# Assignment #1 – Machine Learning – Professor Haugh

## **Jaime Gacitua**

## **jg3499**

## **Monday February 15, 2016**

Question 1.

1. I implemented the Naïve Bayes over the spam dataset using R, and the e1071 library. The code can be found in the appendix.

The dependent variable, to predict, is “spam”.

The 18 independent variables are:

[1] "day.of.week" "time.of.day" "size.kb" "box" "domain" "local"

[7] "digits" "name" "cappct" "special" "credit" "sucker"

[13] "porn" "chain" "username" "large.text" "spampct" "category"

I split the data into 80% training and 20% test sets, and calculate the generalization error. The output for one run of the Naïve Bayes is displayed below, achieving a 95.17% accuracy.

> pred.output

Confusion Matrix and Statistics

Reference

Prediction no yes

no 269 3

yes 18 145

Accuracy : 0.9517

95% CI : (0.9271, 0.9699)

No Information Rate : 0.6598

P-Value [Acc > NIR] : < 2e-16

Kappa : 0.895

Mcnemar's Test P-Value : 0.00225

Sensitivity : 0.9373

Specificity : 0.9797

Pos Pred Value : 0.9890

Neg Pred Value : 0.8896

Prevalence : 0.6598

Detection Rate : 0.6184

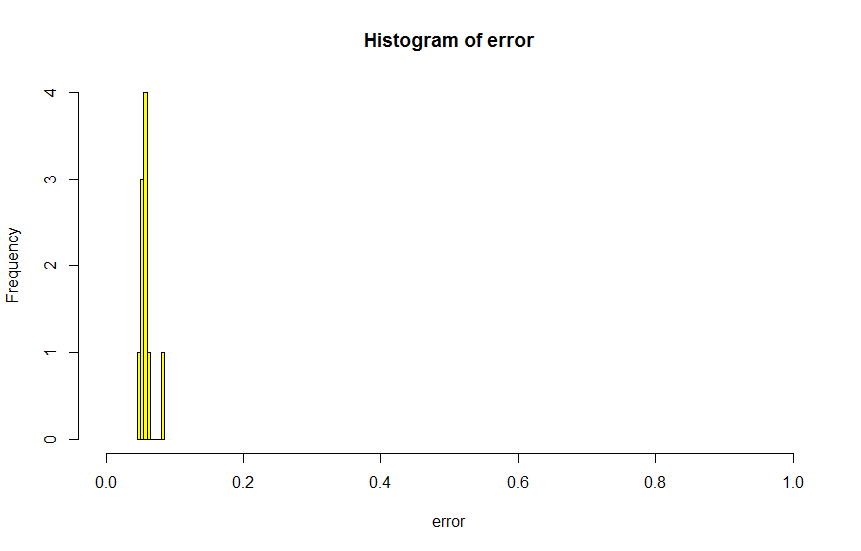
Detection Prevalence : 0.6253

Balanced Accuracy : 0.9585

'Positive' Class : no

1. Running Naïve Bayes for 10 random different training/test sets.

The distribution of can be summarized as a random variable with mean 5.86% and standard deviation 0.9%. Key takeaway here is that the error is very robust to changes in the training/test sets, there are no important outliers.



We have two kinds of errors: False positives and false negatives. In this particular case, it is more costly to have a false positive, since the classifier might filter an important email. The opposite case, a false negative is not ideal, but the relative cost to the user is much lower.

Yes we can adjust the classifier to reflect this. One way to achieve this is to adjust the decision criteria when we predict an email spam or not spam, using the a posteriori probabilities. By default, Naïve Bayes classifies to the category that has the highest probability. I would make a change, adding a new parameter **r>0** and classify as spam only if

***Additional Comments***

I tried running with Laplace smoothing, and also by controlling over the missing values for “spampct”. I also ran n=1000 repetitions, and recorded the errors, depicted in the table below. The key takeaway here is that sometimes adding more information can be detrimental to the accuracy to Naïve Bayes.

Mean error base 0.05935632

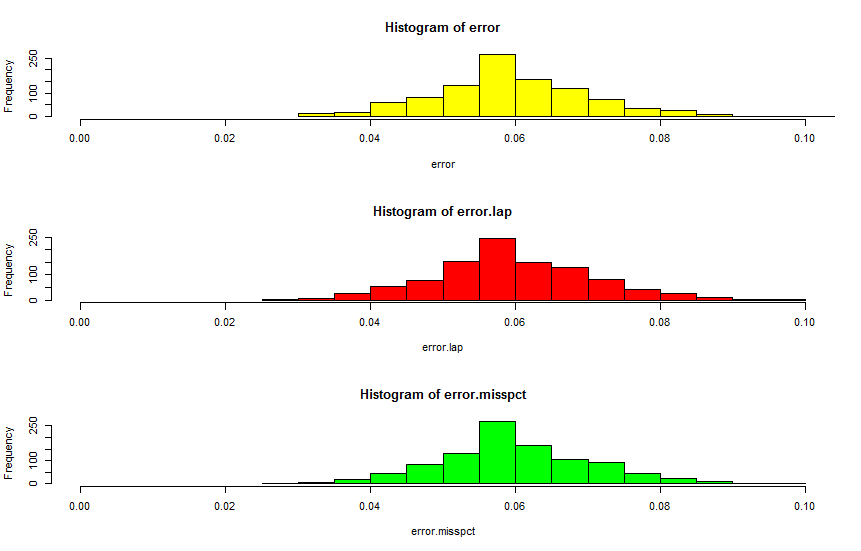
Mean error laplace 0.05941149

Mean error misspct 0.06003678

Var error base 0.01063725

Var error laplace 0.01064671

Var error misspct 0.01054006



Problem 2

Maximizing the Raleigh quotient:

To handle this maximization problem we do a transformation, considering B and W are positive definite matrices.

We define

Replacing in the Raleigh quotient,

The solution to this max problem has infinite solutions,

For any , and ,

To choose a certain solution, we will conveniently constrain y to the unit sphere:

Solving to find y\*, we define the lagrangian:

Takin derivative with respect to y,

Setting derivative equal to zero,

Since C is a symmetric matrix we can add the terms and transpose the equation,

This is the equation for finding the eigenvector corresponding to the largest eigenvalue for the matrix

If we replace C and y using their definitions before,

**This the equation to find the eigenvector a\* of the largest eigenvalue of the matrix**

(b)

We have , the covariance matrix of x, which represents the within class variance.

can be expressed by its eigen decomposition:

The matrix M represents the class centroids, with dimensions K x p (K classes and p features)

The covariance matrix of M, representing the between-class covariance is called B:

When we sphere the data, we do the transformation

We can also compute the sphered class centroids

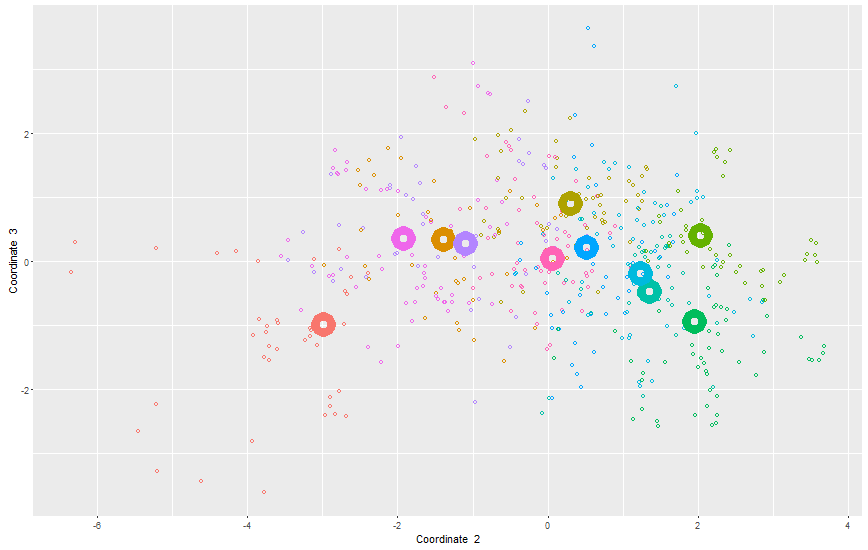
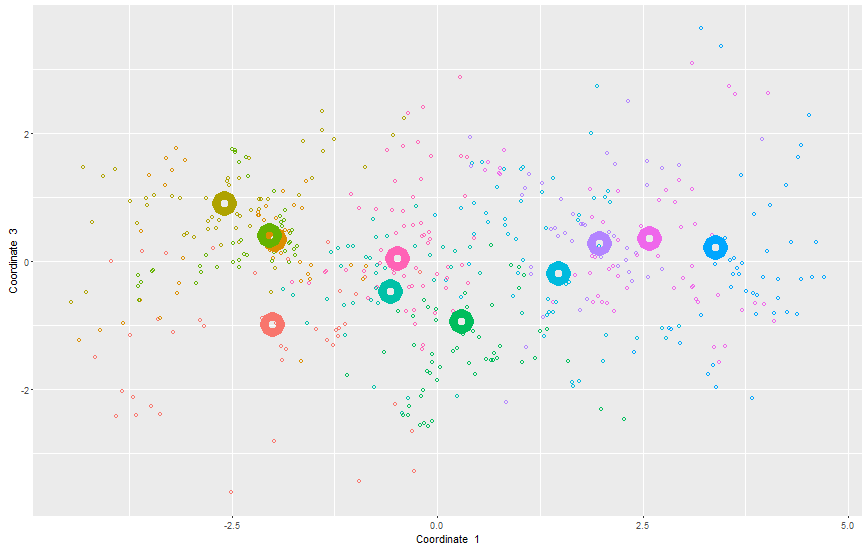
And the covariance matrix of , in this case ,

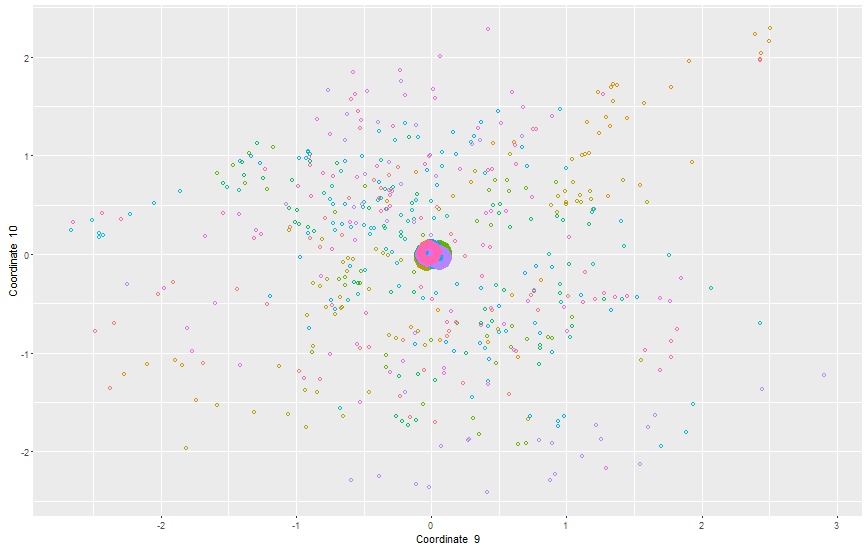
We can now calculate the covariance matrix of , with its eigen decomposition:

Problem 3.

(a)

The implemented code can be found in the appendix. It allows to select any pair of discriminant coordinates. The 4 graphs presented HTF were reproduced, and displayed below:



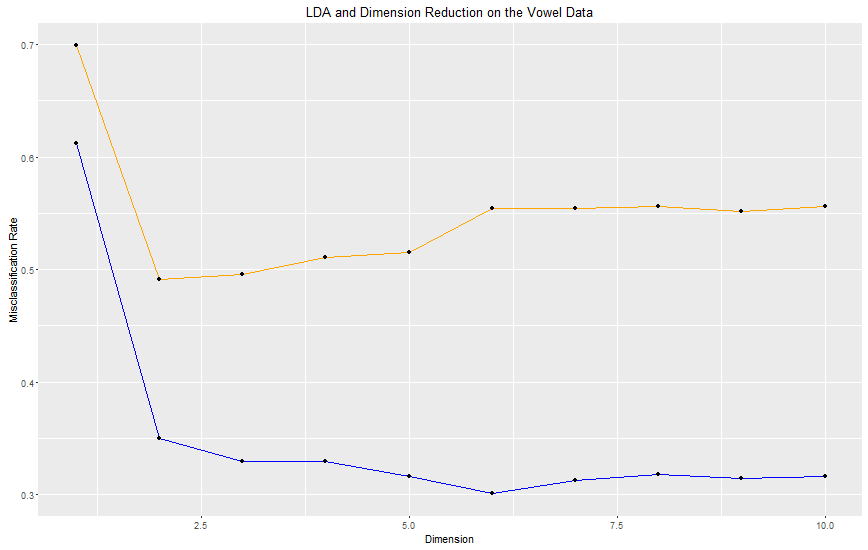


(b)

A reduced-rank LDA was performed on each of the subspaces , successfully reproducing figure 4.10 of HTF:

The orange (top) line represents error in the test data

The blue (bottom) line represents error in the training data



Using 2 dimensions we achieve the least amount of error (50%), so I will use this choice to classify, having the following details. The amount of error is significant, so there is still more work to do to improve the accuracy.

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6 7 8 9 10 11

1 28 18 0 0 0 0 0 0 0 5 0

2 8 16 11 0 0 0 0 0 1 8 0

3 1 0 11 1 0 0 1 0 0 0 0

4 0 0 16 27 0 0 0 0 0 0 0

5 0 0 0 0 19 7 0 0 0 0 0

6 0 0 2 14 11 16 3 0 0 0 4

7 0 0 0 0 12 2 26 7 0 0 1

8 0 0 0 0 0 0 5 26 8 3 0

9 5 0 0 0 0 0 0 3 21 7 5

10 0 0 0 0 0 0 0 6 6 13 0

11 0 8 2 0 0 17 7 0 6 6 32

Overall Statistics

Accuracy : 0.5087

95% CI : (0.4621, 0.5551)

No Information Rate : 0.0909

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.4595

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9 Class: 10 Class: 11

Sensitivity 0.66667 0.38095 0.26190 0.64286 0.45238 0.38095 0.61905 0.61905 0.50000 0.30952 0.76190

Specificity 0.94524 0.93333 0.99286 0.96190 0.98333 0.91905 0.94762 0.96190 0.95238 0.97143 0.89048

Pos Pred Value 0.54902 0.36364 0.78571 0.62791 0.73077 0.32000 0.54167 0.61905 0.51220 0.52000 0.41026

Neg Pred Value 0.96594 0.93780 0.93080 0.96420 0.94725 0.93689 0.96135 0.96190 0.95012 0.93364 0.97396

Prevalence 0.09091 0.09091 0.09091 0.09091 0.09091 0.09091 0.09091 0.09091 0.09091 0.09091 0.09091

Detection Rate 0.06061 0.03463 0.02381 0.05844 0.04113 0.03463 0.05628 0.05628 0.04545 0.02814 0.06926

Detection Prevalence 0.11039 0.09524 0.03030 0.09307 0.05628 0.10823 0.10390 0.09091 0.08874 0.05411 0.16883

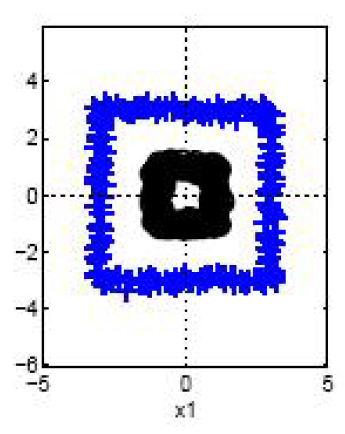
Balanced Accuracy 0.80595 0.65714 0.62738 0.80238 0.71786 0.65000 0.78333 0.79048 0.72619 0.64048 0.82619

Problem 6.

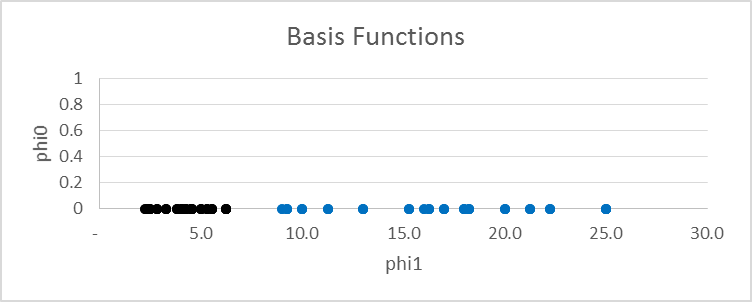
(a)

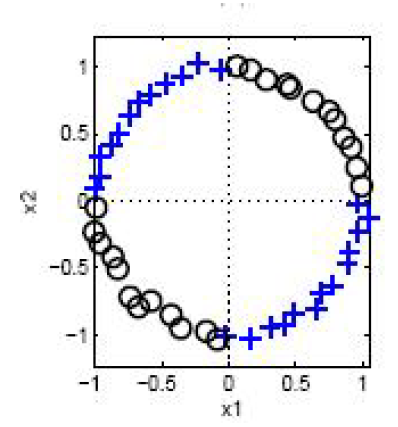
In this case, the classes can be separated according to their distance to the (0,0) point in the x1, x2 plane.

The proposed basis function is based on the Euclidean distance,



The resulting dataset, using , renders the following plot, making the classes linearly separable by a line

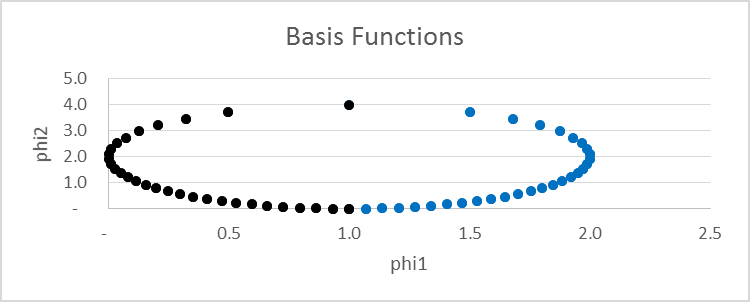




To separate these classes, we can use again the Euclidean distance

will be based on the distance to the line , therefore

will be based on the distance to the line , therefore

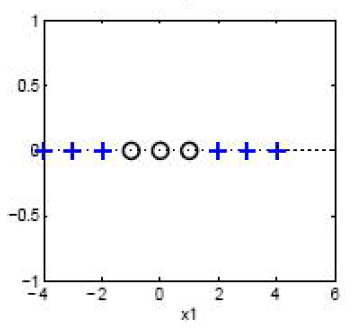


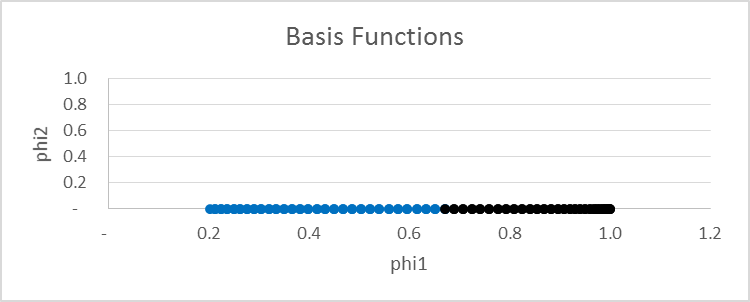
The two classes become linearly separable by the function

(c)

In this opportunity, we can use the basis function , and ,

The classes are linearly separable by the line





## Appendix

## Code for Question 1

library(e1071)

library(caret)

rep <- 10

error = numeric(length=rep)

error.lap = numeric(length=rep)

error.misspct = numeric(length=rep)

error.misspct.lap = numeric(length=rep)

spamdata\_pre <- read.csv(".\\spam.csv")

spamdata <- spamdata\_pre[,3:21]

names(spamdata) #variable column headings

summary(spamdata)

# Check is missing spampct is relevant

spamdata$spampct.present <- factor(!is.na(spamdata$spampct))

# Check if it is working properly

head(spamdata)

# spampct seems to be important, because when it is not present, we have 1062 cases that email is not spam.

table(spamdata$spam, spamdata$spampct.present)

for(t in 1:rep){

sub <- sample(nrow(spamdata), floor(nrow(spamdata) \* 0.8))

trainData <- spamdata[sub, ]

testData <- spamdata[-sub, ]

naive.model <- naiveBayes(spam~., data = trainData, laplace = 0)

naive.model.lap <- naiveBayes(spam~., data = trainData, laplace = 3)

naive.model.misspct <- naiveBayes(spam~., data = trainData, laplace = 0)

# Independent variables

x\_test <- testData[,1:18]

# Independent variables + dummy variable representing missing spampct value

x\_test.misspct <- testData[,1:18]

x\_test.misspct$spampct.present <- testData$spampct.present

# y dependent variables

y\_test <- testData[,19]

pred <- predict(naive.model, x\_test)

pred.lap <- predict(naive.model.lap, x\_test)

pred.misspct <- predict(naive.model.misspct, x\_test.misspct)

pred.output <- confusionMatrix(pred, y\_test)

pred.output.lap <- confusionMatrix(pred.lap, y\_test)

pred.output.misspct <- confusionMatrix(pred.misspct, y\_test)

#print(naive.model)

#print(pred.output)

error[t] <- 1-pred.output$overall[1]

error.lap[t] <- 1-pred.output.lap$overall[1]

error.misspct[t] <- 1-pred.output.misspct$overall[1]

}

cat('Mean error base',mean(error))

cat('Mean error laplace',mean(error.lap))

cat('Mean error misspct',mean(error.misspct))

cat('Var error base',var(error))

cat('Var error laplace',var(error.lap))

cat('Var error misspct',var(error.misspct))

par(mfrow = c(3,1))

hist(error, col='yellow', xlim=c(0.0,0.1))

hist(error.lap, col='red', xlim=c(0.0,0.1))

hist(error.misspct, col='green', xlim=c(0.0,0.1))

## Code for Question 3 (a)

require(DAAG)

require(ggplot2)

require(MASS)

# This code implements reduced rank LDA (Fisher Discriminant Analysis)

# It can reproduce the subplots of Figure 4.8 in HTF by specifing coordinates a,b

# For example, a=1,b=3 reproduces the top-left sub-figure of Figure 4.8

a=1 # First Fisher coordinate to plot

b=3 # second Fisher coordinate to plot

################################################################

# First download the training data from the HTF website

url<-"http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/vowel.train"

vtrain<-read.table(url,header=TRUE,sep=',')

vtrain<-as.data.frame(vtrain)

# columns are row.names,y,x.1,x.2,x.3,x.4,x.5,x.6,x.7,x.8,x.9,x.10

# y is the class, and x.1 to x.10 are predictors

# Now download the test data

url<-"http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/vowel.test"

vtest<-read.table(url,header=TRUE,sep=',')

vtest<-as.data.frame(vtest)

#################################################################

# Now find the Fisher discriminant directions using the "lda" function

ldatrain<-lda(y~x.1+x.2+x.3+x.4+x.5+x.6+x.7+x.8+x.9+x.10, data=vtrain)

ldatrain.means <- ldatrain$means

ldatrain.scaling <- ldatrain$scaling

vtrain.lda.values <- data.frame(predict(ldatrain, vtrain[,3:12]))

vtrain.lda.values.means <- data.frame(predict(ldatrain, data.frame(ldatrain.means)))

plot.data <- subset(vtrain.lda.values, select= c(paste('x.LD',a,sep=''), paste('x.LD',b,sep='')))

plot.data.means <- subset(vtrain.lda.values.means, select= c(paste('x.LD',a,sep=''), paste('x.LD',b,sep='')))

p <- ggplot() +

geom\_point(data = plot.data, aes(

x = plot.data[,1],

y = plot.data[,2],

color=factor(vtrain.lda.values$class), shape=1)) +

xlab(paste('Coordinate ',a)) +

ylab(paste('Coordinate ',b)) +

geom\_point(data = plot.data.means, aes(

x = plot.data.means[,1],

y = plot.data.means[,2],

colour=factor(vtrain.lda.values.means$class), shape=1, stroke=5, size=10)) +

scale\_shape\_identity() + theme(legend.position="none")

p