Lab Topic 2 Block 1 Machine Learning

Hugo Knape & Zahra Jalil Pour & Niklas Larsson

11/12/2020

State of contribution

Assignment1: Zahra Jalilpour

Assignment2: Niklas Larsson

Assignment3: Hugo Knap

# Assignment 1.

title: “Untitled” author: “z” date: ‘2020-11-20’ output: html_document: default pdf_document: default latex_engine: xelatex —

Assignment 1: LDA and logistic regression

In LDA , features in each class has multivariate normal distribution and common variance \$ \$. The covariance matrix is the same in all class $Cov(X) = \Sigma$. Random variable X is a vector $X = (X_1, X_2, \dots, X_p)$ Mean of each class is :

$$\hat{\mu}_k = \frac{1}{\#\{i\ ;\ y_i = k\}} \sum_{i\ ;\ y_i = k} x_i$$

Perior probabity or \$ pi_k\$ is equal to:

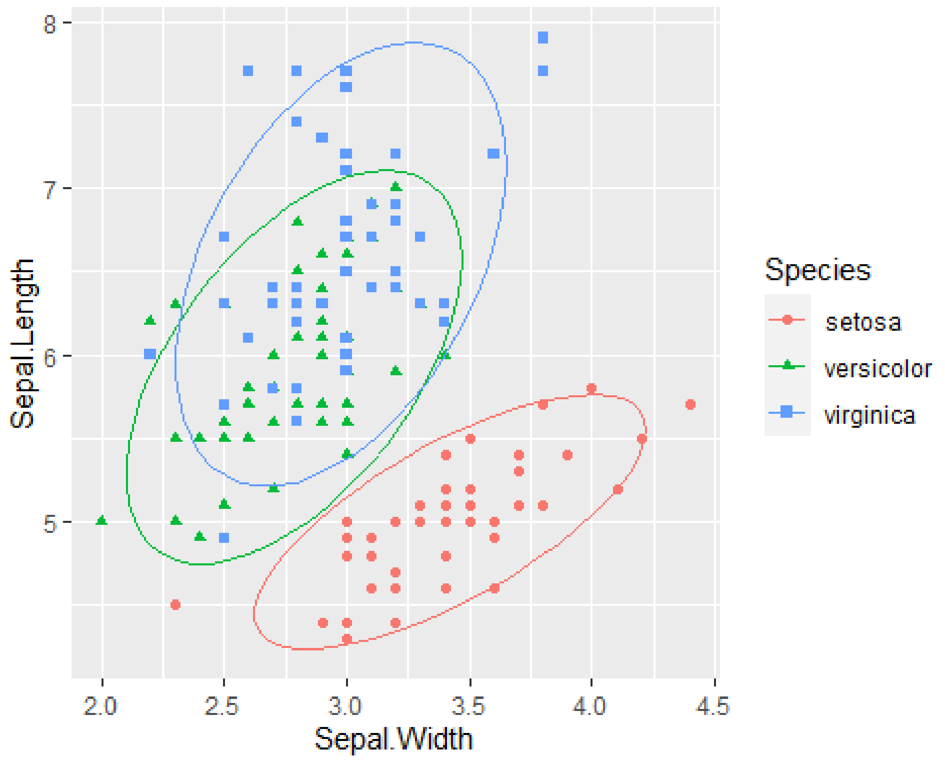
$$\hat{\pi}_k = \frac{\{i\ ;\ y_i = k\}}{n}$$

X follows multivariate Gaussian distribution.

Task 1

Make a scatterplot

```
ggplot(iris, aes(x=Sepal.Width , y= Sepal.Length, shape= Species,
color=Species)) + geom_point() +stat_ellipse()
```



Here we can see that we have more than two classes. Then LDA is the preferred linear classification technique, and LDA is a simple model for preparation and application. Here we can not use logistic regression, because we have more than two classes. Although we can use multi logistic regression but is rarely used for this purpose. LDA is a stable model when the classes are separated well. Also LDA generally is used for small data sets.

Task 2

2-a

Compute mean, covariance matrix and prior probabilities for each class:

```
## [1] "Covariance matrix in Setosa class"

##              Sepal.Length Sepal.Width
## Sepal.Length   0.12424898  0.09921633
## Sepal.Width    0.09921633  0.14368980

## [1] "Covariance matrix in Versicolor class"

##              Sepal.Length Sepal.Width
## Sepal.Length   0.26643265  0.08518367
## Sepal.Width    0.08518367  0.09846939

## [1] "Covariance matrix in Virginica class"
##              Sepal.Length Sepal.Width
## Sepal.Length   0.40434286  0.09376327
## Sepal.Width    0.09376327  0.10400408

## [1] "mean for each class"

##              setosa versicolor virginica
## Sepal.Length  5.006      5.936     6.588
## Sepal.Width   3.428      2.770     2.974

## [1] "prior probability for each class: "

## [1] 0.3333333

## [1] 0.3333333

## [1] 0.3333333
```

2-b

pooled covariance matrix:

```
## [1] "degree of freedom: "

##
##     setosa versicolor  virginica
##         49         49         49

## [1] "pooled covariance matrix: "

##              Sepal.Length Sepal.Width
## Sepal.Length   0.26500816  0.09272109
## Sepal.Width    0.09272109  0.11538776
```

2-c

probabilistic model: The probability of $p(X = x|Y = k)$ is given by:

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}}\, exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right)$$

By considering prior probability is $P(Y = k) = \pi_k$, and taking logarithm , we will find Linear discriminant function or Linear score function:

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log(\pi_k)$$

$$\hat{G}(x) = \arg\max_k \delta_k(x)$$

The decision boundary is the set of points in which two classes are equally probable:

$$\delta_k(x) = \delta_l(x)$$

We consider $ _k, _k, $ by MLE as below:

$$\hat{\Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

$$\hat{\pi}_k = \frac{\{i \ ; \ y_i = k\}}{n}$$

2_d

Compute discriminant functions for each class:

```
## [1] "Discriminant matrix for three different species"

##             setosa versicolor virginica
##   [1,] 65.69175   58.09672  54.93699
##   [2,] 53.22218   50.02352  46.49679
##   [3,] 54.90090   47.79962  43.70231
##   [4,] 51.69789   45.01644  40.69200
##   [5,] 66.53111   56.98477  53.53975
##   [6,] 77.32196   67.28187  64.77443
##   [7,] 57.76144   47.52328  43.11160
##   [8,] 62.48874   55.31354  51.92669
##   [9,] 45.29187   39.45008  34.67140
##  [10,] 55.24337   50.85914  47.30332
##  [11,] 73.27960   65.61065  63.16137
##  [12,] 60.12509   51.41841  47.51914
##  [13,] 52.04036   48.07596  44.29302
##  [14,] 46.13122   38.33812  33.27416
##  [15,] 84.07045   75.90775  74.39605
##  [16,] 90.97336   77.30264  75.41841
##  [17,] 77.32196   67.28187  64.77443
##  [18,] 65.69175   58.09672  54.93699
##  [19,] 78.84626   72.28896  70.57921
##  [20,] 71.75530   60.60356  57.35658
##  [21,] 67.21605   63.10381  60.74177
##  [22,] 69.73412   59.76795  56.55005
##  [23,] 61.80380   49.19450  44.72466
##  [24,] 61.64939   56.42550  53.32393
##  [25,] 60.12509   51.41841  47.51914
##  [26,] 54.40401   51.97109  48.70056
##  [27,] 62.48874   55.31354  51.92669
##  [28,] 66.87358   60.04429  57.14076
##  [29,] 64.85240   59.20868  56.33423
##  [30,] 54.90090   47.79962  43.70231
##  [31,] 54.06154   48.91157  45.09955
##  [32,] 67.21605   63.10381  60.74177
##  [33,] 79.00068   65.05797  61.97995
##  [34,] 84.56734   71.73628  69.39780
##  [35,] 55.24337   50.85914  47.30332
## [36,] 58.44638   53.64232  50.31362
## [37,] 70.41906   65.88699  63.75208
## [38,] 65.34928   55.03720  51.33598
## [39,] 47.31305   40.28569  35.47793
## [40,] 63.67057   57.26111  54.13046
## [41,] 64.50993   56.14916  52.73322
## [42,] 34.34660   36.38397  32.03598
## [43,] 51.35542   41.95692  37.09099
## [44,] 64.50993   56.14916  52.73322
## [45,] 71.75530   60.60356  57.35658
## [46,] 52.04036   48.07596  44.29302
## [47,] 71.75530   60.60356  57.35658
## [48,] 53.71907   45.85205  41.49854
## [49,] 72.09777   63.66308  60.95760
## [50,] 60.46756   54.47793  51.12015
## [51,] 82.08291   92.59365  94.38906
## [52,] 74.99195   80.90825  81.16643
## [53,] 78.87990   89.81047  91.37875
## [54,] 46.16487   55.85963  54.07370
## [55,] 68.08905   79.51336  80.14407
## [56,] 58.63443   63.93283  62.51390
## [57,] 75.83131   79.79630  79.76919
## [58,] 41.09509   45.00985  41.65760
## [59,] 71.29206   82.29654  83.15437
## [60,] 50.70412   53.35939  50.68851
## [61,] 34.19218   43.61496  40.63524
## [62,] 65.04045   69.49919  68.53450
## [63,] 50.05282   64.76185  64.28602
## [64,] 65.38292   72.55871  72.13552
## [65,] 59.47379   62.82088  61.11666
## [66,] 76.51625   85.91534  86.97121
## [67,] 61.49497   63.65649  61.92319
## [68,] 57.79508   65.04479  63.91114
## [69,] 52.41647   68.65699  68.69357
## [70,] 51.38906   59.47843  57.89053
## [71,] 69.08282   71.17042  70.14757
## [72,] 63.36174   71.72310  71.32898
## [73,] 59.66185   73.11139  73.31693
## [74,] 63.36174   71.72310  71.32898
## [75,] 68.92840   78.40141  78.74683
## [76,] 73.31324   83.13216  83.96091
## [77,] 71.63453   85.35606  86.75539
## [78,] 74.49507   85.07972  86.16468
## [79,] 64.20110   70.61114  69.93174
## [80,] 54.59207   62.26161  60.90083
## [81,] 48.18605   56.69525  54.88023
## [82,] 48.18605   56.69525  54.88023
## [83,] 57.79508   65.04479  63.91114
## [84,] 60.15873   68.93992  68.31868
## [85,] 59.13132   59.76136  57.51565
##  [86,]  74.30701   74.78921  73.96440
##  [87,]  76.51625   85.91534  86.97121
##  [88,]  55.61948   71.44017  71.70387
##  [89,]  61.49497   63.65649  61.92319
##  [90,]  50.20723   57.53086  55.68676
##  [91,]  52.22841   58.36647  56.49329
##  [92,]  67.40411   73.39432  72.94205
##  [93,]  55.77389   64.20917  63.10461
##  [94,]  40.25573   46.12180  43.05484
##  [95,]  55.43142   61.14965  59.50359
##  [96,]  62.67680   65.60406  64.12696
##  [97,]  60.65562   64.76844  63.32043
##  [98,]  66.56475   74.50628  74.33929
##  [99,]  45.47992   49.74059  46.87167
## [100,]  58.63443   63.93283  62.51390
## [101,]  75.83131   79.79630  79.76919
## [102,]  57.79508   65.04479  63.91114
## [103,]  79.22237   92.86999  94.97977
## [104,]  67.74658   76.45384  76.54306
## [105,]  72.13141   81.18459  81.75713
## [106,]  85.13151  102.60782 105.99862
## [107,]  43.11627   45.84546  42.46413
## [108,]  79.56484   95.92951  98.58078
## [109,]  64.38915   80.90166  82.13202
## [110,]  92.53130   99.83123 102.02273
## [111,]  76.17378   82.85582  83.37020
## [112,]  64.88604   76.73019  77.13377
## [113,]  75.67689   87.02729  88.36845
## [114,]  52.57089   61.42599  60.09430
## [115,]  59.81626   65.88040  64.71767
## [116,]  74.99195   80.90825  81.16643
## [117,]  72.13141   81.18459  81.75713
## [118,] 102.48280  111.24029 114.65465
## [119,]  78.22860  101.21294 104.97627
## [120,]  50.05282   64.76185  64.28602
## [121,]  80.90109   90.64608  92.18529
## [122,]  57.45261   61.98526  60.31013
## [123,]  82.27097  102.88416 106.58933
## [124,]  63.70421   74.78262  74.93000
## [125,]  80.55861   87.58656  88.58427
## [126,]  84.44657   96.48878  98.79660
## [127,]  64.54357   73.67066  73.53276
## [128,]  67.40411   73.39432  72.94205
## [129,]  66.90722   77.56580  77.94030
## [130,]  80.40420   94.81756  97.18354
## [131,]  78.72549   97.04146  99.97802
## [132,] 104.84645  115.13543 119.06219
## [133,]  66.90722   77.56580  77.94030
## [134,]  65.72539   75.61823  75.73653
## [135,]  59.31938   70.05187  69.71592
## [136,]  86.31333  104.55539 108.20240
## [137,]  77.85249   80.63191  80.57572
## [138,]  72.97077   80.07264  80.35989
## [139,]  66.22228   71.44676  70.73828
## [140,]  78.87990   89.81047  91.37875
## [141,]  76.51625   85.91534  86.97121
## [142,]  78.87990   89.81047  91.37875
## [143,]  57.79508   65.04479  63.91114
## [144,]  79.71926   88.69852  89.98151
## [145,]  80.55861   87.58656  88.58427
## [146,]  74.49507   85.07972  86.16468
## [147,]  59.66185   73.11139  73.31693
## [148,]  72.13141   81.18459  81.75713
## [149,]  76.67066   78.68434  78.37195
## [150,]  65.04045   69.49919  68.53450
```

2-e

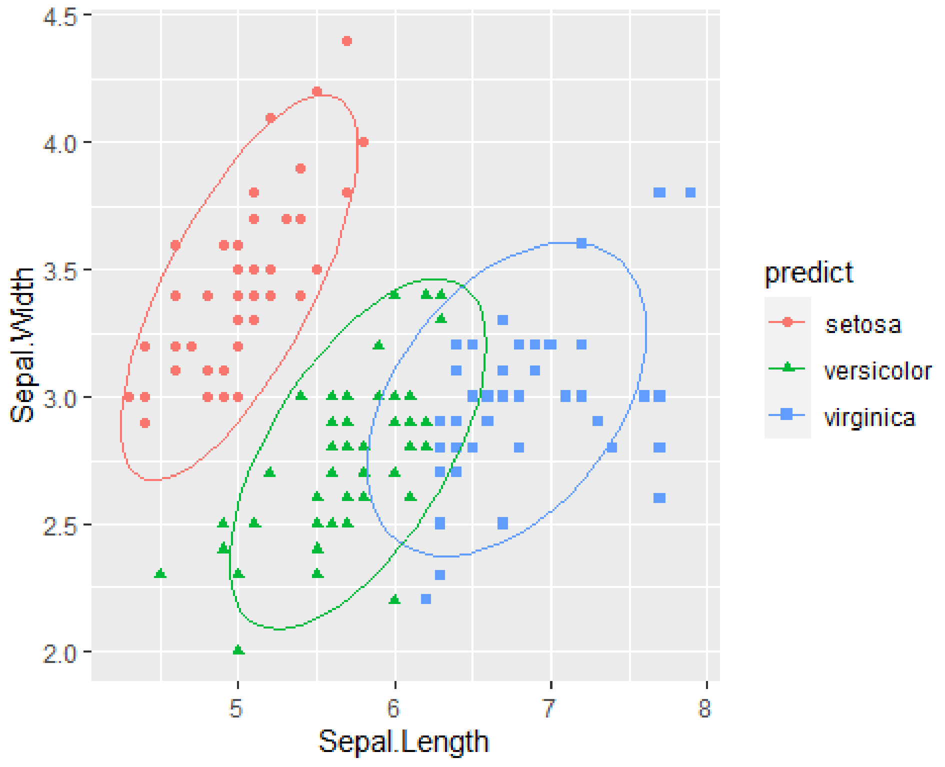
Compute equation of decision boundaries between classes:

```
## [1] decision boundary for Setosa ~ Versicolor
## [2]
## [3]  W1 :

##                   [,1]
## Sepal.Length -7.657399
## Sepal.Width  11.855698

## [1] "W0 :  5.15282159558004"

## [1] decision boundary for Versicolor ~ Verginica
## [2]
## [3]  W1 :

##                    [,1]
## Sepal.Length -2.5620509
## Sepal.Width   0.2908121

## [1] "W0 :  15.2083506778689"

## [1] decision boundary for Verginica ~ Setosa
## [2]
## [3]  W1 :

##                   [,1]
## Sepal.Length  10.21945
## Sepal.Width  -12.14651

## [1] "W0 :   -20.3611722734489"
```

3-a

predict function:

```
library(ggplot2)
D <- cbind(d1,d2,d3)

for(i in row(D)){
  predict <-apply(X = D[,], MARGIN = 1, FUN = which.max)
}
new_df <- iris
new_df <- cbind(new_df, predict)
new_df$predict[new_df$predict == "1"] <- "setosa"
new_df$predict[new_df$predict == "2"] <- "versicolor"
new_df$predict[new_df$predict == "3"] <- "virginica"
ggplot(new_df, aes(x=Sepal.Length , y= Sepal.Width, shape= predict, 
color=predict)) + geom_point()+stat_ellipse()
```



```
Actual<- iris$Species
t_3 <-table(Actual, new_df$predict)
knitr::kable(t_3, caption = "MissClassification matrix for manual 
prediction")
```

MissClassification matrix for manual prediction

	setosa	versicolor	virginica

setosa	49	1	0
versicolor	0	36	14
virginica	0	15	35

```
lda_error_3 = 1- sum(diag(t_3))/sum(t_3)
paste("MSE for manual LDA: " ,lda_error_3)

## [1] "MSE for manual LDA:  0.2"
```

3_b

lda function

```
library(MASS)
new_data3c <- iris
fitted_lda <- lda(Species~Sepal.Length + Sepal.Width, data = new_data3c)
fitted_lda

## Call:
## lda(Species ~ Sepal.Length + Sepal.Width, data = new_data3c)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width
## setosa            5.006       3.428
## versicolor        5.936       2.770
## virginica         6.588       2.974
##
## Coefficients of linear discriminants:
##                    LD1        LD2
## Sepal.Length -2.141178 -0.8152721
## Sepal.Width   2.768109 -2.0960764
##
## Proportion of trace:
##    LD1    LD2
## 0.9628 0.0372

#### Confusion Matrix###
Classification <- predict(fitted_lda, data = new_data3c)$class
Actual<- iris$Species
t <-table(Actual, Classification)
knitr::kable(t,caption = "MissClassification matrix for LDA")
```

MissClassification matrix for LDA

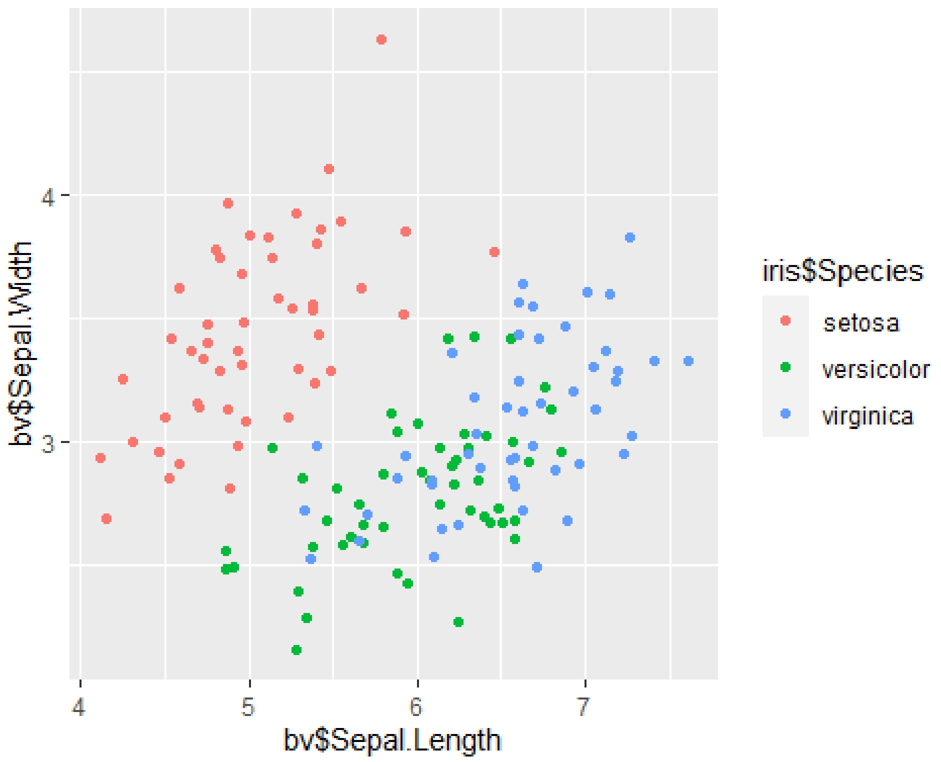
	setosa	versicolor	virginica
setosa	49	1	0

versicolor	0	36	14
virginica	0	15	35

```
lda_error = 1- sum(diag(t))/sum(t)
paste("MSE of LDA model: ", lda_error)

## [1] "MSE of LDA model:  0.2"
```

Miss classification matrix and Accuracy in both model are same.

```
library("mvtnorm")
bvn1 <- as.data.frame(mvrnorm(50, mu =m1, S ))
bvn2 <- as.data.frame(mvrnorm(50, mu=m2, S))
bvn3 <- as.data.frame(mvrnorm(50, mu=m3, S))
bv <- rbind(bvn1,bvn2,bvn3)

ggplot() +
  geom_point(data=bv, aes(bv$Sepal.Length, bv$Sepal.Width,
color=iris$Species))
```



By using new generate data, we can see the Setosa species is classified better than two classes, like other models. ### 5 Logistic Regression

```
library(nnet)
multi_model <- multinom(Species~Sepal.Length + Sepal.Width, data =iris)

## # 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
```

```
multi_model
```

```
## Call:
## multinom(formula = Species ~ Sepal.Length + Sepal.Width, data = iris)
##
## Coefficients:
##              (Intercept) Sepal.Length Sepal.Width
## versicolor    -92.09924     40.40326   -40.58755
## virginica    -105.10096     42.30094   -40.18799
##
## Residual Deviance: 110.425
## AIC: 122.425
```

```
new_data <- iris
new_data$predicted<- predict(multi_model, new_data, type="class")
summary(new_data)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species        predicted 
##  setosa    :50   setosa    :50  
##  versicolor:50   versicolor:51  
##  virginica :50   virginica :49  
##
##
##
```

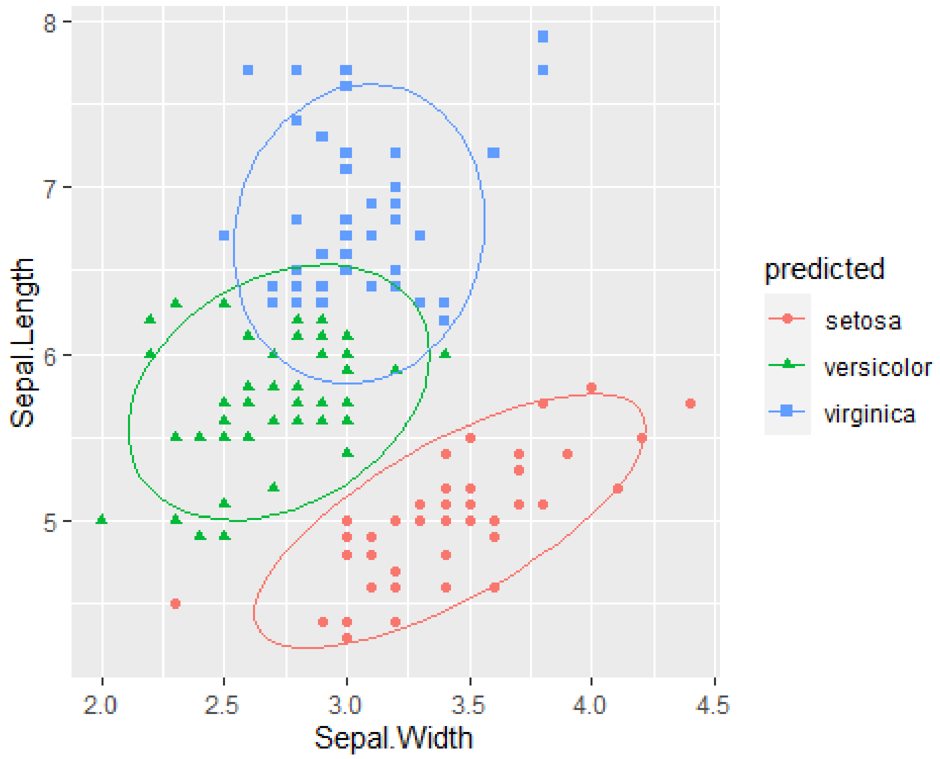
```
t_multi <- table(new_data$Species, new_data$predicted)
t_multi
```

```
##
##               setosa versicolor virginica
##   setosa          50          0         0
##   versicolor      0         38         12
##   virginica       0         13         37
```

```
ME_error <- 1-sum(diag(t_multi))/sum(t_multi)
paste( "ME_error= ", ME_error)
```

```
## [1] "ME_error=  0.166666666666667"
```

```
ggplot(new_data, aes(x=Sepal.Width , y= Sepal.Length, shape= predicted,
color=predicted)) + geom_point() +stat_ellipse()
```



We can see here that ME in logistic regression model is lower than two other models.

Assignment 2. Decision trees and Naïve Bayes for bank marketing

Task 1

Pre-process data by removing “Duration” feature, converting all character columns to categorical classes (factors) and split into train-, validation- and testset.

```
data = read.csv2("bank-full.csv")
data$duration = NULL

character_vars = lapply(data, class) == "character"
data[, character_vars] = lapply(data[, character_vars], as.factor)
#str(data)
```

```r
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]

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

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

#sum(c(dim(train)[1],dim(valid)[1], dim(test)[1])) == dim(data)[1]
```

Task 2

Fit three different models from the given settings.

```r
n = dim(train)[1]
fit_def = tree(formula = y ~., data=train, control = tree.control(nobs = n))
fit_node = tree(formula = y ~., data=train, control = tree.control(nobs = n,
minsize = 7000))
fit_dev = tree(formula = y ~., data=train, control = tree.control(nobs = n,
mindev = 0.0005))

misclass = matrix(0,3,2)
dimnames(misclass) = list( c("Default","Node","Deviance"),
c("Test","Validation") )


pred_def_train = predict(fit_def, newdata = train, type = "class")
pred_def_val = predict(fit_def, newdata = valid, type = "class")
misclass[1,1] = sum(train$y != pred_def_train)/length(train$y)
misclass[1,2] = sum(valid$y != pred_def_val)/length(valid$y)


pred_node_train = predict(fit_node, newdata = train, type = "class")
pred_node_val = predict(fit_node, newdata = valid, type = "class")
misclass[2,1] = sum(train$y != pred_node_train)/length(train$y)
misclass[2,2] = sum(valid$y != pred_node_val)/length(valid$y)


pred_dev_train = predict(fit_dev, newdata = train, type = "class")
pred_dev_val = predict(fit_dev, newdata = valid, type = "class")
misclass[3,1] = sum(train$y != pred_dev_train)/length(train$y)
misclass[3,2] = sum(valid$y != pred_dev_val)/length(valid$y)
```

```
misclass
```

```
##                 Test Validation
## Default  0.10484406  0.1092679
## Node     0.10484406  0.1092679
## Deviance 0.09400575  0.1119221
```

From this result the default/nodesize trees performs best on validation data while deviance model does the opposite. Judging by these result the Deviance-model would be overfitting compared to the other two which also could be seen in the graphical illustrations below. The deviance model do have more potential to be optimized as this tree is much larger. The default and nodesize does only differ by one node which can explain why they perform identically!

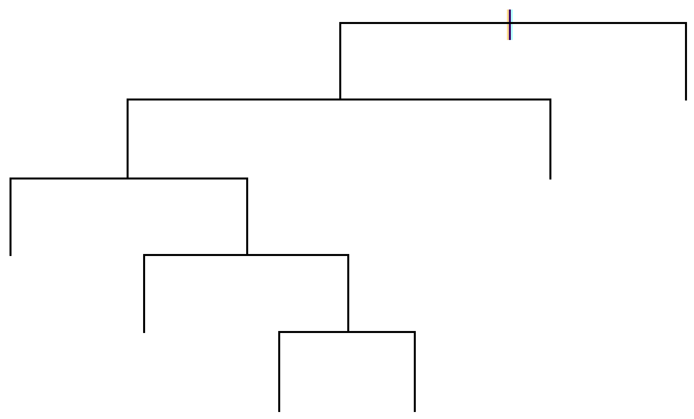
Increasing the minimum node size results in a smaller tree as the data is divided in to larger areas (each area represent a terminal node) and there are less “classification squares” to decide a label between.

Decreasing the deviance forces the model to split nodes more often which will result in a larger tree as seen below. This does introduce more risk for overfitting. The deviance is calculated by

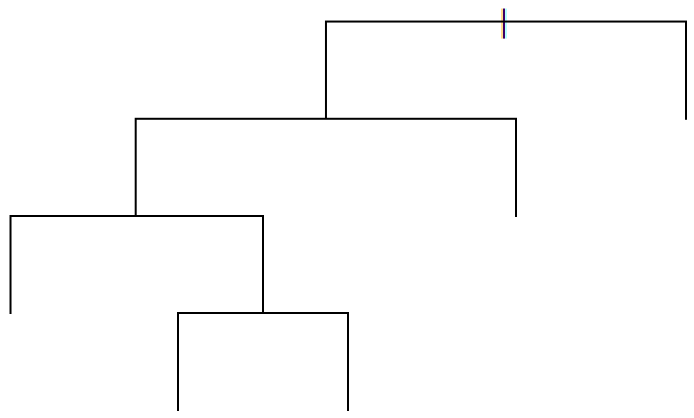
$$Deviance = -\sum_{i=1}^{n} p\,(c_i) log(p(c_i))$$

where $p(c_i)$ is the probability of class c_i in the node. Using a lower value for deviance forces the probability for classification to be higher which results in more splits.

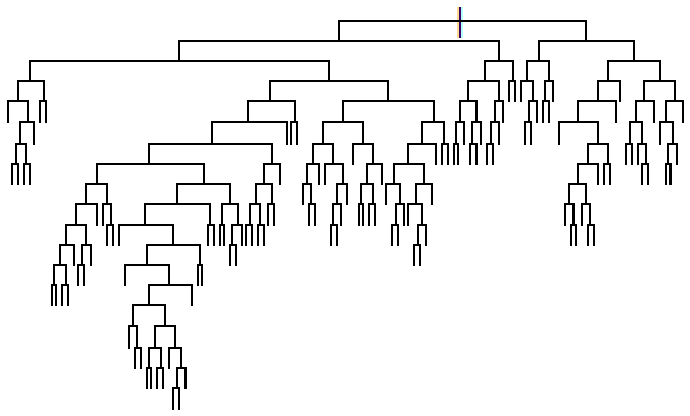
```
plot(fit_def, type = "uniform")
```



```r
plot(fit_node, type = "uniform")
```



```r
plot(fit_dev, type = "uniform")
```

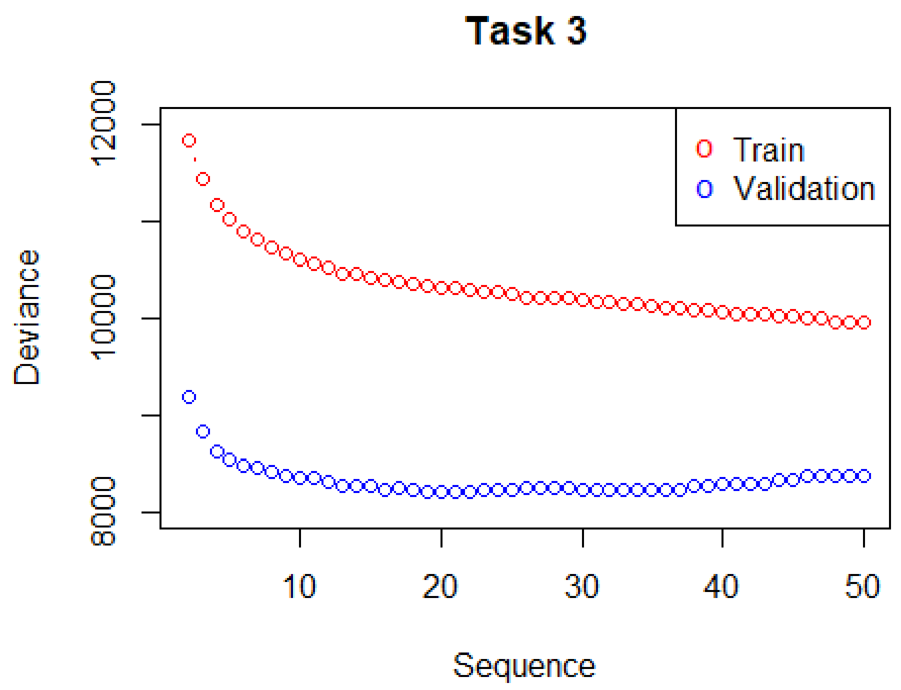


## Task 3

```
trainScore = rep(0,50)
testScore = rep(0,50)

for(i in 2:50){
  prunedTree = prune.tree(fit_dev, best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree" )
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred)
}
```

As seen in the figure below the trees total deviance decreases rapidly during the first part as the model is able to distinguish between more features. As the number of leaves increases the training data's deviance keeps decreasing while the validation data starts to increase after around 20 leafs. This is where the optimal tree size is found. The reason why the training-deviance is larger than the validations is due to the training set being larger therefore more values to sum up.

```
plot(2:50, trainScore[2:50], type = "b", col = "red", ylim = c(8000,12000),
main = "Task 3", xlab = "Sequence", ylab = "Deviance")
points(2:50, testScore[2:50], type = "b", col = "blue")
legend("topright", c("Train","Validation"), pch = c("o","o"), col=
c("red","blue"))
```

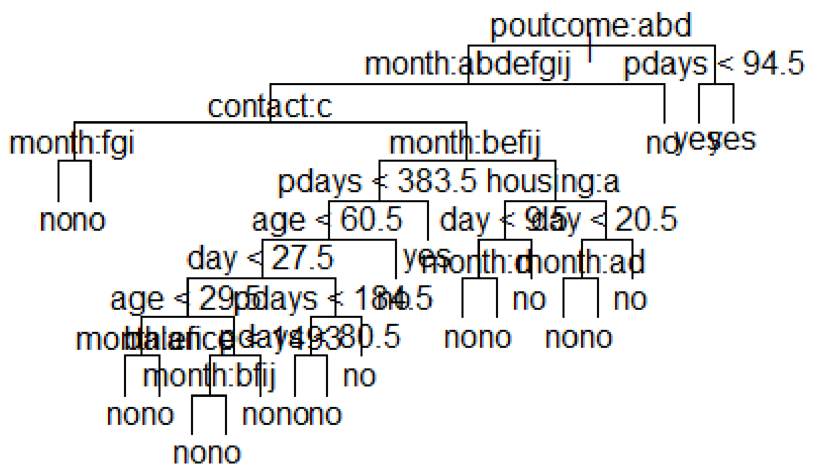


The optimal size (number of leaves/terminal nodes) is found where at the minimum deviance. The optimal number of leaves and the most significant features is shown in the summary of the optimal model:

```
opt_leaf = which.min(testScore[2:50])
opt_mdl = prune.tree(fit_dev, best = opt_leaf)
summary(opt_mdl)

##
## Classification tree:
## snip.tree(tree = fit_dev, nodes = c(581L, 17L, 577L, 79L, 37L,
## 77L, 576L, 153L, 580L, 6L, 1157L, 16L, 5L, 1156L, 156L, 152L,
## 579L, 7L))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"
"balance"
## [8] "housing"
## Number of terminal nodes:  21
## Residual mean deviance:  0.5706 = 10310 / 18060
## Misclassification error rate: 0.1041 = 1882 / 18084
```

The tree structure seems to use more terminal nodes than necessary as there are a lot of nodes which leads to the same label/ typ of terminal node, could be due to the deviance setting. The number of terminal nodes are indeed the same as the optimal which where found above. The outcome feature seems to be the most significant as it is set as the root node. This feature indicate wherever the customer was persuaded to subscribe a term deposit, which seems like a good starting point.

```r
plot(opt_mdl, type = "uniform")
text(opt_mdl)
```



The prediction power of this model is ok, it does get the right answer in about 9/10 cases but its' main failure is by classifying False Negatives.

```r
pred_valid = predict(opt_mdl, newdata = valid, type = "class")
val_error = sum(pred_valid != valid$y)/length(valid$y)
val_conf_matrix = table(valid$y, pred_valid)
val_conf_matrix

##      pred_valid
##          no   yes
##   no  11758   167
##   yes  1319   319

sprintf("Misclassification rate: %.4f", val_error)

## [1] "Misclassification rate: 0.1096"
```

Task 4

By introducing a loss matrix to penalize the False Negative predictions the misclassification rate did increase but the number of False Negative classifications decreased! Using a loss matrix will direct the focus of learning towards the weighting of the matrix, in this case classifying False Negatives gives a five time larger penalty than False Positive and this will affect the overall classification which is why error rate increased.

Using the loss matrix does give a good result as the previous model have been trained on both the training and validation set while this model only have been exposed to the training set, suggesting that it learn faster (but more knowledge about the data is required to set up the loss matrix).

```
lossMatrix = matrix(c(0,5,1,0),2,2)

fit_matrix = rpart(formula = y~., data = train, parms = list(loss =
lossMatrix))
pred_matrix = predict(fit_matrix, newdata = test, type = "class")
error = sum(pred_matrix != test$y)/length(test$y)
cm_matrix = table(test$y, pred_matrix)
cm_matrix
```

```
##      pred_matrix
##          no   yes
##   no  10880  1099
##   yes   807   778
```

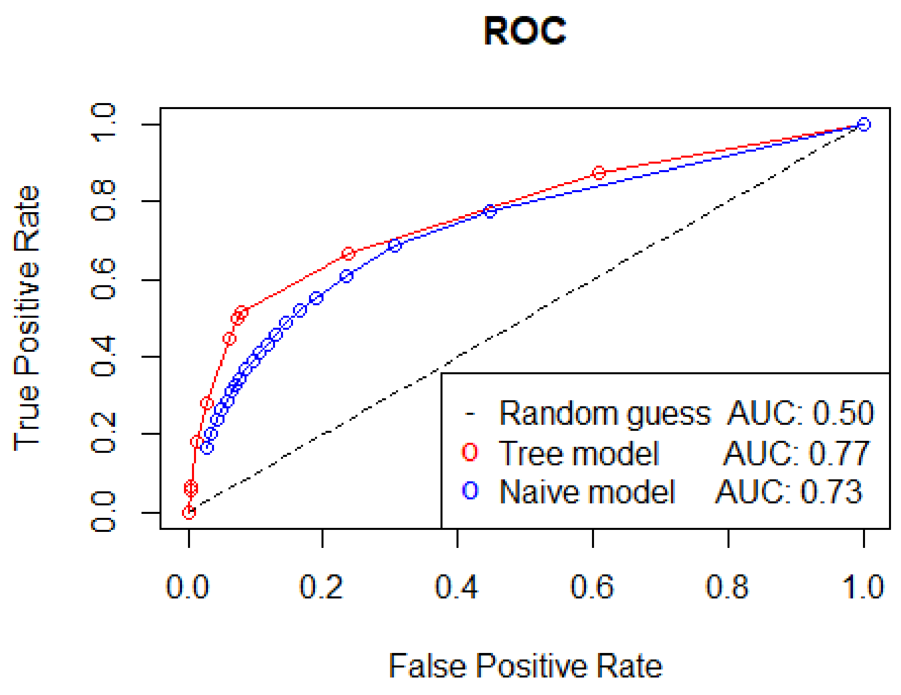
```
sprintf("Misclassification rate: %.4f", error)
```

```
## [1] "Misclassification rate: 0.1405"
```

Task 5

As can be seen in the graph below the optimal tree model seems to be the best model as the "area under curve" (AUC) is greater than the naive's model. Comparing two models AUC could occasionally lead the wrong assumption within certain regions but in practice the AUC-measure performs well as a general comparison. The AUC is calculated using the trapezoidal rule. In this case the tree models ROC curve is always above the naive's which also indicates that the tree model would perform best. Both models are well above the dashed line which could be interpret as a minimum boarder for a models predictive power, if a models is below this line the predictive power is worse than random guessing.

```
fit_naive = naiveBayes(formula = y~., data = train)

opt_pred = predict(opt_mdl, newdata = test, type = "vector")
naive_pred = predict(fit_naive, newdata = test, type = "raw")

thres = seq(0, 0.95, 0.05)
TPR_tree = rep(0,length(thres))
FPR_tree = rep(0,length(thres))
TPR_naive = rep(0,length(thres))
FPR_naive = rep(0,length(thres))
y = test$y


for(i in 1:length(thres)){
  TP_tree = 0
  FP_tree = 0
  TP_naive = 0
  FP_naive = 0

  for(j in 1:length(test$y)){

    if(opt_pred[j,2] >= thres[i]){
      if(y[j] == "yes"){
        TP_tree = TP_tree + 1
        }
      else{
        FP_tree = FP_tree + 1
        }
      }

    if(naive_pred[j,2] >= thres[i]){
      if(y[j] == "yes"){
        TP_naive = TP_naive + 1
        }
      else{
        FP_naive = FP_naive + 1
        }
    }

  }
  TPR_tree[i] = TP_tree/sum(test$y == "yes")
  FPR_tree[i] = FP_tree/sum(test$y == "no")

  TPR_naive[i] = TP_naive/sum(test$y == "yes")
  FPR_naive[i] = FP_naive/sum(test$y == "no")
}

sub_model = seq(0,1,0.01)
AUC_tree = sum(abs(diff(FPR_tree)) * (head(TPR_tree,-1)+tail(TPR_tree,-1)))/2
AUC_naive = sum(abs(diff(FPR_naive)) * (head(TPR_naive,-1)+tail(TPR_naive,-
1)))/2
AUC_sub = sum(abs(diff(sub_model)) * (head(sub_model,-1)+tail(sub_model,-
1)))/2

plot(sub_model, sub_model, col = "black", type = "l", lty = 2, main = "ROC",
xlab = "False Positive Rate", ylab = "True Positive Rate")
lines(FPR_tree, TPR_tree, col = "red", type = "o")
lines(FPR_naive, TPR_naive, col = "blue", type = "o")

leg_text = c(sprintf("Random guess  AUC: %.2f", AUC_sub),sprintf("Tree model
AUC: %.2f", AUC_tree),sprintf("Naive model     AUC: %.2f", AUC_naive))

legend("bottomright", legend = leg_text, pch = c("-","o","o"), col =
c("black", "red", "blue"))
```



Assignment 3: Principal components for crime level analysis.

Task 1

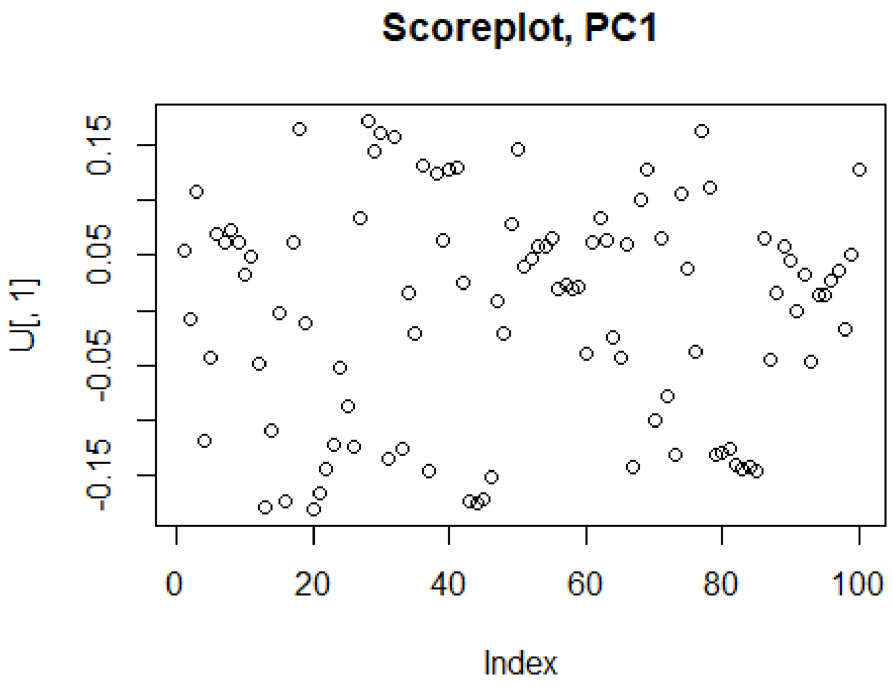
```
communties <- read.csv("communities.csv")
communties <- communties[, -communties$ViolentCrimesPerPop]
#1
df <- (scale(communties))
s <- cov(df)
s.eigen <- eigen(s)
save <- c()
for (v in s.eigen$values) {
  h<- (v / sum(s.eigen$values))
  save <- c(save, h)
  }
save[1]+save[2]

## [1] 0.4230255

nintyfive <- 0
i <- 1
while (nintyfive < 0.95) {
 nintyfive <-  save[i] + nintyfive
 i <- i + 1
}
i-1
```

```
## [1] 35
```

The first to components explains around 42 percent of the variation. It needs 35 features to obtain at least 95% of variance in the data.

Task 2

```
#PRINCOMP
res <- princomp(df)
lambda <- res$sdev^2
#Plot
U <- res$loadings
x <- res$scores

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



The Scoreplot over PC1 shows that it seems around 15 features has a higher absolute value then 0.15 which means that they have a notable contribution to the first component.

The 5 features that has the biggest contribution to the first component are shown in the table down. We can see that many of they features has to do with Income and family which is a very logical relationship to crime level

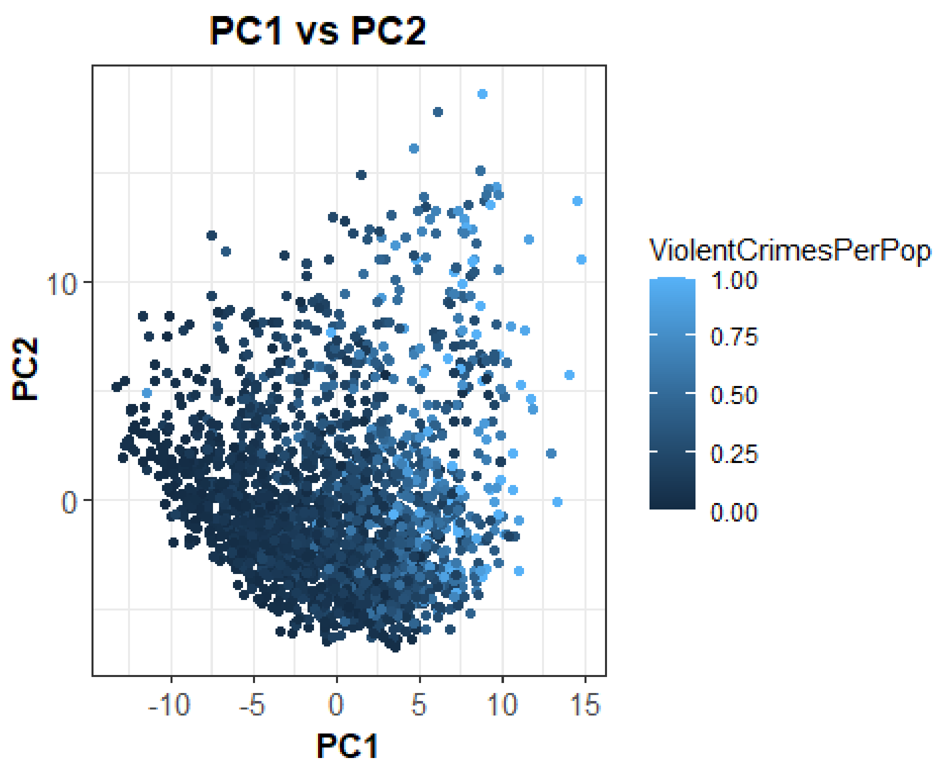
```
#Absolute values
features <- as.data.frame(U[,1])
features$variable <- rownames(features)
features$`U[, 1]` <- abs(features$`U[, 1]`)
```

```
newdata <- features[order(features[,1],decreasing = TRUE ),]
newdata[1:5, ]
```

```
##                 U[, 1]    variable
## medFamInc   0.1802364   medFamInc
## medIncome   0.1788053   medIncome
## PctKids2Par 0.1754391 PctKids2Par
## pctWInvInc  0.1736648  pctWInvInc
## PctFam2Par  0.1723885  PctFam2Par
```

The 5 features that has the biggest contribution to the first component are shown in the table above. We can see that many of they features has to do with Income and family which is a very logical relationship to crime level

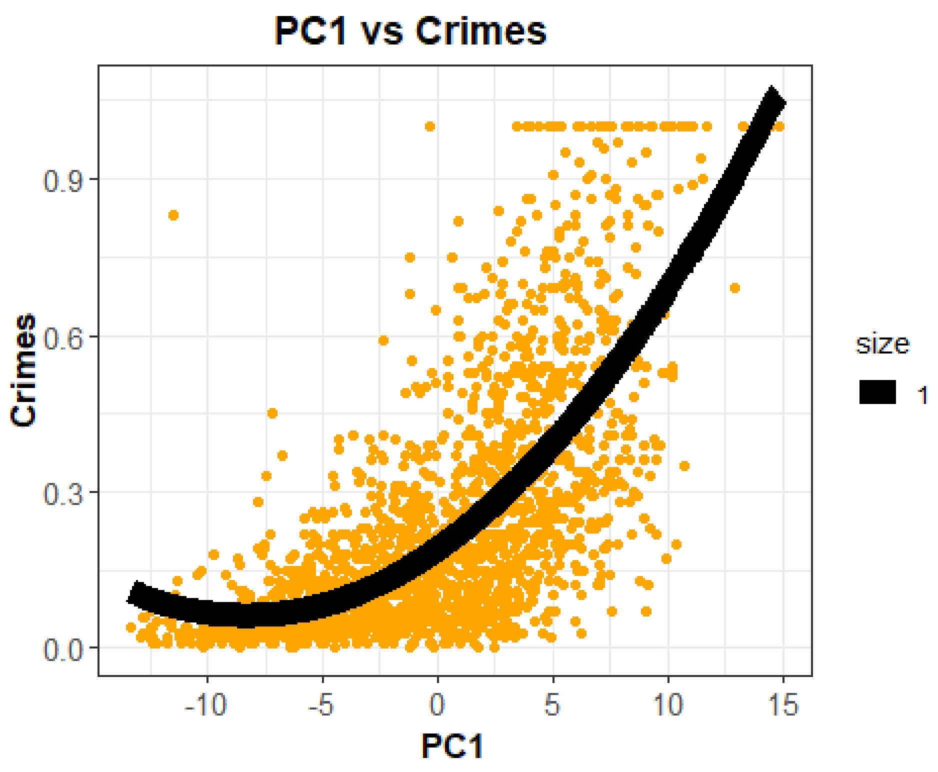
```
#pc1 AGAINST PC2

pc12 <- as.data.frame(x[,1:2])
pc12$crimes <- communties$ViolentCrimesPerPop
library(ggplot2)
p1 <- ggplot(data = pc12, aes(x=Comp.1, y=Comp.2, color=crimes))+
  geom_point() +
  theme_bw() +
  labs(title = "PC1 vs PC2"  , x = "PC1" , y = "PC2", colour =
"ViolentCrimesPerPop")  +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold"))
+
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold"))
+
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))
p1
```



The plot above shows the PC1 versus PC2 with color of Violentcrimesperpop. The picture shows that the PC1 is pretty good to separate low and high values on violentcrimesperpop. It's hard to see any pattern between PC1 and PC2.

Task 3

```r
model <- lm(crimes ~ poly(Comp.1,2), data = pc12)
pc12$pred <- (model$fitted.values)

ggplot(pc12, aes(x=Comp.1, y = crimes)) +
  geom_point(color= "orange")  + theme_bw() +
  geom_line(aes(y=pred, lwd=1)) +
    labs(title = "PC1 vs Crimes"  , x = "PC1" , y = "Crimes")  +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold"))
+
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold"))
+
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))
```



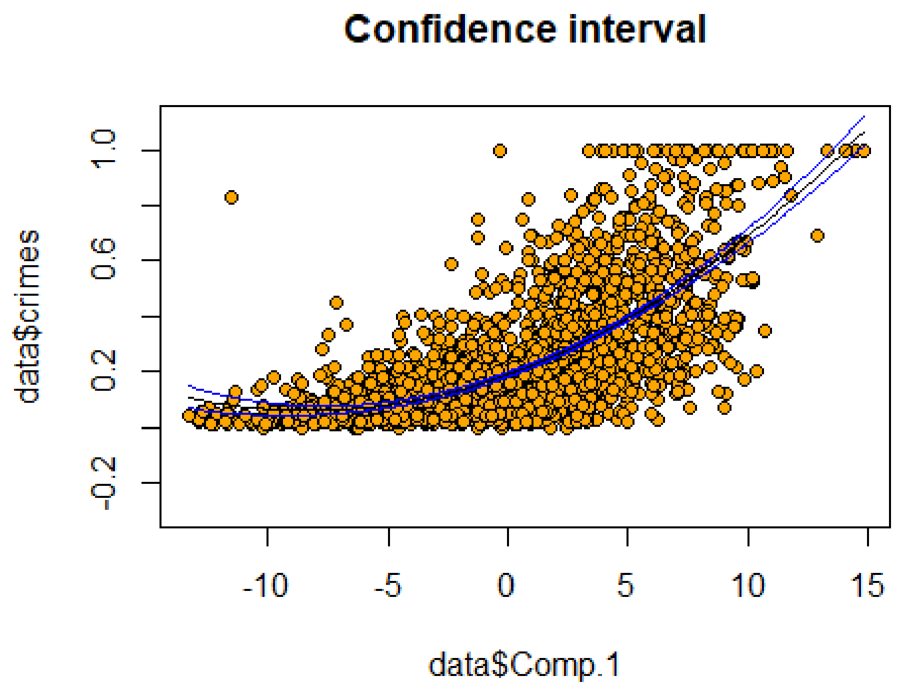
The plot above shows the violentcrimesperpop versus PC1 with the fitted values from the model above as the black line. The graph shows that the model seems to capture the connection between the target and the feature pretty good.

Task 4

a)

```
library(boot)
data <- pc12
data2 <- data[order(data$Comp.1),]
mle <- lm(crimes ~ poly(Comp.1,2), data = data2)
rng <- function(data,mle2){
  n <- nrow(data)
  data1<- data.frame(crimes=data$crimes,Comp.1=data$Comp.1)
  data1$crimes <- rnorm(n,predict(mle2, newdata=data1),sd(mle2$residuals))
  return(data1)}


f1=function(data1){
  res=lm(crimes~poly(Comp.1,degree=2),data1)
  ViolentCrimesPerPopP=predict(res,newdata=data2)
  return(ViolentCrimesPerPopP)}

fit <- lm(crimes ~ poly(Comp.1,2), data = data2)
crimesP=predict(fit)
```

```r
res2 <- boot(data2, statistic=f1, R=3000, mle = fit, ran.gen=rng  ,
sim="parametric")
e2 <- envelope(res2)


plot(data$Comp.1,data$crimes , pch=21, bg="orange", ylim=c(-0.3,1.1), main =
"Confidence interval")
points(data2$Comp.1,crimesP,type="l") #plot fitted line
#plot confidence bands
points(data2$Comp.1,e2$point[2,], type="l", col="blue")
points(data2$Comp.1,e2$point[1,], type="l", col="blue")
```

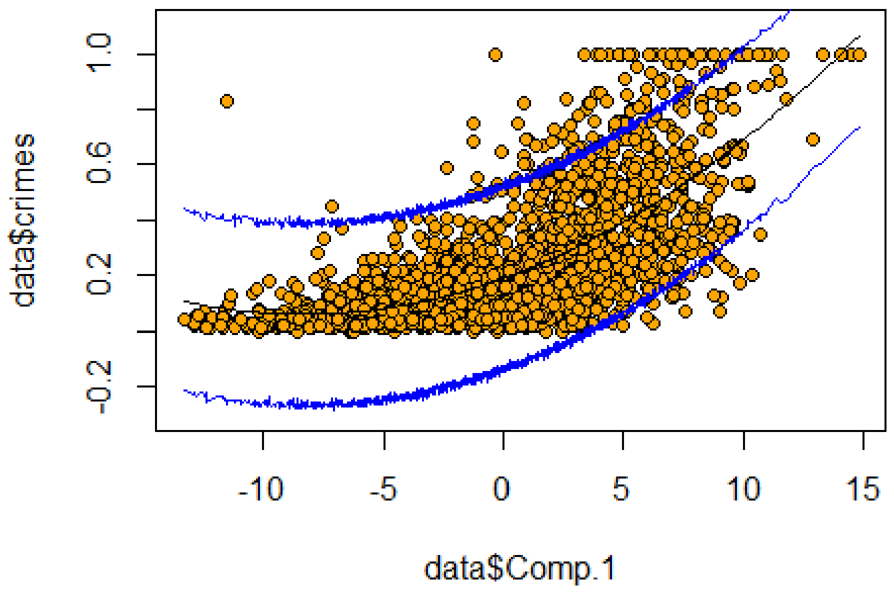


The plot above shows the plot from task 3 but now with confidence intervals. You can see that the confidence intervals are wider in the beginning and the end. That is because is fewer points there compared to in the middle of the plot which gives them more uncertainty.

b)

```r
f1 <- function(data1){
  res <- lm(crimes ~ poly(Comp.1,2), data = data1)
  crimesP <- predict(res, newdata = data2 )
  n <- length(data2$crimes)
  predictedP <- rnorm(n, crimesP, sd(mle$residuals))
  return(predictedP)
}
```

```
rng <- function(data, mle ){
  data1 <- data.frame(crimes=data$crimes, Comp.1=data$Comp.1)
  n <- length(data$crimes)
  data1$Crimes <- rnorm(n, predict(mle, newdata = data1), sd(mle$residuals))
  return(data1)
}



res <- boot(data2, statistic=f1, R=3000,
         mle=mle,ran.gen=rng, sim="parametric")

e <- envelope(res) #compute prediction bands

## Warning in envelope(res): unable to achieve requested overall error rate

fit <- lm(crimes ~ poly(Comp.1,2), data = data2)
crimesP=predict(fit)
plot(data$Comp.1,data$crimes , pch=21, bg="orange", ylim=c(-0.3,1.1))
points(data2$Comp.1,crimesP,type="l") #plot fitted line
#plot prediction bands
points(data2$Comp.1,e$point[2,], type="l", col="blue")
points(data2$Comp.1,e$point[1,], type="l", col="blue")
```



The plot above shows the plot from task 3 but now with prediction intervals. You can see that the predictions bands have almost the same width all over the plot and that the intervals alot of the points in the plot. This is because prediction is made out of every fitted point instead of confidence interval which is made out of the fitted line.

Appendix:

```r
knitr::opts_chunk$set(echo = TRUE)
library(kknn)
library(ggplot2)
library(e1071)
library(tree)
library(rpart)
library(ggplot2)
library(datasets)

ggplot(iris, aes(x=Sepal.Width , y= Sepal.Length, shape= Species,
color=Species)) + geom_point() +stat_ellipse()

data(iris)
### covariance matrix per each class
paste("Covariance matrix in Setosa class")
(var(subset(iris,subset=Species=='setosa',select=c(1:2))) -> S1)
paste("Covariance matrix in Versicolor class")
(var(subset(iris,subset=Species=='versicolor',select=c(1:2))) -> S2)
paste("Covariance matrix in Virginica class")
(var(subset(iris,subset=Species=='virginica',select=c(1:2))) -> S3)


### mean per each class

m1 <- colMeans(subset(iris,subset=Species=='setosa',select=c(1:2)))

m2 <- colMeans(subset(iris,subset=Species=='versicolor',select=c(1:2)))

m3 <- colMeans(subset(iris,subset=Species=='virginica',select=c(1:2)))

paste("mean for each class")
results <- cbind(m1,m2,m3)
colnames(results) <- c("setosa","versicolor","virginica")
rownames(results) <- c("Sepal.Length", "Sepal.Width")
results


### prior probability per each class
paste("prior probability for each class: ")
(nrow(subset(iris,subset=Species=='setosa')) / nrow(iris) -> prio_prob1)
(nrow(subset(iris,subset=Species=='versicolor')) / nrow(iris) -> prio_prob2)
```

```r
(nrow(subset(iris,subset=Species=='virginica')) / nrow(iris) -> prio_prob3)

paste("degree of freedom: ")
(table(iris$Species)-1 -> dof)


paste("pooled covariance matrix: ")
((dof[1]*S1+dof[2]*S2+dof[3]*S3)/sum(dof) -> S)
multi_gusi_model <- function(x,mean,sigma){
  p <- ncol(x)
  sig_inv <- solve(sigma)
  state1 <- (sqrt((2*pi)^p) *(det(sigma)))
  state2 <- -(1/2)*(as.matrix(x-mean) %*% solve(sigma) %*% t(x-mean) )
  model <- (1/state1) * exp(state2)
  return(model)
}

discriminant <- function(x,mean_i, pooled_cov, prior_i){
  #mu <- as.matrix(mean_i)
  #browser()
  state1 <- as.matrix(x) %*% solve(pooled_cov) %*% mean_i
  state2 <- 0.5 * t(mean_i) %*% solve(pooled_cov) %*% mean_i
  d <- state1 - as.numeric(state2) + as.numeric(log(prior_i))
  return(d)
}
#example for one class
#pooled <- ((dof[1]*S1+dof[2]*S2+dof[3]*S3)/sum(dof) -> S)
prior <- (nrow(subset(iris,subset=Species=='setosa')) / nrow(iris) ->
prio_prob1)
d1 <-discriminant(iris[c(1,2)] ,matrix(m1,ncol = 1),S,prio_prob1)

d2 <-discriminant(iris[c(1,2)] ,matrix(m2,ncol = 1),S,prio_prob2)

d3 <-discriminant(iris[c(1,2)] ,matrix(m3,ncol = 1),S,prio_prob3)

D_mat <- cbind(d1,d2,d3)
colnames(D_mat) <- c("setosa","versicolor","virginica")
paste("Discriminant matrix for three different species" )
D_mat
w0 <- function(pi_k, pi_l, mu_k, mu_l, pooled_cov){
  m1 <- as.matrix(m1)
  m2 <- as.matrix(m2)
  w0 <- -(1/2)*t(mu_k + mu_l) %*% solve(pooled_cov) %*% (mu_k - mu_l)
  return(w0)
}
w1 <-function(pi_k, pi_l, mu_k, mu_l, pooled_cov){
  m1 <- as.matrix(m1)
  m2 <- as.matrix(m2)
  w1 <-  solve(pooled_cov) %*% (mu_k - mu_l)
  return(as.matrix(w1))
}
msg1 <- paste("decision boundary for Setosa ~ Versicolor\n\n")
msg2 <- paste(msg1, "W1 : ")
noquote(strsplit(msg2, "\n")[[1]])
w1(0.33,0.33,m1,m2,S)
paste("W0 : ", w0(0.33,0.33,m1,m2,S))

msg1 <- paste("decision boundary for Versicolor ~ Verginica\n\n")
msg2 <- paste(msg1, "W1 : ")
noquote(strsplit(msg2, "\n")[[1]])
w1(0.33,0.33,m2,m3,S)
paste("W0 : ",w0(0.33,0.33,m2,m3,S))

msg1 <- paste("decision boundary for Verginica ~ Setosa\n\n")
msg2 <- paste(msg1, "W1 : ")
noquote(strsplit(msg2, "\n")[[1]])
w1(0.33,0.33,m3,m1,S)
paste("W0 :  ", w0(0.33,0.33,m3,m1,S))
library(ggplot2)
D <- cbind(d1,d2,d3)

for(i in row(D)){
  predict <-apply(X = D[,], MARGIN = 1, FUN = which.max)
}
new_df <- iris
new_df <- cbind(new_df, predict)
new_df$predict[new_df$predict == "1"] <- "setosa"
new_df$predict[new_df$predict == "2"] <- "versicolor"
new_df$predict[new_df$predict == "3"] <- "virginica"
ggplot(new_df, aes(x=Sepal.Length , y= Sepal.Width, shape= predict,
color=predict)) + geom_point()+stat_ellipse()

Actual<- iris$Species
t_3 <-table(Actual, new_df$predict)
knitr::kable(t_3, caption = "MissClassification matrix for manual
prediction")
lda_error_3 = 1- sum(diag(t_3))/sum(t_3)
paste("MSE for manual LDA: " ,lda_error_3)
library(MASS)
new_data3c <- iris
fitted_lda <- lda(Species~Sepal.Length + Sepal.Width, data = new_data3c)
fitted_lda
#### Confusion Matrix###
Classification <- predict(fitted_lda, data = new_data3c)$class
Actual<- iris$Species
t <-table(Actual, Classification)
knitr::kable(t,caption = "MissClassification matrix for LDA")
lda_error = 1- sum(diag(t))/sum(t)
```

```r
paste("MSE of LDA model: ", lda_error)

library("mvtnorm")
bvn1 <- as.data.frame(mvrnorm(50, mu =m1, S ))
bvn2 <- as.data.frame(mvrnorm(50, mu=m2, S))
bvn3 <- as.data.frame(mvrnorm(50, mu=m3, S))
bv <- rbind(bvn1,bvn2,bvn3)

ggplot() +
  geom_point(data=bv, aes(bv$Sepal.Length, bv$Sepal.Width,
color=iris$Species))
library(nnet)
multi_model <- multinom(Species~Sepal.Length + Sepal.Width, data =iris)
multi_model
new_data <- iris
new_data$predicted<- predict(multi_model, new_data, type="class")
summary(new_data)
t_multi <- table(new_data$Species, new_data$predicted)
t_multi
ME_error <- 1-sum(diag(t_multi))/sum(t_multi)
paste( "ME_error= ", ME_error)


ggplot(new_data, aes(x=Sepal.Width , y= Sepal.Length, shape= predicted,
color=predicted)) + geom_point() +stat_ellipse()


data = read.csv2("bank-full.csv")
data$duration = NULL

character_vars = lapply(data, class) == "character"
data[, character_vars] = lapply(data[, character_vars], as.factor)
#str(data)


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

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

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

#sum(c(dim(train)[1],dim(valid)[1], dim(test)[1])) == dim(data)[1]
```

```r
n = dim(train)[1]
fit_def = tree(formula = y ~., data=train, control = tree.control(nobs = n))
fit_node = tree(formula = y ~., data=train, control = tree.control(nobs = n,
minsize = 7000))
fit_dev = tree(formula = y ~., data=train, control = tree.control(nobs = n,
mindev = 0.0005))

misclass = matrix(0,3,2)
dimnames(misclass) = list( c("Default","Node","Deviance"),
c("Test","Validation") )


pred_def_train = predict(fit_def, newdata = train, type = "class")
pred_def_val = predict(fit_def, newdata = valid, type = "class")
misclass[1,1] = sum(train$y != pred_def_train)/length(train$y)
misclass[1,2] = sum(valid$y != pred_def_val)/length(valid$y)


pred_node_train = predict(fit_node, newdata = train, type = "class")
pred_node_val = predict(fit_node, newdata = valid, type = "class")
misclass[2,1] = sum(train$y != pred_node_train)/length(train$y)
misclass[2,2] = sum(valid$y != pred_node_val)/length(valid$y)


pred_dev_train = predict(fit_dev, newdata = train, type = "class")
pred_dev_val = predict(fit_dev, newdata = valid, type = "class")
misclass[3,1] = sum(train$y != pred_dev_train)/length(train$y)
misclass[3,2] = sum(valid$y != pred_dev_val)/length(valid$y)

misclass

plot(fit_def, type = "uniform")
plot(fit_node, type = "uniform")
plot(fit_dev, type = "uniform")

trainScore = rep(0,50)
testScore = rep(0,50)

for(i in 2:50){
  prunedTree = prune.tree(fit_dev, best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree" )
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred)
}

plot(2:50, trainScore[2:50], type = "b", col = "red", ylim = c(8000,12000),
main = "Task 3", xlab = "Sequence", ylab = "Deviance")
points(2:50, testScore[2:50], type = "b", col = "blue")
```

```r
legend("topright", c("Train","Validation"), pch = c("o","o"), col=
c("red","blue"))

opt_leaf = which.min(testScore[2:50])
opt_mdl = prune.tree(fit_dev, best = opt_leaf)
summary(opt_mdl)
plot(opt_mdl, type = "uniform")
text(opt_mdl)
pred_valid = predict(opt_mdl, newdata = valid, type = "class")
val_error = sum(pred_valid != valid$y)/length(valid$y)
val_conf_matrix = table(valid$y, pred_valid)
val_conf_matrix
sprintf("Misclassification rate: %.4f", val_error)


lossMatrix = matrix(c(0,5,1,0),2,2)

fit_matrix = rpart(formula = y~., data = train, parms = list(loss =
lossMatrix))
pred_matrix = predict(fit_matrix, newdata = test, type = "class")
error = sum(pred_matrix != test$y)/length(test$y)
cm_matrix = table(test$y, pred_matrix)
cm_matrix
sprintf("Misclassification rate: %.4f", error)


fit_naive = naiveBayes(formula = y~., data = train)

opt_pred = predict(opt_mdl, newdata = test, type = "vector")
naive_pred = predict(fit_naive, newdata = test, type = "raw")

thres = seq(0, 0.95, 0.05)
TPR_tree = rep(0,length(thres))
FPR_tree = rep(0,length(thres))
TPR_naive = rep(0,length(thres))
FPR_naive = rep(0,length(thres))
y = test$y


for(i in 1:length(thres)){
  TP_tree = 0
  FP_tree = 0
  TP_naive = 0
  FP_naive = 0

  for(j in 1:length(test$y)){

    if(opt_pred[j,2] >= thres[i]){
      if(y[j] == "yes"){
        TP_tree = TP_tree + 1
        }
      else{
        FP_tree = FP_tree + 1
        }
      }

    if(naive_pred[j,2] >= thres[i]){
      if(y[j] == "yes"){
        TP_naive = TP_naive + 1
        }
      else{
        FP_naive = FP_naive + 1
        }
    }

  }
  TPR_tree[i] = TP_tree/sum(test$y == "yes")
  FPR_tree[i] = FP_tree/sum(test$y == "no")

  TPR_naive[i] = TP_naive/sum(test$y == "yes")
  FPR_naive[i] = FP_naive/sum(test$y == "no")
}


sub_model = seq(0,1,0.01)
AUC_tree = sum(abs(diff(FPR_tree)) * (head(TPR_tree,-1)+tail(TPR_tree,-1)))/2
AUC_naive = sum(abs(diff(FPR_naive)) * (head(TPR_naive,-1)+tail(TPR_naive,-
1)))/2
AUC_sub = sum(abs(diff(sub_model)) * (head(sub_model,-1)+tail(sub_model,-
1)))/2

plot(sub_model, sub_model, col = "black", type = "l", lty = 2, main = "ROC",
xlab = "False Positive Rate", ylab = "True Positive Rate")
lines(FPR_tree, TPR_tree, col = "red", type = "o")
lines(FPR_naive, TPR_naive, col = "blue", type = "o")

leg_text = c(sprintf("Random guess  AUC: %.2f", AUC_sub),sprintf("Tree model
AUC: %.2f", AUC_tree),sprintf("Naive model     AUC: %.2f", AUC_naive))

legend("bottomright", legend = leg_text, pch = c("-","o","o"), col =
c("black", "red", "blue"))

communties <- read.csv("communities.csv")
communties <- communties[, -communties$ViolentCrimesPerPop]
#1
df <- (scale(communties))
s <- cov(df)
s.eigen <- eigen(s)
```

```r
save <- c()
for (v in s.eigen$values) {
  h<- (v / sum(s.eigen$values))
  save <- c(save, h)
  }
save[1]+save[2]

nintyfive <- 0
i <- 1
while (nintyfive < 0.95) {
 nintyfive <-  save[i] + nintyfive
 i <- i + 1
}
i-1
#PRINCOMP
res <- princomp(df)
lambda <- res$sdev^2
#Plot
U <- res$loadings
x <- res$scores

plot(U[,1], main="Scoreplot, PC1")
#Absolute values
features <- as.data.frame(U[,1])
features$variable <- rownames(features)
features$`U[, 1]` <- abs(features$`U[, 1]`)
newdata <- features[order(features[,1],decreasing = TRUE ),]
newdata[1:5, ]
#pc1 AGAINST PC2

pc12 <- as.data.frame(x[,1:2])
pc12$crimes <- communties$ViolentCrimesPerPop
library(ggplot2)
p1 <- ggplot(data = pc12, aes(x=Comp.1, y=Comp.2, color=crimes))+
  geom_point() +
  theme_bw() +
  labs(title = "PC1 vs PC2"  , x = "PC1" , y = "PC2", colour =
"ViolentCrimesPerPop")  +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold"))
+
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold"))
+
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))
p1
model <- lm(crimes ~ poly(Comp.1,2), data = pc12)
pc12$pred <- (model$fitted.values)
```

```r
ggplot(pc12, aes(x=Comp.1, y = crimes)) +
  geom_point(color= "orange")  + theme_bw() +
  geom_line(aes(y=pred, lwd=1)) +
    labs(title = "PC1 vs Crimes"   , x = "PC1" , y = "Crimes")  +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold"))
+
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold"))
+
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))

library(boot)
data <- pc12
data2 <- data[order(data$Comp.1),]
mle <- lm(crimes ~ poly(Comp.1,2), data = data2)
rng <- function(data,mle2){
  n <- nrow(data)
  data1<- data.frame(crimes=data$crimes,Comp.1=data$Comp.1)
  data1$crimes <- rnorm(n,predict(mle2, newdata=data1),sd(mle2$residuals))
  return(data1)}


f1=function(data1){
  res=lm(crimes~poly(Comp.1,degree=2),data1)
  ViolentCrimesPerPopP=predict(res,newdata=data2)
  return(ViolentCrimesPerPopP)}

fit <- lm(crimes ~ poly(Comp.1,2), data = data2)
crimesP=predict(fit)

res2 <- boot(data2, statistic=f1, R=3000, mle = fit, ran.gen=rng  ,
sim="parametric")
e2 <- envelope(res2)


plot(data$Comp.1,data$crimes , pch=21, bg="orange", ylim=c(-0.3,1.1), main =
"Confidence interval")
points(data2$Comp.1,crimesP,type="l") #plot fitted line
#plot confidence bands
points(data2$Comp.1,e2$point[2,], type="l", col="blue")
points(data2$Comp.1,e2$point[1,], type="l", col="blue")

f1 <- function(data1){
  res <- lm(crimes ~ poly(Comp.1,2), data = data1)
  crimesP <- predict(res, newdata = data2 )
  n <- length(data2$crimes)
  predictedP <- rnorm(n, crimesP, sd(mle$residuals))
  return(predictedP)
}


rng <- function(data, mle ){
  data1 <- data.frame(crimes=data$crimes, Comp.1=data$Comp.1)
  n <- length(data$crimes)
  data1$Crimes <- rnorm(n, predict(mle, newdata = data1), sd(mle$residuals))
  return(data1)
}



res <- boot(data2, statistic=f1, R=3000,
         mle=mle,ran.gen=rng, sim="parametric")

e <- envelope(res) #compute prediction bands
fit <- lm(crimes ~ poly(Comp.1,2), data = data2)
crimesP=predict(fit)
plot(data$Comp.1,data$crimes , pch=21, bg="orange", ylim=c(-0.3,1.1))
points(data2$Comp.1,crimesP,type="l") #plot fitted line
#plot prediction bands
points(data2$Comp.1,e$point[2,], type="l", col="blue")
points(data2$Comp.1,e$point[1,], type="l", col="blue")
```