932A99 - Group K7 - Lab 2

Hoda Fakharzadehjahromy, Otto Moen & Ravinder Reddy Atla

December 6, 2020

Statement of contribution: Assignment 1 was contributed mostly by Ravinder, assignment 2 by Otto and assignment 3 by Hoda.

1. Assignment 1 - LDA and logistic regression

2. Assignment 2 - Decision trees and Naïve Bayes for bank marketing

This assignment consists of fitting decision trees and Naïve Bayes models for a data set containin information about marketing campaigns carried out by a Portuguese bank to get clients to subscribe to one of their products.

2.1 - Import and divide data

The first step is to import the data into R and divide it into training (40 percent), validation (30 percent) and test (30 percent) sets for use in the following sections. The variable named "duration" is also removed. This is done with the following code:

```
bank <- read.csv2("bank-full.csv")

# Remove the "duration" variable
bank <- bank[,-12]

# Convert character variables to factors
for (var in 1:ncol(bank)) {
  if(is.character(bank[,var])){
    bank[,var] <- as.factor(bank[,var])
  }
}

# Train, validation, test sets
n=dim(bank)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=bank[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=bank[id2,]

id3=setdiff(id1,id2)
test=bank[id3,]
```

2.2 - Decision trees

With the data processing completed we can now begin fitting decision trees. In total three trees are fitted using different settings. For each tree the misclassification rate is calculated for the training and validation sets.

2.2 a) Default settings

The first tree is fitted with the default settings:

```
tree_default <- tree(formula = y~.,
                     data = train)
```

```
trainmis_default <- summary(tree_default)[[7]][1] / summary(tree_default)[[7]][2]
validmis_default <- sum(predict(tree_default,
                                newdata = valid,
                                type = "class") != valid$y) / nrow(valid)
```

With the default settings the misclassification rate for the training set is 0.105 and for the validation set it is 0.109.

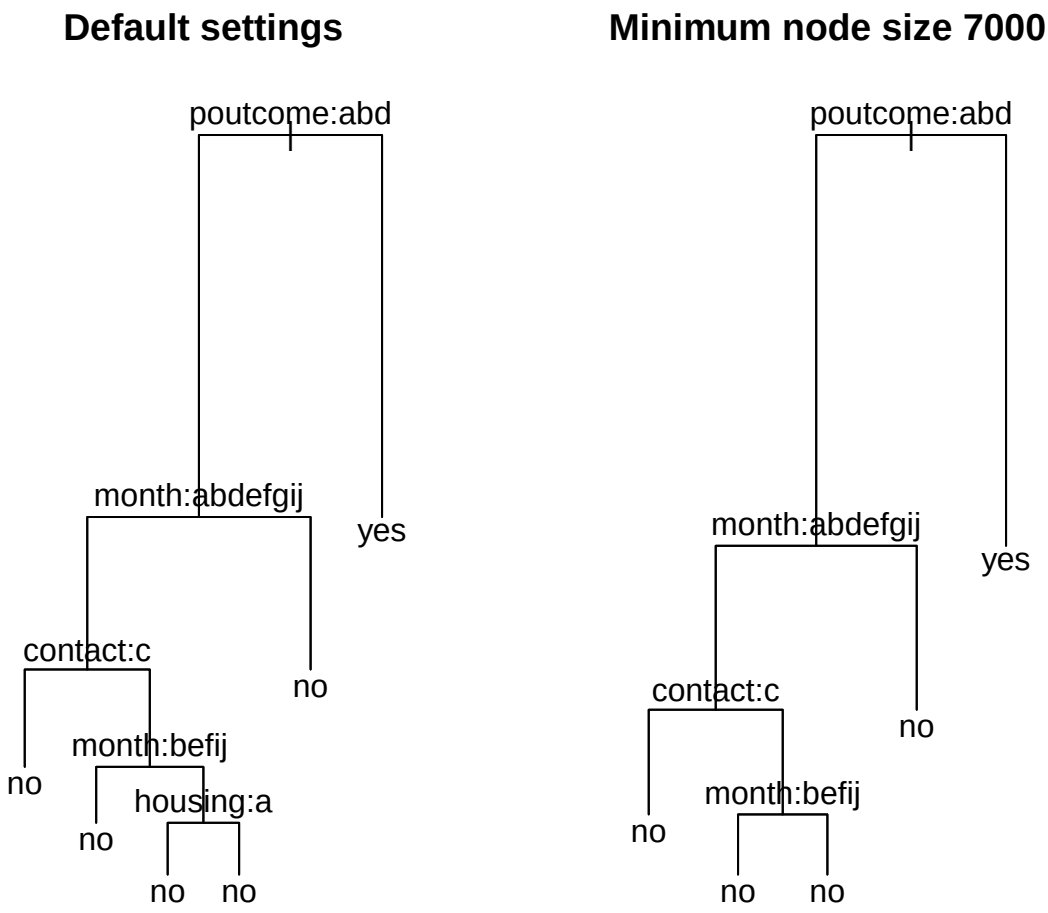
2.2 b) Minimum node size 7000

The second tree is fitted by setting the minimum amount of observations allowed in a leaf to 7000:

```
tree_nodesize <- tree(formula = y~.,
                      data = train,
                      control = tree.control(nobs = 18084,
                                             minsize = 7000))

trainmis_nodesize <- summary(tree_nodesize)[[7]][1] / summary(tree_nodesize)[[7]][2]
validmis_nodesize <- sum(predict(tree_nodesize,
                                 newdata = valid,
                                 type = "class") != valid$y) / nrow(valid)
```

With these settings the misclassification rate for the training set is 0.105 and for the validation set it is 0.109. As can be observed these values are the same as for the default settings. To see why this is the case the two decision trees are plotted with the following code:

```
par(mfrow=c(1,2))
plot(tree_default)
title("Default settings")
text(tree_default)
plot(tree_nodesize)
title("Minimum node size 7000")
text(tree_nodesize)
```



As the plot the plot shows that the difference when setting the minimum node size to 7000 is that the final split, based on the housing variable, no longer occurs. However, in both leaves following this split the outcome was no. As such removing these two leaves from the tree has no outcome on the final predictions, as the outcome of the parent node for the two removed leaves is also no.

2.2 c) Minimum deviance 0.0005

The third and final tree is fitted by setting the minimum allowed deviance in a leaf to 0.0005:

```
tree_deviance <- tree(formula = y~.,
                      data = train,
                      control = tree.control(nobs = 18084,
                                             mindev = 0.0005))

trainmis_deviance <- summary(tree_deviance)[[7]][1] / summary(tree_deviance)[[7]][2]
validmis_deviance <- sum(predict(tree_deviance,
                                 newdata = valid,
```

```
                                    type = "class") != valid$y) / nrow(valid)
```

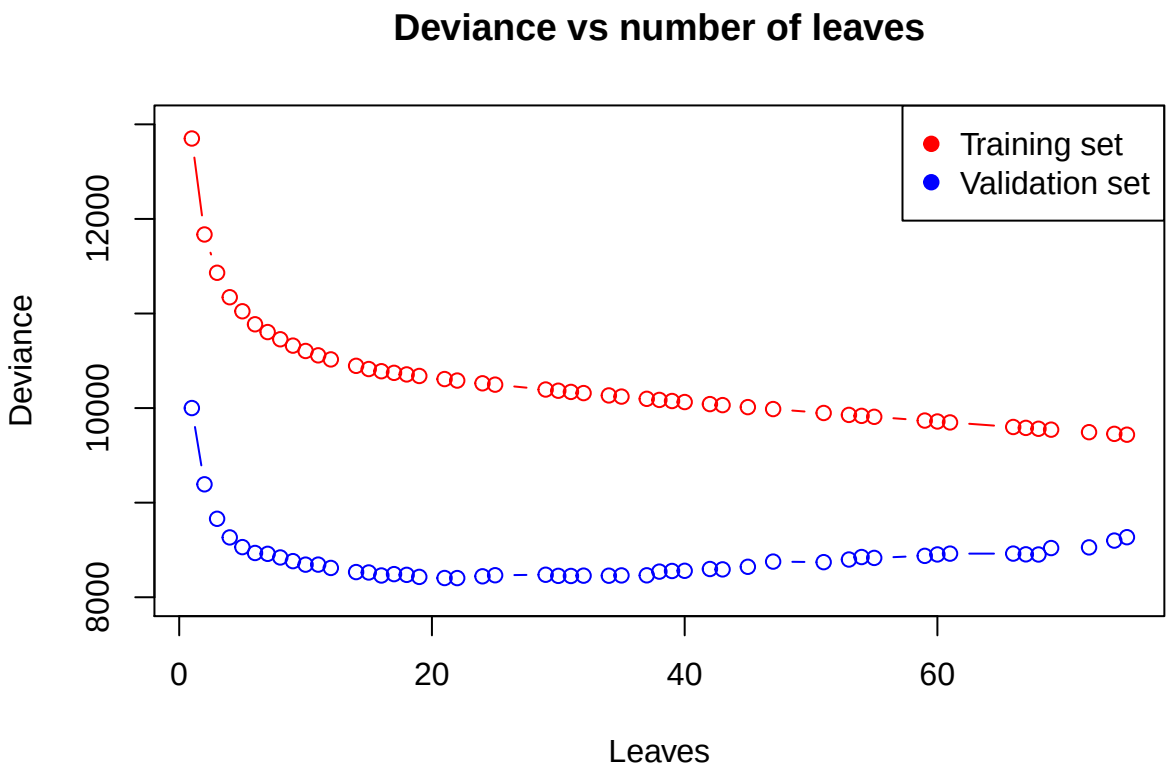
With these settings the misclassification rate for the training set is 0.094 and for the validation set it is 0.112. These values suggest that these settings have caused some overfitting, as the error for the training set is better compared to the previous trees, but for the validation set the error is worse. This is the expected result when allowing for a lower deviance, as this leads to nodes with higher purity, i.e. nodes with a higher rate of observations from only one class. This results in a more complicated model, which performs better on the data it was trained on, but the complicated structure is unlikely to be replicated in a general setting and as such performs worse on new data.

Out of these three models the best one would appear to be the second model, where the minimum node size was set to 7000. The reason for this is that it performs just as well as the model using the default settings while also being a simpler model with fewer leaves.

2.3 - Optimal amount of leaves for model 2.2 c

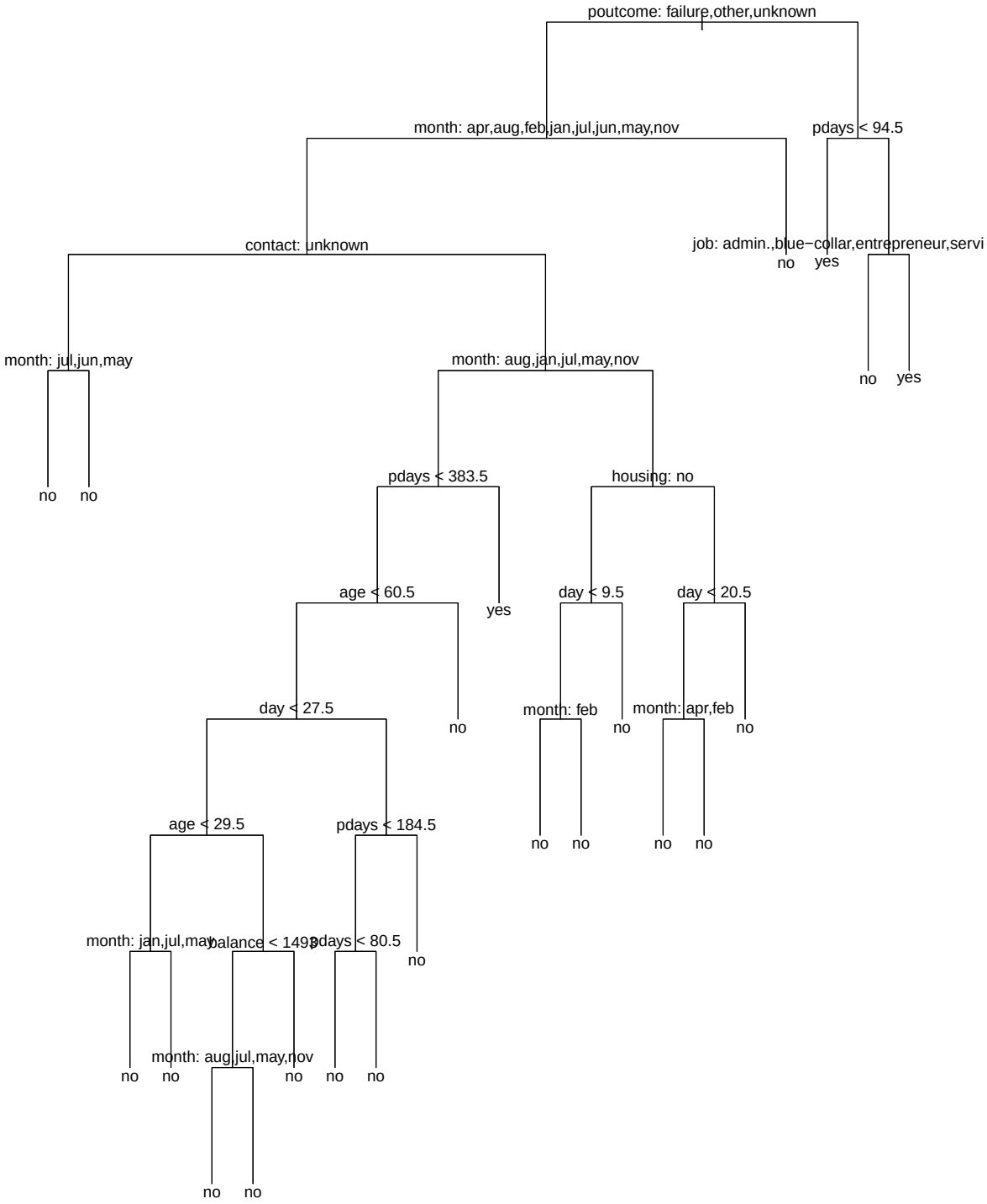
In this section we take the overfitted model from 2.2 c and try to prune it to find an optimal amount of leaves, up to a maximum of 50 leaves. This is done by pruning the tree from the maximum amount of leaves down to the root node and calculating the deviance in each step. This is done with the following code:

```
trainprune <- prune.tree(tree_deviance)
validprune <- prune.tree(tree_deviance, newdata = valid)

plot(rev(trainprune$size)[1:50],
     rev(trainprune$dev)[1:50],
     type = "b",
     col="red",
     ylim = c(8000,13000),
     main = "Deviance vs number of leaves",
     xlab = "Leaves",
     ylab = "Deviance")
points(rev(validprune$size)[1:50],
       rev(validprune$dev)[1:50],
       type = "b",
       col="blue")
legend("topright",
       legend = c("Training set", "Validation set"),
       col = c("red","blue"),
       pch = c(19,19))
```



From the graph it can be observed that the smallest deviance for the validation set is obtained when the amount of leaves is 22. To explore this tree further it is plotted with the following code:

```
finaltree <- prune.tree(tree_deviance, best = 22)
plot(finaltree, type = "uniform")
title("Tree with optimal amount of leaves")
text(finaltree, pretty = 0)
```



One of the most important variables is poutcome, outcome of previous marketing campaign, with the

previous campaign being a success the main reason for an observation to be classified as "yes". It can still be the case that the client has not subscribed even though the previous campaign was a success, that is if the person is employed as an admin, blue-collar, entrepreneur, services or a technician. It should however be noted here that the probability for "no" is just 0.507 compared to 0.493 for "yes" so it is possible that the "no" vote here is due to random chance when the training set was partitioned.

It can also be observed that there is one situation where even if the outcome of the previous campaign was not a success the client is predicted to have subscribed. The variable seemingly responsible for this is pdays, the number of days that passed by after the client was last contacted from a previous campaign. If the previous campaign was not a success and the client was contacted by cellular or telephone in the months of August, January, July, May or November, then if more than 383.5 days have passed since the last contact the observation is predicted to be "yes". Finally the confusion matrix and misclassification rate for the test set is calculated with the following code:

```
yfit <- predict(finaltree, newdata = test, type = "class")
table("True"=test$y, "Predicted"=yfit)
```

```
##      Predicted
## True     no   yes
##   no  11872   107
##   yes  1371   214
```

```
1-(sum(diag(table("True"=test$y, "Predicted"=yfit)))/nrow(test))
```

```
## [1] 0.1089649
```

While misclassification for the test set is only 0.109 the confusion matrix shows that the predictive power for this model seems to be rather poor in terms of being able to correctly classify "yes" outcomes. With only 214 correct predictions and 1371 incorrect predictions for yes this does not appear to be a very good model.

2.4 - Classification with new loss function

In this section we instead attempt to find the optimal tree from 2.2 c by using a different loss function:

```
matrix(c(0,1,5,0), nrow=2, byrow = T)
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    5    0
```

to punish the model more for incorrectly predicting "yes" as "no" which was observed to be a major issue in the previous section. This is done with the following code:

```
loss <- prune.tree(tree_deviance, method = "misclass", loss = matrix(c(0,1,5,0), nrow=2, byrow = T))
pruned_loss <- prune.tree(tree_deviance, best = loss$size[which.min(loss$dev)])
fit_loss <- predict(pruned_loss, newdata = test, type = "class")
table("True"=test$y, "Predicted"=fit_loss)
```

```
##      Predicted
## True     no   yes
##   no  11779   200
##   yes  1262   323
```

```
1-(sum(diag(table("True"=test$y, "Predicted"=fit_loss)))/nrow(test))
```

```
## [1] 0.1077853
```

It can be observed that the chosen model is slightly better at predicting "yes" correctly, with 109 more correct predictions. However, as a result of this 93 extra "no" outcomes have now been incorrectly predicted as "yes"

such that the overall misclassification error has only decreased by 0.0011796. This is because as the penalty for incorrectly predicting "yes" as "no" is now five times greater than the reverse situation the model will to a greater extent predict "yes" in cases where the probabilities for the two outcomes are similar.

2.5 - Optimal tree versus Naïve Bayes model

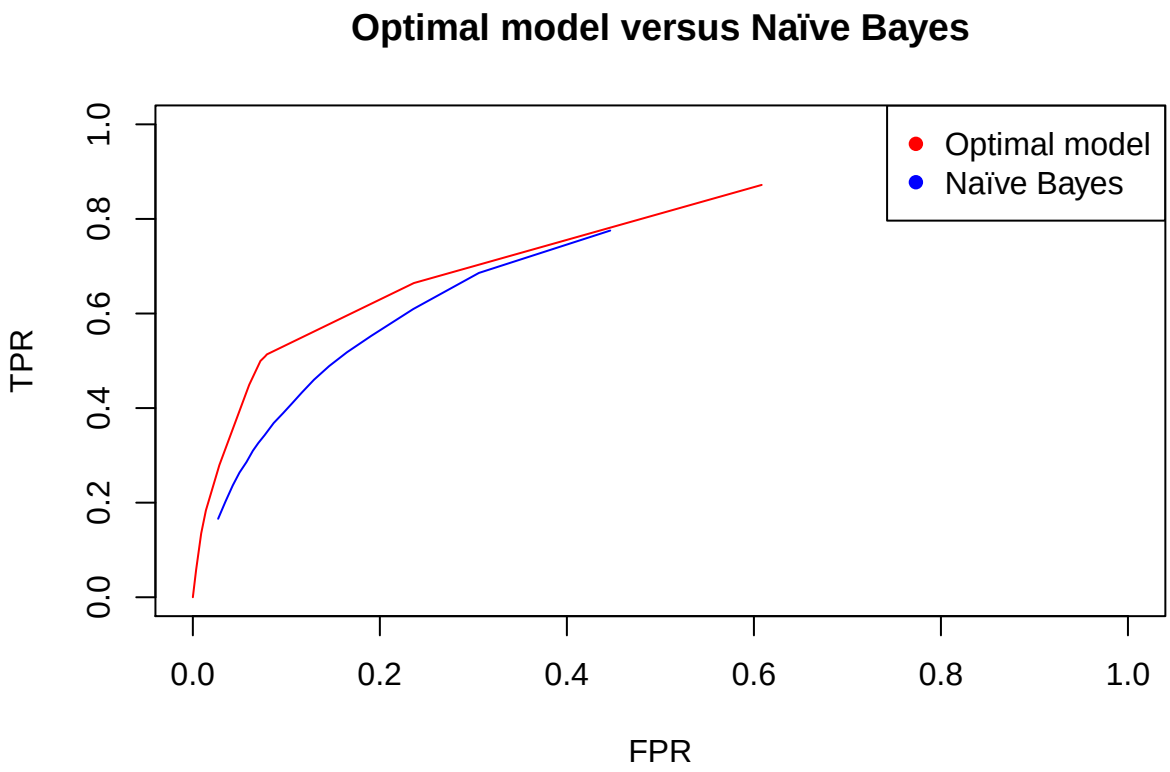
The final step of this assignment is to compare how the optimal tree from the previous section performs versus a Naïve Bayes model by classifying the test data with the following principle:

$$\hat{Y} = 1 \;\; if \;\; p(Y = yes|X) > \pi, \;\; otherwise \;\; \hat{Y} = 0$$

where $\pi = 0.05, 0.1, 0.15, ..., 0.9, 0.95$. For each value of π the TPR and FPR values are calculated and used to plot the ROC curves for the two models. This is done with the following code:

```
# Optimal model
testprobs <- predict(finaltree, newdata = test)
probseq <- seq(from=0.05, to=0.95, by=0.05)
testpreds <- data.frame(matrix(0, nrow = nrow(test), ncol = length(probseq)))

for (obs in 1:nrow(test)) {
  for (prob in 1:length(probseq)) {
    if(testprobs[obs,2]>probseq[prob]){
      testpreds[obs,prob] <- "yes"
    }else{
      testpreds[obs,prob] <- "no"
    }
  }
}

tpr <- 0
fpr <- 0
for (i in 1:19) {
  t <- table( "True"= test$y, "Prediction" = testpreds[,i])
  if(dim(t)[2]==2){
    tpr[i] <- t[2,2]/(t[2,2]+t[2,1])
    fpr[i] <- t[1,2]/(t[1,2]+t[1,1])
  }else{
    tpr[i] <- 0
    fpr[i] <- 0
  }
}

# Naive Bayes
naive_bayes <- naiveBayes(y~.,
                          data = train)
naive_bayes_preds <- predict(naive_bayes,
                             newdata = test,
                             type = "raw")
testpreds_bayes <- data.frame(matrix(0, nrow = nrow(test), ncol = length(probseq)))

for (obs in 1:nrow(test)) {
  for (prob in 1:length(probseq)) {
    if(naive_bayes_preds[obs,2]>probseq[prob]){
      testpreds_bayes[obs,prob] <- "yes"
```

```
    }else{
      testpreds_bayes[obs,prob] <- "no"
    }
  }
}

tpr_bayes <- 0
fpr_bayes <- 0

for (i in 1:19) {
  t <- table("True"= test$y, "Prediction" = testpreds_bayes[,i])
  if(dim(t)[2]==2){
    tpr_bayes[i] <- t[2,2]/(t[2,2]+t[2,1])
    fpr_bayes[i] <- t[1,2]/(t[1,2]+t[1,1])
  }else{
    tpr_bayes[i] <- 0
    fpr_bayes[i] <- 0
  }
}

plot(fpr, tpr,
     type = "l",
     col="red",
     xlim = c(0,1),
     ylim = c(0,1),
     main = "Optimal model versus Naïve Bayes",
     xlab = "FPR",
     ylab = "TPR")
lines(fpr_bayes, tpr_bayes, col="blue")
legend("topright",
       legend = c("Optimal model","Naïve Bayes"),
       col = c("red","blue"),
       pch = 16)
```



The plot shows that the ROC curve for the optimal model has a greater area underneath it compared to the Naïve Bayes model, suggesting that it is a better classifier.

Assignment 3: Principal components for crime level analysis.

3.1. ***Scale all variables except of ViolentCrimesPerPop and implement PCA by using function eigen(). Report how many features are needed to obtain at least 95% of variance in the data. What is the proportion of variation explained by each of the first two principal components?***

```
library(ggplot2)
library(viridis)
library(wesanderson)
library(tidyverse)

Data <- read.csv("communities.csv") #1994 X 101
 # 3.1
 df1 <- Data[,-ncol(Data)]
 #scaling  and centering the data
 df1 <- scale(df1,center = TRUE,scale = TRUE) # 1994 x 100
 # performing PCA with eigen()

 # center with 'colMeans()'
 center_colmeans <- function(x) {
   xcenter = colMeans(x)
   x - rep(xcenter, rep.int(nrow(x), ncol(x)))
 }
 df1<-center_colmeans(df1)

 # step1: Computing Cov xx.T
 x_tilda <- cov(df1)
 # step2: computing eigen vector of covariance matrix

 df2 <-eigen(x_tilda)
 e_values <- df2$values

# step 3: getting new coordinate for x
 Z <- df1 %*% df2$vectors
 v1<- 0
 temp1 <- sum(e_values) # sum of the e_values is 100
 for (i in 1:length(e_values)) {
   if (v1 <(0.95*sum(e_values))) {
     v1 <- v1 + e_values[i]
     index <-i
     }
 }
 cat("we need ",index, "components to describe 95% of variance in the data\n")
```

```
## we need  35 components to describe 95% of variance in the data
```

```
 temp2 <- round(e_values[1]+e_values[2])
 cat("component 1 and 2 describe about ", temp2,"% of the variance in the data\n")
```

```
## component 1 and 2 describe about  42 % of the variance in the data
```

```
 cat("component 1 describes ",round(e_values[1]),"% of the variance in the data \n")
```

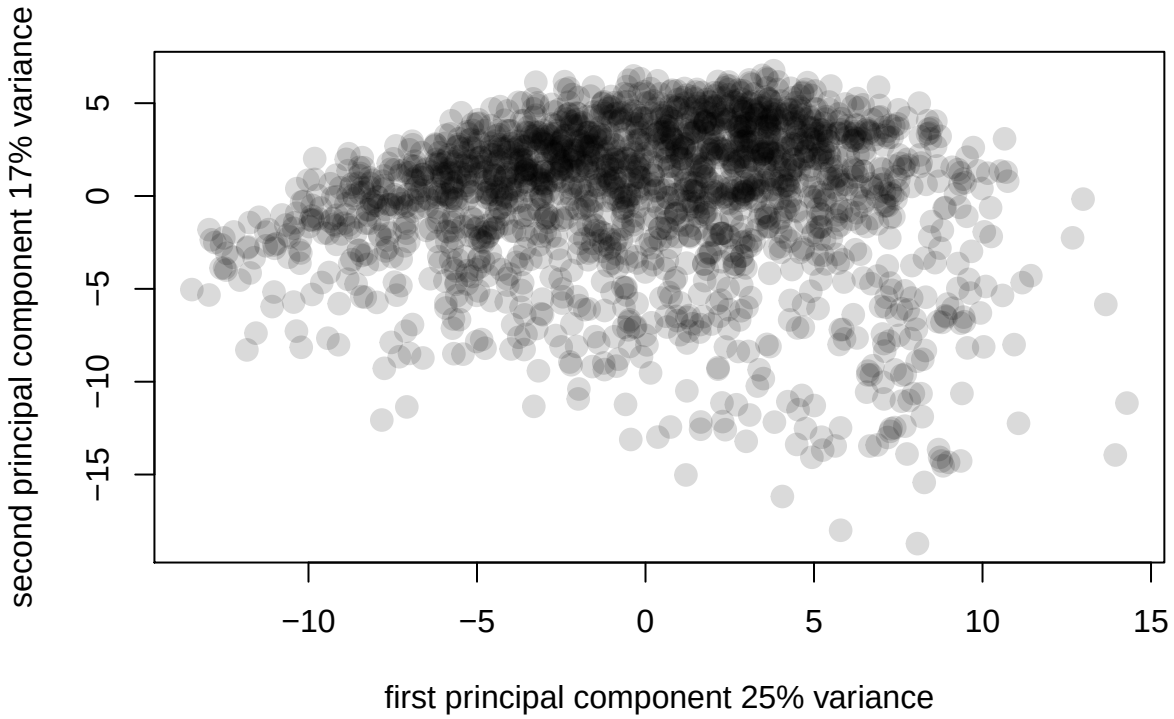
```
## component 1 describes  25 % of the variance in the data
```

```
cat("component 2 describes ",round(e_values[2]), "% of the variance in the data \n")
```

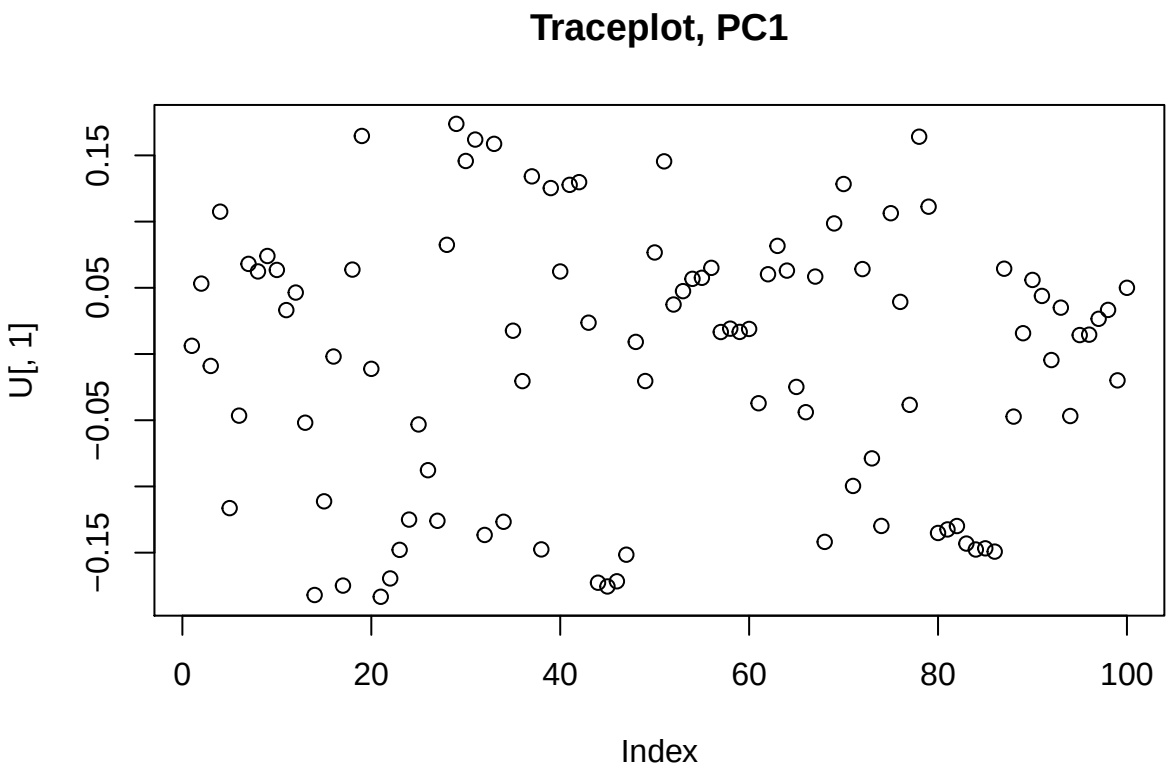
```
## component 2 describes  17 % of the variance in the data
```

3.2 Repeat PCA analysis by using princomp() function and make the score plot of the first principle component. Do many features have a notable contribution to this component? Report which 5 features contribute mostly (by the absolute value) to the first principle component. Comment whether these features have anything in common and whether they may have a logical relationship to the crime level. Also provide a plot of the PC scores in the coordinates (PC1, PC2) in which the color of the points is given by ViolentCrimesPerPop. Analyse this plot

```
PCA <- princomp(df1,scores = TRUE)
survey <- summary(PCA)
U <- PCA$loadings
pca1_plot2_1 <- plot(PCA$scores[, 1], PCA$scores[, 2],
                     cex = 1.5,
                     xlab = "first principal component 25% variance",
                     ylab = "second principal component 17% variance"
, pch = 19, col = rgb(0, 0, 0, 0.15))
```



```r
plot(U[,1], main="Traceplot, PC1")
```




```r
pos <- which(abs(U[,1]) >= 0.15)
cat(length(pos) ,"feature has a notable contribution to pc1 as they have
    absolute value greater than 0.15")
```

```
## 13 feature has a notable contribution to pc1 as they have
##     absolute value greater than 0.15
```

```r
cat("Name of top contributing features: \n")
```

```
## Name of top contributing features:
```

```r
print(names(head(U[,1][pos],5)))
```

```
## [1] "medIncome"   "pctWInvInc"  "pctWPubAsst" "medFamInc"   "perCapInc"
```

According to UCI website these columns are :

- medIncome: median household income
- pctWInvInc: percentage of households with investment / rent income in 1989
- pctWPubAsst: percentage of households with public assistance income in 1989
- medFamInc: median family income (differs from household income for non-family households)
- perCapInc: per capita income

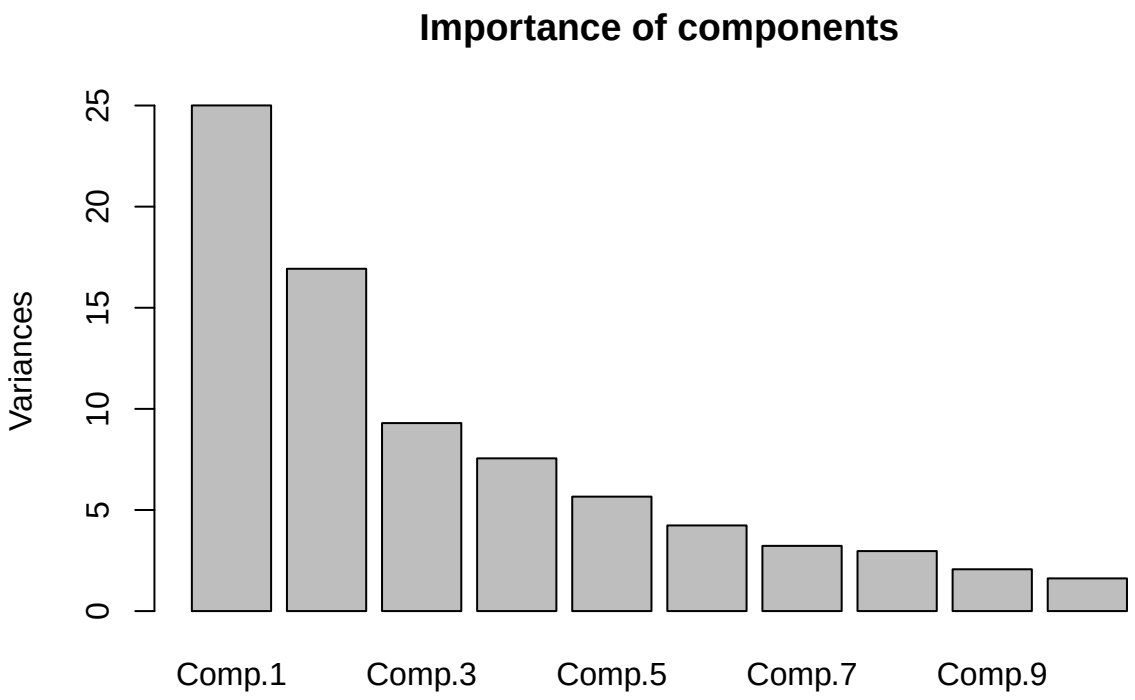
We can see that the value of household income and investment had a direct effect on the value of violent crime. So, we clearly see that household welfare is an important factor and these feature have a logical

relationship to the crime level.

```
new_df <- data.frame(PCA$scores[, 1:2])
 gg <- ggplot (new_df) +
    geom_point (( aes(x =new_df[,1] , y = new_df[,2],colour = Data$ViolentCrimesPerPop ))) +
      xlab (" first principal component 25% variance ")+
   ylab (" second principal component 17% variance")+ ggtitle("PC scores")
 gg+   theme_minimal()+  scale_color_viridis(option = "D")
```

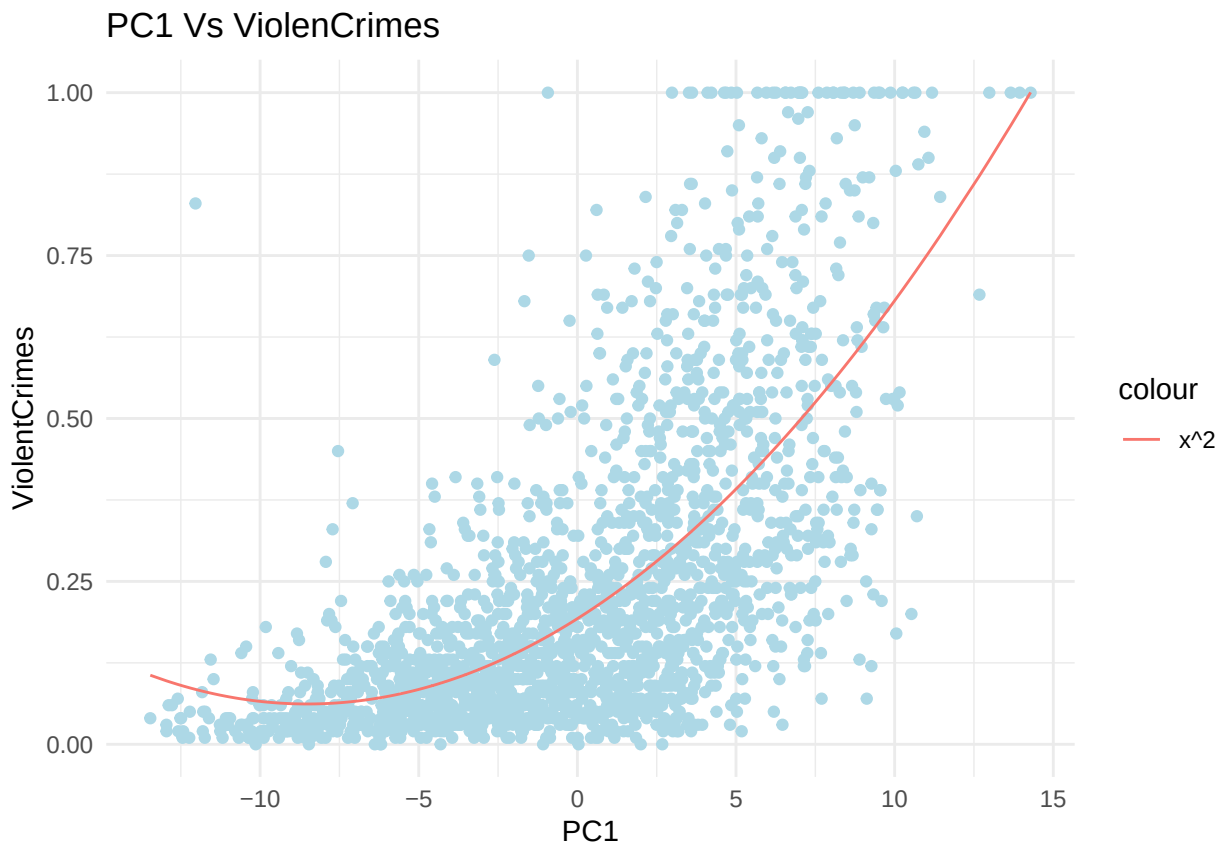
In the following plot we can see the contribution of each component. As expected, PC1 has the largest value.

```
plot(PCA,main = "Importance of components")
```



3.3 ***Assume a second order polynomial regression model in which ViolentCrimesPerPop is target and PC1 is the feature. Compute this model using lm() function (hint: use poly() function within the formula), make a scatterplot of the target versus the feature and present also the predicted values in this plot. Can the target be well explained by this feature? Does the model seem to capture the connection between the target and the feature?***

```
# 3.3

data_t = data.frame("X" = new_df[,1],
                    "Y" = Data$ViolentCrimesPerPop)
x.2 <- lm(Y ~ poly(poly(X, 2)) ,
          data = data_t)

gg <- ggplot(data_t, aes(x = X, y = Y)) +
   geom_point(color="lightblue")+
   ggtitle("PC1 Vs ViolenCrimes")+
   geom_line(aes(x = X, y = predict(x.2), colour = "x^2")) +
   theme_minimal() +
   xlab("PC1") + ylab("ViolentCrimes")
gg
```

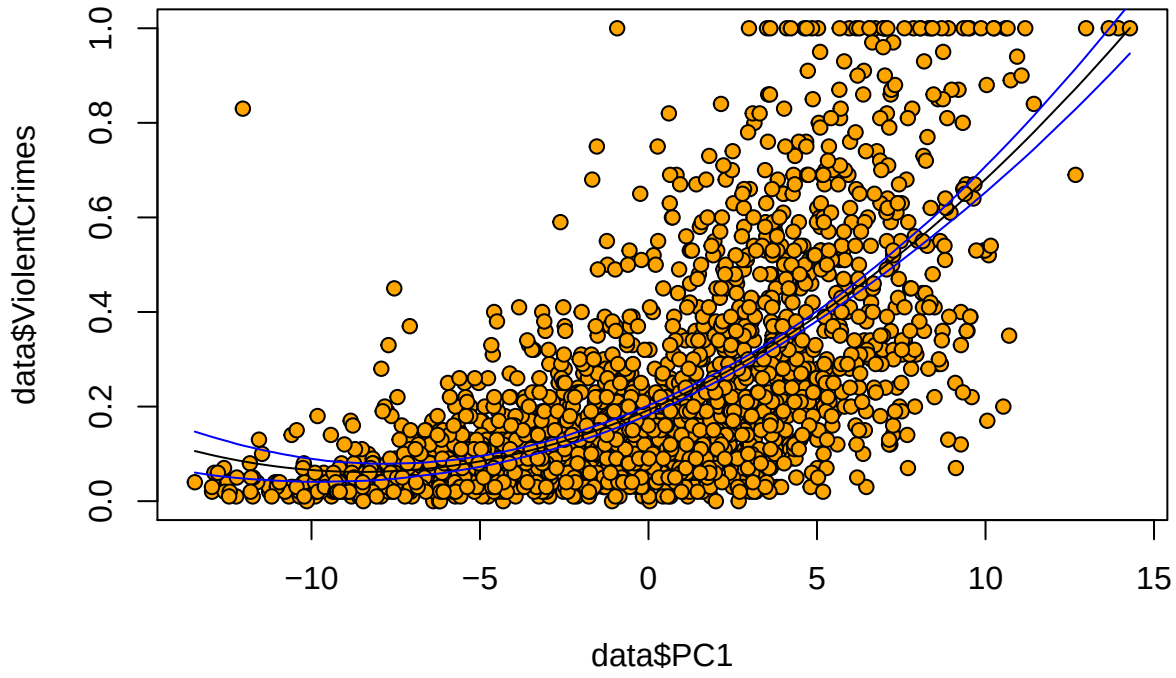


From the plot, it is reasonable to fit a quadratic linear regression to our data set. It is not a perfect model of our data but the general behavior of the data seems to be captured by a second-degree polynomial. ## 3.4 *Use parametric bootstrap to estimate the confidence and prediction bands from the model from step 3 and add these bands into the plot from step 3. What can be concluded by looking at a) confidence intervals b) prediction intervals?*

a)

```
# 3.4
library(boot)
# based on slide 27 in lecture 2c block 1

data <- new_df
data$ViolentCrimes <- Data$ViolentCrimesPerPop
colnames(data)<-c('PC1','PC2','ViolentCrimes')

data2=data[order(data$PC1),]
mle <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)

rng <- function(data,mle){
   n <- length(data$ViolentCrimes)
   data1=data.frame(ViolentCrimes=data$ViolentCrimes, PC1=data$PC1)
   #generate new ViolentCrimes
   data1$ViolentCrimes<- rnorm(n,predict(mle, newdata=data1),sd(mle$residuals))
   return(data1)}
```

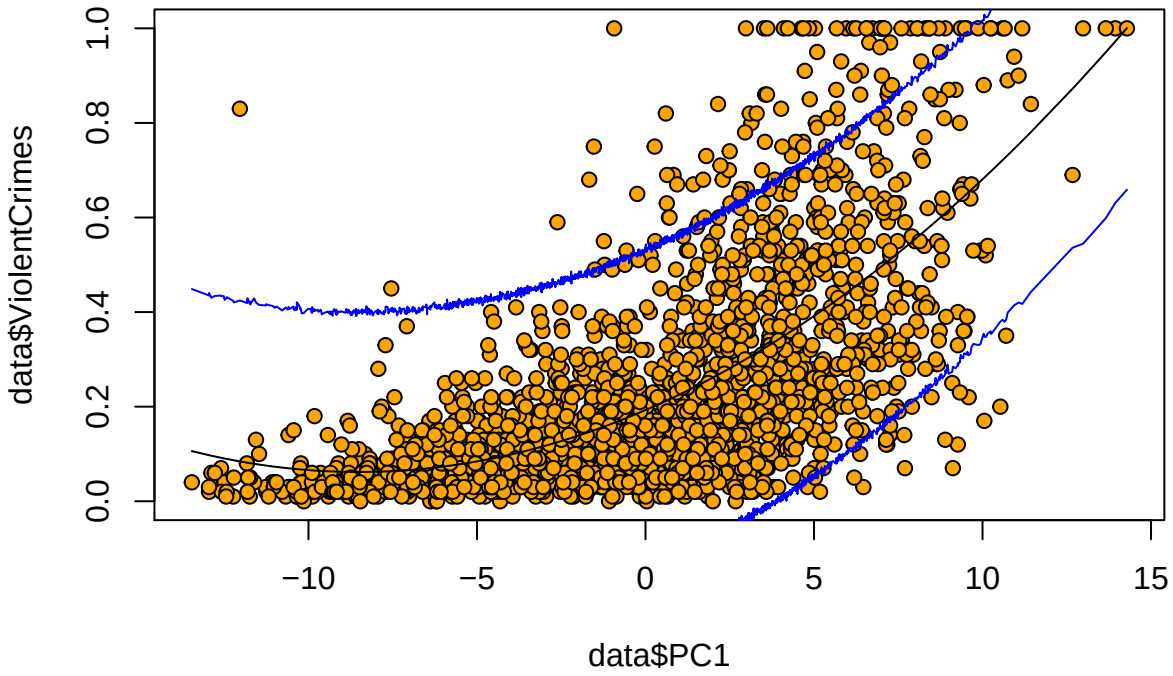
```r
f1<-function(data1){
   res= lm(ViolentCrimes ~ poly(PC1,2), data = data1)
   ##predict values for all PC1 values from the original data
   ViolentCrimesP=predict(res,newdata=data2)
   return(ViolentCrimesP)}

res=boot(data2, statistic=f1, R=1000, mle = mle,ran.gen=rng  , sim="parametric")
e <- envelope(res)


fit <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)
ViolentCrimesP=predict(fit)
plot(data$PC1,data$ViolentCrimes , pch=21, bg="orange")
points(data2$PC1,ViolentCrimesP,type="l") #plot fitted line
#plot confidence bands
points(data2$PC1,e$point[2,], type="l", col="blue")
points(data2$PC1,e$point[1,], type="l", col="blue")
```



In this assignment we performed a parametric bootstrap.We repeat this bootstrap method 1000 times in order to calculate bands.The model does fall inside the confidence band. The confidence band is larger at the both ends, that is because we have less data in these regions as a result, we have little knowledge about the effect and further information is needed.

b)

```
# Prediction bands
mle <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)
f1=function(data1){
   # fit polynomial model

   res=lm(ViolentCrimes ~ poly(PC1,2), data=data1)
   #predict values for all PC1 values from the original data

   ViolentCrimesP=predict(res,newdata=data2)
   n=length(data2$ViolentCrimes)
   predictedP=rnorm(n,ViolentCrimesP,
                    sd(mle$residuals))
   return(predictedP)
}
res=boot(data2, statistic=f1, R=10000,
         mle=mle,ran.gen=rng, sim="parametric")
e <- envelope(res)


fit <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)
ViolentCrimesP=predict(fit)
plot(data$PC1,data$ViolentCrimes , pch=21, bg="orange")
points(data2$PC1,ViolentCrimesP,type="l")
points(data2$PC1,e$point[2,], type="l", col="blue")
points(data2$PC1,e$point[1,], type="l", col="blue")
```



As expected the prediction band is wider than the confidence interval. There are still some data that fall out of the prediction interval.It is logical to consider them as noise. Due to the fact that, the prediction interval is a range that is more likely to contain the response value of the new observation under our second-degree polynomial regression.

Appendix:

```
knitr::opts_chunk$set(echo = TRUE)
library(tree)
library(e1071)
bank <- read.csv2("bank-full.csv")

# Remove the "duration" variable
bank <- bank[,-12]

# Convert character variables to factors
for (var in 1:ncol(bank)) {
  if(is.character(bank[,var])){
    bank[,var] <- as.factor(bank[,var])
  }
}

# Train, validation, test sets
n=dim(bank)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=bank[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=bank[id2,]

id3=setdiff(id1,id2)
test=bank[id3,]
tree_default <- tree(formula = y~.,
                     data = train)

trainmis_default <- summary(tree_default)[[7]][1] / summary(tree_default)[[7]][2]
validmis_default <- sum(predict(tree_default,
                                newdata = valid,
                                type = "class") != valid$y) / nrow(valid)
tree_nodesize <- tree(formula = y~.,
                      data = train,
                      control = tree.control(nobs = 18084,
                                             minsize = 7000))

trainmis_nodesize <- summary(tree_nodesize)[[7]][1] / summary(tree_nodesize)[[7]][2]
validmis_nodesize <- sum(predict(tree_nodesize,
                                 newdata = valid,
                                 type = "class") != valid$y) / nrow(valid)
par(mfrow=c(1,2))
plot(tree_default)
title("Default settings")
text(tree_default)
plot(tree_nodesize)
title("Minimum node size 7000")
text(tree_nodesize)
```

```r
tree_deviance <- tree(formula = y~.,
                      data = train,
                      control = tree.control(nobs = 18084,
                                             mindev = 0.0005))

trainmis_deviance <- summary(tree_deviance)[[7]][1] / summary(tree_deviance)[[7]][2]
validmis_deviance <- sum(predict(tree_deviance,
                                 newdata = valid,
                                 type = "class") != valid$y) / nrow(valid)
trainprune <- prune.tree(tree_deviance)
validprune <- prune.tree(tree_deviance, newdata = valid)

plot(rev(trainprune$size)[1:50],
     rev(trainprune$dev)[1:50],
     type = "b",
     col="red",
     ylim = c(8000,13000),
     main = "Deviance vs number of leaves",
     xlab = "Leaves",
     ylab = "Deviance")
points(rev(validprune$size)[1:50],
       rev(validprune$dev)[1:50],
       type = "b",
       col="blue")
legend("topright",
       legend = c("Training set", "Validation set"),
       col = c("red","blue"),
       pch = c(19,19))
finaltree <- prune.tree(tree_deviance, best = 22)
plot(finaltree, type = "uniform")
title("Tree with optimal amount of leaves")
text(finaltree, pretty = 0)
yfit <- predict(finaltree, newdata = test, type = "class")
table("True"=test$y, "Predicted"=yfit)
1-(sum(diag(table("True"=test$y, "Predicted"=yfit)))/nrow(test))
matrix(c(0,1,5,0), nrow=2, byrow = T)
loss <- prune.tree(tree_deviance, method = "misclass", loss = matrix(c(0,1,5,0), nrow=2, byrow = T))
pruned_loss <- prune.tree(tree_deviance, best = loss$size[which.min(loss$dev)])
fit_loss <- predict(pruned_loss, newdata = test, type = "class")
table("True"=test$y, "Predicted"=fit_loss)
1-(sum(diag(table("True"=test$y, "Predicted"=fit_loss)))/nrow(test))
# Optimal model
testprobs <- predict(finaltree, newdata = test)
probseq <- seq(from=0.05, to=0.95, by=0.05)
testpreds <- data.frame(matrix(0, nrow = nrow(test), ncol = length(probseq)))

for (obs in 1:nrow(test)) {
  for (prob in 1:length(probseq)) {
    if(testprobs[obs,2]>probseq[prob]){
      testpreds[obs,prob] <- "yes"
    }else{
      testpreds[obs,prob] <- "no"
    }
```

```
  }
}

tpr <- 0
fpr <- 0
for (i in 1:19) {
  t <- table( "True"= test$y, "Prediction" = testpreds[,i])
  if(dim(t)[2]==2){
    tpr[i] <- t[2,2]/(t[2,2]+t[2,1])
    fpr[i] <- t[1,2]/(t[1,2]+t[1,1])
  }else{
    tpr[i] <- 0
    fpr[i] <- 0
  }
}

# Naive Bayes
naive_bayes <- naiveBayes(y~.,
                          data = train)
naive_bayes_preds <- predict(naive_bayes,
                             newdata = test,
                             type = "raw")
testpreds_bayes <- data.frame(matrix(0, nrow = nrow(test), ncol = length(probseq)))

for (obs in 1:nrow(test)) {
  for (prob in 1:length(probseq)) {
    if(naive_bayes_preds[obs,2]>probseq[prob]){
      testpreds_bayes[obs,prob] <- "yes"
    }else{
      testpreds_bayes[obs,prob] <- "no"
    }
  }
}

tpr_bayes <- 0
fpr_bayes <- 0

for (i in 1:19) {
  t <- table("True"= test$y, "Prediction" = testpreds_bayes[,i])
  if(dim(t)[2]==2){
    tpr_bayes[i] <- t[2,2]/(t[2,2]+t[2,1])
    fpr_bayes[i] <- t[1,2]/(t[1,2]+t[1,1])
  }else{
    tpr_bayes[i] <- 0
    fpr_bayes[i] <- 0
  }
}

plot(fpr, tpr,
     type = "l",
     col="red",
     xlim = c(0,1),
     ylim = c(0,1),
```

```
     main = "Optimal model versus Naïve Bayes",
     xlab = "FPR",
     ylab = "TPR")
lines(fpr_bayes, tpr_bayes, col="blue")
legend("topright",
       legend = c("Optimal model","Naïve Bayes"),
       col = c("red","blue"),
       pch = 16)
################################
#Assignment 3
################################
library(ggplot2)
library(viridis)
library(wesanderson)
library(tidyverse)

Data <- read.csv("communities.csv") #1994 X 101
 # 3.1
 df1 <- Data[,-ncol(Data)]
 #scaling  and centering the data
 df1 <- scale(df1,center = TRUE,scale = TRUE) # 1994 x 100
 # performing PCA with eigen()

 # center with 'colMeans()'
 center_colmeans <- function(x) {
   xcenter = colMeans(x)
   x - rep(xcenter, rep.int(nrow(x), ncol(x)))
 }
 df1<-center_colmeans(df1)

 # step1: Computing Cov xx.T
 x_tilda <- cov(df1)
 # step2: computing eigen vector of covariance matrix

 df2 <-eigen(x_tilda)
 e_values <- df2$values

# step 3: getting new coordinate for x
 Z <- df1 %*% df2$vectors
 v1<- 0
 temp1 <- sum(e_values) # sum of the e_values is 100
 for (i in 1:length(e_values)) {
   if (v1 <(0.95*sum(e_values))) {
     v1 <- v1 + e_values[i]
     index <-i
     }
 }
 cat("we need ",index, "components to describe 95% of variance in the data\n")
 temp2 <- round(e_values[1]+e_values[2])
 cat("component 1 and 2 describe about ", temp2,"% of the variance in the data\n")
 cat("component 1 describes ",round(e_values[1]),"% of the variance in the data \n")
 cat("component 2 describes ",round(e_values[2]), "% of the variance in the data \n")
```

```r
PCA <- princomp(df1,scores = TRUE)
survey <- summary(PCA)
U <- PCA$loadings
pca1_plot2_1 <- plot(PCA$scores[, 1], PCA$scores[, 2],
                     cex = 1.5,
                     xlab = "first principal component 25% variance",
                     ylab = "second principal component 17% variance"
, pch = 19, col = rgb(0, 0, 0, 0.15))

plot(U[,1], main="Traceplot, PC1")
pos <- which(abs(U[,1]) >= 0.15)
cat(length(pos) ,"feature has a notable contribution to pc1 as they have
    absolute value greater than 0.15")
cat("Name of top contributing features: \n")
print(names(head(U[,1][pos],5)))

new_df <- data.frame(PCA$scores[, 1:2])
 gg <- ggplot (new_df) +
    geom_point (( aes(x =new_df[,1] , y = new_df[,2],colour = Data$ViolentCrimesPerPop ))) +
      xlab (" first principal component 25% variance ")+
   ylab (" second principal component 17% variance")+ ggtitle("PC scores")
 gg+   theme_minimal()+  scale_color_viridis(option = "D")

plot(PCA,main = "Importance of components")

# 3.3

data_t = data.frame("X" = new_df[,1],
                    "Y" = Data$ViolentCrimesPerPop)
x.2 <- lm(Y ~ poly(poly(X, 2)) ,
          data = data_t)

gg <- ggplot(data_t, aes(x = X, y = Y)) +
   geom_point(color="lightblue")+
   ggtitle("PC1 Vs ViolenCrimes")+
   geom_line(aes(x = X, y = predict(x.2), colour = "x^2")) +
   theme_minimal() +
   xlab("PC1") + ylab("ViolentCrimes")
gg
# 3.4
library(boot)
# based on slide 27 in lecture 2c block 1

data <- new_df
data$ViolentCrimes <- Data$ViolentCrimesPerPop
colnames(data)<-c('PC1','PC2','ViolentCrimes')

data2=data[order(data$PC1),]
mle <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)

rng <- function(data,mle){
   n <- length(data$ViolentCrimes)
   data1=data.frame(ViolentCrimes=data$ViolentCrimes, PC1=data$PC1)
```

```
   #generate new ViolentCrimes
   data1$ViolentCrimes<- rnorm(n,predict(mle, newdata=data1),sd(mle$residuals))
   return(data1)}


f1<-function(data1){
   res= lm(ViolentCrimes ~ poly(PC1,2), data = data1)
   ##predict values for all PC1 values from the original data
   ViolentCrimesP=predict(res,newdata=data2)
   return(ViolentCrimesP)}

res=boot(data2, statistic=f1, R=1000, mle = mle,ran.gen=rng   , sim="parametric")
e <- envelope(res)


fit <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)
ViolentCrimesP=predict(fit)
plot(data$PC1,data$ViolentCrimes , pch=21, bg="orange")
points(data2$PC1,ViolentCrimesP,type="l") #plot fitted line
#plot confidence bands
points(data2$PC1,e$point[2,], type="l", col="blue")
points(data2$PC1,e$point[1,], type="l", col="blue")


# Prediction bands
mle <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)
f1=function(data1){
   # fit polynomial model

   res=lm(ViolentCrimes ~ poly(PC1,2), data=data1)
   #predict values for all PC1 values from the original data

   ViolentCrimesP=predict(res,newdata=data2)
   n=length(data2$ViolentCrimes)
   predictedP=rnorm(n,ViolentCrimesP,
                    sd(mle$residuals))
   return(predictedP)
}
res=boot(data2, statistic=f1, R=10000,
         mle=mle,ran.gen=rng, sim="parametric")
e <- envelope(res)


fit <- lm(ViolentCrimes ~ poly(PC1,2), data = data2)
ViolentCrimesP=predict(fit)
plot(data$PC1,data$ViolentCrimes , pch=21, bg="orange")
points(data2$PC1,ViolentCrimesP,type="l")
points(data2$PC1,e$point[2,], type="l", col="blue")
points(data2$PC1,e$point[1,], type="l", col="blue")
```