Machine Learning Lab Report 02

Daisuke Ronald Ssegwanyi Yamashita (ronss490)
Greeshma Jeev Koothuparambil(greko370) Johan Lundin (johlu023)
Sangeeth Sankunny Menon(sansa237)"

2023-12-05

Assignment 1. Explicit regularization

1. Assume that Fat can be modeled as a linear regression in which absorbance characteristics (Channels) are used as features. Report the underlying probabilistic model, fit the linear regression to the training data and estimate the training and test errors. Comment on the quality of fit and prediction and therefore on the quality of model.

```
library(glmnet)
library(ggplot2)

data <- read.csv("C:/Users/sange/OneDrive/Documents/tecator.csv")


set.seed(123)

train_indices <- sample(nrow(data), nrow(data) * 0.5)
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

model <- lm(Fat ~ . - Sample - Protein - Moisture, data = train_data)

# Summary of the model
summary(model)
```

```
##
## Call:
## lm(formula = Fat ~ . - Sample - Protein - Moisture, data = train_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.291326 -0.048237  0.002514  0.053190  0.212745
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.808e+01  7.733e+00  -3.631 0.010955 *
## Channel1     1.705e+04  3.631e+03   4.696 0.003339 **
## Channel2     1.306e+04  1.313e+04   0.995 0.358182
## Channel3    -1.099e+05  2.480e+04  -4.431 0.004418 **
## Channel4     1.432e+05  3.156e+04   4.537 0.003945 **
```

```
## Channel5    -3.006e+04  2.958e+04  -1.016 0.348822
## Channel6    -1.122e+05  3.001e+04  -3.740 0.009617 **
## Channel7     9.124e+04  2.240e+04   4.073 0.006550 **
## Channel8    -3.613e+04  2.323e+04  -1.555 0.170938
## Channel9    -2.604e+04  1.083e+04  -2.404 0.052982 .
## Channel10    6.652e+04  1.721e+04   3.865 0.008310 **
## Channel11    6.746e+04  2.494e+04   2.705 0.035321 *
## Channel12   -1.885e+05  4.693e+04  -4.016 0.006992 **
## Channel13    1.372e+05  4.791e+04   2.863 0.028694 *
## Channel14   -1.029e+04  4.638e+04  -0.222 0.831754
## Channel15    2.809e+04  3.901e+04   0.720 0.498646
## Channel16   -8.904e+04  1.947e+04  -4.574 0.003794 **
## Channel17    5.752e+04  1.582e+04   3.635 0.010903 *
## Channel18    7.531e+03  1.070e+04   0.704 0.507876
## Channel19   -1.042e+05  2.501e+04  -4.166 0.005903 **
## Channel20    1.938e+05  5.009e+04   3.870 0.008268 **
## Channel21   -2.875e+05  5.885e+04  -4.885 0.002753 **
## Channel22    3.439e+05  6.483e+04   5.304 0.001822 **
## Channel23   -2.710e+05  7.885e+04  -3.437 0.013848 *
## Channel24    1.515e+05  6.975e+04   2.171 0.072913 .
## Channel25   -7.109e+04  3.094e+04  -2.298 0.061265 .
## Channel26    3.102e+04  1.232e+04   2.518 0.045400 *
## Channel27   -7.414e+04  1.128e+04  -6.572 0.000595 ***
## Channel28    1.101e+05  2.071e+04   5.314 0.001806 **
## Channel29   -7.170e+04  2.713e+04  -2.643 0.038382 *
## Channel30    1.195e+05  5.645e+04   2.116 0.078691 .
## Channel31   -2.644e+05  9.057e+04  -2.920 0.026644 *
## Channel32    2.629e+05  7.468e+04   3.520 0.012516 *
## Channel33   -7.671e+04  2.884e+04  -2.660 0.037534 *
## Channel34   -4.644e+04  1.539e+04  -3.018 0.023460 *
## Channel35   -1.484e+04  1.715e+04  -0.866 0.419929
## Channel36    8.384e+04  1.950e+04   4.299 0.005101 **
## Channel37   -2.802e+04  1.736e+04  -1.614 0.157620
## Channel38   -7.844e+04  3.363e+04  -2.332 0.058467 .
## Channel39    1.403e+05  4.045e+04   3.470 0.013305 *
## Channel40   -1.113e+05  2.747e+04  -4.052 0.006712 **
## Channel41    4.747e+04  3.837e+04   1.237 0.262307
## Channel42   -1.888e+04  4.068e+04  -0.464 0.658999
## Channel43    2.981e+04  3.457e+04   0.862 0.421667
## Channel44   -6.898e+04  2.481e+04  -2.781 0.031972 *
## Channel45    9.701e+04  1.961e+04   4.948 0.002584 **
## Channel46   -4.328e+04  1.567e+04  -2.763 0.032737 *
## Channel47   -8.598e+03  1.211e+04  -0.710 0.504486
## Channel48   -3.549e+04  1.466e+04  -2.420 0.051842 .
## Channel49    5.941e+04  1.415e+04   4.200 0.005689 **
## Channel50   -5.696e+04  1.905e+04  -2.990 0.024330 *
## Channel51    8.796e+04  3.564e+04   2.468 0.048573 *
## Channel52   -8.216e+04  3.509e+04  -2.342 0.057706 .
## Channel53    3.060e+04  2.749e+04   1.113 0.308273
## Channel54   -3.251e+03  3.119e+04  -0.104 0.920383
## Channel55    2.891e+03  2.742e+04   0.105 0.919472
## Channel56    1.407e+03  1.106e+04   0.127 0.902897
## Channel57   -3.375e+03  1.352e+04  -0.250 0.811142
## Channel58    7.148e+03  1.404e+04   0.509 0.628905
```

```
## Channel59   -1.718e+04  1.258e+04  -1.366 0.221039
## Channel60   -1.304e+04  7.533e+03  -1.731 0.134135
## Channel61    3.965e+04  7.926e+03   5.002 0.002446 **
## Channel62    1.184e+04  1.726e+04   0.686 0.518290
## Channel63   -4.416e+04  2.263e+04  -1.951 0.098904 .
## Channel64    7.167e+04  3.036e+04   2.361 0.056224 .
## Channel65   -1.269e+05  4.568e+04  -2.779 0.032035 *
## Channel66    6.806e+04  4.683e+04   1.453 0.196374
## Channel67    3.277e+04  3.661e+04   0.895 0.405250
## Channel68   -5.417e+04  2.796e+04  -1.938 0.100782
## Channel69    2.730e+04  1.900e+04   1.437 0.200642
## Channel70    1.249e+04  2.832e+04   0.441 0.674666
## Channel71    3.731e+04  3.027e+04   1.233 0.263777
## Channel72   -1.096e+05  2.301e+04  -4.763 0.003118 **
## Channel73    8.525e+04  1.543e+04   5.525 0.001480 **
## Channel74   -6.593e+04  1.469e+04  -4.487 0.004161 **
## Channel75    2.916e+04  1.423e+04   2.049 0.086400 .
## Channel76    3.867e+04  1.026e+04   3.770 0.009287 **
## Channel77   -3.638e+04  1.702e+04  -2.138 0.076378 .
## Channel78    9.437e+03  1.485e+04   0.636 0.548492
## Channel79    3.247e+04  1.622e+04   2.001 0.092281 .
## Channel80   -6.291e+04  2.433e+04  -2.585 0.041480 *
## Channel81   -7.852e+03  1.519e+04  -0.517 0.623642
## Channel82    1.232e+05  3.902e+04   3.158 0.019615 *
## Channel83   -1.410e+05  3.632e+04  -3.882 0.008153 **
## Channel84    2.338e+04  1.523e+04   1.535 0.175634
## Channel85    5.765e+04  2.410e+04   2.392 0.053898 .
## Channel86   -6.535e+04  3.364e+04  -1.943 0.100070
## Channel87    5.474e+04  2.290e+04   2.391 0.053977 .
## Channel88   -4.864e+04  2.062e+04  -2.359 0.056396 .
## Channel89    4.658e+04  2.198e+04   2.120 0.078337 .
## Channel90   -2.108e+04  4.240e+04  -0.497 0.636730
## Channel91    5.957e+04  6.013e+04   0.991 0.360083
## Channel92   -3.924e+04  4.976e+04  -0.789 0.460385
## Channel93    3.180e+04  3.321e+04   0.958 0.375285
## Channel94   -1.040e+05  3.172e+04  -3.280 0.016829 *
## Channel95    9.782e+04  2.216e+04   4.415 0.004496 **
## Channel96   -1.073e+05  2.094e+04  -5.123 0.002171 **
## Channel97    8.059e+04  1.787e+04   4.508 0.004067 **
## Channel98   -1.958e+03  1.305e+04  -0.150 0.885696
## Channel99   -2.519e+04  2.061e+04  -1.222 0.267491
## Channel100   1.406e+04  6.836e+03   2.056 0.085476 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3981 on 6 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9989
## F-statistic: 955.4 on 100 and 6 DF,  p-value: 5.461e-09
```

```r
train_predictions <- predict(model, train_data)
test_predictions <- predict(model, test_data)

train_error <- mean((train_predictions - train_data$Fat)^2)
test_error <- mean((test_predictions - test_data$Fat)^2)
```

```r
cat("Training Error:", train_error, "\n")
```

```
## Training Error: 0.008887299
```

```r
cat("Test Error:", test_error, "\n")
```

```
## Test Error: 275.0871
```

A low training error typically indicates that the model fits the training data well.A higher test error compared to the training error indicates a potential issue of overfitting. Therefore, there is limitation in the model's predictive capabilities.

2. Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features. Report the cost function that should be optimized in this scenario.

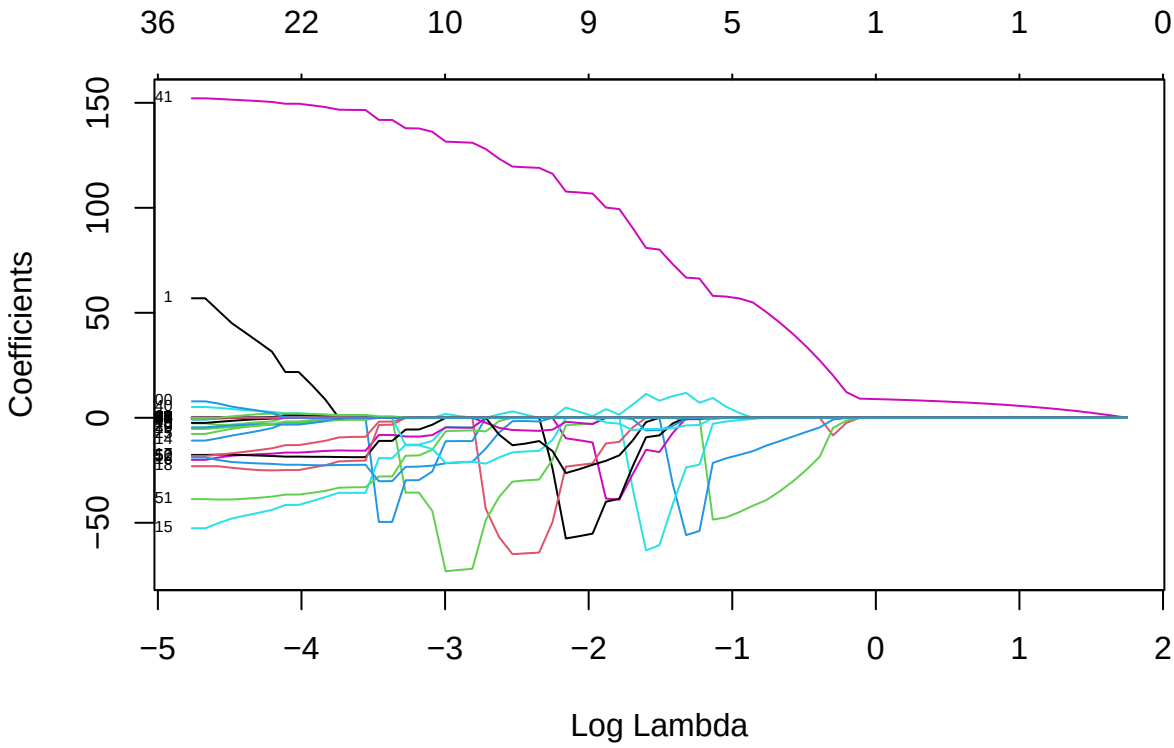
```r
cost <- function(X, y, lambda, theta){

  residuals <- X %*% theta - y

  mse_part <- sqrt(sum(residuals^2)) / length(theta)

  regularization_part <- lambda * sum(abs(theta))

  total_cost <- mse_part + regularization_part

  return(total_cost)
}
```

Cost Function :

$$\hat{\theta} = \frac{1}{n}||y - X\theta||_2^2 + \lambda||\theta||_1$$

3. Fit the LASSO regression model to the training data. Present a plot illustrating how the regression coefficients depend on the log of penalty factor (log λ) and interpret this plot. What value of the penalty factor can be chosen if we want to select a model with only three features?

```r
train_features <- as.matrix(train_data[,2:101])
train_target <- train_data$Fat

lasso_model <- glmnet(train_data[,2:101], train_data$Fat, family = "gaussian",alpha = 1)
plot(lasso_model, xvar = "lambda",label = TRUE)
```



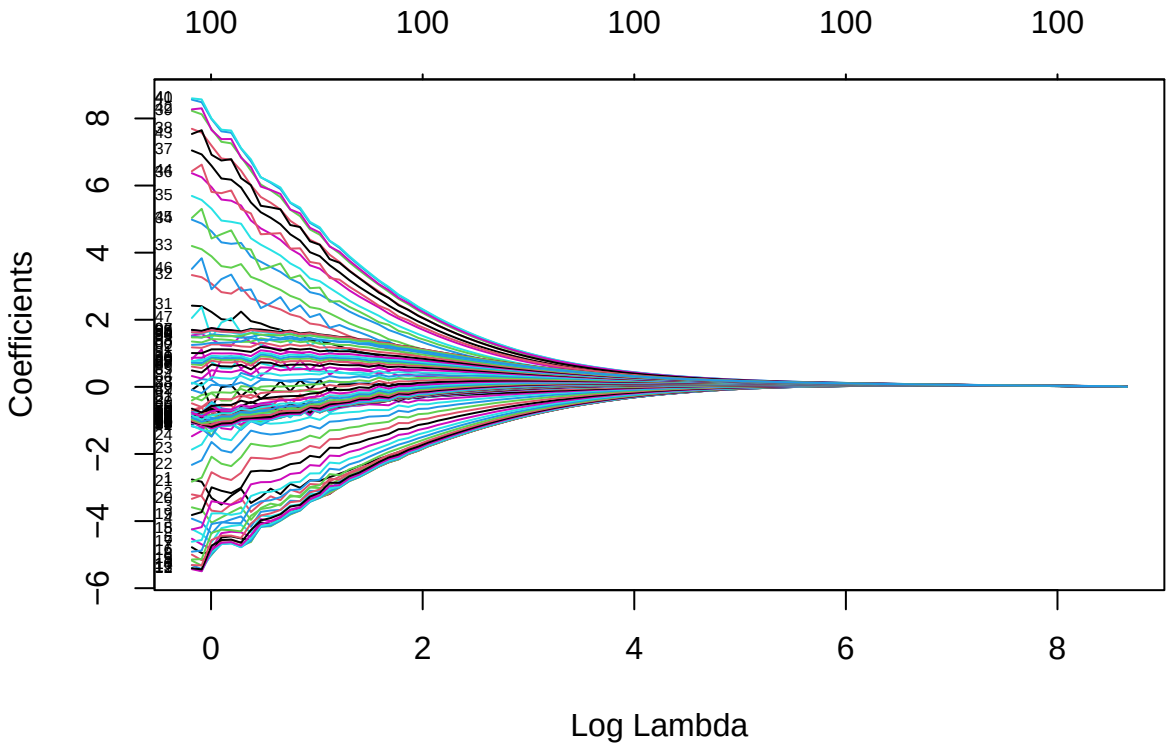

```
nonzero_coeffs <- predict(lasso_model, s = c(lasso_model$lambda), type = "nonzero")
three_feature_lambda <- sapply(nonzero_coeffs, length) == 3
selected_lambda <- lasso_model$lambda[min(which(three_feature_lambda))]
cat("Penalty factor for three features:", selected_lambda, "\n")
```

```
## Penalty factor for three features: 0.6764156
```

The selected penalty factor 0.6764156 strikes a balance, retaining three features in the model while setting the other coefficients to zero.

4. Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4. Conclusions

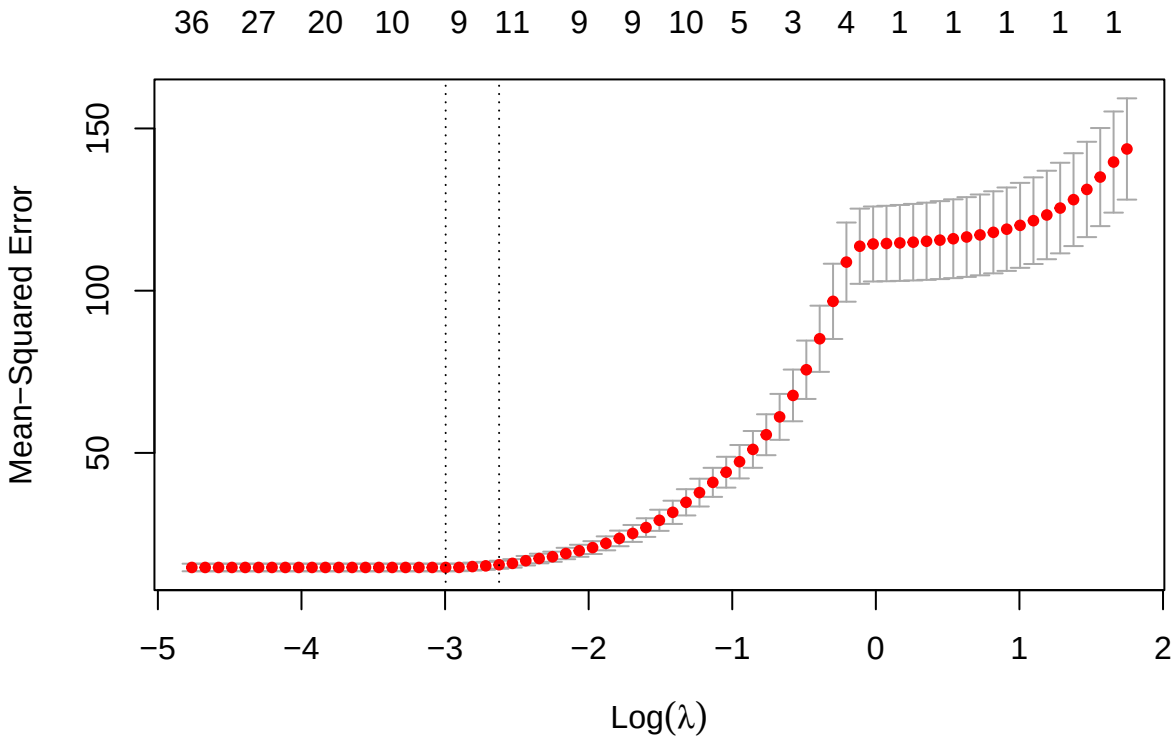
```
ridge_model <- glmnet(x = train_data[,2:101], y = train_data$Fat, alpha = 0)
plot(ridge_model, xvar = "lambda", label = TRUE)
```



Comparison - A higher lambda value in Ridge regression leads to the convergence of all coefficients toward zero. However, unlike LASSO regression, Ridge regression does not aim to filter certain coefficients to exactly zero, especially when the lambda value is small; instead, it tends to proportionally scale them down at relatively similar rates.

5. Use cross-validation with default number of folds to compute the optimal LASSO model. Present a plot showing the dependence of the CV score on log λ and comment how the CV score changes with log λ. Report the optimal λ and how many variables were chosen in this model. Does the information displayed in the plot suggests that the optimal λvalue results in a statistically significantly better prediction than log $\lambda = -4$? Finally, create a scatter plot of the original test versus predicted test values for the model corresponding to optimal lambda and comment whether the model predictions are good.

```
cv_model <- cv.glmnet(x = as.matrix(train_data[,2:101]), y = as.matrix(train_data$Fat),
                      alpha = 1)  # alpha = 1 for LASSO

plot(cv_model)
```




```r
lse_lambda <- which(cv_model$glmnet.fit$lambda == cv_model$lambda.1se)
cv_model$lambda.1se
```

```
## [1] 0.07252983
```

```r
optimal_lambda <- cv_model$lambda.min
cat("Optimal lambda:", optimal_lambda, "\n")
```

```
## Optimal lambda: 0.049992
```

```r
cv_model[["glmnet.fit"]][["df"]][c(lse_lambda, optimal_lambda)]
```

```
## [1] 11
```

```r
num_variables_chosen <- sum(coef(cv_model$glmnet.fit, s = optimal_lambda) != 0)
cat("Number of variables chosen in the model:", num_variables_chosen, "\n")
```

```
## Number of variables chosen in the model: 12
```

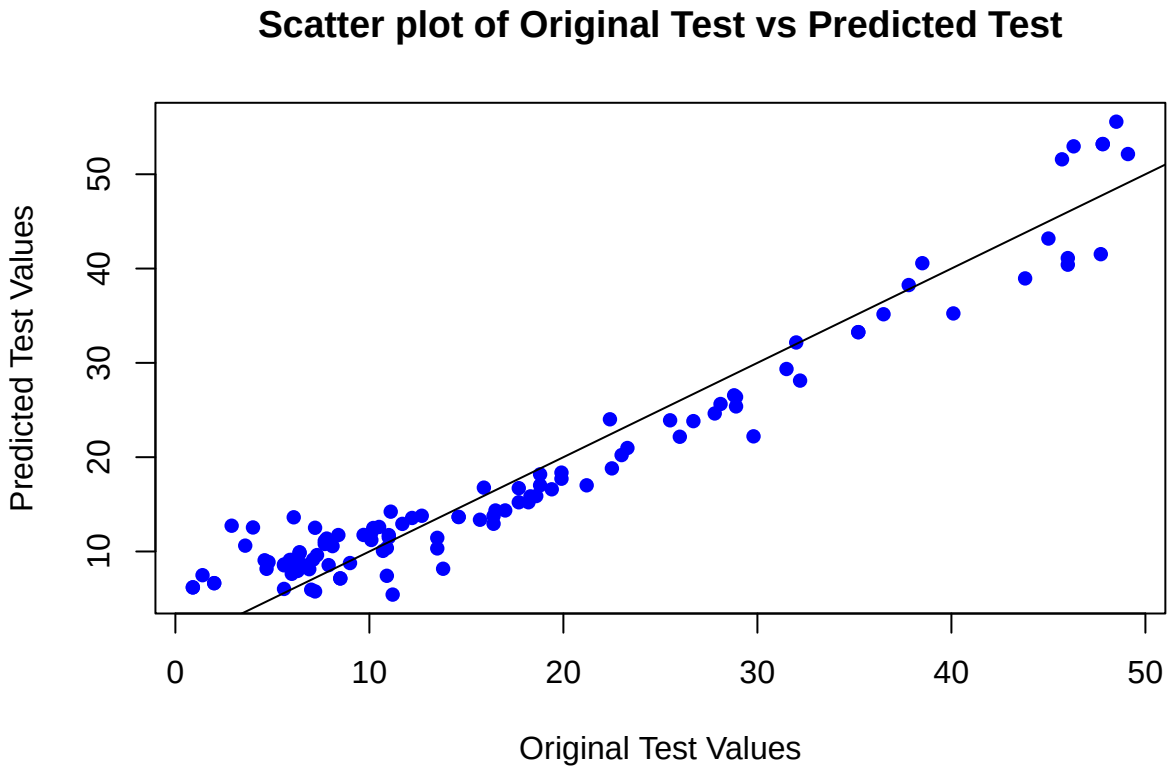
```r
log_lambda_minus_4 <- which(cv_model$lambda == exp(-4))
cv_score_optimal_lambda <- cv_model$cvm[cv_model$lambda == optimal_lambda]
cv_score_lambda_minus_4 <- cv_model$cvm[log_lambda_minus_4]

cat("\nCV Score for optimal lambda:", cv_score_optimal_lambda, "\n")
```

```
##
## CV Score for optimal lambda: 14.6899
```

```
cat("CV Score for log lambda = -4:", cv_score_lambda_minus_4, "\n")
```

```
## CV Score for log lambda = -4:
```

```
test_predictions_optimal <- predict(cv_model, newx = as.matrix(test_data[,2:101]))

plot(test_data$Fat, test_predictions_optimal,
     xlab = "Original Test Values", ylab = "Predicted Test Values",
     main = "Scatter plot of Original Test vs Predicted Test",col="blue",pch = 16)
abline(0, 1, col = "black")
```



0.049992 *doesn't show a significant improvement compared to* $log\lambda = -4$.

Regarding model predictions ,we can see only small differences between what we observed and the optimized predictions. This suggest that the optimized prediction is good and better.

Assignment 2. Decision trees and logistic regression for bank marketing

1. Import the data to R, remove variable "duration" and divide into training/ validation/ test as 40/30/30: use data partitioning code specified in Lecture 2a.

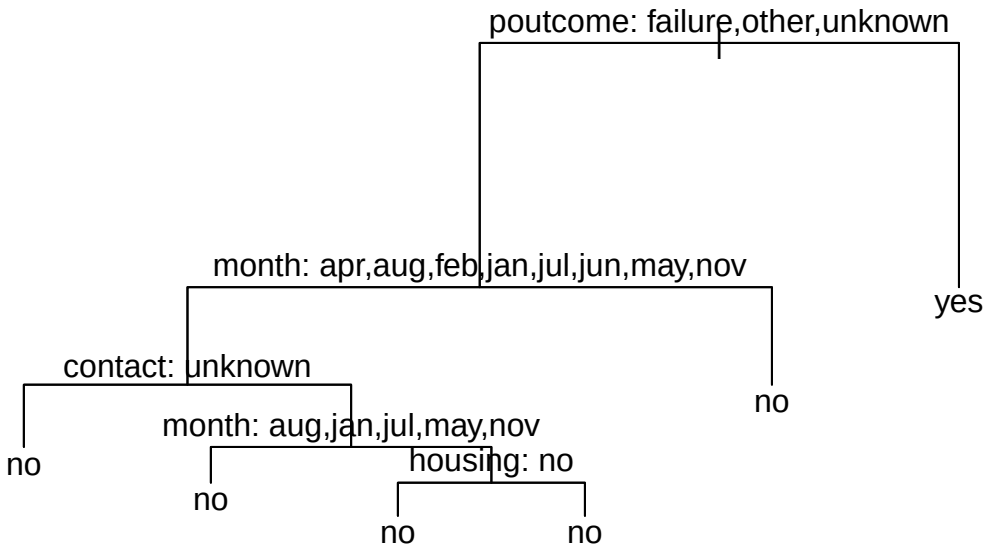
```r
# This dataset has 21 variables altogether.
# Output(target) variable is variable no. 21(categorical variable)
# has the client subscribed a term deposit? (binary: "yes", "no").

## Question 1.

# Inport the data to R, remove variable "duration" and divide it into
# training/validation/test as 40/30/30.

# Load the necessary packages.
library(caret)
library(ggplot2)
library(tree)
library(rpart)


# Import data to R and remove variable "duration".
bank <- read.csv("C:/Users/sange/OneDrive/Documents/bank-full.csv", header = TRUE,sep = ";")
bank <- bank[ , c(-12)]
bank <- data.frame(unclass(bank), stringsAsFactors = TRUE)


# Train data.
n <- nrow(bank)
set.seed(12345)
id <- sample(1 : n, floor(n * 0.4))
train <- bank[id, ]

# Validation data.
id1 <- setdiff(1 : n, id)
set.seed(12345)
id2 <- sample(id1 , floor(n * 0.3))
valid <- bank[id2 , ]

# Test data.
id3 <- setdiff(id1, id2)
test <- bank[id3, ]
```

2. Fit decision trees to the training data so that you change the default settings one by one (i.e. not simultaneously):
a. Decision Tree with default settings.
b. Decision Tree with smallest allowed node size equal to 7000.
c. Decision trees minimum deviance to 0.0005.
and report the misclassification rates for the training and validation data. Which model is the best one among these three? Report how changing the deviance and node size affected the size of the trees and explain why.

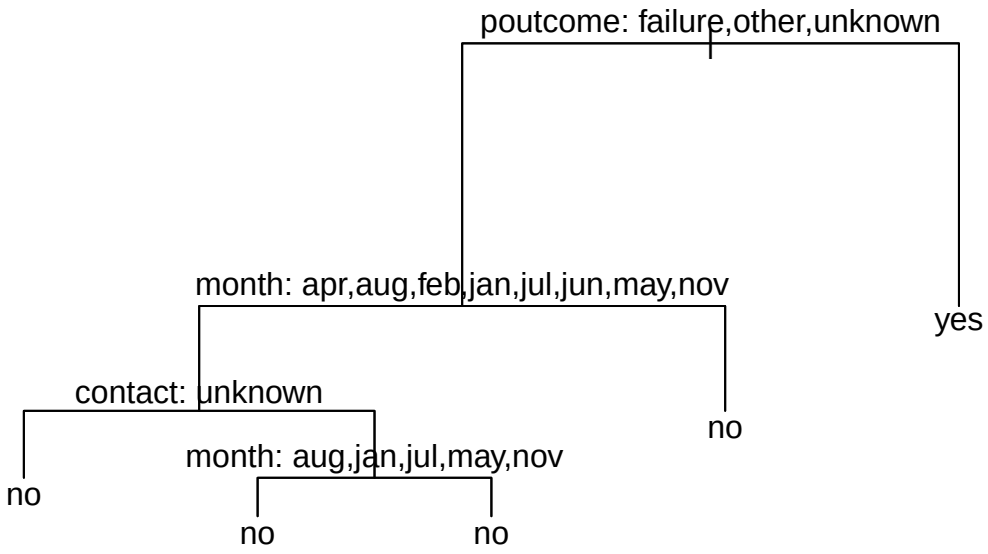
```r
## Question 2

# Fit decision tress to the training data so that you change the defaultsettings
# one by one.

## Part a) Decision tree with default settings.
```

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


## Part b) Decision tree with smallest allowed node size is 7000.


size7000_tree <- tree(y ~., data = train,
                      control = tree.control(nobs = nrow(train), minsize = 7000) )

## Part c) Decision tree with minimum deviance is 0.0005.


mindeviance_tree <- tree(y ~., data = train,
                         control = tree.control(nobs = nrow(train), mindev = 0.0005) )
```

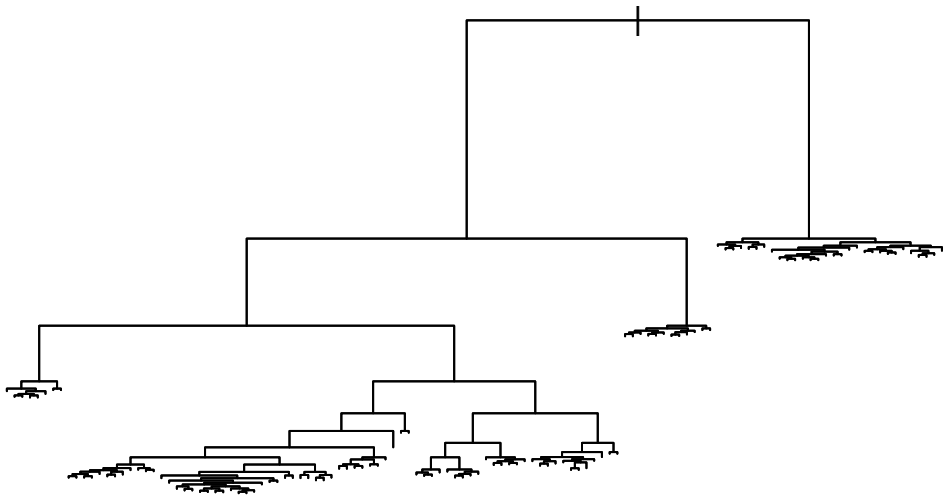
The decision tree with default setting looks as follows:



Decision tree with smallest allowed node size is 7000 looks as follows:



Decision tree with minimum deviance is 0.0005 looks as follows:



```r
# Misclassification error for the training data on default.
summary_default <- summary(default_tree)
misclass_default_train_error <-summary_default$misclass[1]/summary_default$misclass[2]

# Misclassification error for the validation data on default.

predict_default <- predict(default_tree, newdata = valid, type = "class")
misclass_default <- table(valid$y, predict_default)
total <- sum(misclass_default)
diag(misclass_default) <- 0
misclass_default_valid_error <- sum(misclass_default)/total

# Misclassification error for the training data on node size = 7000.
summary_size7000 <- summary(size7000_tree)
misclass_size7000_train_error <-summary_size7000$misclass[1]/summary_size7000$misclass[2]

# Misclassification error for the validation data on node size = 7000.

predict_size7000 <- predict(size7000_tree, newdata = valid, type = "class")
misclass_size7000 <- table(valid$y, predict_size7000)
total <- sum(misclass_size7000)
diag(misclass_size7000) <- 0
misclass_size7000_valid_error <- sum(misclass_size7000)/total

# Misclassification error for the training data on minimum deviance is 0.0005.
summary_mindeviance <- summary(mindeviance_tree)
```

```
misclass_mindeviance_train_error <-summary_mindeviance$misclass[1]/summary_mindeviance$misclass[2]

# Misclassification error for the validation data on minimum deviance is 0.0005.

predict_mindeviance <- predict(mindeviance_tree, newdata = valid, type = "class")
misclass_mindeviance <- table(valid$y, predict_mindeviance)
total <- sum(misclass_mindeviance)
diag(misclass_mindeviance) <- 0
misclass_mindeviance_valid_error <- sum(misclass_mindeviance)/total

#Error Dataframe
trainerror <- c(misclass_default_train_error, misclass_size7000_train_error,
                misclass_mindeviance_train_error)
validerror <- c(misclass_default_valid_error, misclass_size7000_valid_error,
                misclass_mindeviance_valid_error)
Data <- c("Default", "Size 7000", "Minimum Deviance 0.0005")
errordf <- data.frame(Data, trainerror, validerror)
colnames(errordf) <- c("Data","Train Error", "Valid Error")
```

The calculated misclassification errors can be tabulated as follows:

Data	Train Error	Valid Error
Default	0.1048441	0.1092679
Size 7000	0.1048441	0.1092679
Minimum Deviance 0.0005	0.0936187	0.1118484

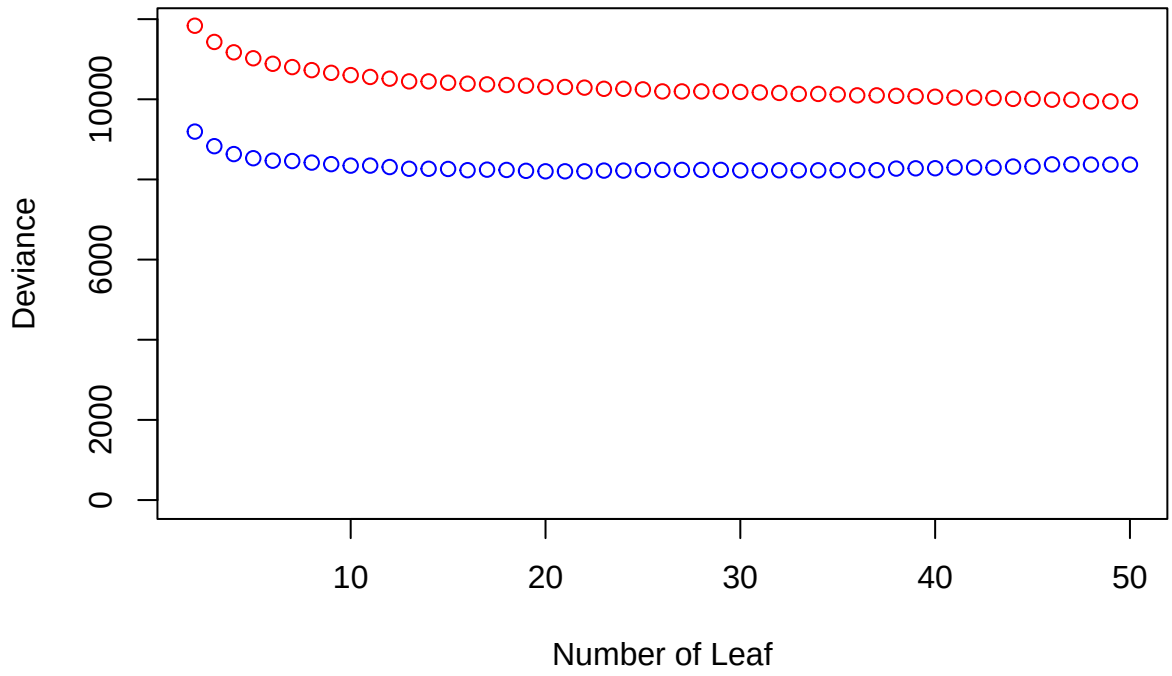
From the table we can choose on default as well as size 7000 model as the best option as both of them give minimum misclassification error for both train and valid data.
The default minimum size of the node is 10 for tree in R. By increasing the value, the capacity of the node to hold more observations increases thus reducing the size of the tree formed. Meanwhile the default minimum deviance of the tree in R is 0.01 and 0.0005 being lesser than 0.01 ensures that the within-node deviance is atleast 0.0005 times of that of the root node causing the tree to grow larger in depth.

3. Use training and validation sets to choose the optimal tree depth in the model 2c: study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and the validation data on the number of leaves and interpret this graph in terms of bias-variance tradeoff. Report the optimal amount of leaves and which variables seem to be most important for decision making in this tree. Interpret the information provided by the tree structure (not everything but most important findings).

```
##Question 3

# finding optimal tree depth

trainScore <- rep(0,50)
validScore <- rep(0,50)
for(i in 2:50) {
  prunedTree <- prune.tree(mindeviance_tree,best=i)
  pred <- predict(prunedTree, newdata=valid,
                  type="tree")
  trainScore[i] <- deviance(prunedTree)
  validScore[i] <- deviance(pred)
}
```

The dependency of deviance on number of leaves can be studied from the below graph:

```
plot(2:50, trainScore[2:50], type="b", col="red",
     ylim=c(0,11800),xlab= "Number of Leaf",ylab= "Deviance")
points(2:50, validScore[2:50], type="b", col="blue")
```

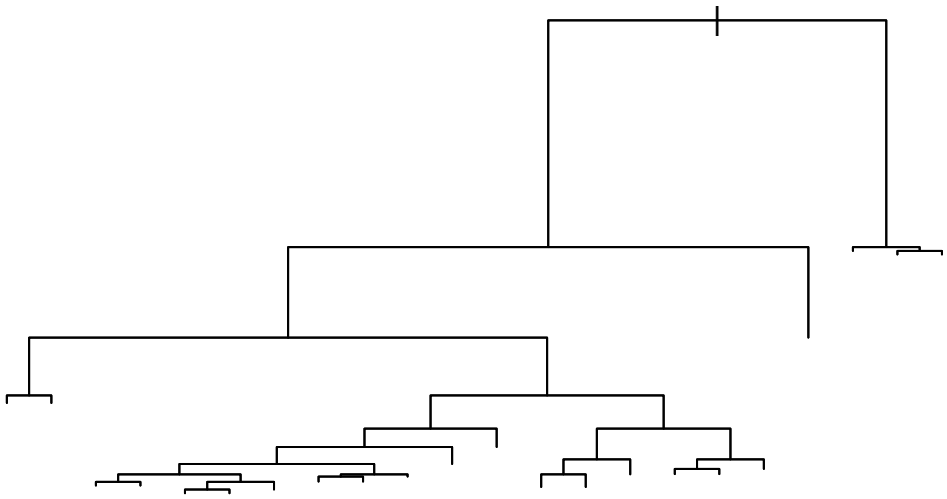


From the graph we can see that the plot for validation data decreases consistently till around 20 and maintains the value and increases around past 35. In the graph there is an exponential decrease of bias till 10 and then it almost maintains the value. The variance can be found to be decreasing consistently even though there is an increase in bias around after 35. Taking the minimum value of validScore will give us the least value which is 21 and since the for loop starts from iteration 2 the optimal number of leaves can be calculated as 21+1 =22.

```
# Finding the optimal number of leaves.
best <- which.min(validScore[2:50])
```

The variables used for the tree are "poutcome" , "month" , "contact" , "pdays" , "age" , "day" , "balance" "housing" and "job".The Optimal number of leaves is 22. So the optimal tree is pruned from the minimum deviance tree and is done as below:

```
# Optimal tree

prunedTreeFinal <- prune.tree(mindeviance_tree, best= (best + 1))
plot(prunedTreeFinal)
```



The root node is split based on poutcome where all successes falls in one category and others falls in second branch. The second is divided based on month where most of the y values from March, September, October and December ends up in no subscription. The probability with which a subscription = "yes" is decided with a comparitively lower probabilities.

4. Estimate the confusion matrix, accuracy and F1 score for the test data by using the optimal model from step 3. Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

```
## Question 4.

## Estimate the Confusion matrix, accuracy and F1 score for the test data.

# Predict using optimal tree.
prediction_test <- predict(prunedTreeFinal, newdata=test,
                           type="class")

# Create the confusion matrix.
confusion_matrix <- table(test$y, prediction_test)

# Use confusion matrix to calculate precision.
precision <- confusion_matrix[2,2]/sum(confusion_matrix[ , 2])

# Use confusion matrix to calculate recall.
recall <- confusion_matrix[2,2]/sum(confusion_matrix[2 , ])

# Use confusion matrix to calculate accuracy.
```

```
accuracy <- (confusion_matrix[2,2] + confusion_matrix[1,1]) /
  (confusion_matrix[1,1] + confusion_matrix[1,2] + confusion_matrix[2,1] + confusion_matrix[2,2])


# Calculate F1 score.
F1_score <- 2 * (precision * recall)/ (precision + recall)
```

The confusion matrix looks like this:

```
##      prediction_test
##          no   yes
##   no  11872   107
##   yes  1371   214
```

The Accuracy is : 0.8910351

The F1 Score is : 0.224554

F1 score is a better option than accuracy because the data has imbalanced classes. Our interest rests on case where the person has subscribed a term deposit. But the number of subscribed data is less.

5. Perform a decision tree classification of the test data with the following loss matrix,

$$L = \begin{array}{c} Observed \\ yes \\ no \end{array} \begin{array}{c} Predicted \\ \begin{pmatrix} 0 & 5 \\ 1 & 0 \end{pmatrix} \end{array}$$

and report the confusion matrix for the test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
## Question 5.

#Perform a decision tree classification of the test data with given loss matrix.

# Create the given loss matrix.
loss_matrix <- matrix(c(0, 1, 5, 0), nrow = 2, ncol = 2, byrow = T)

# Create a decision tree classification of the test data.
model <- rpart(y~.,  train, parms = list(loss = loss_matrix),method = "class")

# Make predictions on test data.

predict_test <- predict(model, newdata = test, type = "class")

# Create the confusion matrix.
custom_loss_cm <- table(test$y, predict_test)

# Use confusion matrix to calculate precision.
precision_loss <- custom_loss_cm[2,2]/sum(custom_loss_cm[ , 2])

# Use confusion matrix to calculate recall.
recall_loss <- custom_loss_cm[2,2]/sum(custom_loss_cm[2 , ])

# Use confusion matrix to calculate accuracy.
```

```
accuracy_loss<- (custom_loss_cm[2,2] + custom_loss_cm[1,1]) /
  (custom_loss_cm[1,1] + custom_loss_cm[1,2] + custom_loss_cm[2,1] + custom_loss_cm[2,2])


# Calculate F1 score.
F1_score_loss <- 2 * (precision_loss * recall_loss)/ (precision_loss + recall_loss)
```

The confusion matrix looks like this:

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

The Accuracy for Loss Matrix Decision Tree is : 0.859481

The F1 Score for Loss Matrix Decision Tree is : 0.4494512

The Accuracy for Optimal Pruned Tree is : 0.8910351

The F1 Score for Optimal Pruned Tree is : 0.224554

When the positive class represents less than 5% of the total data, the chances of more negative class prediction increases. Therefore explicit input of loss matrix is advised so as to improve positive class predictions[Source: https://datamining.togaware.com/survivor/Loss_Matrix.html *]. Here the total number of class "No" is 16018 and Class "Yes" is 2066 which forms only 0.114% of the total train data. So giving a loss function is ideal. Although the Accuracy is getting reduced, we can see an increase in the F1-score for the decision tree with explicit loss matrix which is a good trait. An increased F1-Score implies that the model is attaining a better balance between positive and negative class predictions.*

6. Use the optimal tree and a logistic regression model to classify the test data by using the following principle:

$$\hat{Y} = yes,\ if\ p(Y = \text{"}yes\text{"}|X > \pi),\ no,\ otherwise.$$

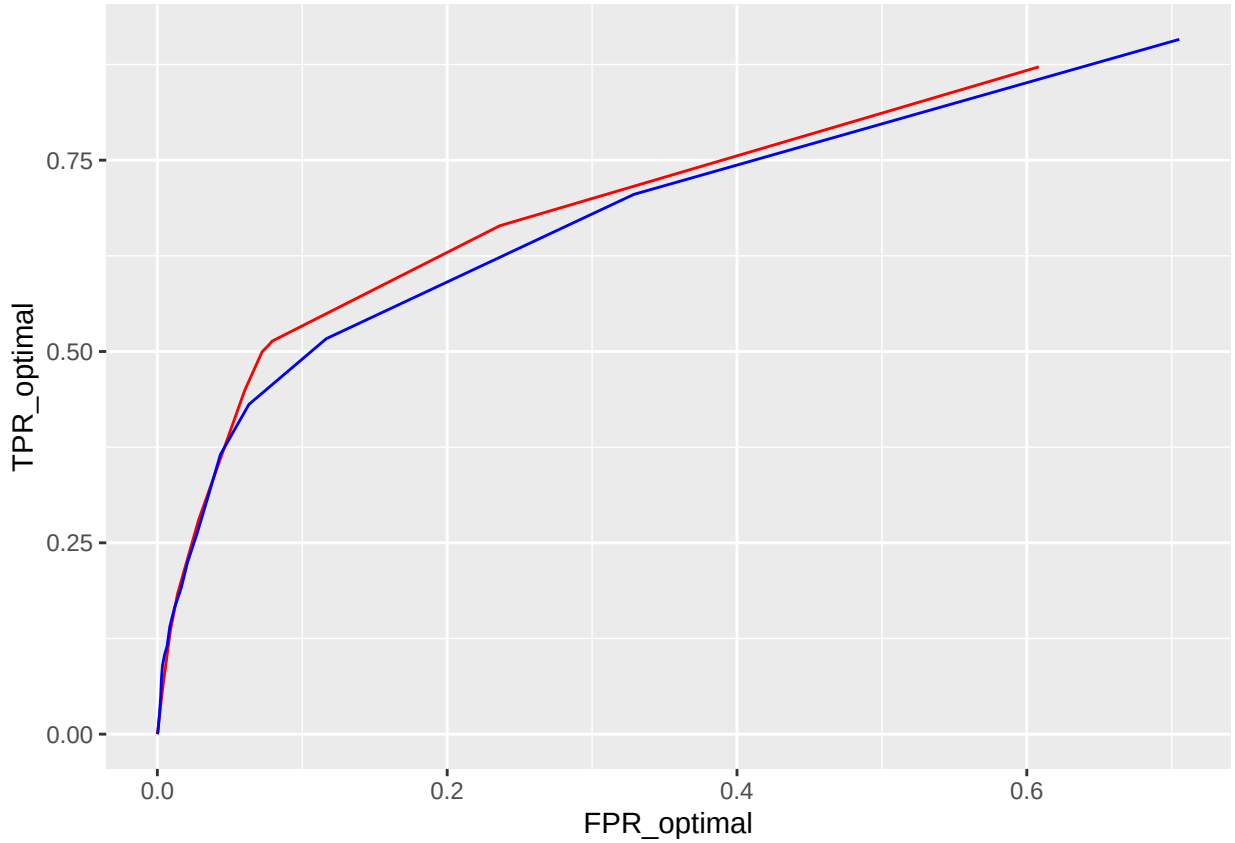
where π = 0.05,0.1,0.15, ... 0.9,0.95. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion? Why precision- recall curve could be a better option here?

```
## Question 6.

pi_value <- seq(0.05, 0.95, by = 0.05)

TPR_optimal <- c()
FPR_optimal <- c()
prec_optimal <- c()
recall_optimal <- c()
predict_optimal <- predict(prunedTreeFinal, newdata = test)


TPR_glm <- c()
FPR_glm <- c()
prec_glm <- c()
recall_glm <- c()
```

```
glm_model <- glm(y ~., data = train, family = "binomial")
predict_glm <- predict(glm_model, test, type = "response")

for (i in 1:length(pi_value)) {
  #assume 1 is no and yes is 2
  principle_predict_optimal <- factor(levels = c("no", "yes"))
  principle_predict_optimal[which(predict_optimal[,2]> pi_value[i])] <- "yes"
  principle_predict_optimal[which(predict_optimal[,2]<= pi_value[i])] <- "no"
  cm_optimal <- table(test$y, principle_predict_optimal)

  TPR_optimal[i] <- cm_optimal[2,2]/sum(cm_optimal[2 , ])
  FPR_optimal[i] <- cm_optimal[1,2]/sum(cm_optimal[1 , ])

  prec_optimal[i] <- cm_optimal[2,2]/sum(cm_optimal[ , 2])
  recall_optimal[i] <- cm_optimal[2,2]/sum(cm_optimal[2 , ])

  # glm
  principle_predict_glm <- factor(levels = c("no", "yes"))
  principle_predict_glm[which(predict_glm > pi_value[i])] <- "yes"
  principle_predict_glm[which(predict_glm <= pi_value[i])] <- "no"
  principle_predict_glm <- as.factor(principle_predict_glm)
  cm_glm <- table(as.numeric(test$y), principle_predict_glm)

  TPR_glm[i] <- cm_glm[2,2]/sum(cm_glm[2 , ])
  FPR_glm[i] <- cm_glm[1,2]/sum(cm_glm[1 , ])

  prec_glm[i] <- cm_glm[2,2]/sum(cm_glm[ , 2])
  recall_glm[i] <- cm_glm[2,2]/sum(cm_glm[2 , ])
}
```

The chart given below shows the ROC Curve for the Decision Tree (Red) and GLM (Blue)

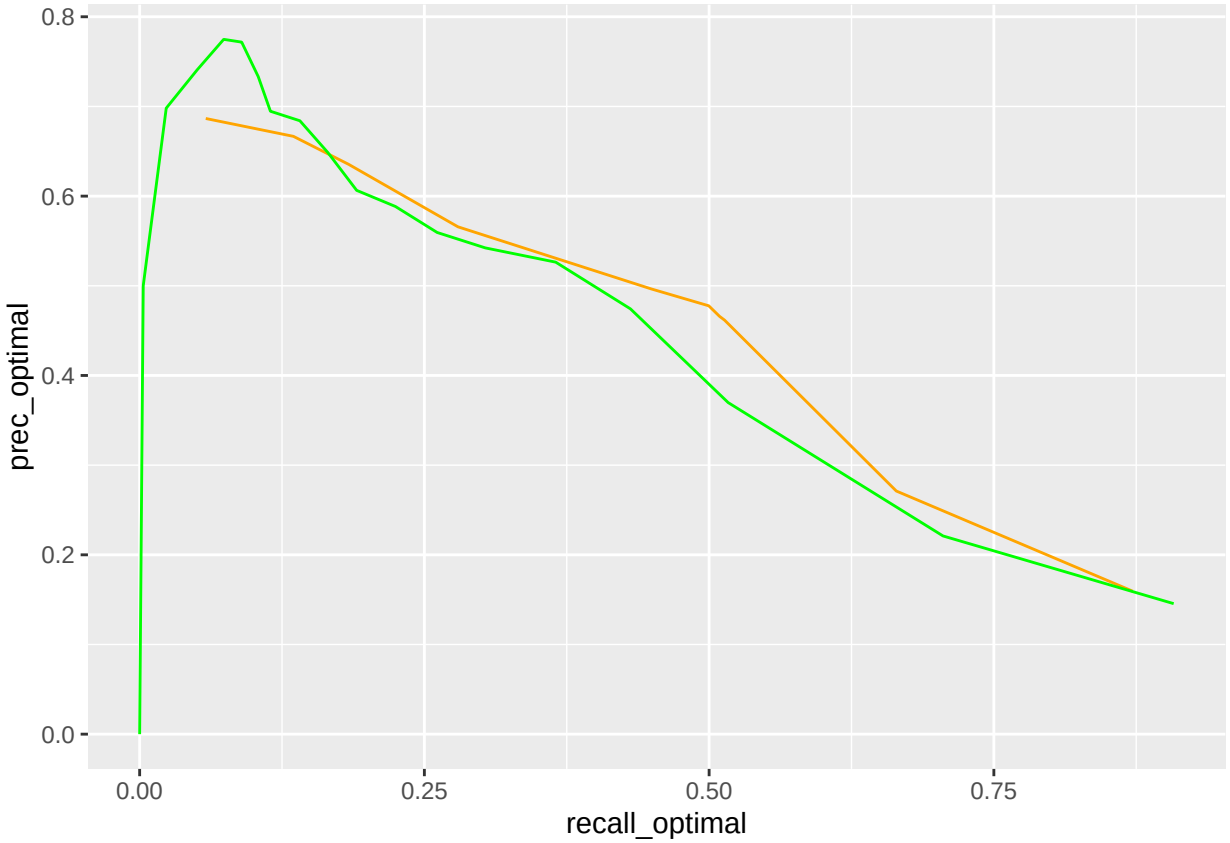
```
#Plotting the ROC Curve:
ROC_data <- data.frame(TPR_optimal, FPR_optimal, TPR_glm, FPR_glm)
ggplot(ROC_data)+
  geom_line(mapping = aes(x= FPR_optimal, y= TPR_optimal), colour = "red")+
  geom_line(mapping = aes(x= FPR_glm, y= TPR_glm), colour = "blue")
```



The area under the Decision Tree Curve is greater than the GLM model Curve.

The graph below is the Precision-Recall Curve for the Decision Tree (Red) and GLM (Blue)

```
#Plotting the Precision - Recall curve:
PRdata <- data.frame(prec_optimal, recall_optimal, prec_glm, recall_glm)
ggplot(PRdata)+
  geom_line(mapping = aes(x= recall_optimal, y= prec_optimal), colour = "orange")+
  geom_line(mapping = aes(x= recall_glm, y= prec_glm), colour = "green")
```

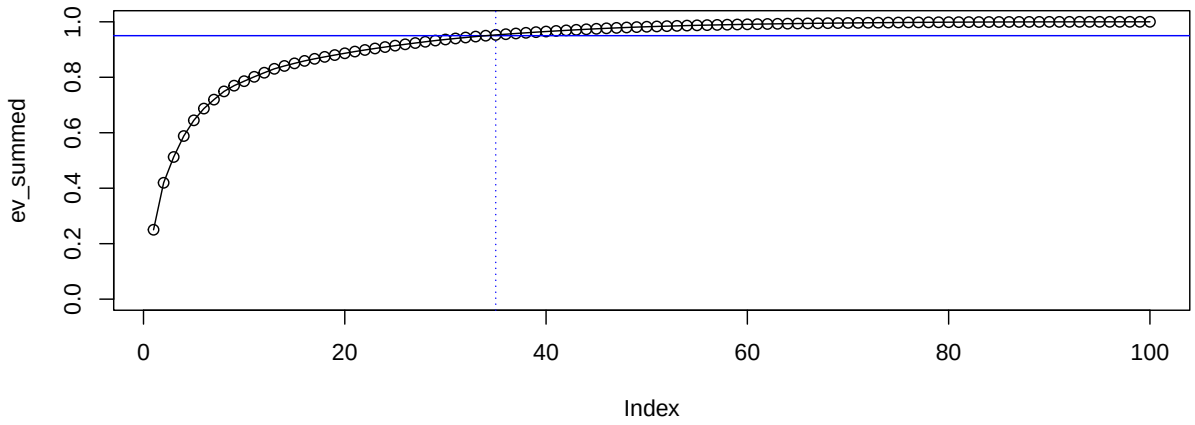


Precision-Recall Curve is preferred over ROC Curve as the data is heavily imbalanced. The ROC curve can produce a good curve by increasing the number of accurate predictions for the majority class. The precision and recall give importance to the minority class and would give focus on the performance of the model.

Assignment 3. Principal components and implicit regularization

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

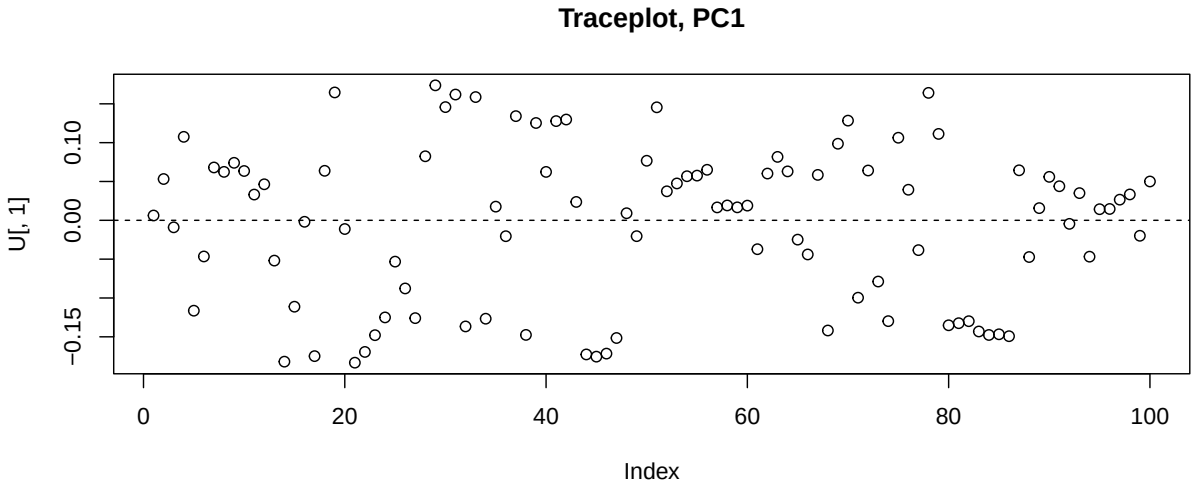
All variables except ViolentCrimesPerPop was scaled with the caret library and PCA was implemented by using the eigen-function.



In the plot above the cumulative sum of the principal components can be seen. PC1 explains 25% of the variation and PC2 explains 16%, so only those two components explain the proportion of variation with 42%. In order to explain 95% (the blue line in the plot above) 35 components is needed.

2. Repeat PCA analysis by using princomp() function and make the trace plot of the first principle component. Do many features have a notable contribution to this component? Report which 5 features contribute mostly (by the absolute value) to the first principle component. Comment whether these features have anything in common and whether they may have a logical relationship to the crime level. Also provide a plot of the PC scores in the coordinates (PC1, PC2) in which the color of the points is given by ViolentCrimesPerPop. Analyse this plot (hint: use ggplot2 package).

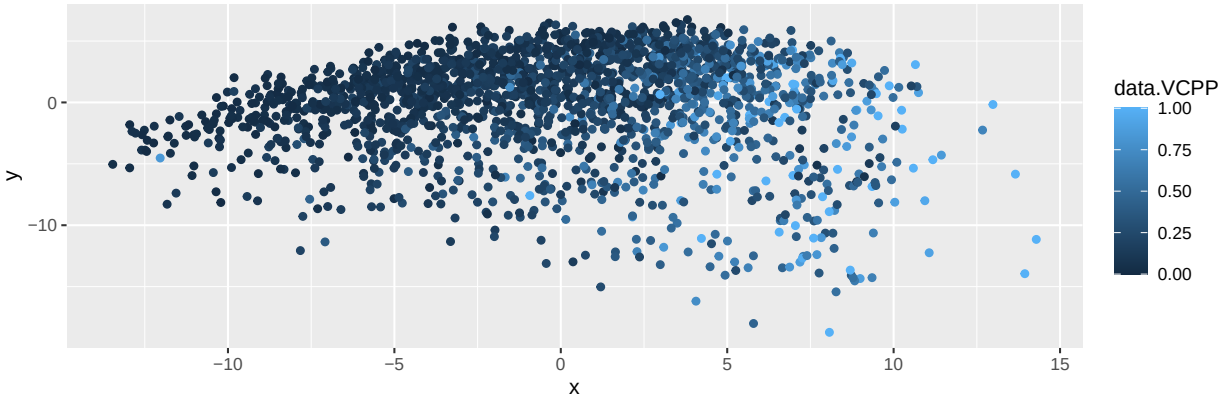
Repeating the PCA analysis but with the princomp-function in R. The contributions to the first principal component can be seen in the traceplot below.



By taking the absolute values of the contributing features and sorting them by magnitude give the five features that contribute mostly to PC1:

- Median family income (medFamInc)
- Median household income (medIncome)
- Percentage of kids that lives with two parents (PctKids2Par)
- Percentage of households with investment income (pctWInvInc)
- Percentage of people under the poverty level (pctPopUnderPov)

It seems likely that these features do have an impact on the crime rate. Poor neighborhoods with absent parents is likely to have a higher rate of violent crimes.



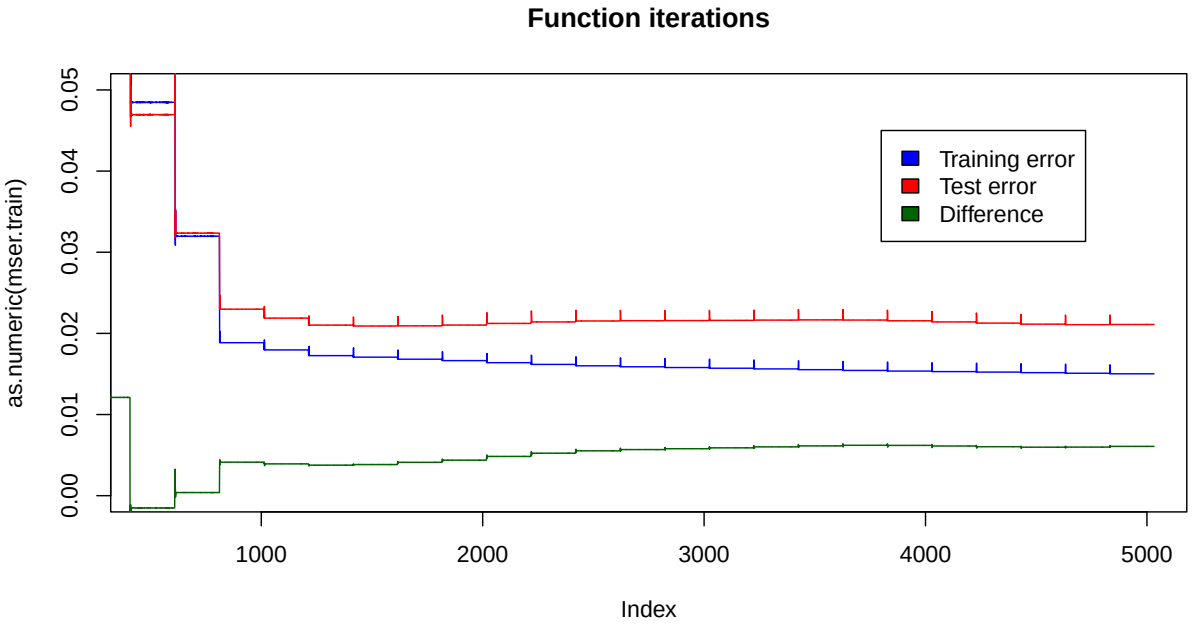
The plot above show the scores in PC1 (x) and PC2 (y) coordinates with the color of the points scaled by the ViolentCrimesPerPop.

3. Split the original data into training and test (50/50) and scale both features and response appropriately, and estimate a linear regression model from training data in which ViolentCrimesPerPop is target and all other data columns are features. Compute training and test errors for these data and comment on the quality of model.

The mean square error for the training data is 0.275 and the test data 0.425. That are relatively high mean square errors and with large difference between the training and test data. Thus this model is probably not good enough to predict the crime rate with precision.

4. Implement a function that depends on parameter vector θ and represents the cost function for linear regression without intercept on the training data set. Afterwards, use BFGS method (optim() function without gradient specified) to optimize this cost with starting point $ ^0 $ = 0 and compute training and test errors for every iteration number. Present a plot showing dependence of both errors on the iteration number and comment which iteration number is optimal according to the early stopping criterion. Compute the training and test error in the optimal model, compare them with results in step 3 and make conclusions.
a. Hint 1: don't store parameters from each iteration (otherwise it will take a lot of memory), instead compute and store test errors directly.
b. Hint 2: discard some amount of initial iterations, like 500, in your plot to make the dependencies visible.

Implementing a cost function that depend on a parameter vector (θ) and using the optim-function with the BFGS method to optimize the parameter vector to minimize the cost function. The cost function used is the same as before (mean square error)



In the plot above the mean square error for the training and test data is shown, and it can be seen how the error decrease for each iteration but at around iteration 1500 the error for the test data is starting to increase. Since further iterations will only decrease the error in the training data but increase the error in the data there is no point in further iterations, and the parameter vector (θ) given in the interval of approximately 1200-1500 seems to be a good model.

Compared to the linear regression model this model gives significantly lower mean square error values (0.018 and 0.022 compared to 0.275 and 0.425) and is thus a more reliable model to predict the crime rate.

STATEMENT OF CONTRIBUTION

The first Assignment was coded and analysed by Sangeeth Sankunny Menon. The second Assignment was coded and analysed by both Daisuke Ronald Ssegwanyi Yamashita and Greeshma Jeev Koothuparambil. The third assignment was coded and analysed by Johan Lundin.

The RMD code for each assignment was coded by respective coders and was put together and compiled by Greeshma Jeev.

APPENDIX

Assignment 1

```
library(glmnet)
library(ggplot2)
```

```r
data <- read.csv("C:/Users/sange/OneDrive/Documents/tecator.csv")


set.seed(123)

train_indices <- sample(nrow(data), nrow(data) * 0.5)
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

model <- lm(Fat ~ . - Sample - Protein - Moisture, data = train_data)

# Summary of the model
summary(model)

train_predictions <- predict(model, train_data)
test_predictions <- predict(model, test_data)

train_error <- mean((train_predictions - train_data$Fat)^2)
test_error <- mean((test_predictions - test_data$Fat)^2)


cat("Training Error:", train_error, "\n")
cat("Test Error:", test_error, "\n")

cost <- function(X, y, lambda, theta){

  residuals <- X %*% theta - y

  mse_part <- sqrt(sum(residuals^2)) / length(theta)

  regularization_part <- lambda * sum(abs(theta))

  total_cost <- mse_part + regularization_part

  return(total_cost)
}

train_features <- as.matrix(train_data[,2:101])
train_target <- train_data$Fat

lasso_model <- glmnet(train_data[,2:101], train_data$Fat, family = "gaussian",alpha = 1)
plot(lasso_model, xvar = "lambda",label = TRUE)

nonzero_coeffs <- predict(lasso_model, s = c(lasso_model$lambda), type = "nonzero")
three_feature_lambda <- sapply(nonzero_coeffs, length) == 3
selected_lambda <- lasso_model$lambda[min(which(three_feature_lambda))]
cat("Penalty factor for three features:", selected_lambda, "\n")

ridge_model <- glmnet(x = train_data[,2:101], y = train_data$Fat, alpha = 0)
plot(ridge_model, xvar = "lambda", label = TRUE)

cv_model <- cv.glmnet(x = as.matrix(train_data[,2:101]), y = as.matrix(train_data$Fat), alpha = 1)  # a
```

```r
plot(cv_model)

lse_lambda <- which(cv_model$glmnet.fit$lambda == cv_model$lambda.1se)
cv_model$lambda.1se
optimal_lambda <- cv_model$lambda.min
cat("Optimal lambda:", optimal_lambda, "\n")
cv_model[["glmnet.fit"]][["df"]][c(lse_lambda, optimal_lambda)]

num_variables_chosen <- sum(coef(cv_model$glmnet.fit, s = optimal_lambda) != 0)
cat("Number of variables chosen in the model:", num_variables_chosen, "\n")

log_lambda_minus_4 <- which(cv_model$lambda == exp(-4))
cv_score_optimal_lambda <- cv_model$cvm[cv_model$lambda == optimal_lambda]
cv_score_lambda_minus_4 <- cv_model$cvm[log_lambda_minus_4]

cat("\nCV Score for optimal lambda:", cv_score_optimal_lambda, "\n")
cat("CV Score for log lambda = -4:", cv_score_lambda_minus_4, "\n")

test_predictions_optimal <- predict(cv_model, newx = as.matrix(test_data[,2:101]))

plot(test_data$Fat, test_predictions_optimal, xlab = "Original Test Values", ylab = "Predicted Test Valu
abline(0, 1, col = "black")
```

Assignment 2

```r
#### Assignment 2: Decision trees and logistic regression for bank marketing.

# This dataset has 21 variables altogether.
# Output(target) variable is variable no. 21(categorical variable)
# has the client subscribed a term deposit? (binary: "yes", "no").

## Question 1.

# Inport the data to R, remove variable "duration" and divide it into
# training/validation/test as 40/30/30.

# Load the necessary packages.
library(caret)
library(ggplot2)
library(tree)
library(rpart)


# Import data to R and remove variable "duration".
bank <- read.csv("bank-full.csv", header = TRUE,sep = ";")
bank <- bank[ , c(-12)]
bank <- data.frame(unclass(bank), stringsAsFactors = TRUE)
```

```r
# Train data.
n <- nrow(bank)
set.seed(12345)
id <- sample(1 : n, floor(n * 0.4))
train <- bank[id, ]

# Validation data.
id1 <- setdiff(1 : n, id)
set.seed(12345)
id2 <- sample(id1 , floor(n * 0.3))
valid <- bank[id2 , ]

# Test data.
id3 <- setdiff(id1, id2)
test <- bank[id3, ]


## Question 2

# Fit decision tress to the training data so that you change the defaultsettings
# one by one.

## Part a) Decision tree with default settings.


default_tree <- tree(y ~., data = train)
plot(default_tree)
text(default_tree, pretty = 0)


## Part b) Decision tree with smallest allowed node size is 7000.


size7000_tree <- tree(y ~., data = train, control = tree.control(nobs = nrow(train), minsize = 7000) )
plot(size7000_tree)
text(size7000_tree, pretty = 0)


## Part c) Decision tree with minimum deviance is 0.0005.


mindeviance_tree <- tree(y ~., data = train, control = tree.control(nobs = nrow(train), mindev = 0.0005)
plot(mindeviance_tree)


# Misclassification error for the training data on default.
summary_default <- summary(default_tree)
misclass_default_train_error <-summary_default$misclass[1]/summary_default$misclass[2]

# Misclassification error for the validation data on default.

predict_default <- predict(default_tree, newdata = valid, type = "class")
misclass_default <- table(valid$y, predict_default)
```

```r
total <- sum(misclass_default)
diag(misclass_default) <- 0
misclass_default_valid_error <- sum(misclass_default)/total

# Misclassification error for the training data on node size = 7000.
summary_size7000 <- summary(size7000_tree)
misclass_size7000_train_error <-summary_size7000$misclass[1]/summary_size7000$misclass[2]

# Misclassification error for the validation data on node size = 7000.

predict_size7000 <- predict(size7000_tree, newdata = valid, type = "class")
misclass_size7000 <- table(valid$y, predict_size7000)
total <- sum(misclass_size7000)
diag(misclass_size7000) <- 0
misclass_size7000_valid_error <- sum(misclass_size7000)/total

# Misclassification error for the training data on minimum deviance is 0.0005.
summary_mindeviance <- summary(mindeviance_tree)
misclass_mindeviance_train_error <-summary_mindeviance$misclass[1]/summary_mindeviance$misclass[2]

# Misclassification error for the validation data on minimum deviance is 0.0005.

predict_mindeviance <- predict(mindeviance_tree, newdata = valid, type = "class")
misclass_mindeviance <- table(valid$y, predict_mindeviance)
total <- sum(misclass_mindeviance)
diag(misclass_mindeviance) <- 0
misclass_mindeviance_valid_error <- sum(misclass_mindeviance)/total

trainerror <- c(misclass_default_train_error, misclass_size7000_train_error,misclass_mindeviance_train_
validerror <- c(misclass_default_valid_error, misclass_size7000_valid_error,misclass_mindeviance_valid_
errordf <- data.frame(trainerror, validerror)
colnames(errordf) <- c("Train Error", "Valid Error")
row.names(errordf) <- c("Default", "Size 7000", "Minimum Deviance 0.005")

##Question 3

# finding optimal tree depth

trainScore <- rep(0,50)
validScore <- rep(0,50)
for(i in 2:50) {
  prunedTree <- prune.tree(mindeviance_tree,best=i)
  pred <- predict(prunedTree, newdata=valid,
                  type="tree")
  trainScore[i] <- deviance(prunedTree)
  validScore[i] <- deviance(pred)
}
plot(2:50, trainScore[2:50], type="s", col="red",
     ylim=c(0,11800),xlab= "Leaf",ylab= "Deviance")
points(2:50, validScore[2:50], type="s", col="blue")

# Finding the optimal number of leaves.
best <- which.min(validScore[2:50])
```

```r
# Optimal number of leaves is 22.



# Optimal tree

prunedTreeFinal <- prune.tree(mindeviance_tree, best= (best + 1))

plot(prunedTreeFinal)


## Question 4.

## Estimate the Confusion matrix, accuracy and F1 score for the test data.

# Predict using optimal tree.
prediction_test <- predict(prunedTreeFinal, newdata=test,
                           type="class")

# Create the confusion matrix.
confusion_matrix <- table(test$y, prediction_test)

# Use confusion matrix to calculate precision.
precision <- confusion_matrix[2,2]/sum(confusion_matrix[ , 2])

# Use confusion matrix to calculate recall.
recall <- confusion_matrix[2,2]/sum(confusion_matrix[2 , ])

# Use confusion matrix to calculate accuracy.
accuracy <- (confusion_matrix[2,2] + confusion_matrix[1,1]) /
  (confusion_matrix[1,1] + confusion_matrix[1,2] + confusion_matrix[2,1] + confusion_matrix[2,2])


# Calculate F1 score.
F1_score <- 2 * (precision * recall)/ (precision + recall)

## F1 score is a better option than accuracy because the data has imbalanced
## classes. We are giving preference to the class where the person has subscribed a term deposit.

## Question 5.

#Perform a decision tree classification of the test data with given loss matrix.

# Create the given loss matrix.
loss_matrix <- matrix(c(0, 1, 5, 0), nrow = 2, ncol = 2, byrow = T)

# Create a decision tree classification of the test data.
model <- rpart(y~.,  train, parms = list(loss = loss_matrix),method = "class")

# Make predictions on test data.

predict_test <- predict(model, newdata = test, type = "class")
```

```r
# Create the confusion matrix.
custom_loss_cm <- table(test$y, predict_test)

# Use confusion matrix to calculate precision.
precision_loss <- custom_loss_cm[2,2]/sum(custom_loss_cm[ , 2])

# Use confusion matrix to calculate recall.
recall_loss <- custom_loss_cm[2,2]/sum(custom_loss_cm[2 , ])

# Use confusion matrix to calculate accuracy.
accuracy_loss<- (custom_loss_cm[2,2] + custom_loss_cm[1,1]) /
  (custom_loss_cm[1,1] + custom_loss_cm[1,2] + custom_loss_cm[2,1] + custom_loss_cm[2,2])


# Calculate F1 score.
F1_score_loss <- 2 * (precision_loss * recall_loss)/ (precision_loss + recall_loss)


## Question 6.

pi_value <- seq(0.05, 0.95, by = 0.05)

TPR_optimal <- c()
FPR_optimal <- c()
prec_optimal <- c()
recall_optimal <- c()
predict_optimal <- predict(prunedTreeFinal, newdata = test)


TPR_glm <- c()
FPR_glm <- c()
prec_glm <- c()
recall_glm <- c()

glm_model <- glm(y ~., data = train, family = "binomial")
predict_glm <- predict(glm_model, test, type = "response")

for (i in 1:length(pi_value)) {
  #assume 1 is no and yes is 2
  principle_predict_optimal <- factor(levels = c("no", "yes"))
  principle_predict_optimal[which(predict_optimal[,2]> pi_value[i])] <- "yes"
  principle_predict_optimal[which(predict_optimal[,2]<= pi_value[i])] <- "no"
  cm_optimal <- table(test$y, principle_predict_optimal)

  TPR_optimal[i] <- cm_optimal[2,2]/sum(cm_optimal[2 , ])
  FPR_optimal[i] <- cm_optimal[1,2]/sum(cm_optimal[1 , ])

  prec_optimal[i] <- cm_optimal[2,2]/sum(cm_optimal[ , 2])
  recall_optimal[i] <- cm_optimal[2,2]/sum(cm_optimal[2 , ])

  # glm
  principle_predict_glm <- factor(levels = c("no", "yes"))
  principle_predict_glm[which(predict_glm > pi_value[i])] <- "yes"
```

```
  principle_predict_glm[which(predict_glm <= pi_value[i])] <- "no"
  principle_predict_glm <- as.factor(principle_predict_glm)
  cm_glm <- table(as.numeric(test$y), principle_predict_glm)

  TPR_glm[i] <- cm_glm[2,2]/sum(cm_glm[2 , ])
  FPR_glm[i] <- cm_glm[1,2]/sum(cm_glm[1 , ])

  prec_glm[i] <- cm_glm[2,2]/sum(cm_glm[ , 2])
  recall_glm[i] <- cm_glm[2,2]/sum(cm_glm[2 , ])
}

#Plotting the ROC Curve:
ROC_data <- data.frame(TPR_optimal, FPR_optimal, TPR_glm, FPR_glm)
ggplot(ROC_data)+
  geom_line(mapping = aes(x= FPR_optimal, y= TPR_optimal), colour = "red")+
  geom_line(mapping = aes(x= FPR_glm, y= TPR_glm), colour = "blue")

#Plotting the Precision - Recall curve:
PRdata <- data.frame(prec_optimal, recall_optimal, prec_glm, recall_glm)
ggplot(PRdata)+
  geom_line(mapping = aes(x= recall_optimal, y= prec_optimal), colour = "orange")+
  geom_line(mapping = aes(x= recall_glm, y= prec_glm), colour = "green")
```

Assignment 3

```
# COMMINITIES AND CRIME
# https://archive.ics.uci.edu/dataset/183/communities+and+crime


# Clear the workspace, set directory and setting the seed for later
rm(list = ls())
set.seed(12345)

# Reading data file communities.csv (meaning of features in the link on line 4)
data = read.csv("communities.csv", header = TRUE)

#-------------------------------------------------------------------------------
# TASK 1
#-------------------------------------------------------------------------------

# a) Scaling all data (except ViolentCrimesPerPop)...
#-------------------------------------------------------------------------------

# Saving the column of Violent crimes per population (for later use)
data.VCPP = data$ViolentCrimesPerPop

# Copy the data and keep only the features and remove the target (removes the
# columnViolentCrimesPerPop)
data.feat = data
```

```r
data.feat$ViolentCrimesPerPop=c()

# Scaling data (see Lecture 1D, slide 15-16)
library(caret)

scaler      = preProcess(data.feat)
data.scaled = predict(scaler,data.feat)

# b) and implement PCA (using eigen)
#-------------------------------------------------------------------------------

# Covariance-matrix
S = cov(data.scaled)

# Eigen-values
ev <- eigen(S)$values

# Summing the eigenvalues as a percentage
ev_summed = cumsum(ev) / sum(ev)

# 35 features is needed (gives 95,3%)
ev_summed[35]

# Plotting the summed values and showing the 95% line
plot1 = plot(ev_summed, ylim = c(0,1))
abline(h=0.95, col="blue")
abline(v=35, col="blue", lty=3)
lines(ev_summed)

# The first and second component contribution
PC1.contribute = ev[1] / sum(ev)
PC2.contribute =  ev[2] / sum(ev)

plot1 = plot(ev_summed, ylim = c(0,1))
abline(h=0.95, col="blue")
abline(v=35, col="blue", lty=3)
lines(ev_summed)

#-------------------------------------------------------------------------------
# TASK 2
#-------------------------------------------------------------------------------

# a)  Repeat PCA analysis by using princomp() function...
#-------------------------------------------------------------------------------

res = princomp(data.scaled)

U = res$loadings

# ...and make the trace plot of the trace plot of PC1
plot2 = plot(U[,1], main="Traceplot, PC1")
abline(h=0, lty=2)
```

```r
# c) Report which 5 features contribute mostly (by the  absolute value) to PC1
#-------------------------------------------------------------------------------
U.abs = abs(U[,1])

U.ord = order(U.abs)

start = nrow(U)-5
end = nrow(U)

for (i in start:end)
 {
  x = colnames(data.scaled[U.ord[i]])
  print(x)
 }

# d) Also provide a plot of the PC scores in the coordinates (PC1, PC2) in which
# the color of the points is given by ViolentCrimesPerPop
#-------------------------------------------------------------------------------
Z = as.data.frame(res$scores)

library(ggplot2)

x = Z[,1]                  # PC1 in the x-axis
y = Z[,2]                  # PC2 on the y-axis

plot3 = ggplot(Z, aes(x, y) ) + geom_point(aes(color = data.VCPP))

plot2 = plot(U[,1], main="Traceplot, PC1")
abline(h=0, lty=2)

ggplot(Z, aes(x, y) ) + geom_point(aes(color = data.VCPP))

#-------------------------------------------------------------------------------
# TASK 3
#-------------------------------------------------------------------------------

# a) Split the original data into training and test (50/50) and scale both features
# and response appropriately...
#-------------------------------------------------------------------------------

n  = nrow(data)
id = sample(1:n, floor(n*0.5))

train = data[id,]
test  = data[-id,]

scaler       = preProcess(train)
train.scaled = predict(scaler,train)
test.scaled  = predict(scaler,test)

# b) ...and estimate a linear regression model from  training data in which
# ViolentCrimesPerPop is target and all other data  columns are features.
#-------------------------------------------------------------------------------
```

```r
linregmod =lm(train.scaled$ViolentCrimesPerPop~., data=train.scaled)

# c) Compute training and test errors for these data
#-------------------------------------------------------------------------------

# Using the linear regression model to predict the results for train and test
pred.train = predict(linregmod,train.scaled, type="response")
pred.test  = predict(linregmod,test.scaled, type="response")

# Taking the actual value from the data and subtracting the predicted value from
# the model, then squaring it. Summing all the squared values and dividing by number
# of data points to get the mean

# Training data
sqer.train = (train.scaled$ViolentCrimesPerPop - pred.train)^2
mser.train.lm = sum(sqer.train)/length(sqer.train)

# Test data
sqer.test = (test.scaled$ViolentCrimesPerPop - pred.test)^2
mser.test.lm = sum(sqer.test)/length(sqer.test)

#-------------------------------------------------------------------------------
# TASK 4
#-------------------------------------------------------------------------------

# a) Implement a function that depends on parameter vector theta and represents the
# cost function for linear regression without intercept on the training data set
#-------------------------------------------------------------------------------

# Copy the data and keep only the features and remove the target (removes the
# columnViolentCrimesPerPop)
train.feat = train
train.VCCP = train.feat$ViolentCrimesPerPop
train.feat$ViolentCrimesPerPop = c()

test.feat = test
test.VCCP = test.feat$ViolentCrimesPerPop
test.feat$ViolentCrimesPerPop = c()

# Initialize list to save (according to Lecture 2D, slide 33)
mser.train=list()
mser.test =list()
thetas    =list()
k=0

# Cost-function (mean squared error for the training data, mser.train)
J = function(theta)
{

  # For the training data
  h = rowSums (t(t(train.feat) * theta))
  sqer.train = (train.VCCP- h)^2
  mser.train  = sum(sqer.train)/length(sqer.train)
```

```
  # For the test data
  h = rowSums (t(t(test.feat) * theta))
  sqer.test = (test.VCCP- h)^2
  mser.test  = sum(sqer.test)/length(sqer.test)

  # In order to plot iterations (according to Lecture 2D, slide 33)
  .GlobalEnv$k= .GlobalEnv$k+1
  .GlobalEnv$mser.train[[k]]=mser.train
  .GlobalEnv$mser.test[[k]]=mser.test
  #.GlobalEnv$thetas[[k]]=theta <- Does not need to know actual values of theta

  return(mser.train)

}

# b) use BFGS method (optim() function without gradient specified) to
# optimize this cost with starting point theta_0 = 0 and compute training and
# test errors for every iteration number
#-------------------------------------------------------------------------------

theta_0 = rep(0, length(train.feat))

output = optim(par= theta_0, fn= J, method = "BFGS", control=list(maxit=25))

# Stopping criteria?

# c) Present a plot showing dependence of  both errors on the iteration number
#-------------------------------------------------------------------------------

plot4 = plot(as.numeric(mser.train), type="l", main="Function iterations",
             xlim = c(500, 5000), ylim = c(0, 0.05), col="blue")

lines(as.numeric(mser.test), type="l", col="red")

lines((as.numeric(mser.test)-as.numeric(mser.train)), type="l", col="darkgreen",
      xlim = c(500, 5000), ylim = c(0, 0.1))

legend(3800, 0.045, legend=c("Training error", "Test error", "Difference"),
       fill = c("blue","red", "darkgreen") )
# d) Compute the training and  test error in the optimal model, compare them with
# results in step 3 and make conclusions.
#-------------------------------------------------------------------------------

# Mean square errors for the linear regression model
print(mser.train.lm)
print(mser.test.lm)

# Just taking some value by visually inspecting the plot. It seems that the min
# difference (with low errors) between training and test is in the interval of
# approximately 1000 to 1600, so just picked iteration 1200 to get some values
mser.train[1200]
mser.test[1200]
```

```
plot4 = plot(as.numeric(mser.train), type="l", main="Function iterations", xlim = c(500, 5000), ylim = c
lines(as.numeric(mser.test), type="l", col="red")
lines((as.numeric(mser.test)-as.numeric(mser.train)), type="l", col="darkgreen",xlim = c(500, 5000), yli
legend(3800, 0.045, legend=c("Training error", "Test error", "Difference"), fill = c("blue","red", "dark
```
