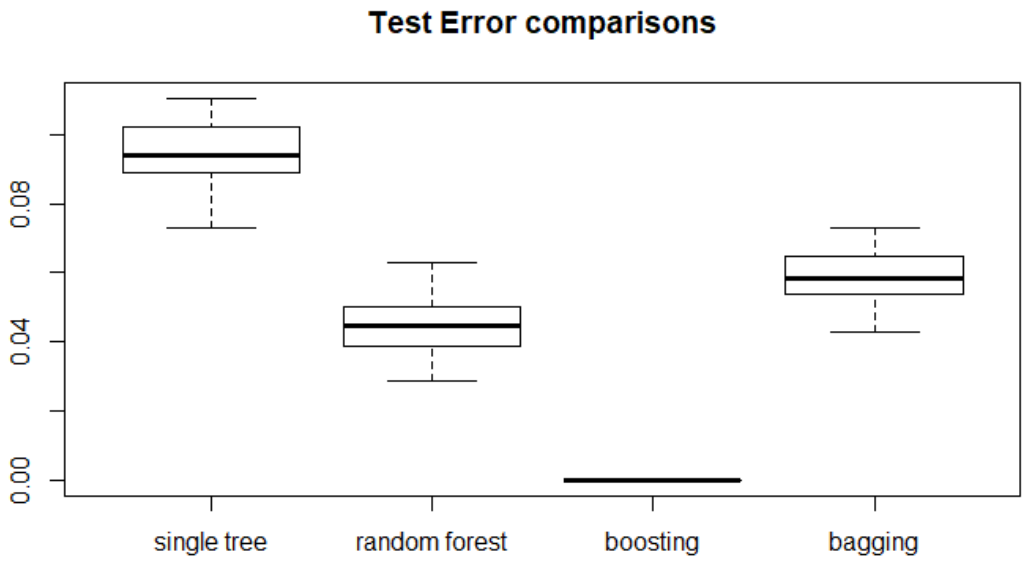


```r
#Calculating average test errors
avg.err <- round(apply(err.bc, 2, mean),4)
avg.err.bc <- avg.err
avg.acc <- 1-avg.err.bc


#Computing gain in accuracy and reduction in error relative to single tree.

gain_rf.bc     <- round(100*((avg.acc[1]-avg.acc[2])/avg.acc[1]),2)
decrease_rf.bc <- round(100*((avg.err[1]-avg.err[2])/avg.err[1]),2)
cat('\n The gain in accuracy for Random forest is: ',  gain_rf.bc ,'%\n')

##
##  The gain in accuracy for Random forest is:  -1.69 %

cat('\n The reduction in error for Random forest is: ',  decrease_rf.bc
,'%\n')

##
##  The reduction in error for Random forest is:  25.9 %

gain_boost.bc     <- round(100*((avg.acc[1]-avg.acc[3])/avg.acc[1]),2)
decrease_boost.bc <- round(100*((avg.err[1]-avg.err[3])/avg.err[1]),2)
cat('\n The gain in accuracy for boosting is: ',  gain_boost.bc ,'%\n')

##
##  The gain in accuracy for boosting  is:  -2.79 %

cat('\n The reduction in error for boosting is: ',  decrease_boost.bc ,'%\n')

##
##  The reduction in error for boosting is:  42.67 %

gain_bag.bc     <- round(100*((avg.acc[1]-avg.acc[4])/avg.acc[1]),2)
decrease_bag.bc <- round(100*((avg.err[1]-avg.err[4])/avg.err[1]),2)


cat('\n The gain in accuracy for bagging is: ',  gain_bag.bc ,'%\n')

##
##  The gain in accuracy for bagging is:  -2.06 %

cat('\n The reduction in error for bagging is: ',  decrease_bag.bc ,'%\n')

##
##  The reduction in error for bagging is:  31.43 %
```

```r
#DNA dataset

#using stratified holdout


data("DNA")


stratified.holdout <- function(y, ptr)
{
  n              <- length(y)
  labels         <- unique(y)       # Obtain classifiers
  id.tr.dna <- id.te.dna <- NULL
  # Loop once for each unique label value
  for(j in 1:length(labels))
  {
    sj    <- which(y==labels[j])  # Grab all rows of label type j
    nj    <- length(sj)           # Count of label j rows to calc proportion
below

    id.tr.dna <- c(id.tr.dna, (sample(sample(sample(sj))))[1:round(nj*ptr)])
  }                                # Concatenates each label type together 1
by 1

  id.te.dna  <- (1:n) [-id.tr.dna]          # Obtain and Shuffle test indices
to randomize

  return(list(idx1=id.tr.dna,idx2=id.te.dna))
}


library(kernlab)

library(MASS)

library(dplyr)
```

```
XY <- DNA


pos.dna   <- ncol(XY)
colnames(XY)[1pos.dna] <- 'Y.dna'

R <- 40

test.err.dna <- matrix(1, nrow=R ,ncol=4)

Ptr= 3/4

for(r in 1:R)
{


 hold <- stratified.holdout(as.factor(XY[,pos.dna]), ptr)
  id.tr.dna <- hold$idx1
  id.te.dna<- hold$idx2



  tree.xy.dna <- rpart(Y.dna~., data=XY[id.tr.dna,])
  test.err.dna[r, 1] <-1-
sum(diag(prop.table(table(XY[id.te.dna,pos.dna],predict(tree.xy.dna,XY[id.te.
dna,-pos.dna], type="class"))))


 forest.xy.dna <- randomForest(Y.dna~., data=XY[id.tr.dna,])
  test.err.dna[r, 2] <-1-
sum(diag(prop.table(table(XY[id.te.dna,pos.dna],predict(tree.xy.dna,XY[id.te.
dna,-pos.dna], type="class"))))


 #boost.xy.dna <- ada(Y.dna~., data=XY[id.tr.dna,])
  #test.err.dna[r, 3] <-1-
sum(diag(prop.table(table(XY[id.te.dna,pos.dna],predict(boost.xy.dna,XY[id.te
.dna,-pos.dna], type="class"))))


 bagging.xy.dna <- bagging(Y.dna~., data=XY[id.tr.dna,])
  test.err.dna[r, 4] <-1-
sum(diag(prop.table(table(XY[id.te.dna,pos.dna],predict(bagging.xy.dna,XY[id.
te.dna,-pos.dna], type="class"))))


}


#Boxplots
```

```
boxplot(test.err.dna, names = c('single tree','random
forest','boosting','bagging'), main='Test Error comparisons:DNA data')
```




```
#Calculating average test errors
avg.err.dna <- round(apply(test.err.dna, 2, mean),4)
avg_err_dna <- avg.err.dna
avg.acc.dna <- 1-avg.err.dna


#Computing gain in accuracy and reduction in error relative to single tree.

gain_rf.dna    <- round(100*((avg.acc.dna[1]-
avg.acc.dna[2])/avg.acc.dna[1]),2)
decrease_rf.dna <- round(100*((avg.err.dna[1]-
avg.err.dna[2])/avg.err.dna[1]),2)
cat('\n The gain in accuracy for Random forest is: ',  gain_rf.dna ,'%\n')
```

```
##
##  The gain in accuracy for Random forest is:   -5.55 %
```

```
cat('\n The reduction in error for Random forest is: ',  decrease_rf.dna
,'%\n')
```

```
##  The reduction in error for Random forest is:  52.95%
```

```
gain_boost.dna    <- round(100*((avg.acc.dna[1]-
avg.acc.dna[3])/avg.acc.dna[1]),2)
decrease_boost.dna <- round(100*((avg.err.dna[1]-
avg.err.dna[3])/avg.err.dna[1]),2)
cat('\n The gain in accuracy for boosting is: ',  gain_boost.dna ,'%\n')
```

```
##
##  The gain in accuracy for boosting is:  -10.47 %
```

```
cat('\n The reduction in error for boosting is: ',  decrease_boost.dna
,'%\n')
```

```
##
##  The reduction in error for boosting is:  100 %
```

```
gain_bag.dna    <- round(100*((avg.acc.dna[1]-
avg.acc.dna[4])/avg.acc.dna[1]),2)
decrease_bag.dna <- round(100*((avg.err.dna[1]-
avg.err.dna[4])/avg.err.dna[1]),2)
cat('\n The gain in accuracy for bagging is: ',  gain_bag.dna ,'%\n')
```

```
##
##  The gain in accuracy for bagging is:  -3.94 %
```

```
cat('\n The reduction in error for bagging is: ',  decrease_bag.dna ,'%\n')
```

```
##
##  The reduction in error for bagging is:  37.66 %
```

```
#Calculating the tables

#4.3
#Average test errors for all the data sets across all the methods.

spam.<-c(0.1063,0.0497,0.0503,0.0589,"RandomForest")
prostate.<-c(1,0.6658,0.6421,0.6421,"Boosting")
breastCancer.<-c(0.0629,0.0500,0.349,0.0436,"Boosting")
dna.<-c(0.948,0.0446,0,0.0591,"Boosting")


Avg_error_table<-data.frame(spam.,prostate.,breastCancer.,dna.)
row.names(Avg_error_table)<-c("Tree","randomforest","Boost","Bag","Winner")
Avg_error_table
```

```
##                     spam. prostate. breastCancer.     dna.
## Tree               0.1063         1        0.0629    0.948
```

```
## randomforest       0.0497    0.6658          0.05   0.0446
## Boost              0.0503    0.6421         0.349        0
## Bag                0.0589    0.6421        0.0436   0.0591
## Winner       RandomForest  Boosting      Boosting Boosting
```

```
#4.4
#Reductions in error relative to a single tree for all the datasets

spam.1<-c(52.64,43.82,51.77)
prostate.1<-c(NA,NA,NA)
breastcancer.1<-c(25.9,42.67,31.43)
dna.1<-c(52.95,37.66,100)

error_reduction_table<-data.frame(spam.1,prostate.1,breastcancer.1,dna.1)
row.names(error_reduction_table)<-c("Randomforest","Bag","boosting")
error_reduction_table
```

```
##              spam.1 prostate.1 breastcancer.1  dna.1
## Randomforest  52.64         NA          25.90  52.95
## Bag           43.82         NA          42.67  37.66
## boosting      51.77         NA          31.43 100.00
```

```
#4.5

#Gain in Accuracy relative to a single tree for all the datasets

spam.2<-c(-6.13,-5.1,-6.03)
prostate.2<-c(30.13,34.47,32.76)
breastcancer.2<-c(-1.69,-2.06,-2.79)
dna.2<-c(-5.55,-3.94,-10.47)

Accuracy_gain_table<-data.frame(spam.2,prostate.2,breastcancer.2,dna.2)
row.names(Accuracy_gain_table)<-c("Randomforest","Bag","boosting")
Accuracy_gain_table
```

```
##              spam.2 prostate.2 breastcancer.2  dna.2
## Randomforest  -6.13      30.13          -1.69  -5.55
## Bag           -5.10      34.47          -2.06  -3.94
## boosting      -6.03      32.76          -2.79 -10.47
```

```
##4.5

n/p for all the datasets are as follows:

spam : 79.32 >1

Prostate : 0.15768 < 1

BreastCancer : 69.9 > 1
```

DNA: 17.6 >1

Analysis:

All except the prostate dataset have the n/p ratio less than 1 and the effect of this can be clearly seen in its predictive performance.

For the Prostate dataset n<<<<<p and we can compare its performance with other datasets by looking at the average error,reduction in error and gain in accuracy tables calculated above.

It does not do very well in the comparison to the other datasets. This could be due to the curse of dimensionality.

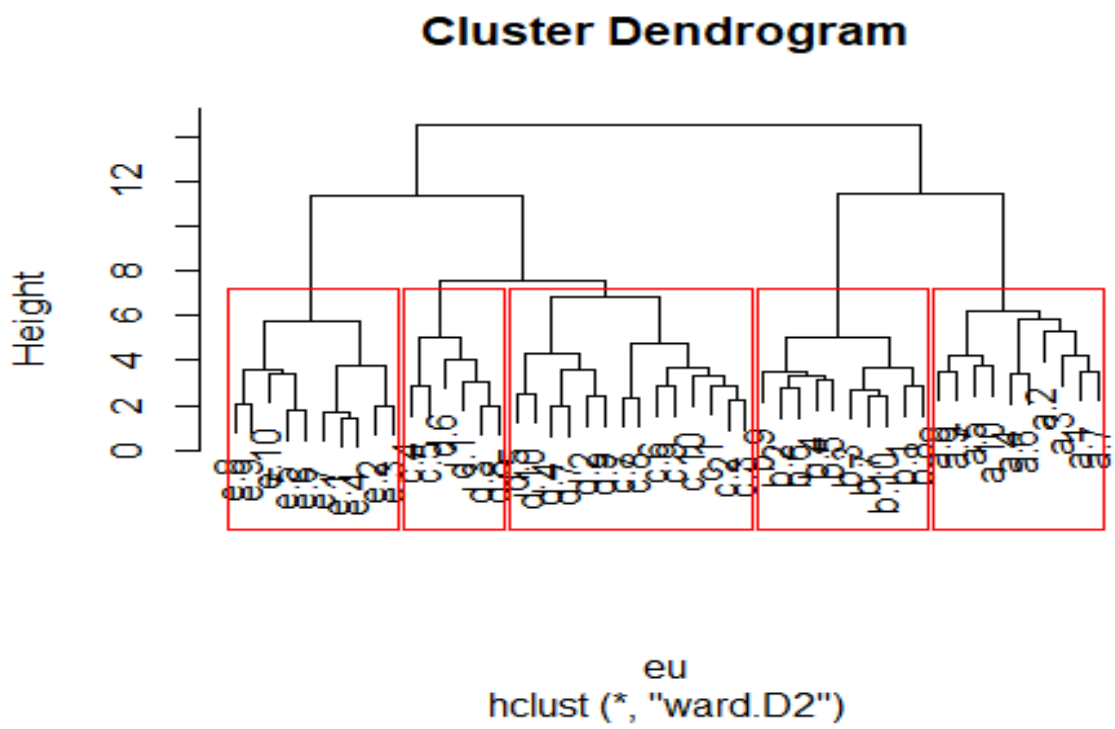
Comparing all the Methods: Single tree, random forest, bagging and boosting, we can definitely say that the performance of single tree is low in comparison to others.

Random forest and boosting (mostly) does pretty well.

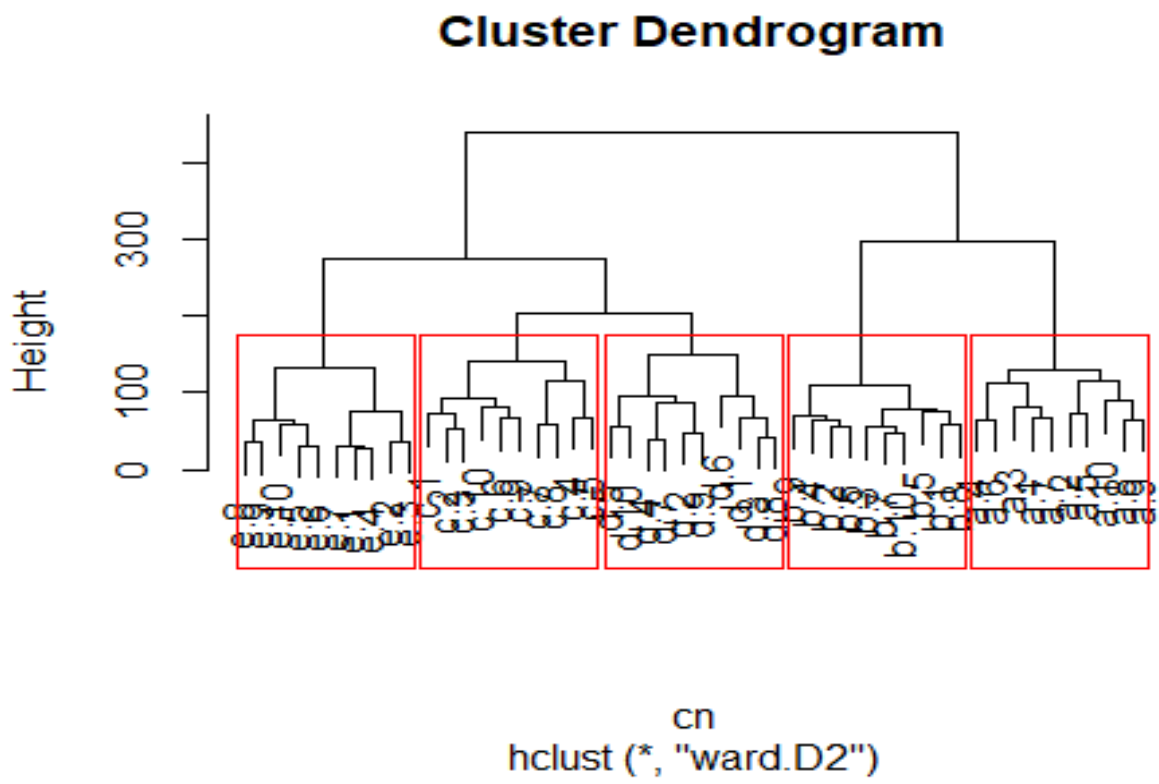
Clustering

```
library(cluster)
library(factoextra) #For K-Means and K-Medoids Cluster plots.

## Loading required package: ggplot2

## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at
https://goo.gl/13EFCZ

library(fossil)

## Loading required package: sp

## Loading required package: maps

##
## Attaching package: 'maps'

## The following object is masked from 'package:cluster':
##
##     votes.repub

## Loading required package: shapefiles

## Loading required package: foreign

##
## Attaching package: 'shapefiles'

## The following objects are masked from 'package:foreign':
##
##     read.dbf, write.dbf



#Taking data in.

orl_data<-read.csv(file.choose(),header = T,sep=",")
#subsetting the data for first 5 people
first_5 <- orl_data[1:50,]

##6.1
#Heirarchial clustering using Euclidean distance and ward linkage:

#calculating euclidean distance
eu<- dist(first_5,method = "euclidean")

#clustering using hclust function
eu_clust<-hclust(eu,method = "ward.D2")
#plotting the dendogram
```

```r
plot(eu_clust)
#drawing rectangles for clusters in dendogram
rect.hclust(eu_clust,k=5,border = "red")
```

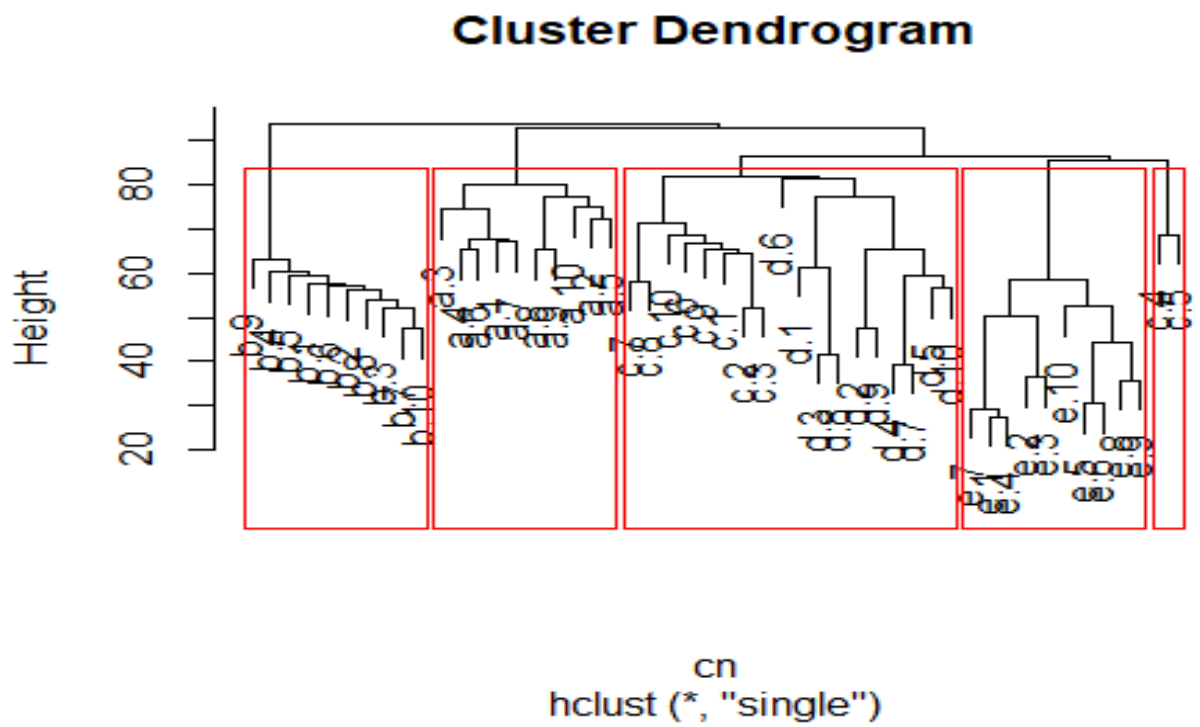



```r
#Heirarchial clustering using Canberra distance and ward linkage:

#calculating canberra distance
cn<- dist(first_5,method = "canberra")

#clustering using hclust function
cn_clust<-hclust(cn,method = "ward.D2")
#plotting the dendogram
plot(cn_clust)
#drawing rectangles for clusters in dendogram
rect.hclust(cn_clust,k=5,border = "red")
```

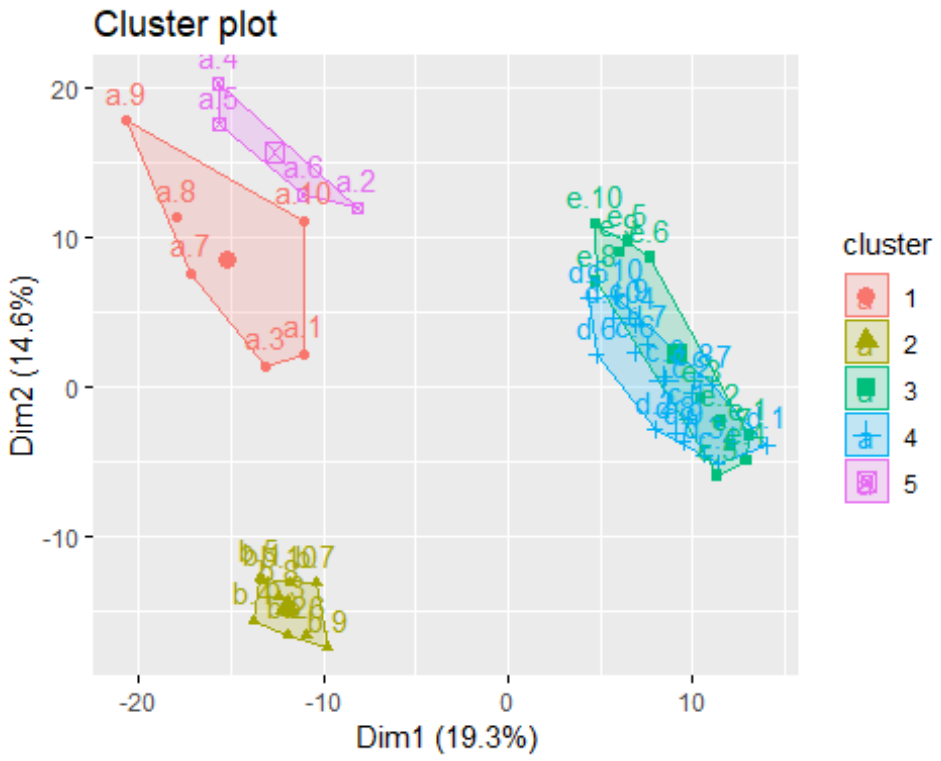



```
##6.3
## Euclidean distance fails to cluster the two people labelled as 'c' and 'd'
correctly.
##Whereas the canberra distance clusters them perfectly as two different
people for any measurements.


#Function to calculate sorensen distance.

sorenson_distance <- function(data=tr)
                     {
                     k<-nrow(data)
                     dist_matrix<-matrix(0,k)

                     for (i in 1:nrow(data))
                     {
                       for (j in 1:nrow(data))
                       {

                         if (i==j)
                         {
                           dist_matrix[i,j]==0
                         }
```

```
                        else
                        {
                          for (k in ncol(data))
                          {
                            dist_matrix[i,j]== sum(abs(data[i,k]-
data[j,k])/
                                     (abs(data[i,k]+data[j,k])))
                          }

                        }
                      }
                    }
                    return(dist_matrix)
                    }


#sorensen_dist <-sorenson_distance(first_5)
#print(sorensen_dist)

#sorensen_cluster <-hclust(sorensen_dist,method = "ward.D2")
#plot(sorensen_cluster)
#rect.hclust(sorensen_cluster,k=5,border = "red")



#Heirarchial clustering using Canberra distance and single linkage:

#clustering using hclust function
cn_clust_single<-hclust(cn,method = "single")
#plotting the dendogram
plot(cn_clust_single)
#drawing rectangles for clusters in dendogram
rect.hclust(cn_clust_single,k=5,border = "red")
```



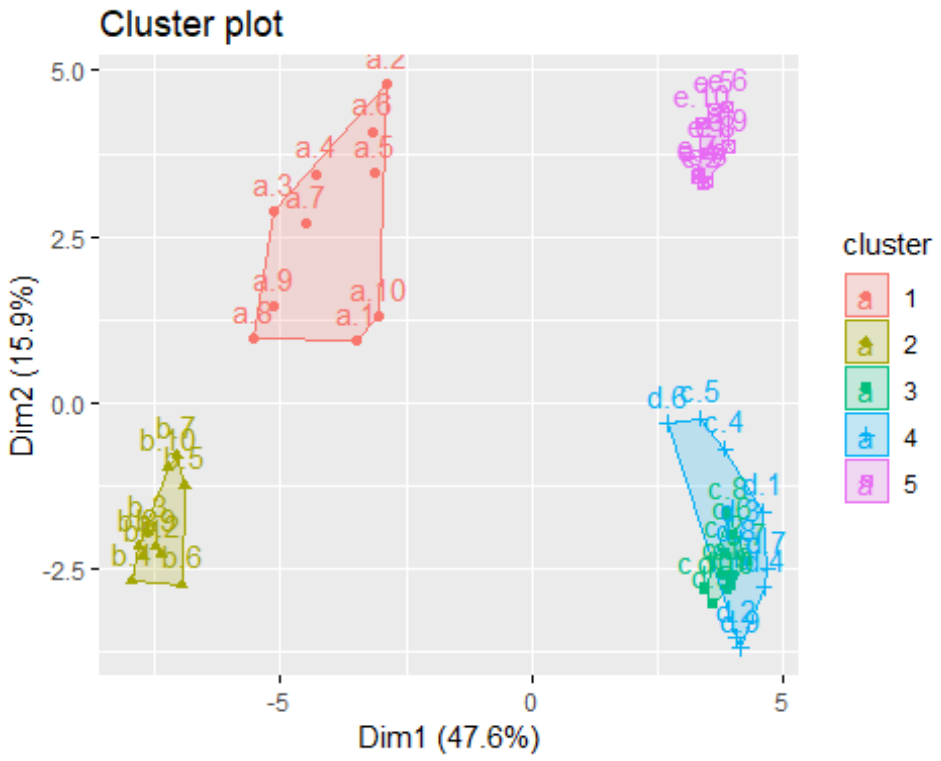

```
#The single linkage takes the shortest distance and thus fails to find the
difference between teh two people labelled as 'c' and 'd' and even 'e' and
thus does not cluster their respective measurements correctly.


KMeans_clust<-kmeans(first_5,5)
fviz_cluster(KMeans_clust,first_5)
```




```
KMeans_clust$size
```

```
## [1]  6 10 11 19  4
```

```
#KMedoids
K_Medoids_clust<-pam(cn,5,diss = F) #using canberra distance.
fviz_cluster(K_Medoids_clust)
```




```
K_Medoids_clust$clusinfo
```

```
##      size max_diss   av_diss diameter separation
## [1,]   10 183.5490 137.26591 234.7811   204.6580
## [2,]   10 140.1910  93.21788 153.5192   204.6580
## [3,]   10 162.2154 116.56343 199.1833   103.3477
## [4,]   10 193.7253 126.26877 239.1010   103.3477
## [5,]   10 161.2982 103.13737 190.3802   182.0318
```

```
#We see that by using Medoids we get all the clusters of exact size 10.
#K-means does not cluster as well as K-medoids.

#*************************************************************************
**#
```

1. plotting the data
points.

```
library(ggplot2)
news<-read.csv(file.choose(),header = T,sep=",")
news$y<-as.factor(news$y)
ggplot(news,aes(x1,x2,color=y,title=))+geom_point()
```

table(news$y)

0 1

300 300

The data has equal number of class probabilites.

$\delta j(x) = Pr[Y = j|x] = \frac{\pi_j p(\mathbf{x}|y=j)}{p(\mathbf{x})}$

Here, πj = 0.5 for both classes

3.

The estimator is given as follows,

$$\widehat{\pi}_j(n_j) = \frac{\sum_{i=1}^{n} I(Y_i = j)}{\sum_{i=1}^{n} I(Y_i)}$$

4.

4.1

We know that the Scott's rule of thumb is given as,

$\sqrt{\mathrm{H}_{jj}} = \sqrt{h_j} = m^{\frac{-1}{p+4}} \widehat{\sigma}_{x_j}$ and $\widehat{\sigma}_{x_j} = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(x_{ij} - \bar{x}_j)^2}$.

Choosing a bandwith matrix proportional to the covariance matrix, we get

$\hat{H} = \mathrm{m}^{\frac{-1}{p+4}} \widehat{\mathbf{\Sigma}}^{1/2}$

$$\widehat{[Y=j|\mathbf{x}]} = \frac{\widehat{\pi}_j \widehat{p(\mathbf{x}|y=j)}}{\widehat{p(\mathbf{x})}} = \frac{.5\widehat{p(\mathbf{x}|y=j)}}{\widehat{p(\mathbf{x})}} = \delta_j$$

By making substitutions and Solving further, we get

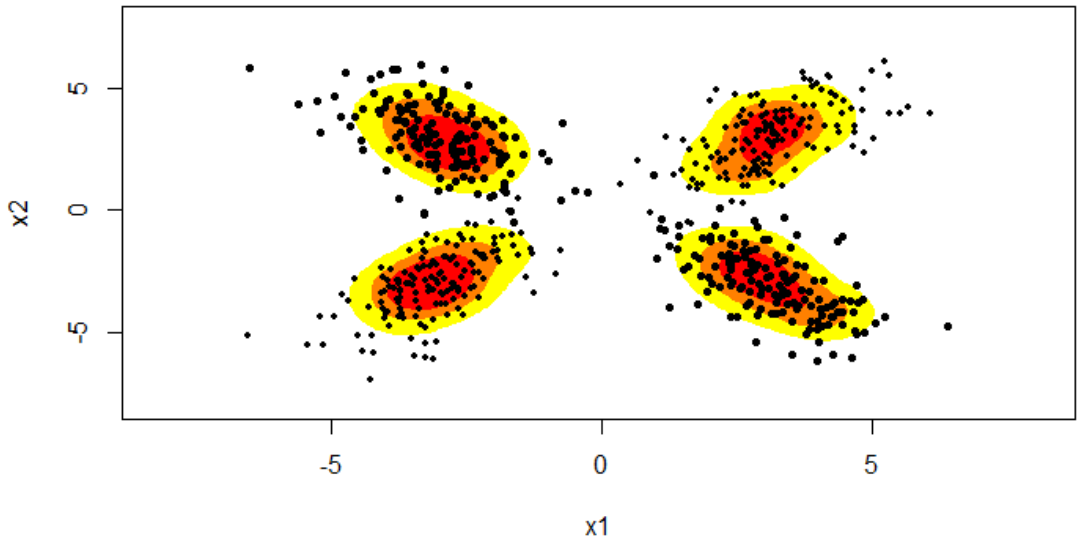
$$\widehat{[Y=j|\mathbf{x}]} = \frac{\widehat{\pi}_j \frac{1}{mh}\sum_{i=1}^{m} K\left(\frac{x-x_{ij}}{h}\right)}{\frac{1}{n}\sum_{i=1}^{m} K_{\mathrm{H}}(\mathbf{x}-\mathbf{x}_i)} = \frac{.5p\frac{1}{mh}\sum_{i=1}^{m} K\left(\frac{x-x_{ij}}{h}\right)}{\frac{1}{n}\sum_{i=1}^{m} K_{\mathrm{H}}(\mathbf{x}-\mathbf{x}_i)} = \widehat{\delta_j(\mathbf{x})}$$

∀j ∈ R,H = diag$(h_1^2, h_2^2, \cdots, h_p^2)$

Isotropic

```
library(ks)
xy<-news
x1 <- xy[which(xy$y==1),-3] # Class 1 observations
x0 <- xy[which(xy$y==0),-3] # Class 0 observations
m0 <- 300
m1 <- 300
p <- 2
var01 <- var(x0$x1)
var02 <- var(x0$x2)
cov0102 <- cov(x0$x2, x0$x1)
var11 <- var(x1$x1)
var12 <- var(x1$x2)
cov1112 <- cov(x1$x1, x1$x2)
var_matrix0 <- matrix(c(sqrt(var01), sqrt(abs(cov0102)), sqrt(abs(cov0102)),
```

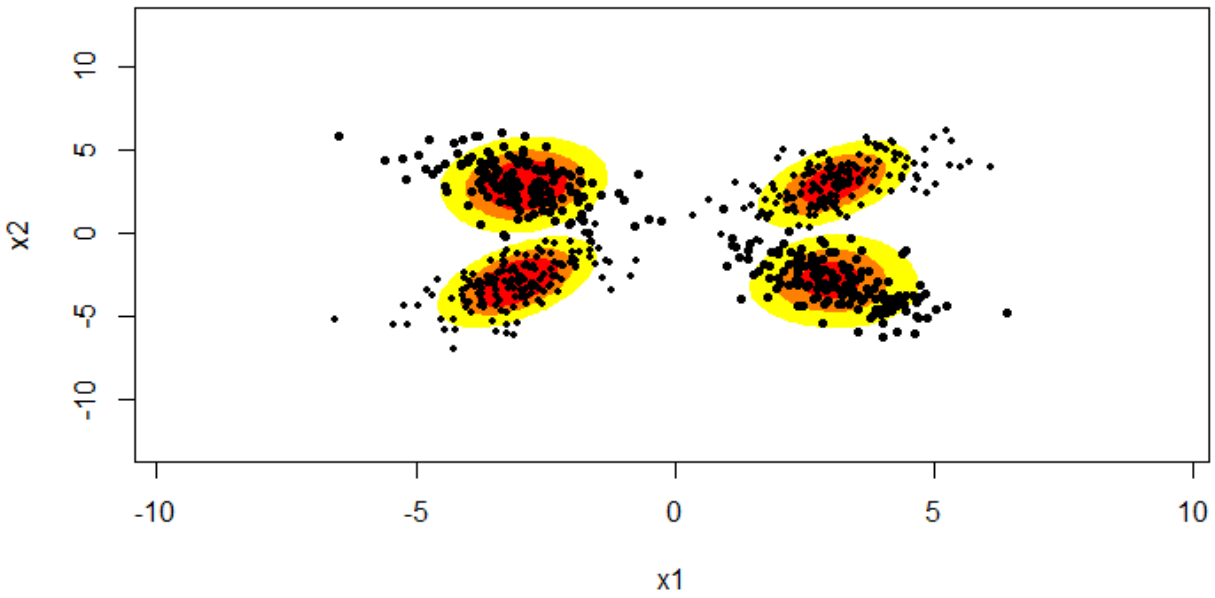
```
 var_matrix1 <- matrix(c(sqrt(var11), sqrt(abs(cov1112)), sqrt(abs(cov1112)),
 iso0 <- (m0^(-1/(p+4)))*matrix(c(1, 0, 0, 1), nrow=2, ncol=2)
 iso1 <- (m1 ^(-1/(p+4)))*matrix(c(1, 0, 0, 1), nrow=2, ncol=2)
 abmin <- min(min(x1$x1),min(x0$x1))
 abmax <- max(max(x1$x1),max(x0$x1))
 ormin <- min(min(x1$x2),min(x0$x2))
 ormax <- max(max(x1$x2),max(x0$x2))
 n1 <- nrow(x1)
 n0 <- nrow(x0)
 H1 <- iso1 # Estimate the H matrix
 dens1 <- kde(x=x1, H=H1)     # Estimate the density
 d1<- predict(dens1, x=x1) # Predict/Estimate densities of points
 H0<- iso0  # Estimate the H matrix
 dens0 <- kde(x=x0, H=H0)     # Estimate the density
d0<- predict(dens0, x=x0) # Predict/Estimate densities of points
 d1 <- predict(dens1, x=rbind(x1,x0)) # density of x in class 1
 d0 <- predict(dens0, x=rbind(x1,x0)) # density of x in class 2
 p1 <- d1/(d1+d0) # Prob[Y=1|x]
 p0 <- 1 - p1
 plot(dens0, display="filled.contour2")
points(x0, cex=0.7, pch=16)
 plot(dens1, display="filled.contour2",
+     xlim=c(abmin, abmax), ylim=c(ormin, ormax),
+     add=TRUE)
 points(x1, cex=0.5, pch=16)
 #plot(dens0, display="filled.contour2", add=TRUE)
 #points(x0, cex=0.7, pch=16)
```




```
 y    <- xy$y
 yhat.np <- ifelse(p1>0.5,1,0) # Predicted class with nonparametric kernel
 err.np <- mean(ifelse(y!=yhat.np,1,0))
 table(y, yhat.np)
yhat.np y   0 1 0 295 5
1 5 295
```

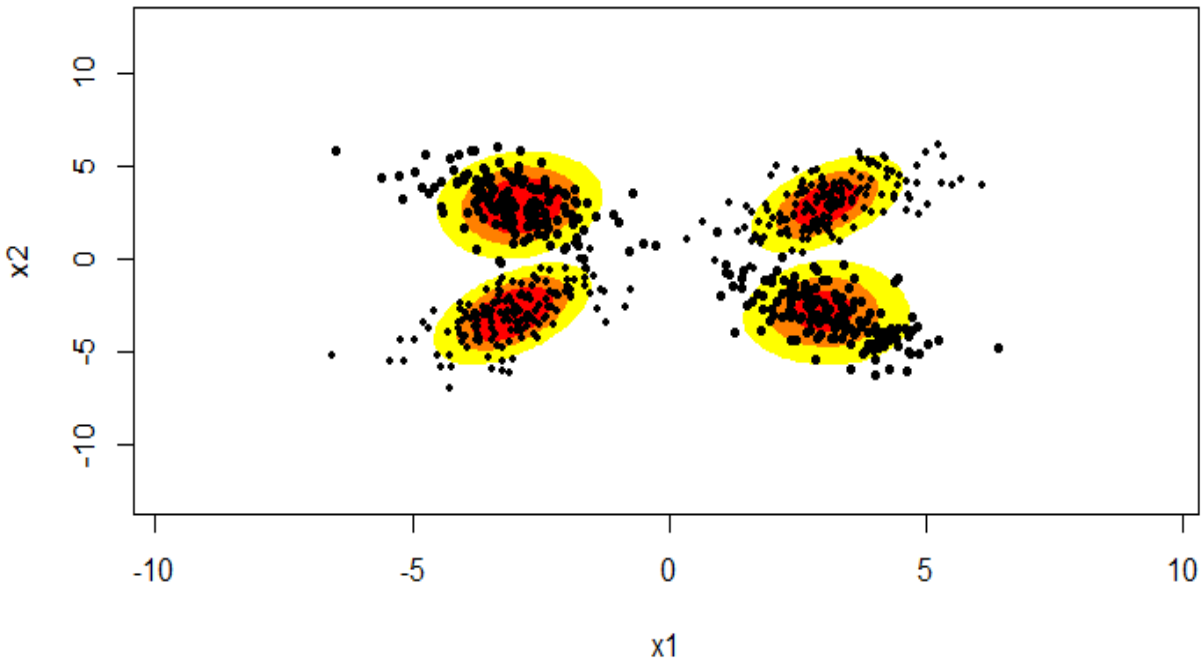
```
x1 <- xy[which(xy$y==1),-3] # Class 1 observations
 x0 <- xy[which(xy$y==0),-3] # Class 0 observations
 m0 <- 300
 m1 <- 300
 p <- 2
 var01 <- var(x0$x1)
 var02 <- var(x0$x2)
 cov0102 <- cov(x0$x2, x0$x1)
 var11 <- var(x1$x1)
 var12 <- var(x1$x2)
 cov1112 <- cov(x1$x1, x1$x2)
 var_matrix0 <- matrix(c(sqrt(var01), sqrt(abs(cov0102)), sqrt(abs(cov0102)),
 var_matrix1 <- matrix(c(sqrt(var11), sqrt(abs(cov1112)), sqrt(abs(cov1112)),
 scott0 <- (m0^(-1/(p+4)))*var_matrix0
 scott1 <- (m1^(-1/(p+4)))*var_matrix1
 abmin <- min(min(x1$x1),min(x0$x1))
 abmax <- max(max(x1$x1),max(x0$x1))
 ormin <- min(min(x1$x2),min(x0$x2))
 ormax <- max(max(x1$x2),max(x0$x2))
 n1 <- nrow(x1)
 n0 <- nrow(x0)
 H1 <- scott1     # Estimate the H matrix
 dens1 <- kde(x=x1, H=H1)     # Estimate the density
 d1<- predict(dens1, x=x1) # Predict/Estimate densities of points
 H0<- scott0      # Estimate the H matrix
 dens0 <- kde(x=x0, H=H0)     # Estimate the density
d0    <- predict(dens0, x=x0) # Predict/Estimate densities of points
```

```
d1 <- predict(dens1, x=rbind(x1,x0)) # density of x in class 1
 d0 <- predict(dens0, x=rbind(x1,x0)) # density of x in class 2
 p1 <- d1/(d1+d0) # Prob[Y=1|x]
 p0 <- 1 - p1
 plot(dens0, display="filled.contour2")  points(x0, cex=0.7, pch=16)
 plot(dens1, display="filled.contour2",
+     xlim=c(abmin, abmax), ylim=c(ormin, ormax),
+     add=TRUE)
points(x1, cex=0.5, pch=16)
 #plot(dens0, display="filled.contour2", add=TRUE)
 #points(x0, cex=0.7, pch=16)
```



```
 y<- xy$y
yhat.np <- ifelse(p1>0.5,1,0) # Predicted class with nonparametric kernel
 err.np <- mean(ifelse(y!=yhat.np,1,0))

 table(y, yhat.np)
yhat.np
y     0 1
288 12
1 299



c.)
##Silverman

x1 <- xy[which(xy$y==1),-3] # Class 1 observations
 x0 <- xy[which(xy$y==0),-3] # Class 0 observations
 m0 <- 300
 m1 <- 300
 p <- 2
 var01 <- var(x0$x1)
 var02 <- var(x0$x2)
 cov0102 <- cov(x0$x2, x0$x1)
 var11 <- var(x1$x1)
 var12 <- var(x1$x2)
 cov1112 <- cov(x1$x1, x1$x2)
 var_matrix0 <- matrix(c(sqrt(var01), sqrt(abs(cov0102)), sqrt(abs(cov0102)),
 var_matrix1 <- matrix(c(sqrt(var11), sqrt(abs(cov1112)), sqrt(abs(cov1112)),
```

```
 silvermann0 <- ((4/(p+2))^(1/(p+4)))*(m0^(-1/(p+4)))*var_matrix0
 silvermann1 <- ((4/(p+2))^(1/(p+4)))*(m1 ^(-1/(p+4)))*var_matrix1
 abmin <- min(min(x1$x1),min(x0$x1))
 abmax <- max(max(x1$x1),max(x0$x1))
 ormin <- min(min(x1$x2),min(x0$x2))
 ormax <- max(max(x1$x2),max(x0$x2))
 n1 <- nrow(x1)
 n0 <- nrow(x0)
 H1 <- silvermann1      # Estimate the H matrix
 dens1 <- kde(x=x1, H=H1)     # Estimate the density
 d1   <- predict(dens1, x=x1) # Predict/Estimate densities of points
 H0   <- silvermann0    # Estimate the H matrix
 dens0 <- kde(x=x0, H=H0)     # Estimate the density
 d0   <- predict(dens0, x=x0) # Predict/Estimate densities of points
 d1 <- predict(dens1, x=rbind(x1,x0)) # density of x in class 1
 d0 <- predict(dens0, x=rbind(x1,x0)) # density of x in class 2
 p1 <- d1/(d1+d0) # Prob[Y=1|x]
 p0 <- 1 - p1
 plot(dens0, display="filled.contour2") points(x0, cex=0.7, pch=16)
plot(dens1, display="filled.contour2",
+     xlim=c(abmin, abmax), ylim=c(ormin, ormax),
+     add=TRUE)
 points(x1, cex=0.5, pch=16)
 #plot(dens0, display="filled.contour2", add=TRUE)
 #points(x0, cex=0.7, pch=16)
```




```
 y<- xy$y
 yhat.np <- ifelse(p1>0.5,1,0) # Predicted class with nonparametric kernel
 err.np <- mean(ifelse(y!=yhat.np,1,0))
```

```
 table(y, yhat.np)
yhat.np
y     0 1
288 12
1 299
```

4.5

The confusion matrix of Isotropic method is better than the other two methods. Therefore we use Isotropic as the best classifier.

5.

5.1

δj(x) = P[Y = j|x]

$$P[x|Y=j] = \sum_{j=1}^{2} \alpha_j \phi_p(\mathbf{x}; \nu_j, \Psi_j), \qquad \alpha_j > 0 \qquad \sum_{j=1}^{2} \alpha_k = 1, \ j = 1, 2$$

,

$$\phi_p(\mathbf{x}; \nu_j, \Psi_j) = \frac{1}{(2\pi)^p |\psi_j|} exp\left[\frac{1}{2}(x-\nu_j)\psi_j^{-1}(x-\nu_j)\right]$$

$$P[Y=j|x] = \frac{P[Y=j]P[X|Y=j}{P[X=x]}] \approx P[Y=j]P[X|Y=j]$$

$$\delta_j(x) = P[Y=j|x] \approx \pi_j P[x|y=j] \approx \sum_{j=1}^{2} \pi_j \alpha_j \phi_p(\mathbf{x}; \nu_j, \Psi_j)$$

5.2

```
#codes taken from the code file:'density-estimate-nonpar-and-mixture-1.R' >
library(ks)
# Class conditional densities via mixtures of Gaussians


require(mixtools)
library(ggplot2)
xy<-read.csv(file.choose(),header = T,sep=",")
x1 <- xy[which(xy$y==1),-3] # Class 1 observations
```

```
x0 <- xy[which(xy$y==0),-3] # Class 0 observations

dens.mix.1 <- mvnormalmixEM(x1) number of iterations= 6

a1 <- dens.mix.1$lambda # Mixing proportions for class 1

m1 <- dens.mix.1$mu

S1 <- dens.mix.1$sigma > dens.mix.0 <- mvnormalmixEM(x0) number of
iterations= 20

# Several errors with and without this part sometimes, its weird

a0 <- dens.mix.0$lambda # Mixing proportions for class 0

m0 <- dens.mix.0$mu

S0 <- dens.mix.0$sigma

x <- as.matrix(rbind(x1,x0))

d1m <- a1[1]*dmvnorm(x, m1[[1]],S1[[1]])+a1[2]*dmvnorm(x, m1[[2]],S1[[2]])

d0m <- a0[1]*dmvnorm(x, m0[[1]],S0[[1]])+a0[2]*dmvnorm(x, m0[[2]],S0[[2]])

p1m <- d1m/(d1m+d0m)

yhat.mix <- ifelse(p1m>0.5, 1, 0) # Predicted class with nonparametric ker



# err.mix <- mean(ifelse(xy$y!=yhat.mix, 1,0))


#y.tab <- as.table(yhat.mix)

#y.tab
```

$$\widehat{\delta_j(x)} = P[\widehat{Y = j}|x] = \hat{\pi}_j P[\widehat{x|y = j}] = \frac{n_j}{n}\sum_{j=1}^{2}\alpha_j\phi_p(\mathbf{x};\hat{\nu_j},\hat{\Psi}_j) = \frac{1}{n}\sum_{j=1}^{2} n_j\alpha_j\phi_p(\mathbf{x};\hat{\nu_j},\hat{\Psi}_j)$$

Where $\phi_p(\mathbf{x};\hat{\nu_j},\hat{\Psi}_j) = \frac{1}{\sqrt{(2\pi)^p det(\Psi_k)}} exp\left[\frac{1}{2}(x-\nu_j)\psi_j^{-1}(x-\nu_j)\right]$

6.

```
#Codes taken from the R file:'mixture-discriminant-analysis-1.R' >
```

```
 library(MASS)
library(car)
library(kernlab)
library(rpart)
library(randomForest)
library(class)
library(ada)
library(rda)
library(e1071)
library(nnet)
library(ks)
library(mixtools)
library(MASS)
 x<-read.csv(file.choose(),header = T,sep=",")
 xy<-x
 x<-x[,-3]
 #splitting teh data into 70-30

 smp_size <- floor(0.70 * nrow(x))
 ## set the seed to make your partition reproducible
 set.seed(123)
 train_ind <- sample(seq_len(nrow(x)), size = smp_size)
 train <- x[train_ind, ]
 test <- x[-train_ind, ]
 x1 <- x[1:300, ]
 x2 <- x[301:600,]
 km.x1 <- kmeans(x1, 2)
 mu1 <- list(km.x1$center[1,], km.x1$center[2,])
```

```
 mix1 <- mvnormalmixEM(x1, mu=mu1) number of iterations= 5
 km.x2 <- kmeans(x2, 2)
 mu2 = list(km.x2$center[1,], km.x2$center[2,])
 mix2 <- mvnormalmixEM(x2, mu=mu2) number of iterations= 7
 mix.da <- function(xnew,m1,m2,prior)
+ {
+ delta1 <- prior*(m1$lambda[[1]]*dmvnorm(as.matrix(xnew), m1$mu[[1]], m1$sig
+ delta2 <- (1-prior)*(m2$lambda[[1]]*dmvnorm(as.matrix(xnew), m2$mu[[1]], m2
+ res <- ifelse(delta1>delta2, 1, 0)
+ return(res)
+ }
 mix.da(x[50, ], mix1, mix2,.5)
[1] 1
 mix.da(x[350, ], mix1, mix2,.5)
[1] 0
 # Mixture of Gaussian
 y <- c(rep(1,300), rep(0,300))
 yhat.mix.da <-mix.da(x, mix1, mix2,.5)

 table(y, yhat.mix.da)
yhat.mix.da
y     0 1
294 6
5 295
 #lda
 yhat.lda <- predict(lda(x,y),x)$class

 table(y, yhat.lda)
yhat.lda y  0 1
```

```
152 148
149 151
# qda
 yhat.qda <- predict(qda(x,y),x)$class

 table(y, yhat.qda)
yhat.qda y  0 1
294 6
5 295
# naive bayes
 bayes.predict <- predict(naiveBayes(x,y),x, type='raw')
 nb <- ifelse(bayes.predict[,2]>0.5,1,0)

 table(y, nb)
nb
y     0 1
171 129
166 134
```

7. Mixtures of gaussian and QDA perform really well whereas LDA and Naive do not do very well.

9.

```
library(ggplot2)
set.seed(10121967)
 # clear screen
```

```
graphics.off()
# Functions

unitlength <-function(xx)
+     {
+     n <- nrow(xx)
+     p <- ncol(xx)
+     aa <- matrix(rep(apply(xx,2,mean), n), ncol=p, byrow=TRUE)
+     bb <- sqrt((n-1)*matrix(rep(apply(xx,2,var), n), ncol=p, byrow=TRUE))
+     return((xx-aa)/bb)
+     }
# Standardizing the data

standard <-function(xx)
+     {
+     n <- nrow(xx)
+     p <- ncol(xx)
+     aa <- matrix(rep(apply(xx,2,mean), n), ncol=p, byrow=TRUE)
+     bb <- sqrt(matrix(rep(apply(xx,2,var), n), ncol=p, byrow=TRUE))
+     return((xx-aa)/bb)
+     }
 # Unit cube

cube <-function(xx)
+     {
+     n <- nrow(xx)
+     p <- ncol(xx)
```

```
+     aa <- matrix(rep(apply(xx,2,min), n), ncol=p, byrow=TRUE)
+     bb <- matrix(rep(apply(xx,2,max), n), ncol=p, byrow=TRUE)
+     return((xx-aa)/(bb-aa))
+     }
 # Neighborhood size

neighborhood <-function(xy)
+     {
+     p <- ncol(xy)-1
+     y <- xy[,p+1]
+     n <- nrow(xy)
+     max.k <- 1
+     err.k <- matrix(0, ncol=max.k, nrow=50)
+
+     for(j in 1:50)
+     {
+
+     id.tr <- sample(1:n, round(.7*n))
+     yhat.te <- knn(xy[id.tr,-(p+1)], xy[-id.tr,-(p+1)], xy[id.tr,(p+1)]
+     err.k[j,] <- sum(diag(prop.table(table(xy[-id.tr,p+1],yhat.te)))) +
+     }
+     merr.k <- apply(err.k, 2, mean)
+     return(min(which(merr.k==min(merr.k))))
+     }
 # Read in the data

xy <- read.csv(file.choose(),header=T,sep=",")
      library(MASS)
```

```
      library(car)
      library(kernlab)
      library(rpart)
      library(randomForest)
      library(class)
      library(ada)
      library(rda)
      library(e1071)
      library(nnet)
      library(ks)
p <- ncol(xy)-1
pos <- 1+p
y <- xy[,pos]
n <- nrow(xy)
colnames(xy)[pos] <- 'y'
xy$y <- as.factor(ifelse(xy$y==unique(xy$y)[1],'success','failure'))
 # Standardize the predictor variables
xy[,-pos] <- standard(xy[,-pos])
 # Determine the optimal neighborhood size k.opt by re-sampling

k.opt <- neighborhood(xy)
# Set the total number of replications

R <- 100
 # Initialize the test error vector >
err <- matrix(0, ncol=8, nrow=R)
for(r in 1:R)
+     {
```

```
+
+     id.F        <- which(xy$y == 'failure')
+     n.F         <- length(id.F)
+      id.F.tr <- sample(sample(sample(id.F)))[1:round(0.7*n.F)]
+      id.F.te <- setdiff(id.F, id.F.tr)
+
+      id.S <- which(xy$y == 'success')
+      n.S <- length(id.S)
+      id.S.tr <- sample(sample(sample(id.S)))[1:round(0.7*n.S)]
+      id.S.te <- setdiff(id.S, id.S.tr)
+
+      xy.tr <- xy[c(id.F.tr,id.S.tr), ]
+      xy.te <- xy[c(id.F.te,id.S.te), ]
+      ntr <- nrow(xy.tr)
+      nte <- n - ntr
+
+      lda.xy <- lda(y~., data=xy.tr)
+      yhat.lda <- predict(lda.xy, xy.te[,-(p+1)])$class
+      err.lda <- 1-sum(diag(table(xy.te$y, yhat.lda)))/nte
+
+      err[r,1] <- err.lda
+
+      svm.xy <- ksvm(y~., data=xy.tr, kernel='rbfdot')
+      yhat.svm <- predict(svm.xy, xy.te[,-(p+1)])
+      err.svm <- 1-sum(diag(table(xy.te$y, yhat.svm)))/nte
+
+      err[r,2] <- err.svm
+
```

```
+
+     tree.xy <- rpart(y~., data=xy.tr)
+     yhat.tree <- predict(tree.xy, xy.te[,-(p+1)], type='class')
+     err.tree <- 1-sum(diag(table(xy.te$y, yhat.tree)))/nte
+
+     err[r,3] <- err.tree
+
+     forest.xy <- randomForest(y~., data=xy.tr)
+     yhat.forest <- predict(forest.xy, xy.te[,-(p+1)], type='class')
+     err.forest <- 1-sum(diag(table(xy.te$y, yhat.forest)))/nte
+
+     err[r,4] <- err.forest
+
+     gausspr.xy <- gausspr(y~., data=xy.tr)
+     yhat.gausspr <- predict(gausspr.xy, xy.te[,-(p+1)], type='response')
+     err.gausspr <- 1-sum(diag(table(xy.te$y, yhat.gausspr)))/nte
+
```

```
+     err[r,5] <- err.gausspr
+
+     yhat.kNN <- knn(xy.tr[,-(p+1)], xy.te[,-(p+1)], xy.tr[,(p+1)], k=1)
+     err.knn <- 1-sum(diag(table(xy.te$y, yhat.kNN)))/nte
+
+     err[r,6] <- err.knn
+
+     boost.xy <- ada(y~., data=xy.tr)
+     yhat.boost <- predict(boost.xy, xy.te[,-(p+1)])
+     err.boost <- 1-sum(diag(table(xy.te$y, yhat.boost)))/nte
+
+     err[r,7] <- err.boost
+
+     # naive bayes
+     bay.xy<-naiveBayes(y~.,data=xy.tr)
+     yhat.raw.xy<-predict(bay.xy, xy.te, type='raw')
+     yhat.bay.xy<-ifelse(yhat.raw.xy[,2]>0.5,1,0)
+     err.bay<-1-sum(diag(table(xy.te$y, yhat.bay.xy)))/nte
+
+     err[r,8] <- err.bay
```

Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel Using automatic sigma estimation (sigest) for RBF or laplace kernel

Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel
Using automatic sigma estimation (sigest) for RBF or laplace kernel

```
cols <- c(2,3,4,5,2,3,4,5)
metodos <- c('LDA','SVM','CART','Iso','Gauss', 'kNN','Boost','N_Bayes')
windows()
boxplot(err, col=cols, names=metodos)
      windows()
avg.err <- round(apply(err, 2, mean),4)
plot(1:8, avg.err, ylab='Average prediction error',
+     xlab='Method (Classifier)',xlim=c(0,10), ylim=c(0.90*min(avg.err),1.10*
text(1:8, avg.err, col=cols, labels=metodos, pos=4)
#abline(h=bayes.risk, lwd=3, col='red')
```

```
10.
 avg.err
[1] 0.5228 0.0212 0.0451 0.0262 0.0224 0.0289 0.0256 0.5238
```