Computer Lab 2 Block 1

Linköping University

Jakob Fjellström
Mowniesh Asokan
Tim Yuki Washio

December 6, 2020

Mowniesh worked on assignment 1, Jakob on assignment 2, and Yuki on assignment 3.

Assignment 1: LDA and logistic regression

1.

Actually LDA tries to reduce dimensions of the feature set while retaining the information that discriminates output classes. LDA tries to find a decision boundary around each cluster of a class.The goal of a LDA is often to project a feature space (a data-set n-dimensional samples) into a smaller subspace k.

- It is linearly non-separable.
- Iris Setosa is linearly separable from the other two classes.so,that we can draw a line or hyper-plane to classify a each groups.

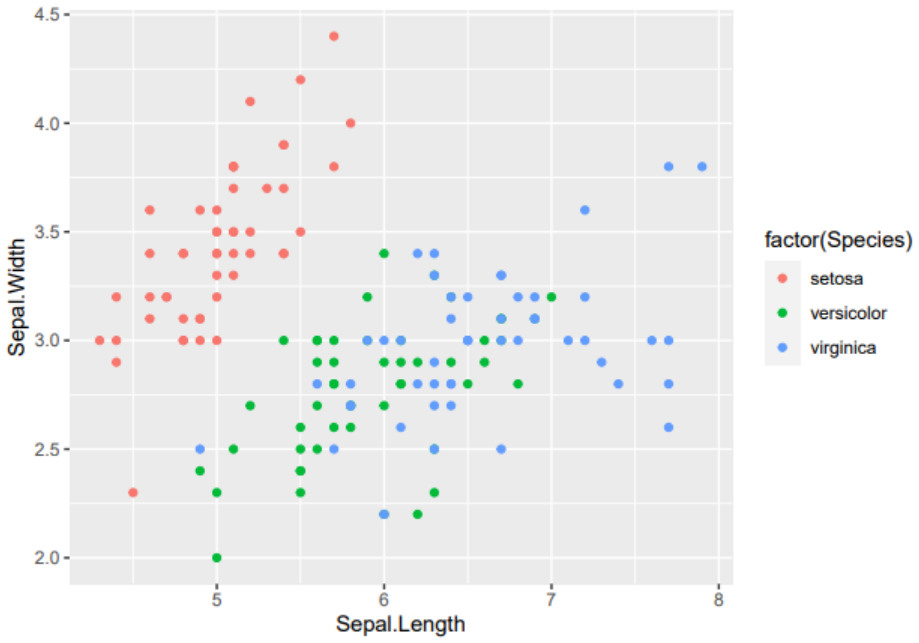


Figure 1

2.

R functions only to implement Linear Discriminant Analysis between the three species based on variables Sepal Length and Sepal Width: Setosa -1 Versicolor-2 Virginca -3.

2a. Mean, Covariance matrices (use cov()) and Prior probabilities per class.

```
## [1] "Prior probabilities of groups:"

## [1] 0.3333333 0.3333333 0.3333333

## [1] "Group means:"

##     [,1]  [,2]  [,3]
## x1 5.006 5.936 6.588
## x2 3.428 2.770 2.974

## [1] "covariance matrix of group 1"

##            x1         x2
## x1 0.12424898 0.09921633
## x2 0.09921633 0.14368980

## [1] "covariance matrix of group 2"

##            x1         x2
## x1 0.26643265 0.08518367
## x2 0.08518367 0.09846939

## [1] "covariance matrix of group 3"

##            x1         x2
## x1 0.40434286 0.09376327
## x2 0.09376327 0.10400408
```

Figure 2

2b.Pooled Covariance Matrix

```
## [1] "Pooled covariance matrix of groups"

##             x1         x2
## x1 0.26500816 0.09272109
## x2 0.09272109 0.11538776
```

Figure 3

2c. Probabilistic Model for LDA.

$$x|y = C_i, \mu_i, \Sigma \sim N(\mu_i, \Sigma)$$
$$y|\pi \sim Multinomial(\pi_1, ...\pi k)$$

Figure 4

2d. Discriminant Function.

```
# Discriminant function
disc_fn<-function(v,p_cv,m_g,p_g)
{
v<-as.matrix(v)
p_cv<-solve(p_cv)
d1<-((v%*%p_cv)%*%(m_g))
d2<-(0.5*t(m_g))%*%(p_cv)%*%(m_g)
t_d<-d1-(as.numeric(d2))+log(p_g)
return(t_d)
}
```

2e. Decision Boundary.

```
## Coefficients

##       Setosa Versicolor Virginica
## x1 47.343881   18.36513  12.21604
## x2 -8.833515   12.24331  17.58185
```

Figure 5

3.

Whether the error obtained using discriminant function is not same as error obtained by using LDA model, but the miss-classification rate of the two model are to be same.

- Both the model and function works perfectly for the class1 (setosa) and class3(virginica) , but it confuse with the class2(versicolor)

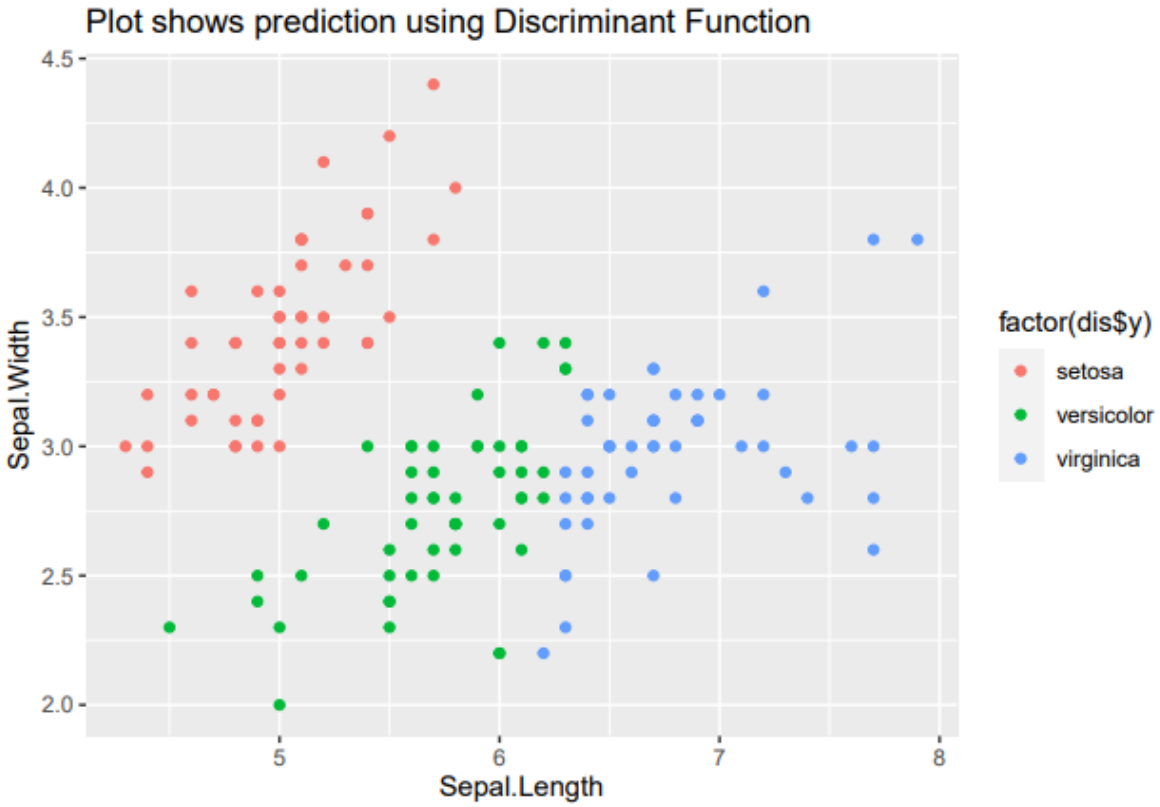


Figure 6

```
## Confusion Matrix of prediction using Discriminant function

##             y
##               1  2  3
##   setosa     49  0  0
##   versicolor  1 36 15
##   virginica   0 14 35

## Misclassification Rate using Discrimiant function

## [1] 0.2

## Confusion Matrix using LDA model

##
##      1  2  3
##   1 49  0  0
##   2  1 36 15
##   3  0 14 35

## Misclassification Rate using LDA model

## [1] 0.2
```

Figure 7

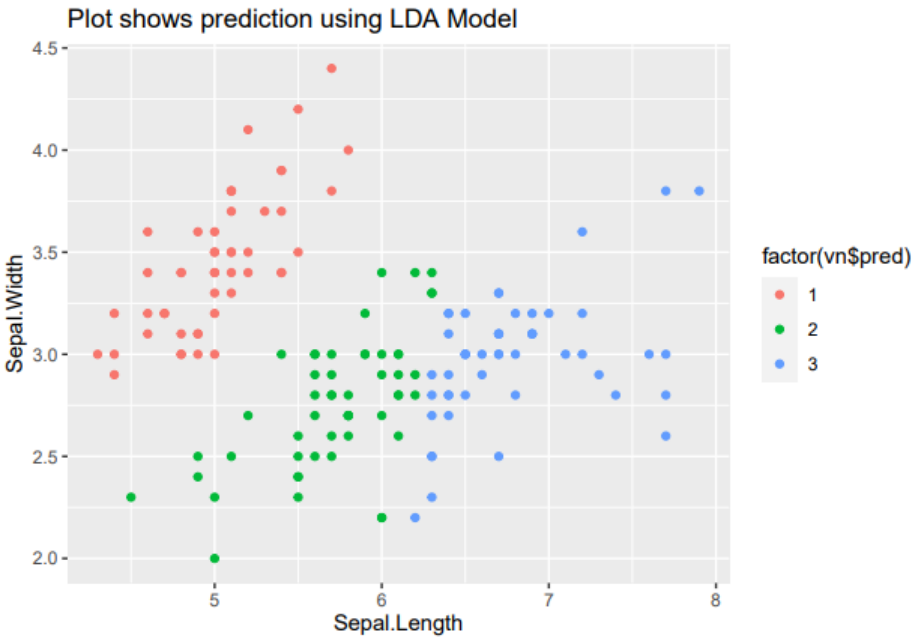


Figure 8

4.Sample the Data

Sample the data of iris using the multivariate normal distribution .By specifying the mean and sigma in the old dataset. we can generate a new sample from same mean and variance of each groups. so,the scatter plot of the generated data and the orginal data is looks similar.

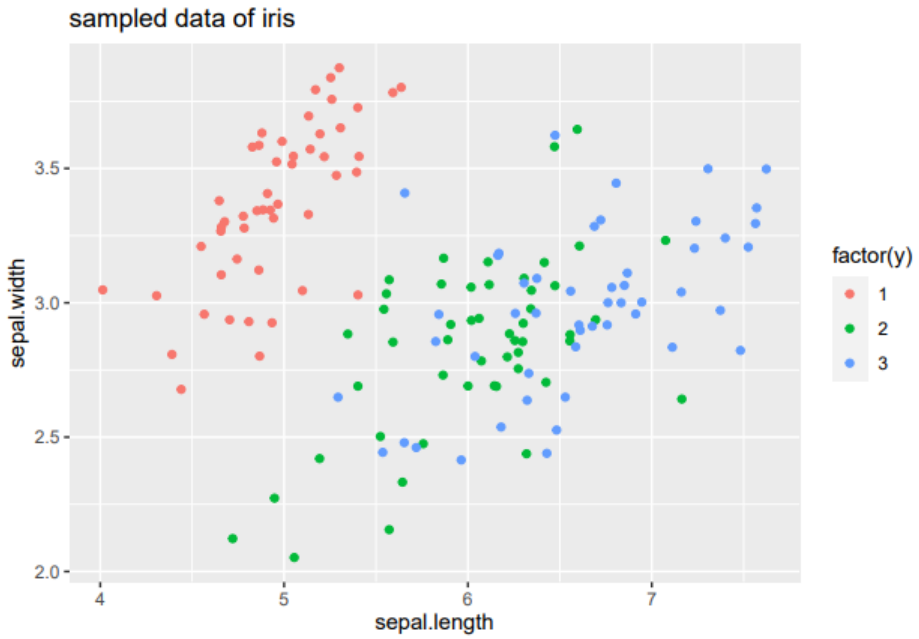


Figure 9

5. Logistic Regression model.

Logistic regression measures the relationship between one or more independent variables (X) and the categorical dependent variable (Y) by estimating probabilities using a logistic(sigmoid) function[1].

- The logistic Regression model is works better than the lda model.Because the miss-classification Rate of this model small compare with LDA model.

```
## # weights:  12 (6 variable)
## initial  value 164.791843
## iter  10 value 62.715967
## iter  20 value 59.808291
## iter  30 value 55.445984
## iter  40 value 55.375704
## iter  50 value 55.346472
## iter  60 value 55.301707
## iter  70 value 55.253532
## iter  80 value 55.243230
## iter  90 value 55.230241
## iter 100 value 55.212479
## final  value 55.212479
## stopped after 100 iterations
```

Figure 10

```
##   [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
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 3 2 3 2 3 2 2 2 2 2 2 3 2 2 2 2 2 2 2
##  [75] 3 3 3 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 3 3 3 2 2 3 3 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
## Levels: 1 2 3
```

Figure 11

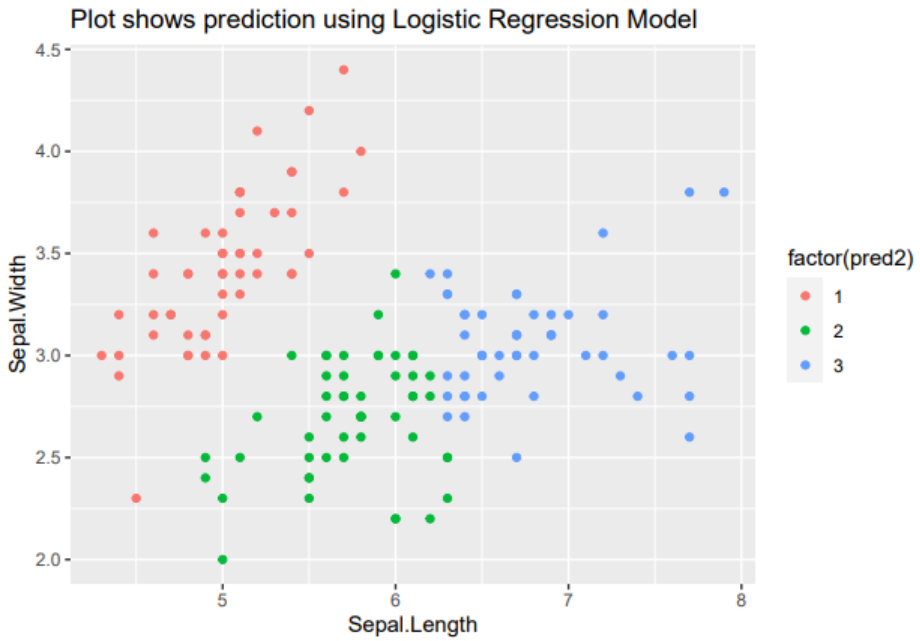


Figure 12

```
##            Reference
## Prediction  1  2  3
##          1 50  0  0
##          2  0 38 13
##          3  0 12 37

## Misclassification Rate using Logistic Regression model

## [1] 0.1666667
```

Figure 13

Assignment 2: Decision trees and Naïve Bayes for bank marketing

1.

Given the data file bank-full.csv, and loaded it into R, I cleaned the data file and then removed the variable "duration". Then I partitioned the data into training/validation/test as 40/30/30. The full code is in the Appendix under "R Code Assignment 2". Below is just the code used i task 1.

```
clean_bank <- clean_bank[,-12]   #remove duration

n <- dim(clean_bank)[1]

set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=clean_bank[id,]

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

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

2.

In this task we should create three different decision trees; one with defualt settings, one with smallest allowed node size equal to 7000, and the third tree with a minimum deviance to 0.0005.
With these we should then report the misclassification rates for both the training & validation data, and then do some quick analyze of the result.
So to start off, I used the tree() function i the tree package, and then fitted the following trees. And since the dataset is filled with quite a few missing values, all of whom are categorical values, we can use the built in function *na.tree.replace*().

```
library(tree)
```

```
default_tree <- tree(formula = y ~ .,
data = na.tree.replace(train))

small_node_tree <- tree(formula = y ~ .,
data = na.tree.replace(train),
control = tree.control(nobs = dim(train)[1], minsize = 7000))

min_deviance_tree <- tree(formula = y ~ .,
data = na.tree.replace(train),
control= tree.control(nobs = dim(train)[1], mindev = 0.0005))
```

Then to find the misclassification rate, we must first find the predicted values from each decision tree. I did this using the base R predict function, with specified type="class", since the response variable isn't numerical.

```
d_f_pred <- predict(default_tree, train,
type="class")
s_n_pred <- predict(small_node_tree, train,
type="class")
m_d_pred <- predict(min_deviance_tree, train,
type="class")
```

And then we find predictions using the validation data.

```
d_f_pred_val <- predict(default_tree, valid,
type="class")
s_n_pred_val <- predict(small_node_tree, valid,
type="class")
m_d_pred_val <- predict(min_deviance_tree, valid,
type="class")
```

And since the classification rate simply is the sum of the diagonal of the confusion matrix, divided by the number of observations, we can easily find the misclassification rate using the below function.

```
MC_rate <- function(y, y_hat){
    n <- length(y)
```

```
    c_m <- table(y, y_hat)
    d_c_m <- sum(diag(c_m))
    mc <- 1 - (d_c_m/n)
    return(mc)
}
```

Below is a table over the misclassifcation rates.

	Train			Valid	
Defualt	Small Node	Min Dev.	Defualt	Small Node	Min Dev.
0.1048441	0.1048441	0.09854015	0.1092679	0.1092679	0.111701

First off, the decision trees does not seem to be overfitted, since their are no large deviance between the misclassification rates of the two datasets. Secondly, we can note changing the deviance to 0.0005 is resulting in more accurate classifications. Then, ofcourse, we can note that setting the small node to 7000 doesn't change the classification rate at all compared to the default setting.

3.

Here we are tasked to use the tree model with low deviance setting, and use it with the training and validation data to find the optimal tree depth. The purpose is to find the best number of terminal nodes by studying the relationship between deviance against tree size. This is illustrated in the figures below.

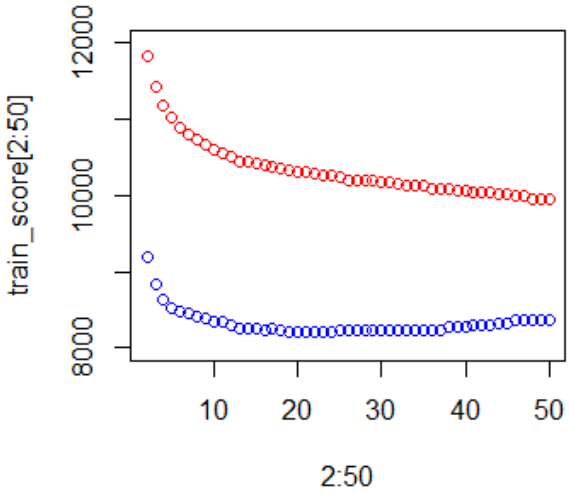


Figure 14: The figure is illustrating how the deviance in the external nodes of the tree model from 2c changes as the number of nodes increases.

The optimal number of nodes is the amount of nodes that yields the lowest deviance when the tree model is evaluated with the validation data. In our case that number is 21, and below is the code to find it.

```
optimal_nodes <- which.min(valid_score[-1])
```

So with this information, we can fit one last tree model and setting the best number of nodes to 21. And when this is done, the model uses the following variables as the most important in order to make decision in the tree.

```
summary(final_tree)
#return variables
"poutcome" "month" "contact" "pdays" "age" "day"
"balance" "housing"
```

Once this is done, we were to create a confusion matrix, and then find the misclassification rate for the test data. Below is the confusion matrix.

	pruned_fit	
	no	yes
no	11540	438
yes	1537	48

This matrix then results in a misclassification rate of 0.1456168. This means that $\sim$ 85% of all predictions on the test data were correct, and hence the model can be said to have a good predictive power.

4.

Here in task 4 we are supposed to use the test data and perform a decision tree classification with the given loss matrix. For this task I tried the other suggested package from the lecture, i.e. the *rpart* package, since that package provided a simpler way to incorporate the loss matrix, compared to the *tree* package.

```
library(rpart)
loss_m <- matrix(data=c(0,1,5,0), nrow=2)
test_tree <- rpart(formula = y ~., data = test,
parms = list(loss = loss_m))
pred_pruned <- predict(test_tree, type="class")
table(test$y, pred_pruned)
```

The above code returned the confusion matrix below.

	pred_pruned	
	no	yes
no	11979	0
yes	1585	0

That confusion matrix then gives us the mislassification rate of 0.1168534, which can be interpret as $\sim$ 11.5% of all classifications the tree does are misclassified. That figure is however $\sim$ 3 percentage points less than for the decision tree in task 3. Now, why is that? So the loss matrix is to weight misclassifications differently. And what that means is that when incorporating the loss matrix, we want to reduce a certain kind of error (for some reason) by penalizing it more. Now, this error - classification error - is either a false positive (FP) or a false negative (FN) and can be visualised below.

	Predicted	
	Class 1	Class 2
Class 1	TP	FN
Class 2	FP	TN

So if we go back to the confusion matrix in task 3, we see that we had 438 cases of the FN error, and here with the following loss matrix

$$\begin{pmatrix} 0 & 5 \\ 1 & 0 \end{pmatrix}$$

we note that it would cost 5 times as much as before (and 5 times more than an FP error!) to make a FN error. This then ofcourse results in many more FP errors, which we can see has happened from the confusion matrix in this task.

5.
Here we are supposed to use the optimal tree from task 3, and the Naive Bayes model to classify the test data.

I wasn't able to finish this final task, however I think the classification I did was correct..? It was the ROC curve I couldn't wrap my head around. We weren't supposed to use a package right? One final question is with regards to handling the variable 'unknown'. That was supposed to be a legit value of the predictors "education" and "job" and so forth, but not a valid value of "poutcome" and "contact"? I set these to NA, and used the tree() na action function to do its work, but then realised when using other functions such as rpart() or naiveBayes(), that those functions would handle NA different. So my question is if there were some specific way we should handle these values that I simply have missed?

Assignment 3. Principal components for crime level analysis

3.1

First, we load and scale the data. Then we make use of the built-in cov() and eigen() function to compute PCA.

```
## To obtain 95% of variance in the data, we need 34 features.
## The proportion of variance explained
##
## PC1: 25.04%
## PC2: 16.94%
```

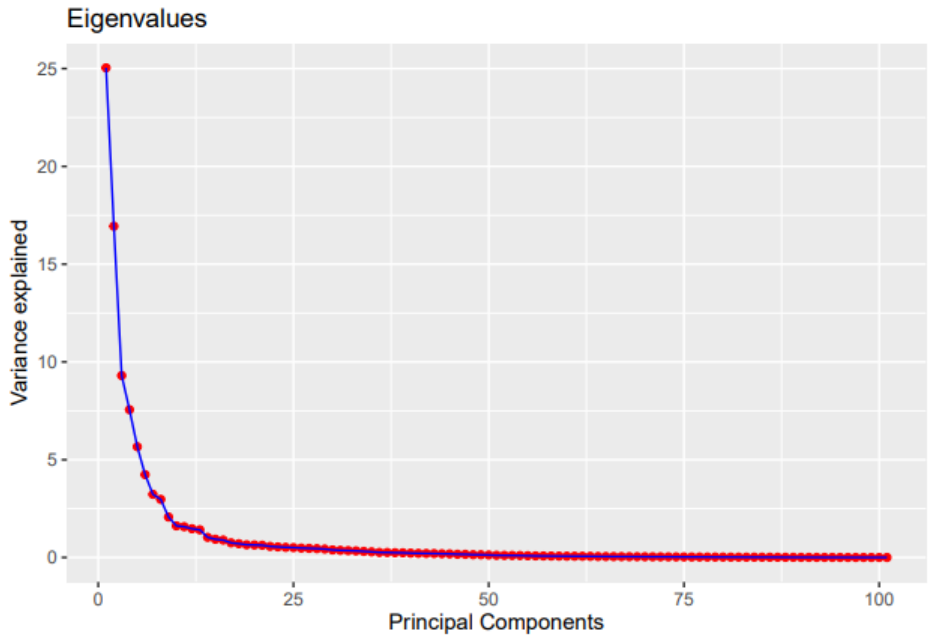
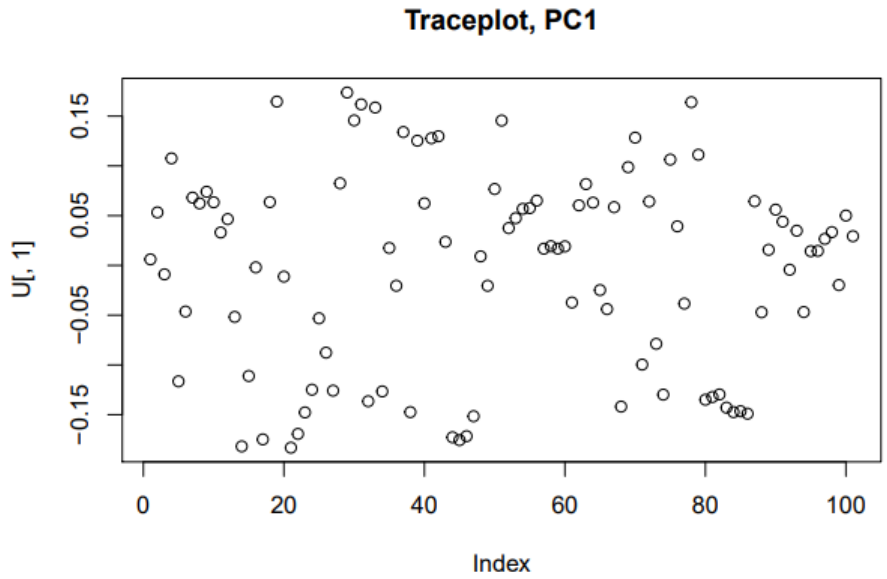


Figure 15

3.2




```
## The top 5 features contributing mostly to the first principle component:
##
## medFamInc
## medIncome
## PctKids2Par
## pctWInvInc
## PctPopUnderPov
```

Figure 16

Most of these features are related to the financial situation of the subject. Therefore, one could guess that financial difficulties could lead to higher crime rates.
Though, we observe multiple more features contributing equally high (0.15) on absolute value to PC1.

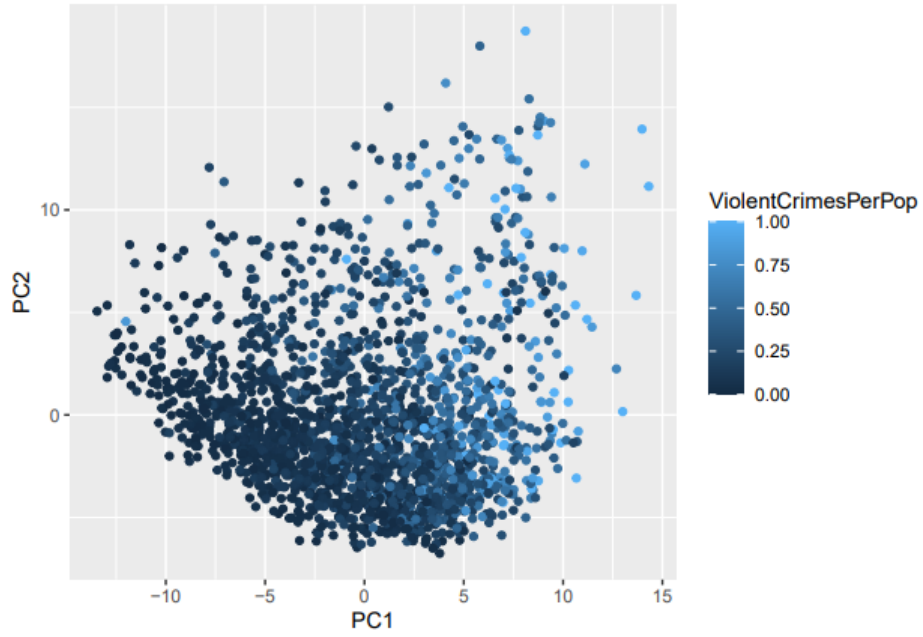


Figure 17

We can observe that a high PC1 value correlates with higher values of ViolentCrimesPerPop. On the other hand the PC2 value doesn"t present a clear effect pattern on ViolentCrimesPerPop.

3.3

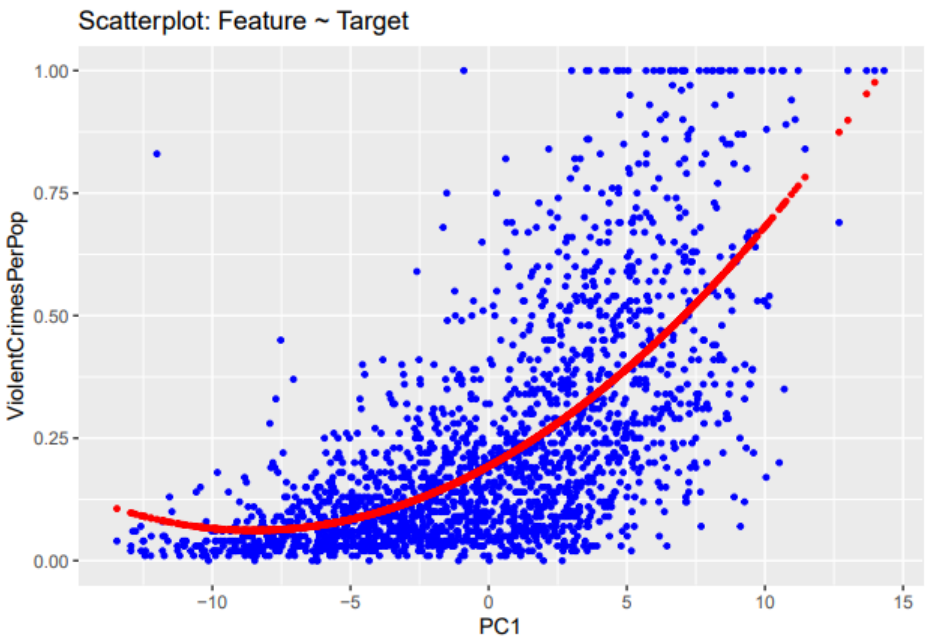


Figure 18

We can observe a pattern of target values that is explained by PC1. Also, the model line mostly fits the point cloud. Though the model line is slightly off, especially when we observe target values greater than 0.5. A lot of noise around the model line can be seen as well.

3.4

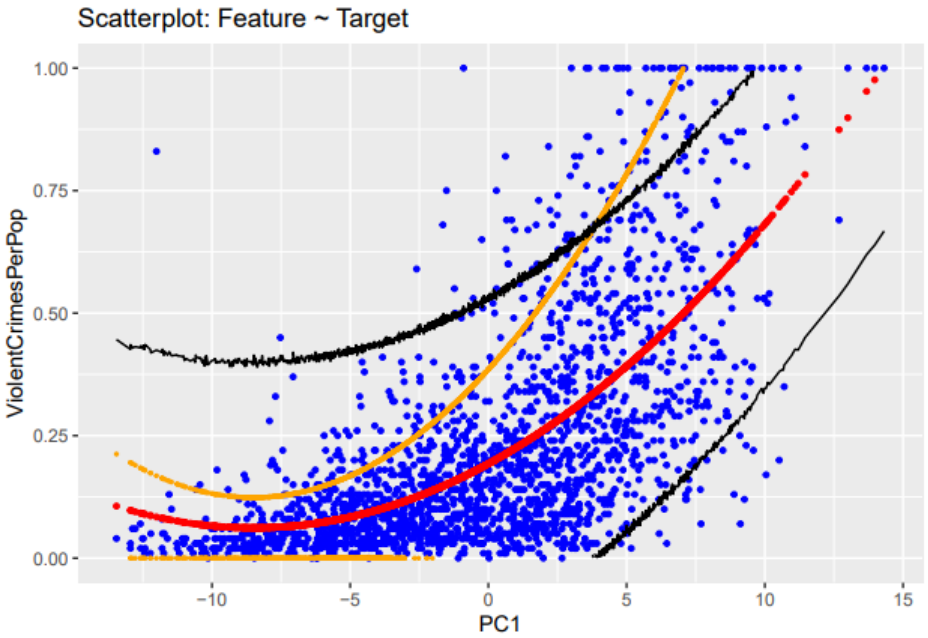


Figure 19

Confidence interval: Unfortunately, the confidence bands from our plot don't seem to make a lot of sense. In general we want our confidence interval to show the likely range of values associated with our MLE.
Prediction interval: The prediction interval seems to be fitting well. It predicts in what range a future individual observation will fall.

Appendix: R Code

Assignment 1

```
library(ggplot2)
library(MASS)
library(mvtnorm)
data=iris
x0=c(data$Sepal.Length)
y0=c(data$Sepal.Width)
z=c(data$Species)
x = data.frame(x1=x0,x2=y0)
y=as.factor(z)

ggplot(data = data, aes(x = Sepal.Length,y = Sepal.Width))
+ geom_point(aes(color = factor(Species)))
# Grouping the Targets
group1_index = which( y == 1 )
group2_index = which( y == 2 )
group3_index = which( y == 3 )

#priors:
prior_group1 = length(group1_index) / length(y)
prior_group2 = length(group2_index) / length(y)
prior_group3 = length(group3_index) / length(y)
print("Prior probabilities of groups:")
print(c(prior_group1, prior_group2,prior_group3))

#means:
mean_group1 = as.matrix(colMeans(x[group1_index, ]))
mean_group2 = as.matrix(colMeans(x[group2_index, ]))
mean_group3 = as.matrix(colMeans(x[group3_index, ]))

print("Group means:")
print(cbind(mean_group1, mean_group2,mean_group3))

# Covariance Matrix
cv1<-cov(x[group1_index, ])
cv2<-cov(x[group2_index, ])
cv3<-cov(x[group3_index, ])
```

```
print("covariance matrix of group 1")
print(cv1)

print("covariance matrix of group 2")
print(cv2)

print("covariance matrix of group 3")
print(cv3)
# Pooled Co-variance Matrix
pooled_cv<-as.matrix((length(group1_index)*cv1)+(length(group2_index)*
+ (length(group3_index)*cv3))/length(y)
print("Pooled covariance matrix of groups")
print(pooled_cv)

# Discriminant function
disc_fn<-function(v,p_cv,m_g,p_g)
{
v<-as.matrix(v)
p_cv<-solve(p_cv)
d1<-((v%*%p_cv)%*%(m_g))
d2<-(0.5*t(m_g))%*%(p_cv)%*%(m_g)
t_d<-d1-(as.numeric(d2))+log(p_g)
return(t_d)
}

# Decision Boundary Function
decision_boundary<-function(x,y,z)
{
y<-solve(y)
m<-(-0.5*t(x))%*%(y)%*%(x)
w_oi<-m+log(z)
w_i<-y%*%x
#cat("w_0i",w_oi,"\n","w_i",w_i,"\n")
}

dc1<- decision_boundary(mean_group1,cv1,prior_group1)
#decision_bndy value of group 1
dc2<- decision_boundary(mean_group2,cv2,prior_group2)
#decision_bndy value of group 2
dc3<- decision_boundary(mean_group3,cv3,prior_group3)
#decision_bndy value of group 3
```

```
dc_b<-data.frame(dc1,dc2,dc3)
colnames(dc_b)<-c("Setosa","Versicolor","Virginica")
cat("Coefficients")
dc_b
#Bind the Discriminant values in the data.frame

d_val1<- disc_fn(x,pooled_cv,mean_group1,prior_group1)
d_val2<- disc_fn(x,pooled_cv,mean_group2,prior_group2)
d_val3<- disc_fn(x,pooled_cv,mean_group3,prior_group3)

disc_val<-cbind(d_val1,d_val2,d_val3)
dis<-as.data.frame(disc_val)

colnames(dis)<-c("setosa","versicolor","virginica")

#Found the target value of this data by max value occurs on the row

m<-colnames(dis)[max.col(dis, ties.method = "first")]

dis$y<-m # add those values as y in the same dataframe

ggplot(data = data, aes(x = Sepal.Length,y = Sepal.Width))
+ geom_point(aes(color = factor(dis$y)))
+ ggtitle("Plot shows prediction using Discriminant Function")

#confusion matrix
cat("Confusion Matrix of prediction using Discriminant function")
table(as.factor(dis$y), y)

#miss-classification rate
missclassrate=function(y,y_i)
{
n=length(y)
v<-1-(sum(diag(table(y,y_i)))/n)
return(v)
}
ms=missclassrate(as.factor(dis$y),y)

cat("Misclassification Rate using Discrimiant function","\n")
ms
```

```
#LDA Model
fit_lda <- lda(y~., data = x)
# ca("coefficients")
# coef(fit_lda)
pred_lda <- predict(fit_lda, x)
vn<-data.frame(original = y, pred = pred_lda$class)
cat("Confusion Matrix using LDA model")
table(vn$pred,vn$original)
ldms=missclassrate(as.factor(vn$pred),y)
cat("Misclassification Rate using LDA model","\n")
ldms


ggplot(data = data, aes(x = Sepal.Length,y = Sepal.Width))
+ geom_point(aes(color = factor(vn$pred)))
+ ggtitle("Plot shows prediction using LDA Model")
# bind the all groups in a single dataframe
bvn1 <- rmvnorm(50, mean = mean_group1, sigma = cv1 )
bvn2 <- rmvnorm(50, mean = mean_group2, sigma = cv2 )
bvn3 <- rmvnorm(50, mean = mean_group3, sigma = cv3 )

bvn<-rbind(bvn1,bvn2,bvn3)
bvn<-as.data.frame(bvn)
bvn$y<-y

sample_data<-bvn[sample(nrow(bvn), 150), ]

colnames(bvn)<-c("sepal.length","sepal.width","species")

ggplot(data =bvn, aes(x = sepal.length,y = sepal.width))
+ geom_point(aes(color = factor(y))) + ggtitle("sampled data of iris")


### 5. Logistic Regression Model
library(nnet)
irisModel<-multinom(y~x1+x2 ,data = x)
m<-summary(irisModel)

pred2<-predict(irisModel,x)
pred2
```

```
ggplot(data = data, aes(x = Sepal.Length,y = Sepal.Width))
+ geom_point(aes(color = factor(pred2)))
+ ggtitle("Plot shows prediction using Logistic Regression Model")

#confusiom Matrix
table("Prediction"=as.factor(pred2), "Reference"=y)

lms=missclassrate(as.factor(pred2),y)
cat("Misclassification Rate using Logistic Regression model","\n")
lms
```

Assignment 2

```
nm <- c("age", "job","marital", "education", "default", "balance",
"housing", "loan", "contact", "day","month","duration",
          "campaign","pdays","previous","poutcome","y")

clean_bank <- tidyr::separate(bank_full,1,nm,sep=";")


clean_bank <- clean_bank[,-12]  #remove duration
#clean_bank[clean_bank == "unknown"] <- NA
na_9 <- which(clean_bank[,9] == "unknown")
na_15 <- which(clean_bank[,15] == "unknown")

clean_bank$contact[na_9] <- NA
clean_bank$poutcome[na_15] <- NA

clean_bank[,1] <- as.numeric(clean_bank[,1])
clean_bank[,2] <- as.factor(clean_bank[,2])
clean_bank[,3] <- as.factor(clean_bank[,3])
clean_bank[,4] <- as.factor(clean_bank[,4])
clean_bank[,5] <- as.factor(clean_bank[,5])
clean_bank[,6] <- as.numeric(clean_bank[,6])
clean_bank[,7] <- as.factor(clean_bank[,7])
clean_bank[,8] <- as.factor(clean_bank[,8])
clean_bank[,9] <- as.factor(clean_bank[,9])
clean_bank[,10] <- as.numeric(clean_bank[,10])
clean_bank[,11] <- as.factor(clean_bank[,11])
```

```
clean_bank[,12] <- as.numeric(clean_bank[,12])
clean_bank[,13] <- as.numeric(clean_bank[,13])
clean_bank[,14] <- as.numeric(clean_bank[,14])
clean_bank[,15] <- as.factor(clean_bank[,15])
#clean_bank[,16] <- ifelse(clean_bank$y == "yes", 1, 0)
clean_bank[,16] <- as.factor(clean_bank$y)


## 1

n <- dim(clean_bank)[1]

set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=clean_bank[id,]

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

id3=setdiff(id1,id2)
test=clean_bank[id3,]

## 2

library(tree)


#creating the three trees
default_tree <- tree(formula = y ~ ., data = na.tree.replace(train))

small_node_tree <- tree(formula = y ~ .,
                        data = na.tree.replace(train),
                        control = tree.control(nobs = dim(train)[1],
                        minsize = 7000))

min_deviance_tree <- tree(formula = y ~ .,
                       data = na.tree.replace(train),
                       control= tree.control(nobs = dim(train)[1],
                       mindev = 0.0005))
```

```
#testing the fitted tree model on the training and validation data

d_f_pred <- predict(default_tree, train, type="class")
s_n_pred <- predict(small_node_tree, train, type="class")
m_d_pred <- predict(min_deviance_tree, train, type="class")

#validation
d_f_pred_val <- predict(default_tree, valid, type="class")
s_n_pred_val <- predict(small_node_tree, valid, type="class")
m_d_pred_val <- predict(min_deviance_tree, valid, type="class")

#find misclassification rates


MC_rate <- function(y, y_hat){
    n <- length(y)
    c_m <- table(y,y_hat)
    d_c_m <- sum(diag(c_m))
    mc <- 1 - (d_c_m/n)
    return(mc)
}
MC_rate(y=train$y, d_f_pred)
MC_rate(y=train$y, s_n_pred)
MC_rate(y=train$y, m_d_pred)

MC_rate(y=valid$y, d_f_pred_val)
MC_rate(y=valid$y, s_n_pred_val)
MC_rate(y=valid$y, m_d_pred_val)


## 3

#creating the same type of in_deviance
tree with the validation data as with train data in 2c


min_deviance_tree <- tree(formula = y ~ .,
                  data = na.tree.replace(train),
                  control= tree.control(nobs = dim(train)[1],
```

```
                mindev = 0.0005))    #tree for train

train_score <- rep(0,50)
valid_score <- rep(0,50)

for(i in 2:50){
  pruned_train <- prune.tree(min_deviance_tree, best=i)
  pred_train <- predict(pruned_train, newdata=na.tree.replace(valid),
  type="tree")
  train_score[i] <- deviance(pruned_train)
  valid_score[i] <- deviance(pred_train)
}


plot(2:50, train_score[2:50], type="b", col="red", ylim=c(8000,12000))
points(2:50, valid_score[2:50], type="b", col="blue")

optimal_nodes <- which.min(valid_score[-1])
### 21 external nodes yields the best

final_tree <- prune.tree(min_deviance_tree, best = optimal_nodes)
pruned_fit <- predict(final_tree, newdata=valid, type="class")

summary(final_tree)          #finding important variables

set.seed(12345)
rm <- sample(1:nrow(test), 1)

table(test$y[-rm], pruned_fit)     #confusion matrix for pruned tree
MC_rate(y = test$y[-rm], pruned_fit)  #Misclasification rate


### 4
library(rpart)
loss_m <- matrix(data=c(0,1,5,0), nrow=2)

test_tree <- rpart(formula = y ~., data = test,
parms = list(loss = loss_m))

pred_pruned <- predict(test_tree, type="class")
```

```
table(test$y,pred_pruned)
MC_rate(test$y, pred_pruned)

## 5
library(e1071)

last_tree <- tree(formula = y ~ ., data = test,
                  control= tree.control(nobs = dim(train)[1],
                  mindev = 0.0005))
naive_class <- naiveBayes(y~., data=test)

## prune tree to 21 nodes according to task 3
pruned_last_tree <- prune.tree(last_tree, best = 21)


#probabiliteis for the models
naive_pred <- predict(naive_class, newdata=test, type="raw")
last_pred <- predict(pruned_last_tree, newdata=test)

class_df <- data.frame(naive_pred[,2],last_pred[,2])
names(class_df) <- c("NaivePred", "TreePred")

## Classifying according to threshold probabilites of 0.05 ... 0.95

tree_class<-c()
naive <- c()
p_i <- c()
j=0
while(j < nrow(last_pred)){
 for(i in seq(0.05,0.95,0.05)){

  tree_class[j] <- ifelse(class_df$TreePred[j] > i, "yes", "no")
  naive[j] <- ifelse(class_df$NaivePred[j] > i, "yes", "no")

  p_i[j] <- i
  j <- j + 1

 }
}
classified <- data.frame(tree_class, naive)
```

```
set.seed(12345)
rm <- sample(1:length(naive), 1)


#function for FPR & TPR  (True positive // sensitivity)
#FPR IS NOT 1- sensitivity!!!


TP_FP <- function(){

   All_yes <- length(which(test$y=="yes"))
   All_no <- length(which(test$y == "no"))

   correct_pred_yes <- length(which(test$y == "yes" &
   classified$tree_class[-rm] == "yes"))

   false_pred_yes <- length(which(test$y == "no" &
   classified$tree_class[-rm] == "yes"))

  TPR <-  correct_pred_yes / All_yes
  FPR <- false_pred_yes/All_no

   res <- data.frame(TPR,FPR)
   names(res) <- c("TPR", "FPR")
   return(res)
}
TP_FP()
```

Assignment 3

```
data = read.csv("communities.csv")
data[, -c(101)] = scale(data[, -c(101)]) # scale except last column

covariance_matrix = cov(data) # covariance matrix
my_eigen = eigen(covariance_matrix) # using eigen()

count_features_obtained_var = function(threshold, eigenvalues) {
variance_obtained = 0
for (i in 1:length(eigenvalues)){
```

```
variance_obtained = variance_obtained + eigenvalues[i]
if (variance_obtained >= threshold) {
return(i)
}
}
}

variance_to_obtain = 95
num_features = count_features_obtained_var(variance_to_obtain,
my_eigen$values)

cat("To obtain ", variance_to_obtain, "% of variance in the data,
we need ", num_features, " features.\n", sep="")

cat("The proportion of variance explained\n\nPC1: ",
round(my_eigen$values[1], 2), "%\nPC2: ",
round(my_eigen$values[2], 2), "%", sep="")

eigenvalues_df = data.frame(values = my_eigen$values)
ggplot(eigenvalues_df, aes(x=as.numeric(rownames(eigenvalues_df)),
y=values)) + geom_point(color="red")
+ geom_line(color="blue") + xlab("Principal Components")
+ ylab("Variance explained") + ggtitle("Eigenvalues")

res = princomp(data) # PCA using princomp()
lambda = res$sdev^2 # eigenvalues
# sprintf("%2.3f",lambda/sum(lambda)*100) # proportion of variation
# screeplot(res)

res = prcomp(data) # PCA using prcomp() because plotting traceplot
only works with prcomp

U = res$rotation
plot(U[,1], main="Traceplot, PC1")

top_five_features = sort((abs(U[1:101,1])), decreasing = TRUE)[1:5]
cat("The top 5 features contributing mostly to the first principle
component:\n", names(top_five_features), sep = "\n")
PC1_vals = res$x[,1]
PC2_vals = res$x[,2]
```

```
df = data.frame(PC1_vals, PC2_vals, data[,101])
colnames(df) = c("PC1", "PC2", "ViolentCrimesPerPop")

ggplot(df, aes(x=PC1, y=PC2, color=ViolentCrimesPerPop))
   + geom_point()

polynomial_model = lm(ViolentCrimesPerPop~poly(PC1_vals, degree = 2),
data=data)

df = data.frame(cbind(PC1_vals, data[,101]))
colnames(df) = c("PC1", "ViolentCrimesPerPop")

model_plot = ggplot(df, aes(PC1, ViolentCrimesPerPop)) +
geom_point(color="blue", size=1) +
geom_point(aes(x=PC1, y=polynomial_model$fitted.values), color="red",
size=1) + ylim(0,1) + ggtitle("Scatterplot: Feature ~ Target")
model_plot

library(boot)

df_order = df[order(df$PC1),]

rng = function(df, polynomial_model) {
df1 = data.frame(ViolentCrimesPerPop=df$ViolentCrimesPerPop,
PC1=df$PC1)
n=length(df$ViolentCrimesPerPop)
df$ViolentCrimesPerPop = rnorm(n, predict(polynomial_model,
newdata=df1),
sd(polynomial_model$residuals))

return(df1)
}

f1 = function(df1){
res = lm(ViolentCrimesPerPop~poly(PC1, degree=2), data=df1)
#fit linear model
target_predicted = predict(res, newdata=df_order)

return(target_predicted)
}
```

```
res = boot(df_order, statistic=f1, R=1000, mle=polynomial_model,
ran.gen=rng, sim="parametric")

# Confidence Bands
confidence_envelope = envelope(res)
fit = lm(ViolentCrimesPerPop~poly(PC1, degree=2), data=df_order)
target_predict = predict(fit)

model_plot_ = model_plot +
geom_point(aes(x=df_order$PC1, y=target_predict -
confidence_envelope$point[1,]), color="orange", size=0.5)
+
geom_point(aes(x=df_order$PC1, y=target_predict +
confidence_envelope$point[2,]), color="orange", size=0.5)

# Prediction Bands
mle = lm(ViolentCrimesPerPop~poly(PC1, degree=2), data=df_order)

f1 = function(df1) {
res = lm(ViolentCrimesPerPop~poly(PC1, degree=2), data=df1)
target_predict = predict(res,newdata=df_order)
n = length(df$ViolentCrimesPerPop)
result = rnorm(n, target_predict, sd(mle$residuals))

return(result)
}

res = boot(df_order, statistic=f1, R=10000, mle=mle, ran.gen=rng,
sim="parametric")

prediction_envelope = envelope(res)

model_plot_ +
geom_line(aes(x=df_order$PC1, y=prediction_envelope$point[2,])) +
geom_line(aes(x=df_order$PC1, y=prediction_envelope$point[1,]))
```