Lab2

Tugce Izci (tugiz798), Kerstin Nilsson (kerni714) and Alan Cacique Tamariz (alaca734)

2022-12-04

Assignment 1: EXPLICIT REGULARIZATION

The "tecator.csv" contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat.For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein.

1) Data divided randomly into train and test (50/50).Fat modeled as a linear regression where Channels were used as features. Training and test errors estimated. To commented on quality of fit and prediction, RMSE was calculated. RMSE is a good measure of how linear regression model accurately fits or predicts the response. Lower values of RMSE shows the fit or the prediction is good.

Linear regression model is

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_p x_p$$

$$y \sim N(\theta^T x, \sigma^2)$$

where

$$\theta = \{\theta_0, \theta_1, ..., \theta_p\}, x = \{1, x_1, x_2, ..., x_p\}$$

The values of the estimated regression coefficients are not displayed here, but code for obtaining the values of the coefficients is available in the Appendix.

Comparing test and training RMSE results, our model had a very low training error but a rather high testing error. This indicates that our regression model is overfitted.

```
#> rmse_training     rmse_test
#>     0.0755587    26.8780472
```

2) Lasso regression is a regularized model of linear regression. Aim of Lasso regression is to avoid overfitting.

Since our linear regression model was overfit; in this question Fat was modeled as a Lasso regression in which all Channels used as features.

This is the cost function that should be minimised in this case.

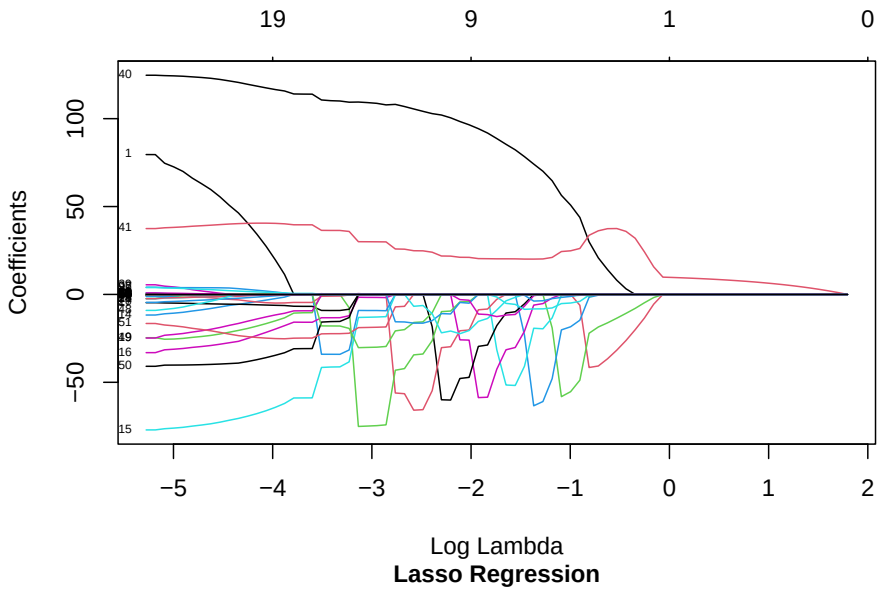
$$J(\theta) = \sum_{i=1}^{n} (\boldsymbol{\theta}^T \boldsymbol{x_i} - y_i)^2 + \lambda \sum_{j=1}^{p} |\theta_j|$$

where
n is the number of observations

p is the number of features
$\theta = \{\theta_0, \theta_1, ..., \theta_p\}$
$x = \{1, x_1, x_2, ..., x_p\}$

In above formula, first part represents the mean square error and the second part consists of a penalty term multiplied by a regularisation parameter λ.

3) Fitted the Lasso regression model to the training data. Plot illustrated regression coefficients depend on $\log \lambda$. In the graph, each curve corresponds to a variable. Numbers which are located top of the graph, indicates the number of non-zero coefficients. While $\log \lambda = 2$ non-zero coefficient value become 0. Lasso regression is pushing the coefficients to the zero.

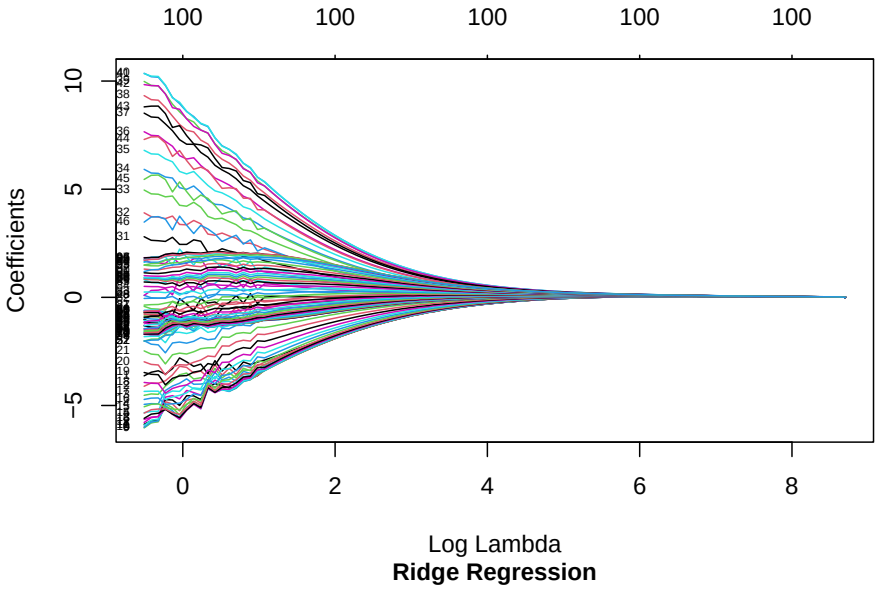



```
#>    Df %Dev Lambda
#> 22  3 29.2  0.853
#> 23  3 38.3  0.777
#> 24  3 45.9  0.708
```

In above seen three features. By selecting model with only three features we could here select the lambda corresponding to the highest deviance and this indicates lower penalty factor in between the features.So taking three features gives $\lambda = 0.708$.

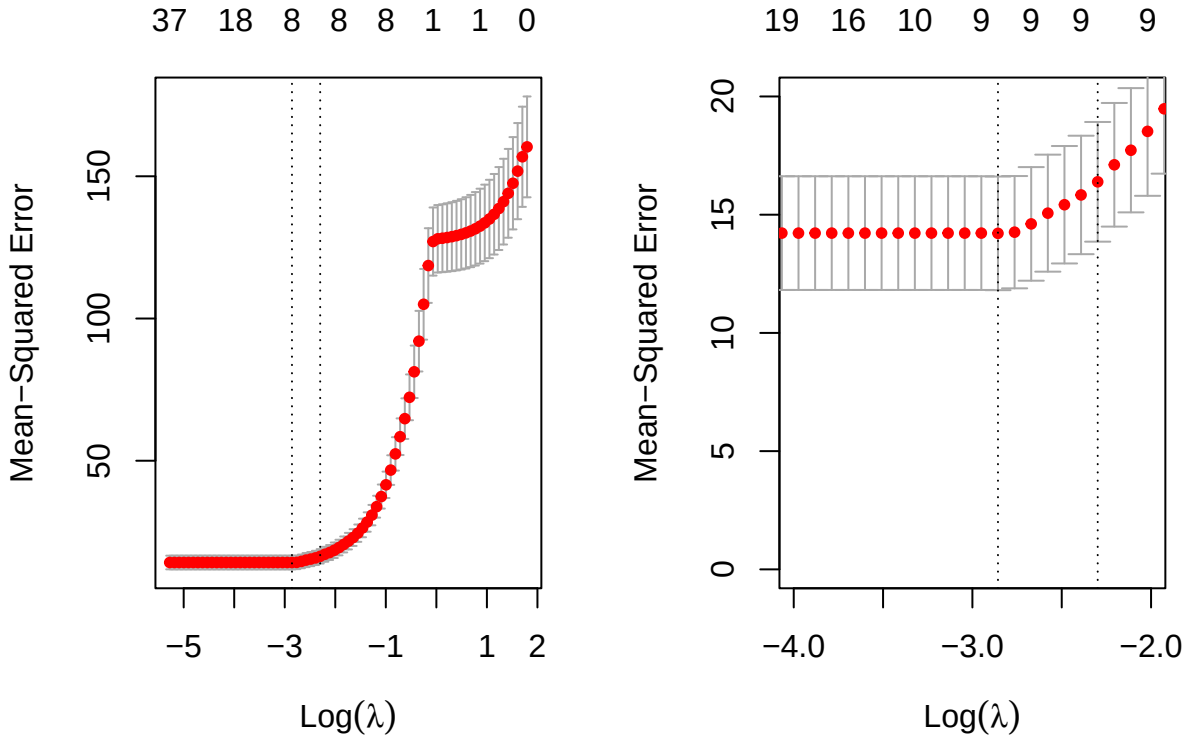
```
#> [1] 0.708
```

4) Fitting the Ridge regression and comparing with Lasso; Ridge regression doesn't perform feature selection and doesn't set the coefficients as equal to zero. Top of the graph, non-zero coefficients can be seen equal to 100.The higher value of lambda, the smaller the values of the coefficients.



5) Cross validation technique was used to compute optimal Lasso model, it was plotted, and optimal λ value was decided.

There are numbers in the top of the graph and these represent the number of non-zero coefficients.



The CV score (MSE) generally decreases with $\log \lambda$, with a steeper descent at $\log \lambda = 0$, until it reaches a

minimum just before $\log \lambda = -3$, where the CV score stays at the same level for the remaining λ sequence.

Our optimal λ value and optimal $\log \lambda$ value is calculated as below.

```
#> [1] 0.057445
```

```
#> [1] -2.8569
```

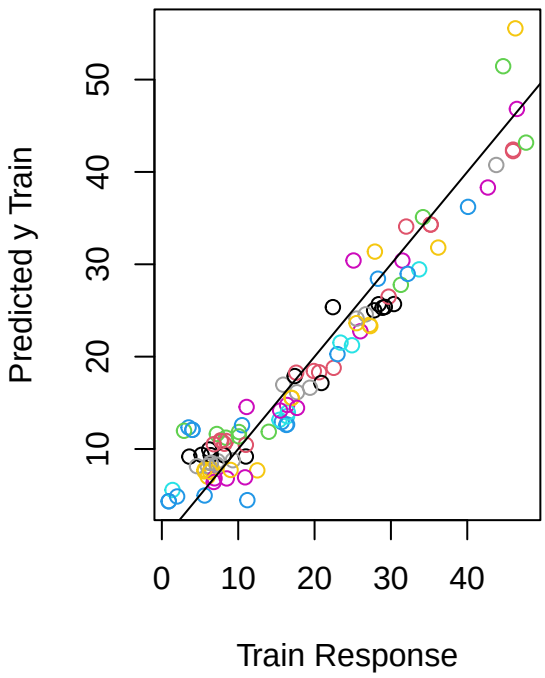
Summary of the result of the CV procedure, including number of variables in the model with optimal λ:

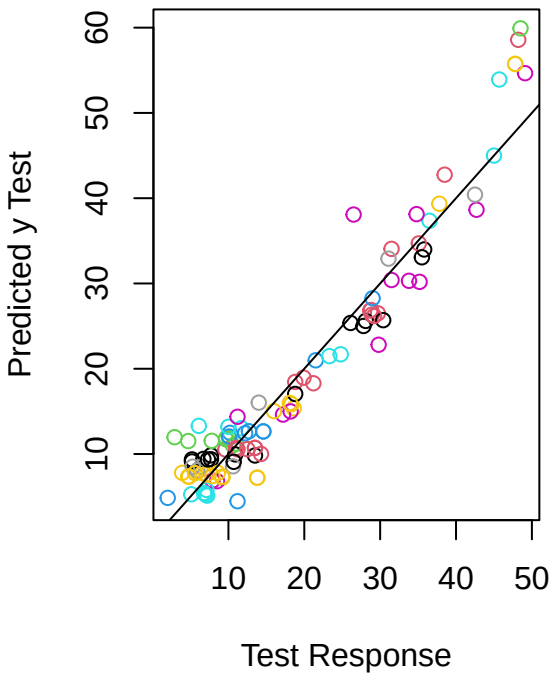
```
#>
#> Call:  cv.glmnet(x = covariates, y = response, alpha = 1, family = "gaussian")
#>
#> Measure: Mean-Squared Error
#>
#>     Lambda Index Measure   SE Nonzero
#> min 0.0574    51    14.2 2.40       8
#> 1se 0.1004    45    16.4 2.53       9
```

The number of variables in the model corresponding to the optimal λ is 8. It can be noted that the CV MSE is around 14 for the optimal λ, which correspond to an RMSE of 3.8 (which is considerably lower than the RMSE for the test set for the linear model).

If we compare $\log \lambda = -4$ and optimal $\log \lambda = -2.86$ the MSE CV values are not statistically different from each other as can be seen in the plot (the interval estimates are the same).Therefore, we cannot say that a model with the optimal λ results in a model with statistically significantly better prediction than a model with $\log \lambda = -4$

Scatter plot for the predicted and real data indicates model predictions seem overall reasonably good. Further, the performance look similar between the training and the test set.


Predicted y Train
10
20
30
40
50
0
10
20
30
40
Train Response


Predicted y Test
10
20
30
40
50
60
10
20
30
40
50
Test Response

Assignment 2. Decision trees and logistic regression for bank marketing

Provided description of data: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Objective: The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y). The class "yes" will thus be considered the positive class.

2.1 Import data and divide into training and test set

Upon reading the data to R, it was noted that the variable "balance" that is included in the dataset is not included in the provided description of the variables. Also, the variable "day" does not correspond to the variable "day_of_week" in the variable list, the variable "day" in the data appears to refer to the day of the month. Since there is no information on year, this variable cannot be converted to information regarding day of the week. It was further noted that some of the variable categories and values in the data differ from those in the description, e.g. in the description for the variable "pdays" it is noted that the value 999 means client was not previously contacted, however the maximum value of the variable pdays in the data is 871, whereas 82% of the observations have the value -1, which likely is the coding for the client was not contacted, or that the information is unknown. Modelling this value as a numeric (low) value could have an impact on the results. The variable "duration" was removed according to the instructions.

There are six numeric input variables: age, balance, day, campaign (number of contacts performed during this campaign), pdays (number of days that passed by after the client was last contacted from a previous campaign) and previous (number of contacts performed before this campaign),

and nine categorical input variables: job, marital (marital status), education, default (has credit in default), housing (has housing loan), loan (has personal loan), contact (contact communication type), month (last contact month of year), and day (day of the month).

As can be seen from table 1, there is imbalance in the class rates.

Table 1: Frequency table of target variable.

y	n	%
no	39922	88.3
yes	5289	11.7

2.2 Fit decision trees to the training data (three different models)

Three different models were fitted to the training data:

m1a - decision tree with default settings.
m1b - decision tree with smallest allowed node size equal to 7000.
m1c - decision tree with minimum deviance equal to 0.0005.

Default settings for controlling tree growth with respect to minimum nodesize and deviance is for nodesize equal to 10 and for deviance equal to 0.01. The interpretation of the minimun deviance is that the within-node deviance must be at least this parameter (mindev) times that of the root node for the node to be split.

Misclassification rates were calculated for the training and validation data sets, see table 2 for results. Confusion matrices for the validation data sets were also calculated, see table 3 for those results. Models

m1a and m1b have the lowest misclassification error for the validation sets, however model m1c is somewhat better at classifying the positive class. By following the principle of selecting the model with the lowest misclassification error for the validation set with the least complexity, model m1b would be selected, as it is less complex than model m1a (see table 4 below).

Table 2: Misclassification rates (%) for the three models.

model	err_train	err_valid
m1a	10.48	10.93
m1b	10.48	10.93
m1c	9.40	11.19

Table 3: Confusion matrices for the three models for the validation dataset.

model		no	yes	%
m1a	no	11772	153	98.7
	yes	1329	309	18.9
m1b	no	11772	153	98.7
	yes	1329	309	18.9
m1c	no	11715	210	98.2
	yes	1308	330	20.1

The trees were further investigated with respect to how changing the deviance and node size affected the size of the trees. Table 4 displays the size and deviance of the trees, as well as the minimum size and minimum deviance for the internal nodes. Adding the criterion smallest allowed node size equal to 7000 decreased the size of the tree by one leaf compared to using the default settings. This can be understood by inspecting the tree structure (code in the Appendix), where there was one internal node of size below 7000, that in model m1b had to be turned into a terminal node.

Changing the criterion of the minimum deviance from the default (0.01) to 0.0005 increased the tree considerably, to 122 leaves. The reason that the model m1c tree is larger is because internal nodes with smaller deviance are allowed. It can be seen in table 4 that the minimum node size and deviance for the internal nodes are much smaller for the model m1c tree than for the other trees.

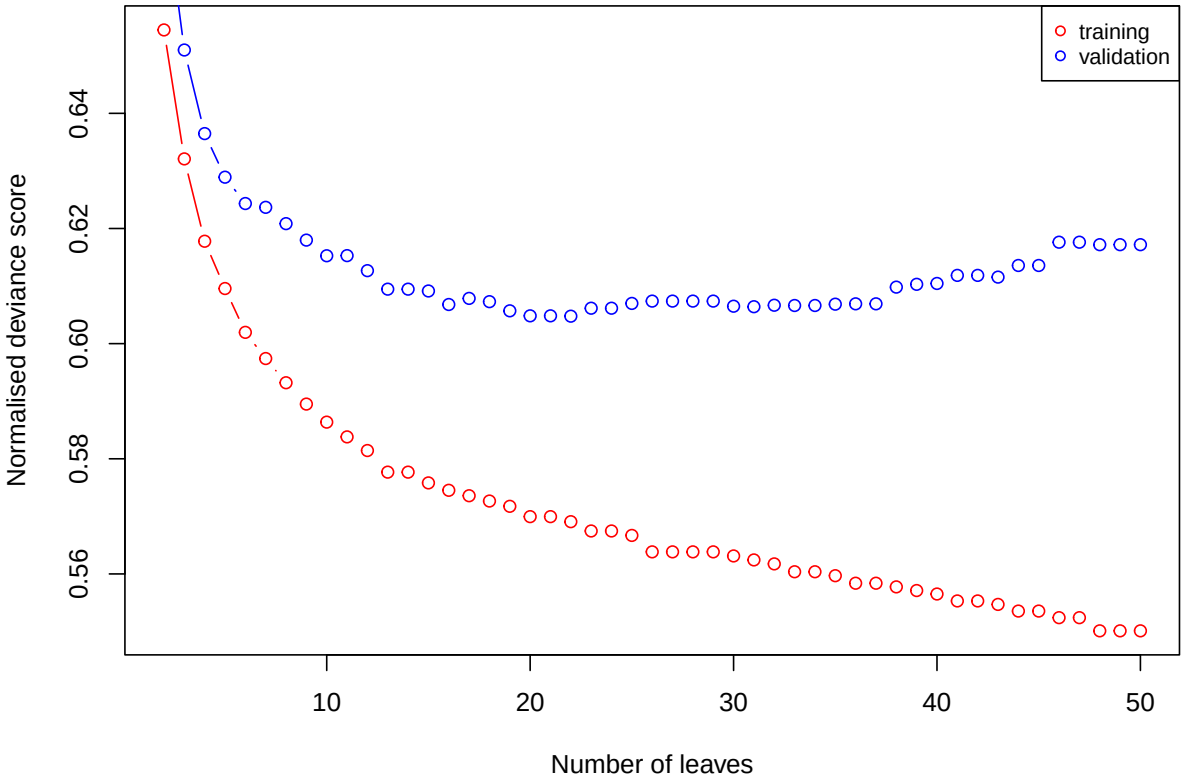
Table 4: Tree size and deviance, and minimum internal node size and deviance.

Model	Tree size	Deviance	Min int. node size	Min int. node deviance
m1a	6	10885.9	2414	2262.1
m1b	5	11023.5	11698	7913.8
m1c	122	9363.3	13	15.0

2.3. Choose the optimal tree depth in the model c in step 2 (model m1c)

The training and validation sets were used to choose the optimal tree depth in the model in step 2 (model 1c). Models were fitted with the training data for up to 50 leaves, and the validation set was used for prediction. Deviance was extracted for the training and validation sets in the models respectively, adjusted for the number of observations in each dataset. Figure 1 shows the deviances for the training and the validation data versus the number of leaves.

Figure 1. Deviances for the training and the validation data vs number of leaves



To the left in the graph, with the smaller amount of trees, are the least complex models that have a higher bias, as can be seen by that both the training and validation sets have a higher deviance. As the models become more and more complex, the deviance decreases for the training set, and for the test set also up till around 20 trees, the bias decreases, but with increasing complexity comes increased variance, which is noted by that the deviance decrease for the validation set levels off, and its deviance starts to also increase gradually from the lowest point at around 20 trees.

The optimal amount of leaves were chosen as the tree with the lowest validation error (and if there had been several trees that had an equally low validation error, the least complex tree would have been chosen).

```
#> Optimal amount of leaves

#> [1] 22
```

Inspecting the structure of the tree (code in Appendix), shows that the variables used in the tree are poutcome, month, contact, pdays, age, day, balance, housing and job. Of these, poutcome, month, pdays, job and contact seem to be the most important.

There were three decision pathways that lead to the classification “yes”: if the outcome of the previous marketing campaign (poutcome) was success and the days since the previous campaign (pdays) was either less than 94.5, or if the previous days were equal to or more than 94.5 days and the job was housemaid, management, retired, self-employed, student, unemployed or unknown the classification was “yes”. The third pathway that lead to the classification “yes” was if the outcome of the previous marketing campaign (poutcome) was failure, other or unknown, the month was any of January, May, July, August or November and contact was with cellular or telephone and the previous days were equal to or more than 383.5.

2.4. Estimate the confusion matrix, accuracy and F1 score for the optimal model

The confusion matrix, accuracy and F1 score were calculated for the test data by using the optimal model from step 3 (model m2c). The accuracy is the fraction or percentage of correctly classified observations, while F1 score is defined as:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

where precision is the proportion (or percentage) of the positive identifications that was actually correct and recall is defined as the proportion (or percentage) of the actual positives that was identified correctly.

Tables 5-7 below show the different metrics.

Table 5: Error rates for model m2c (optimal) model.

err_train	err_valid	err_test
10.4	11.2	10.9

Table 6: Confusion matrix for model m2c model, test dataset.

	no	yes	%
no	11872	107	99.1
yes	1371	214	13.5

Table 7: Accuracy and F1 score for model m2c for test data.

Accuracy	F1
89.1	22.5

In terms of the total percentage of observations that gets correctly predicted, the performance could be considered quite good with 10.9% misclassification error, or 89.1% accuracy, for the test data. However, the prediction rate for the positive class is only 13.5%, which cannot be considered good. The performace of the model is better reflected by the F1 score, which has a value of 22.3.

2.5 Perform a decision tree classification with loss matrix

A decision tree classification using model m2c was performed with a loss matrix, that weighted the positive class 5:1 to the negative class. The confusion matrix and the accuracy and F1 score are shown in tables 8-9 below.

Table 8: Confusion matrix for the loss matrix classification.

	no	yes	%
no	11030	949	92.1
yes	771	814	51.4

Table 9: Accuracy and F1 score for model m2c for test data using loss matrix.

Accuracy	F1
87.3	48.6

The prediction rate has now increased for the positive class, 51.45% compared to 13.5% for the model without the loss matrix, and decreased somewhat for the negative class, 92% compared to 99.1% for the previous model. This is because the loss matrix shifts the r value so that more observations are classified as postive, The accuracy is now 87.3% and the F1 score is 48.6.

2.6 ROC curves for optimal tree and logistic regression

ROC curves were calculated for the optimal tree (model m2c) and a logistic regression model (model 3a). Precision and recall were also calculated and plotted. Confusion matrix and accuracy and F1 for the default r value (0.5) were also calculated for comparison, see tables 10 and 11. The prediction rate for the positive class was a few percentage points higher for the logistic regression model (16.5%) compared to the optimal tree (13.5%).

Table 10: Confusion matrix for logistic regression model (m3a).

	no	yes	%
no	11837	142	98.8
yes	1323	262	16.5

Table 11: Accuracy and F1 score for the logistic regression model (m3a).

Accuracy	F1
89.2	26.3

Figure 2a. ROC

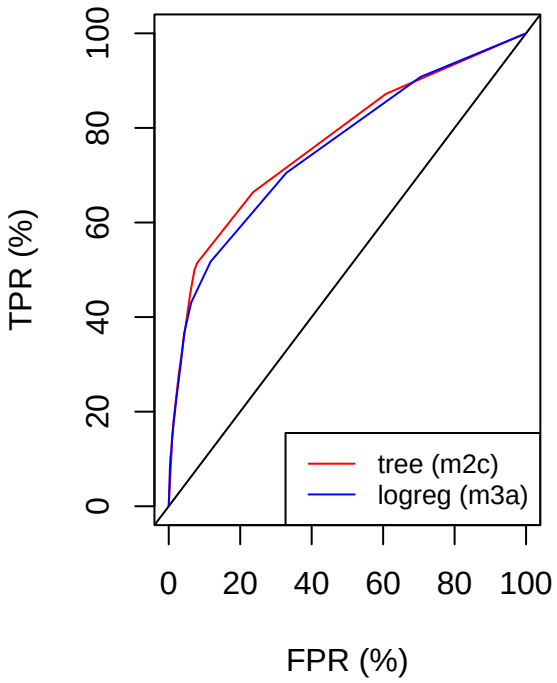
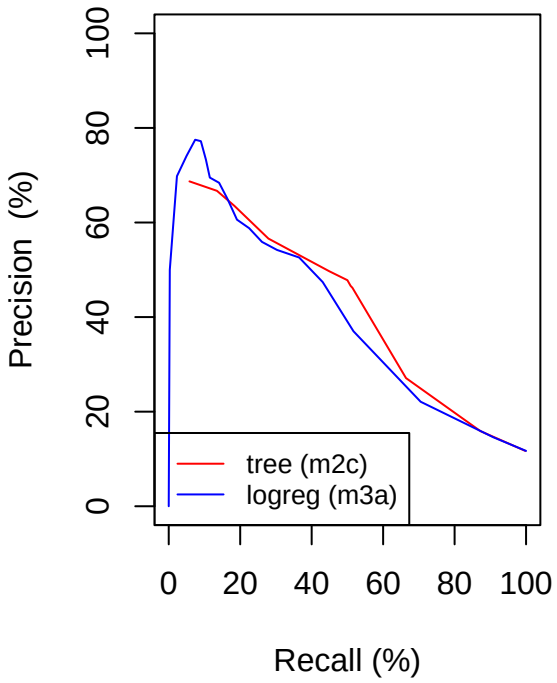


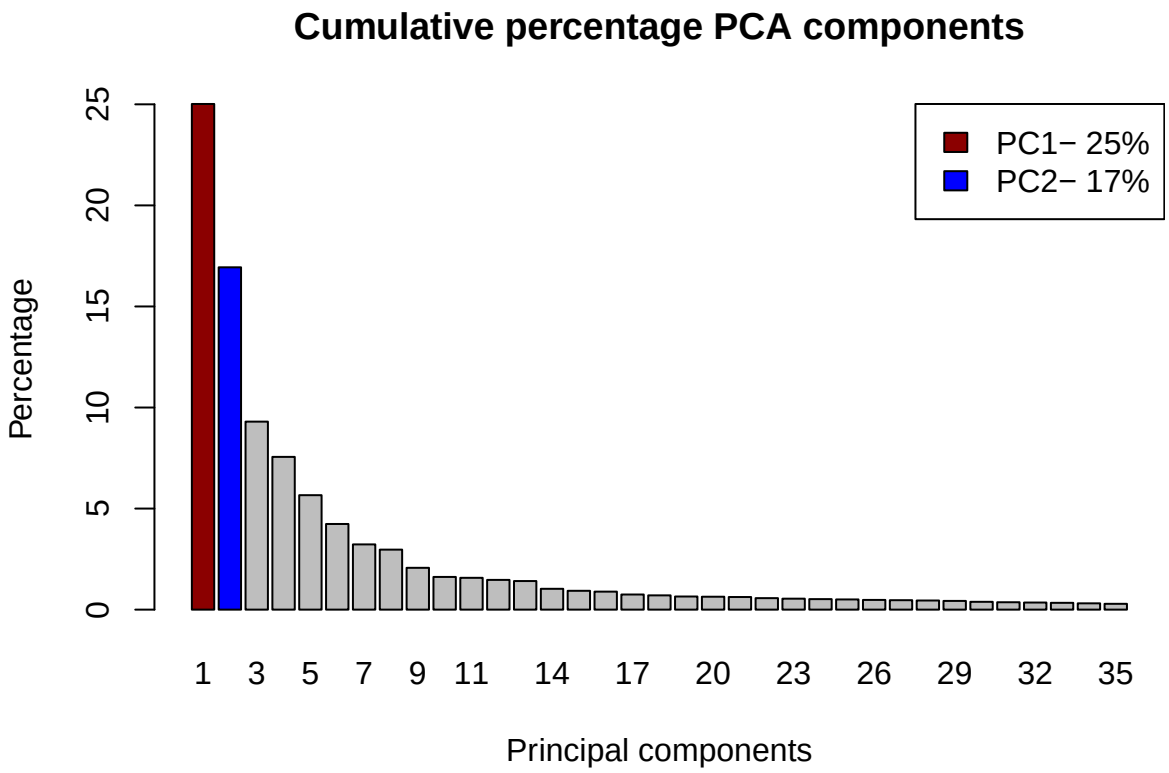
Figure 2b. Precision−Recall



The ROC and precision-recall curves are quite similar for the optimal tree and the logistic regression model, however the decision tree is somewhat better over some ranges of the performance metrics. From the ROC curve, it can be seen that to obtain around 80% true positive rate, almost 60% false positive rate has to be accepted. From the precision-recall curve, it can be seen that when an 80% rate of the true positives/recall is selected, only around 20% of the selected observations will be true positives. The precision-recall curve gives a better idea of the benefits versus the disadvanatges with selecting a certain recall/ true positive rate (or rather, selecting an r value corrsponding to a certain recall).

Assignment 3

3.1. **35 principal components** are needed to obtain at least **95% of variance** in the data. PC1 and PC2 have 25% and 17% of representation respectively. The plot below shows the individual contribution of the 35 principal components to the 95% representation of variance. The combination of PC1 and PC2 represents almost half of it with 42%.



3.2. The 5 features which contribute the most to the PC1 are :

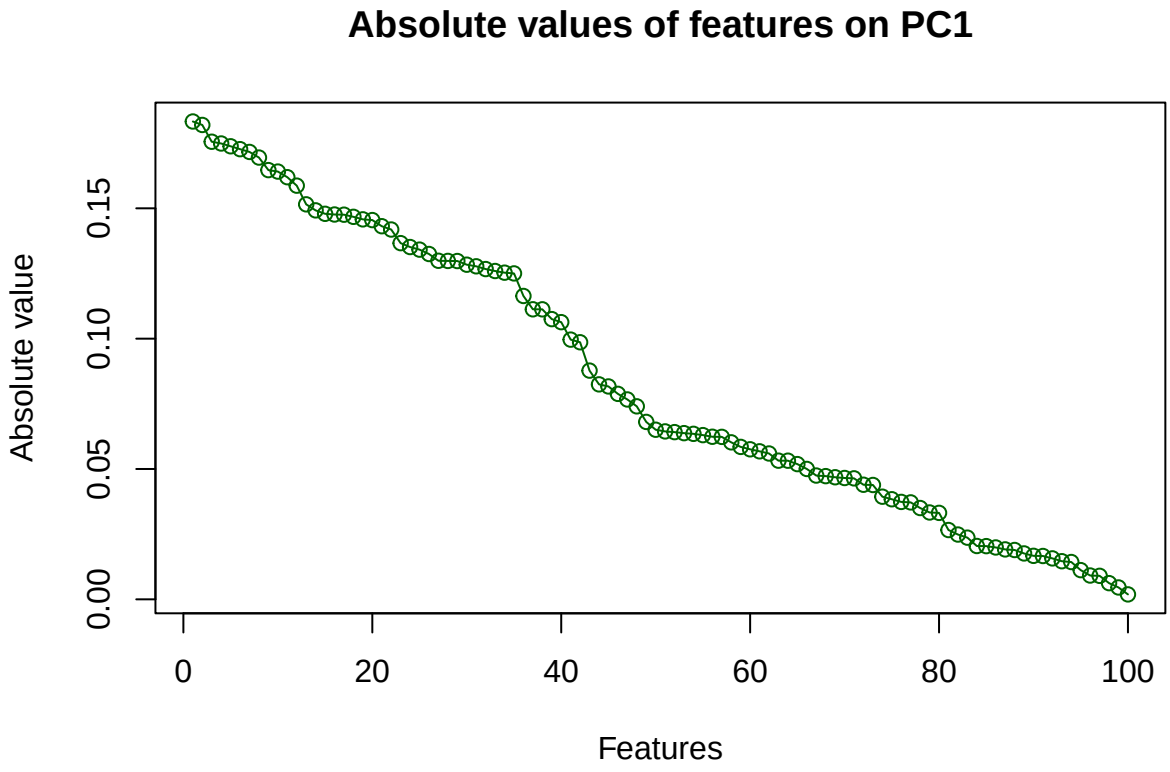
- medFamInc: median family income
- medIncome: median household income
- PctKids2Par: percentage of kids in family housing with two parents
- pctWInvInc: percentage of households with investment / rent income in 1989
- PctPopUnderPov: percentage of people under the poverty level

These features have logical relationship to the crime level, because all are metrics of the economic context so this could also have correlation against one another. The median family income and median household income have an impact on the percentage of people under the poverty level. In addition, the percentage of kids in family housing with two parents, could be related to the median family income and median household income, because it could show the number of adults that work or that make an economic contribution to the family income. Furthermore, PctKids2Par could show if there are adults supervising the kids, and this could decreased the possibilities of the kids to fall into crime.

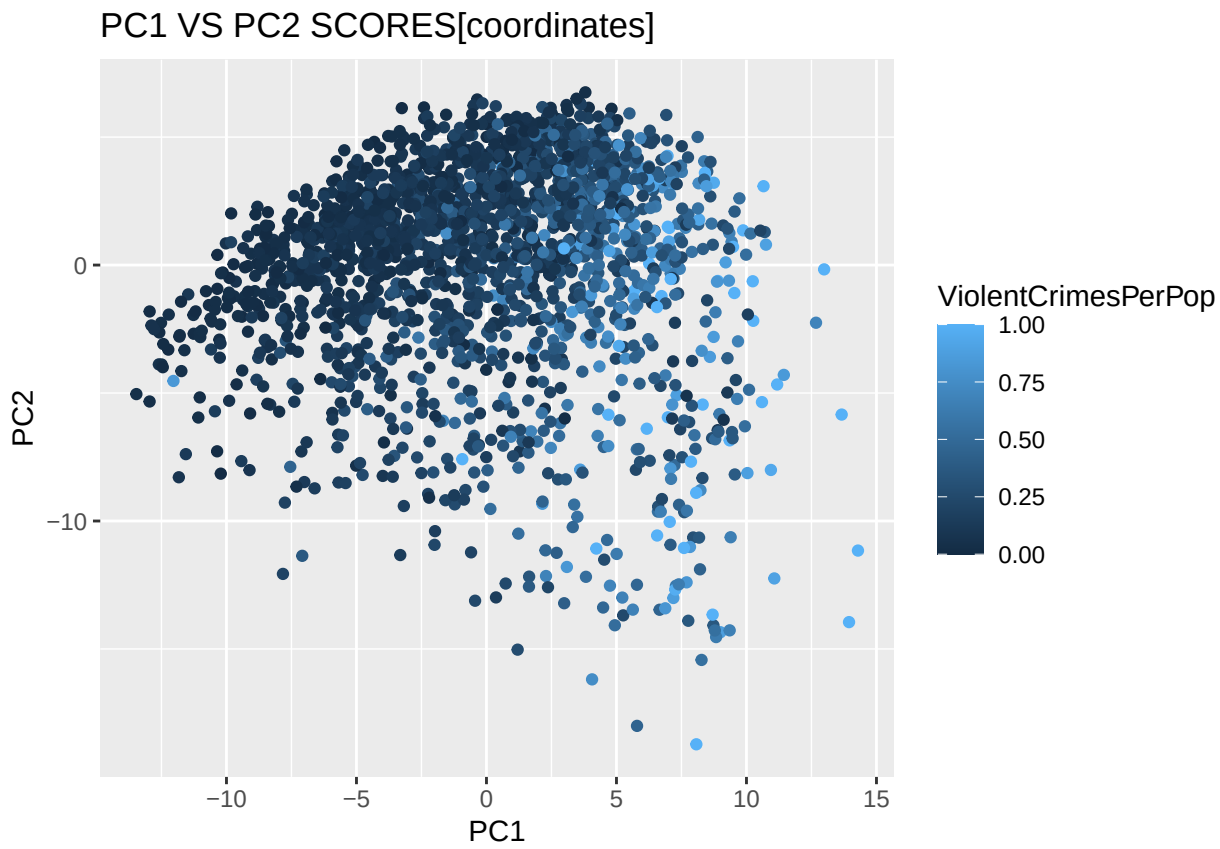
For the Trace plot we took the eigen vectors (*loadings of the princomp() function*), and calculated the absolute value, sort it in a decreasing order. Also, we used this for the 5 features which contribute the most or the 5 with highest magnitude.

From the traceplot it seems that quite a lot of features contribute to PC1, there is more less a gradual scale from the lowest (absolute) value. E.g. there are around 40 features that have a value half of the maximum value or higher.

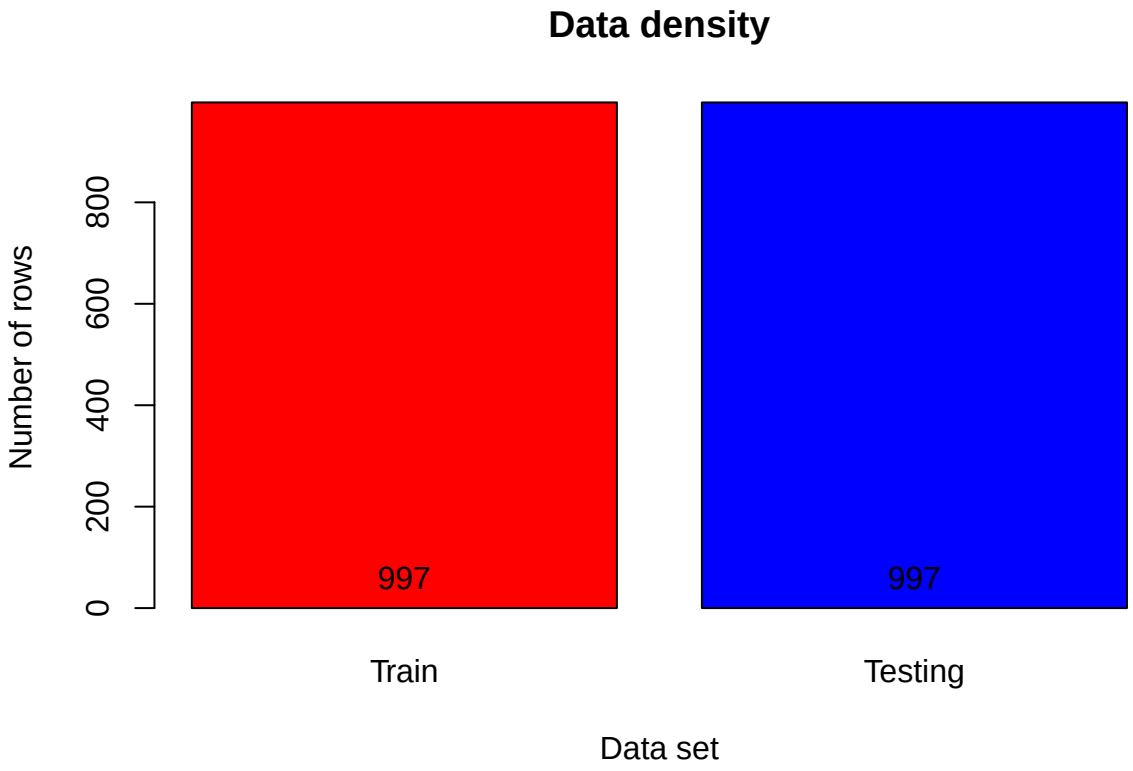
```
#> [1] "5 features which contribute the most for the PC1"

#>      medFamInc      medIncome    PctKids2Par     pctWInvInc PctPopUnderPov
#>        0.18331        0.18198        0.17554        0.17487        0.17380
```



The PC1 vs PC2 scores plot, shows that there is a trend along the mainly first PC according to Violent crimes per 100k people.


PC1 VS PC2 SCORES[coordinates]
PC2
0
−10
−10
−5
0
5
10
15
PC1
ViolentCrimesPerPop
1.00
0.75
0.50
0.25
0.00

3.3



3.3.1 Compute Linear regression model

Comparing the MSE for the training and test set indicates that the model may not be underfitting to a great extent, as the MSE for the test set is somewhat larger, but also not too much overfitting, as the test MSE is not a great deal larger. To obtain a better understanding, we calculated the coefficient of determination for the training and the test set (for the test set we followed the procedure in lecture 2d). For the test was 0.69(69%) and for the training 0.72(72%). The difference between the values of this parameter is not big as for the MSE. Based on these values, we could say that the model seems reasonably good (but it also depends on the application).

The difference again between both parameters is not big as for the MSE. To know if the quality of the model is very good we should have more models to compare with.

```
#> mse_training  mse_testing
#>         0.28         0.42
```

```
#> Coefficient determination Train
```

```
#> [1] 0.72
```

```
#> Coefficient determination Test
```

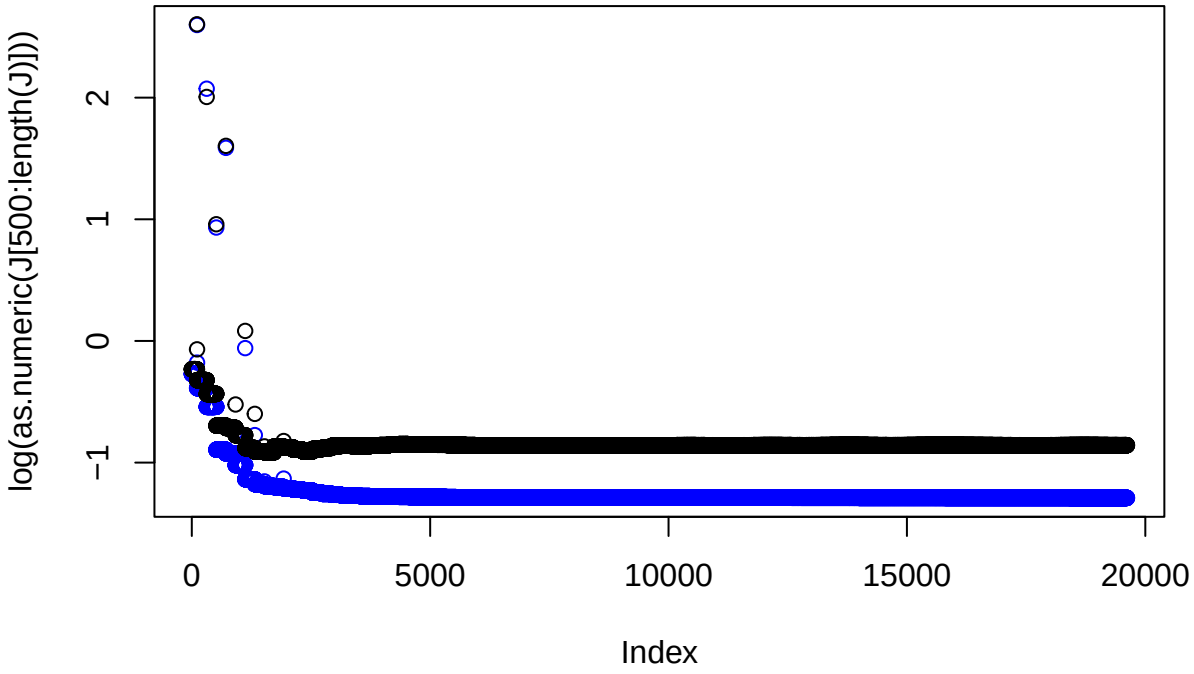
```
#> [1] 0.69
```

3.4. The cost function of the linear regression was implemented

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{\theta}^T \boldsymbol{x_i} - y_i)^2$$

where n is the number of observations.

The values of the MSE for training and test using two different methods, lm() package and optimizing the cost function of the linear regression, are quite similar, however the MSE from test using optim() is slightly smaller, 0.40 compared to 0.42, which indicates that this is a better model. Using optim() required early stopping at iteration number 2182 to obtain the optimal or lowest MSE test.




```
#> Iteration with lowest MSE on test

#> [1] 2182

#> MSE training

#> [1] 0.3033

#> MSE test

#> [1] 0.40023
```

Statement of Contribution

Assignment 1 was contributed by Tugce Izci. Assignment 2 was contributed by Kerstin Nilsson. Assignment 3 was contributed by Alan Cacique.

All three assignments procedures and results were reviewed and discussed before the creation of the final report.

Appendix: All code for this report

```
knitr::opts_chunk$set(comment = "#>",
                      echo = FALSE)
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 1
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

#-------------------------------------------------------------------------------#
#- Set libraries
#-------------------------------------------------------------------------------#
library(tidyverse)
library(caret)
library(dplyr)
library(ggplot2)
library(glmnet)
#-------------------------------------------------------------------------------#
#- 1) Read in data and divide into training, validation and test
#-------------------------------------------------------------------------------#
dir_root <- "C:/Users/kerstin/Documents/LiU/ML/Labs/MLab02_devel"
dir_data <- paste0(dir_root,"/Data/")

file_in <- "tecator.csv"
path_in <- paste0(dir_data,file_in)
data_in <- read.csv(path_in)


#data_in <- read.csv("tecator.csv")
#data_in

#Exclude variables sample and moisture, protein

df <- data_in %>% dplyr::select(-Sample,-Moisture, -Protein)

summary(df)
#-------------------------------------------------------------------------------#
#-2) Randomly order the data set -Divide training and test set 50%
#-------------------------------------------------------------------------------#
set.seed(12345)
n <- nrow(df)
id <- sample(1:n, floor(n*0.5))
train <- df[id, ]
test <- df[-id, ]
#-------------------------------------------------------------------------------#
#-3) Compute a linear regression model from the training data, estimate training and test RMSE
#-------------------------------------------------------------------------------#
#- Linear model
formula <- Fat ~ .
data <- train
m1 <- lm(formula=formula, data=data)
coef(m1)
```

```r
summary(m1)
#- Calculate training and test MSE
error_training <- train$Fat  - predict(m1)
error_test <- test$Fat  - predict(m1, test)

mse_training <- mean(error_training^2)
mse_test <- mean(error_test^2)

#- RMSE error (root mean square error)
rmse_training <- sqrt(mse_training)
rmse_test <- sqrt(mse_test)
rmse <- c(rmse_training = rmse_training, rmse_test = rmse_test)
#- Print rmse for training and test set
(rmse)



#-------------------------------------------------------------------------------#
# 4) Compute LASSO regression, all Channels are used as features. Report cost function
#-------------------------------------------------------------------------------#

# Lasso Regression

covariates <- as.matrix(dplyr::select(train,-Fat))
response <- train$Fat

l_fit <- glmnet(x = covariates,y = response, alpha=1, family="gaussian")
print(l_fit)


#-------------------------------------------------------------------------------#
# 5) Fit LASSO regression to training data, and present a plot
#-------------------------------------------------------------------------------#

# Fitting into training data & plotting

plot(l_fit, xvar = "lambda", sub = "Lasso Regression",  font.sub=2, label = TRUE)

options(digits=5)
coef(l_fit)


#Take 3 features

l_fit_print <- print(l_fit)
l_fit_3vars <- l_fit_print[l_fit_print$Df==3,]
ind_max <- which.max(l_fit_3vars$`%Dev`)
l_fit_3vars
l_fit_3vars[ind_max,3]
#-------------------------------------------------------------------------------#
# 6) Fit RIDGE regression to training data, and present a plot
#-------------------------------------------------------------------------------#
```

```
r_fit <- glmnet(x = covariates,y =response, alpha=0, family="gaussian")

plot(r_fit, sub= "Ridge Regression", xvar = "lambda", font.sub=2, label = TRUE)


#-------------------------------------------------------------------------------#
# 6) Cross Validation
#-------------------------------------------------------------------------------#

#Lasso cross validation
l_cvfit<-cv.glmnet(x = covariates, y = response, alpha=1, family="gaussian")
#Plotting cross validated lasso
par(mfrow=c(1,2))
plot(l_cvfit)
plot(l_cvfit, xlim = c(-4,-2) , ylim=c(0,20))

#Optimal lambda value
l_cvfit$lambda.min

#Log(lambda) value
log(l_cvfit$lambda.min)

coef(l_cvfit, s = "lambda.min")

coef_lambda_min <- coef(l_cvfit, s = "lambda.min")

num_coef_lambda_min <- length(which(abs(coef_lambda_min)>0))-1
#cat("Number of variables that was chosen in the model with the optimal lambda:")
#(num_coef_lambda_min)
l_cvfit


coef_lambda_min <- coef(l_cvfit, s = l_cvfit$lambda.min)


covariates_test <- as.matrix(dplyr::select(test,-Fat))
response_test <- test$Fat


y_pred_train_min_lambda <-  coef_lambda_min[1] + t(as.matrix(coef_lambda_min[-1]))%*%t(as.matrix(covaria



y_pred_test_min_lambda <-  coef_lambda_min[1] + t(as.matrix(coef_lambda_min[-1]))%*%t(as.matrix(covariat


par(mfrow=c(1,2))

plot(response, y_pred_train_min_lambda,
     xlab = "Train Response", ylab ="Predicted y Train" ,
     col= y_pred_train_min_lambda)

abline(0,1)
```

```r
plot(response_test, y_pred_test_min_lambda,
     xlab = "Test Response", ylab ="Predicted y Test" ,
     col= y_pred_test_min_lambda)

abline(0,1)

#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 2
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Set libraries
#-------------------------------------------------------------------------------#
#library(dplyr)
library(tidyverse)
library(tree)
library(ggplot2)

#-------------------------------------------------------------------------------#
#- Functions
#-------------------------------------------------------------------------------#

#- Calculate misclassification rate

# model - model object, can be tree or glm object
# formula - formula object
# data - data which to calculate misclassification rate

# returns list with misclassification rate and confusion matrix

misclass_rate <- function(model, formula, data) {

  y <- unlist(as.vector(data[,all.vars(formula)[1]]))
  levels_y <- levels(y)
  if (class(model)[1]=="tree"){
    y_pred <- predict(model, newdata=data, type="class")
  }
  if (class(model)[1] == "glm"){
    y_prob <- predict(model, newdata=data, type="response")
    y_pred <- ifelse(y_prob>0.5, levels_y[2], levels_y[1])
  }

  y_pred <- factor(y_pred, levels=levels_y)

  cm <- table(y,y_pred)
  err <- ((sum(cm)-sum(diag(cm))) /sum(cm))

  perc_class <- round(100*(diag(cm)/rowSums(cm)),1)
  cm_2 <- cbind(cm, perc_class)
  colnames(cm_2)[3] <- "`%`"
```

```
  res <- list(err=err,cm=cm_2)
  return(res)
}

#- Function to calculate classification metrics
#- Note: assumes cm[1,1] is TP and cm[2,2] is TN
#- cm - confusion matrix
#- returns vector with the following metrics:
#  TPR - true positive rate
#  TNR - true negative rate
#  FPR - false positive rate
#  FNR - false negative rate
#  rec - recall
#  prec - precision
#  acc - accuracy
#  f1 - F1 score
classifier_metrics <- function(cm) {

  TN <- cm[1,1]
  FP <- cm[1,2]
  FN <- cm[2,1]
  TP <- cm[2,2]

  N <- sum(cm[1,1:2])
  P <- sum(cm[2,1:2])

  #- Basic
  TPR <- TP/P
  TNR <- TN/N
  FPR <- FP/N
  FNR <- FN/P

  #- Accuracy: (TP+FP)/(P+N)
  acc <- (TP+TN)/(P+N)

  #- F1 score: (2*prec*rec)/(prec+rec)
  rec <- TP/P
  prec <- TP/(TP+FP)

  f1 <- (2*prec*rec)/(prec+rec)

  measures <- data.frame(TPR=TPR,TNR=TNR,FPR=FPR,FNR=FNR,rec=rec,prec=prec,acc=acc,f1=f1)
  measures <- (round(100*measures,1))

  return(measures)
}
#- function calculate_roc - calculates roc and precision & recall over different
# prob cutoffs (r)

# prob - vector of predicted probabilities
# formula - formula for the model
# data - data (contains target)
# returns list with prob cutoff (pi), TPR_v - true positive rate,
```

```r
# FPR_v - false positive rate, rec_v - recall, prec_v - precision
calculate_roc <- function(prob, formula, data) {
  y <- unlist(as.vector(data[,all.vars(formula)[1]]))
  levels_y <- levels(y)
  pi <- seq(from=0,to=0.95,by=0.05)
  TPR_v <- numeric(length(pi))
  FPR_v <- numeric(length(pi))
  rec_v <- numeric(length(pi))
  prec_v <- numeric(length(pi))

  for (i in 1:length(pi)) {
    y_pred_i <- ifelse(prob>pi[i], levels_y[2], levels_y[1])
    y_pred_i <- factor(y_pred_i,levels=levels_y)
    cm <- table(y,y_pred_i)
    cl_m <- classifier_metrics(cm)
    TPR_v[i] <- cl_m$TPR
    FPR_v[i] <- cl_m$FPR

    rec_v[i] <- cl_m$rec
    prec_v[i] <- cl_m$prec
  }
  res <- list(pi=pi,TPR_v=TPR_v,FPR_v=FPR_v,rec_v=rec_v,prec_v=prec_v)
  return(res)
}
#------------------------------------------------------------------------------#
#- 1 Read data and data management:
#    a) Read in data
#    b) divide into training and test set
#------------------------------------------------------------------------------#

# a) Read in data
dir_root <- "C:/Users/kerstin/Documents/LiU/ML/Labs/MLab02_devel"
dir_data <- paste0(dir_root,"/Data/")

file_in <- "bank-full.csv"
path_in <- paste0(dir_data,file_in)

df_in <- read_csv2(path_in)
#df_in <- read.csv2(path_in)

dim(df_in)
#=> 45211x17

#spec(df_in)

names_df_in <- data.frame(names=names(df_in))

#- Exclude variables that are not to be used

df <- df_in %>% dplyr::select(-duration) %>%
  dplyr::mutate_if(is.character, as.factor)

#- Overview of data
```

```r
str(df)

#- Summary of numerical variables
df_num <- dplyr::select(df,age,balance,day,campaign,pdays,previous)
summary(df_num)

#- Frequency tables of variables
df %>% count(age) %>% mutate(perc = 100*prop.table(n)) %>% print(n = Inf)
df %>% count(pdays) %>% mutate(perc = 100*prop.table(n))
df %>% count(job) %>% mutate(perc = 100*prop.table(n))
df %>% count(marital) %>% mutate(perc = 100*prop.table(n))
df %>% count(education) %>% mutate(perc = 100*prop.table(n))
df %>% count(default) %>% mutate(perc = 100*prop.table(n))
df %>% count(housing) %>% mutate(perc = 100*prop.table(n))
df %>% count(loan) %>% mutate(perc = 100*prop.table(n))
df %>% count(contact) %>% mutate(perc = 100*prop.table(n))
df %>% count(month) %>% mutate(perc = 100*prop.table(n))
df %>% count(campaign) %>% mutate(perc = 100*prop.table(n))%>% print(n = Inf)
df %>% count(poutcome) %>% mutate(perc = 100*prop.table(n))%>% print(n = Inf)

tab_y <- df %>% count(y) %>% mutate(`%` = round(100*prop.table(n),1))

knitr::kable(
  tab_y,
  caption = "Frequency table of target variable."
)
#- Extract levels of y
levels_y <- levels(df$y)


#- b) Divide into training, validation and test

set.seed(12345)

n <- dim(df)[1]

id <- sample(1:n, floor(n*0.4))
train <- df[id,]

id1 <- setdiff(1:n, id)

set.seed(12345)
id2 <- sample(id1, floor(n*0.3))
valid <- df[id2,]

id3 <- setdiff(id1,id2)
test <- df[id3,]


#-------------------------------------------------------------------------------#
#- 2 Fit decision trees
#    a) DT with default settings
#    b) DT with smallest allowed node size equal to 7000
```

```
#    c) DT with minimum deviance 0.0005
#-------------------------------------------------------------------------------#

#- 2a) Fit models
formula <- y ~ .
data <- train

nobs <- nrow(data)

#- Default settings
# tree.control(nobs, mincut = 5, minsize = 10, mindev = 0.01)
# where
# nobs - The number of observations in the training set.
# mincut - The minimum number of observations to include in either child node.
#          This is a weighted quantity; the observational weights are used to
#          compute the 'number'. The default is 5.
# minsize - The smallest allowed node size: a weighted quantity. The default is 10.
# mindev - The within-node deviance must be at least this times that of the root
#           node for the node to be split.

m1a <- tree(formula=formula, data=data)

#- Smallest allowed node size = 7000
m1b <- tree(formula=formula, data=data, control=tree.control(nobs=nobs,
                                                             minsize=7000) )
#- Minimum deviance = 0.0005
#- this means min node deviance is 12850.45*0.0005 = 6.42522
#- default min deviance is 128.5045
m1c <- tree(formula=formula, data=data, control=tree.control(nobs=nobs,
                                                             mindev=0.0005) )

#- 2b) Calculate misclassification rate
res_train_m1a <- misclass_rate(m1a, formula, train)
res_valid_m1a <- misclass_rate(m1a, formula, valid)

res_train_m1b <- misclass_rate(m1b, formula, train)
res_valid_m1b <- misclass_rate(m1b, formula, valid)

res_train_m1c <- misclass_rate(m1c, formula, train)
res_valid_m1c <- misclass_rate(m1c, formula, valid)

#- Compile results
err_train <- c(res_train_m1a$err,res_train_m1b$err,
               res_train_m1c$err)

err_valid <- c(res_valid_m1a$err,res_valid_m1b$err,
               res_valid_m1c$err)

err_train <- round(100*err_train,2)
err_valid <- round(100*err_valid,2)

df_err <- data.frame(model=c("m1a","m1b","m1c"), err_train=err_train,
                     err_valid= err_valid)
```

```
knitr::kable(
  df_err,
  caption = "Misclassification rates (%) for the three models.",
  longtable = FALSE
)

#- Confusion matrices)
models <- c("m1a","","m1b","","m1c","")
target <- rep(c("no","yes"),3)
cm_train_m1abc <- rbind(res_train_m1a$cm,res_train_m1b$cm,res_train_m1c$cm)
cm_train_m1abc_2 <- cbind(model=models,` `=target,cm_train_m1abc)
rownames(cm_train_m1abc_2) <- c()

cm_valid_m1abc <- rbind(res_valid_m1a$cm,res_valid_m1b$cm,res_valid_m1c$cm)
cm_valid_m1abc_2 <- cbind(model=models,` `=target,cm_valid_m1abc)
rownames(cm_valid_m1abc_2) <- c()

#cm_train_tab <- knitr::kable(cm_train_m1abc_2,
#             caption= "Confusion matrices for the three models for the training dataset")

knitr::kable(cm_valid_m1abc_2,
             caption= "Confusion matrices for the three models for the validation dataset.")

# knitr::kable(
#   list(cm_train_m1abc, cm_valid_m1abc),
#   caption = "Confusion matrices for the three models for training (left) and validation (right).")

#- 2c) Inspect the trees

summary_m1a <- summary(m1a)
summary_m1b <- summary(m1b)
summary_m1c <- summary(m1c)

#- Size
size_m1a <- summary_m1a$size
size_m1b <- summary_m1b$size
size_m1c <- summary_m1c$size

#- Deviance
dev_m1a <- summary_m1a$dev
dev_m1b <- summary_m1b$dev
dev_m1c <- summary_m1c$dev

size <- c(m1a=size_m1a,m1b=size_m1b,m1c=size_m1c)

deviance <- round(c(m1a=dev_m1a,m1b=dev_m1b,m1c=dev_m1c),1)

#- Extract information from the internal nodes
m1a_frame <- m1a$frame
m1b_frame <- m1b$frame
m1c_frame <- m1c$frame

m1a_frame_int <- dplyr::filter(m1a_frame,var != "<leaf>")
```

```
m1b_frame_int <- dplyr::filter(m1b_frame,var != "<leaf>")
m1c_frame_int <- dplyr::filter(m1c_frame,var != "<leaf>")

#- min sizes
m1a_min_n <- min(m1a_frame_int$n)
m1b_min_n <- min(m1b_frame_int$n)
m1c_min_n <- min(m1c_frame_int$n)

min_int_size <- c(m1a=m1a_min_n, m1b=m1b_min_n, m1c=m1c_min_n)
#deviance of root node: 12850.000

#- min deviances
m1a_min_dev <- min(m1a_frame_int$dev)
m1b_min_dev <- min(m1b_frame_int$dev)
m1c_min_dev <- min(m1c_frame_int$dev)

min_int_dev <- round(c(m1a=m1a_min_dev, m1b=m1b_min_dev, m1c=m1c_min_dev),1)

models <- c("m1a","m1b","m1c")

summary_m1 <- data.frame(model=models,size=size,dev=deviance,
                         min_int_size=min_int_size,
                         min_int_dev=min_int_dev)

rownames(summary_m1) <- c()
names(summary_m1) <- c("Model","Tree size","Deviance","Min int. node size",
                       "Min int. node deviance")
knitr::kable(summary_m1,caption="Tree size and deviance, and minimum internal node size and deviance.")

#- Plots and summaries of the trees
plot(m1a)
text(m1a,pretty=0)
m1a
summary(m1a)

plot(m1b)
text(m1b,pretty=0)
m1b
summary(m1b)

plot(m1c)
text(m1c,pretty=0)
m1c
summary(m1c)
#-----------------------------------------------------------------------------#
#- 3) Choose optimal tree depth
#-----------------------------------------------------------------------------#

#- Prune tree m1c to sizes 2 to 50
train_score=rep(0,49)
valid_score=rep(0,49)

for(i in 2:50) {
```

```
  pruned_tree <- prune.tree(m1c, best=i)
  pred_valid <- predict(pruned_tree, newdata=valid, type="tree")

  train_score[i] <- deviance(pruned_tree)/nrow(train)
  valid_score[i] <- deviance(pred_valid)/nrow(valid)
}

#- 3.1: graph
plot(2:50, train_score[2:50], type="b", col="red", xlab = "Number of leaves",
     ylab = "Normalised deviance score",
     main=" Figure 1. Deviances for the training and the validation data vs number of leaves")
points(2:50, valid_score[2:50], type="b", col="blue")
legend("topright", legend=c("training", "validation"), col=c("red", "blue"), pch=1:1, cex=0.8)

#- Compile results into data frame
res <- data.frame(size=2:50,train=train_score[2:50],valid=valid_score[2:50])

#- Identify optimal amount of leaves
cat("Optimal amount of leaves")
ind <- which(valid_score[2:50]==min(valid_score[2:50]))
res$size[ind][1]
#=> 22

#- Model for selected tree
m2c <- prune.tree(m1c, best=res$size[ind][1])

#- Error rate
res_train_m2c <- misclass_rate(m2c, formula, train)
res_valid_m2c <- misclass_rate(m2c, formula, valid)

#- Compile results
df_err_m2c <- data.frame(err_train=res_train_m2c$err,
                         err_valid=res_valid_m2c$err)

#- Detailed inspection
m2c
plot(m2c)
text(m2c, pretty=0,cex=0.6)
summary(m2c)
#- Extract frame
m2c_frame <- data.frame(node=rownames(m2c$frame),m2c$frame)

m2c_frame_leaf_yes <- dplyr::filter(m2c_frame,var == "<leaf>" & yval=="yes")
#=> nodes 37, 6 and 15

#=> compare with model 1a
#-------------------------------------------------------------------------------#
#- 4) Estimate confusion matrix, accuracy and F1 score for test data
#-------------------------------------------------------------------------------#

#- Misclassification rates and confusion matrices
res_train_m2c <- misclass_rate(m2c, formula, train)
res_valid_m2c <- misclass_rate(m2c, formula, valid)
```

```
res_test_m2c <- misclass_rate(m2c, formula, test)

#- Compile results
df_err_m2c <- data.frame(err_train=res_train_m2c$err,
                         err_valid=res_valid_m2c$err,
                         err_test=res_test_m2c$err)

df_err_m2c <- round(100*df_err_m2c,1)

knitr::kable(df_err_m2c,caption="Error rates for model m2c (optimal) model.")

#- Confusion matrices
knitr::kable(res_test_m2c$cm,caption="Confusion matrix for model m2c model,
             test dataset.")

#- Calculate classification metrics
#- Accuracy: (TP+FP)/(P+N)
#- F1 score: (2*prec*rec)/(prec+rec)

cm_test_m2c <- res_test_m2c$cm

classifier_metrics_test_m2c <- classifier_metrics(cm_test_m2c)

acc_test_m2c <-classifier_metrics_test_m2c$acc
f1_test_m2c <- classifier_metrics_test_m2c$f1

acc_f1 <- data.frame(Accuracy=acc_test_m2c,F1=f1_test_m2c)

knitr::kable(acc_f1,caption="Accuracy and F1 score for model m2c for test data.")

#-------------------------------------------------------------------------------#
# 5) Perform a decision tree classification of the test data with loss matrix
#-------------------------------------------------------------------------------#

#- Obtain probabilities for the test set
prob_test_m2c <- predict(m2c, newdata=test)

prob_test_m2c_n <- prob_test_m2c[,1]
prob_test_m2c_y <- prob_test_m2c[,2]

#- Apply loss matrix
y_pred_test_m2c_l <- ifelse(prob_test_m2c_y/prob_test_m2c_n>1/5, levels_y[2],
                            levels_y[1])
y_pred_test_m2c_l <- factor(y_pred_test_m2c_l,levels=levels_y)

#- Calculate confusion matrix
cm_test_m2c_l <- table(test$y,y_pred_test_m2c_l)

#- Calculate misclassification rate
perc_class <- round(100*(diag(cm_test_m2c_l)/rowSums(cm_test_m2c_l)),1)
  cm_test_m2c_l_2 <- cbind(cm_test_m2c_l, perc_class)
  colnames(cm_test_m2c_l_2)[3] <- "`%`"
```

```
knitr::kable(cm_test_m2c_l_2, caption="Confusion matrix for the loss matrix classification.")

#- Calcualte metrics including accuracy and F1 score
classifier_metrics_test_m2c_l <- classifier_metrics(cm_test_m2c_l)

acc_test_m2c_l <-classifier_metrics_test_m2c_l$acc
f1_test_m2c_l <- classifier_metrics_test_m2c_l$f1

#- Print table
acc_f1_l <- data.frame(Accuracy=acc_test_m2c_l,F1=f1_test_m2c_l)
knitr::kable(acc_f1_l,caption="Accuracy and F1 score for model m2c for test data using loss matrix.")


#-------------------------------------------------------------------------------#
# 6) Optimal tree combined compared with logistic regression
#-------------------------------------------------------------------------------#

#- Log reg model
m3a <- glm(formula, train, family = "binomial")
prob_test_m3a_y <- predict(m3a, newdata=test, type="response")

#- Calculate misclassification rate and confusion matrices for training and
#  test data
res_train_m3a <- misclass_rate(m3a, formula, train)
res_test_m3a <- misclass_rate(m3a, formula, test)

df_err_m3a <- round(100*data.frame(err_train=res_train_m3a$err,err_test=res_test_m3a$err),1)

#knitr::kable(df_err_m3a, caption="Missclassification rate #(%) for logistic regression model")

knitr::kable(res_test_m3a$cm, caption="Confusion matrix for logistic regression model (m3a).")

#- Calculate classification metrics accuracy and F1 score for logistic
#  regression model
classifier_metrics_test_m3a <- classifier_metrics(res_test_m3a$cm)

acc_test_m3a <-classifier_metrics_test_m3a$acc
f1_test_m3a <- classifier_metrics_test_m3a$f1

acc_f1_m3a <- data.frame(Accuracy=acc_test_m3a,F1=f1_test_m3a)

#- Print table
knitr::kable(acc_f1_m3a, caption="Accuracy and F1 score for the logistic regression model (m3a).")

#- Calculate ROC data for optimal tree (m2c) and logistic regression (m3a)
#  models
roc_m2c <- calculate_roc(prob_test_m2c_y, formula, test)
roc_m3a <- calculate_roc(prob_test_m3a_y, formula, test)

#- Plot ROC & precision-recall curves
par(mfrow=c(1,2))
plot(roc_m2c$FPR_v,roc_m2c$TPR_v,xlab="FPR (%)",ylab="TPR (%)", type="l", xlim=c(0,100),
     ylim=c(0,100), col="red", main="Figure 2a. ROC")
```

```
abline(a=0, b=1)
lines(roc_m3a$FPR_v,roc_m3a$TPR_v, col="blue")
legend("bottomright", legend=c("tree (m2c)", "logreg (m3a)"), col=c("red", "blue"), lty=1:1, cex=0.8)

#- Precision recall
plot(roc_m2c$rec_v,roc_m2c$prec_v,xlab="Recall (%)",ylab="Precision  (%)", type="l",
     xlim=c(0,100), ylim=c(0,100), col="red", main="Figure 2b. Precision-Recall")
lines(roc_m3a$rec_v,roc_m3a$prec_v, col="blue")
legend("bottomleft", legend=c("tree (m2c)", "logreg (m3a)"), col=c("red", "blue"), lty=1:1, cex=0.8)

#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 3
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
library(dplyr)
library(tidyverse) # read_csv
library(caret)
library(ggplot2)
library(GGally)
#Load data

#file_in <- "C:/Users/act_9/Documents/Master/Winter Semester - Ago 2022/Period 2/Machine Learning_732A9
#df <- read.csv(file = file_in, header = TRUE)

dir_root <- "C:/Users/kerstin/Documents/LiU/ML/Labs/MLab02_devel"
dir_data <- paste0(dir_root,"/Data/")

file_in <- "communities.csv"
path_in <- paste0(dir_data,file_in)

df <- read.csv(file = path_in, header = TRUE)

## Scale and center all data with out the target ViolentCrimesPerPop or column 101

scaler <- preProcess(df[-101], method =c("center", "scale") )
df_scale_x<- predict(scaler,df[-101])

# x<- as.matrix(df_scale_x)

## Calculate the covariance S matrix to feed the eigen()
# n<- ncol(x)
# S<- (1/n)*t(x)%*%(x)

## Calculate the covariance with cov formula
S1 <- cov(df_scale_x)

##  Covariance matrix S is quadratic [nxn],eigen needs a square matrix as input
#dim(S) #Is 100x100
## Eigen Values, input the covariance matrix S
e<-eigen(S1, symmetric = FALSE)

# Variances of the components
```

```
lam<- e$values


# Take the proportion
Lambdap<- (lam/length(lam))*100
#Cumulative proportion
cumulative_proportion<-cumsum((lam/length(lam))*100)
# 35 represent the 95% of the data set
represent95 <- which(cumulative_proportion >= 95 & cumulative_proportion<=95.3)

#cumulative_proportion
barplot(Lambdap[1:35], main = "Cumulative percentage PCA components",
        xlab = "Principal components", ylab = "Percentage",
        names.arg = c(1:35), col = c("darkred", "blue", "grey", "grey", "grey", "grey", "grey", "grey",
legend("topright", c("PC1- 25%","PC2- 17%"), fill = c("darkred","blue"))

#- We do not take the the scale and center data, but we activated the
#- cor parameter to TRUE to scale it and center it. We just take the data not the target

res<-princomp(df[-101], cor=TRUE)

#- Maximum value its the more representative for the PC1, PC2,PC3....
#- The loadings are the eigen vectors
#- Principal Components

eigenvectors <- res$loadings
eigenvalues <- res$sdev*res$sdev
scores<- res$scores

#- Absolute values in a decreasing order
U<-sort(abs(eigenvectors[,1]), decreasing = TRUE)


#medFamInc  medIncome   PctKids2Par    pctWInvInc  PctPopUnderPov
print("5 features which contribute the most for the PC1")
U[1:5]

plot(U, main = "Absolute values of features on PC1", xlab= "Features", ylab="Absolute value", col= "dar




options(digits = 2)
df1<- data.frame(scores[,1],scores[,2], df[,101])
colnames(df1)<- c("PC1", "PC2", "ViolentCrimesPerPop")
p<- ggplot(data = df1, aes(x=df1[,1], y= df1[,2], colour= df1[,3])) +
  geom_point() +
  labs(x = "PC1",
       y = "PC2",
       colour= "ViolentCrimesPerPop",
       title = "PC1 VS PC2 SCORES[coordinates]")
p
```

```r
#- Split data
n<-nrow(df)

#- Training 50%
set.seed(12345)
id<- sample(1:n, floor(n*0.5))

trainp <-df[id,]

#- Test 50%
testp<- df[-id,]

#- Plot data distribution
nt <- nrow(trainp)
nte<- nrow(testp)

n_data<- c(Train = nt, Testing =nte)
barp<- barplot(n_data, main = "Data density", xlab = "Data set", ylab = "Number of rows",
        col = c("red", "blue"))
text(barp, 0, round(n_data, 1),cex=1,pos=3)

#- Scale data and target
scalerp<- preProcess(trainp)
trainp_sc <- predict(scalerp, trainp)
testp_sc <- predict(scalerp,testp)

#- Linear model

data_pca <- trainp_sc
linear_model <- lm(ViolentCrimesPerPop~., data = data_pca)

#- Calculate training and test MSE

error_training <- trainp_sc$ViolentCrimesPerPop - predict(linear_model)
error_testing <- testp_sc$ViolentCrimesPerPop - predict(linear_model, testp_sc)

#- MSE
mse_training <- mean(error_training^2)
mse_testing <-mean(error_testing^2)

#- RMSE
rmse_training <- sqrt(mse_training)
rmse_testing <- sqrt(mse_testing)

Mean_sqrt_error <- c(mse_training=mse_training, mse_testing= mse_testing)
Root_mean_sqrt_error <- c(rmse_training=rmse_training, rmse_testing= rmse_testing)

print(Mean_sqrt_error)
#print(Root_mean_sqrt_error)

#- Coefficient of determination Test
y <- testp_sc$ViolentCrimesPerPop
ynew <- predict(linear_model, testp_sc)
```

```
cod<- sum((ynew-mean(y))^2)/sum((y-mean(y))^2)

#- Coefficient of determination Training
y1 <- trainp_sc$ViolentCrimesPerPop
ynew1 <- predict(linear_model, trainp_sc)

cod1<- sum((ynew1-mean(y1))^2)/sum((y1-mean(y1))^2)


cat("Coefficient determination Train","\n")
cod1
cat("Coefficient determination Test", "\n")
cod

# Cost function Linear regression model

J<- list()
Jt<- list()
k <- 0
# First parameter is the first to optimize
cf_lm <- function(theta,x,y,xt,yt){
  x<- as.matrix(x)
  xt <- as.matrix(xt)

  J <- mean((x%*%theta-y)^2) # MSE training
  Jt <-mean((xt%*%theta-yt)^2) # MSE training
  # Saving the MSE for training and test to the Global environment
  .GlobalEnv$k= .GlobalEnv$k+1
  .GlobalEnv$J[[k]]=J
  .GlobalEnv$Jt[[k]]=Jt
  return(J)
}
x <- trainp_sc[,-101]
y <- trainp_sc$ViolentCrimesPerPop
xt <- testp_sc[,-101]
yt <- testp_sc$ViolentCrimesPerPop
optim(par = rep(0,ncol(x)), fn= cf_lm, x=x, y = y, xt=xt, yt=yt, method = "BFGS")


plot(log(as.numeric(J[500:length(J)])), type = "p", col="blue")
points(log(as.numeric(Jt[500:length(Jt)])), type = "p")

#Index  Training
index<-which(as.numeric(Jt)== min(as.numeric(Jt)))

#MSE Value Test
options(digits = 5)

cat("Iteration with lowest MSE on test","\n")
index

cat("MSE training","\n")
as.numeric(J[index])
```

```
cat("MSE test","\n")
as.numeric(Jt[index])
```