Laboration report in Machine Learning

Computer lab 2 block 1

732A99

Simge Cinar
Duc Tran
William Wiik

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

01 december 2023

Contents

1 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. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry. Divide data randomly into train and test (50/50) by using the codes from the lectures.

```r
set.seed(12345)


# Read in data
tecator <- read.csv("tecator.csv")


# Partitioning training data (50%)
n <- dim(tecator)[1]
id <- sample(1:n, floor(n*0.5))
tecator_train <- tecator[id,]


# Partitioning test data (50%)
tecator_test <- tecator[-id,]
```

1.1 Question 1.1

Question:

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.

Answer:

Probabilistic model:

$$\hat{Fat} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + ... + \hat{\theta}_{100} x_{100} + \epsilon$$

```r
# column 1, 103 and 104 are sample, moisture and protein and should
# not be used as features
lm_model <- lm(Fat ~ ., data = tecator_train[,-c(1,103:104)])



pred_train <- predict(lm_model, newdata = tecator_train)
pred_test <- predict(lm_model, newdata = tecator_test)

mse_train <- mean((tecator_train$Fat - pred_train)^2)
mse_test <- mean((tecator_test$Fat - pred_test)^2)
```

```r
df <- data.frame("MSE" = c(mse_train, mse_test))
rownames(df) <- c("Train","Test")

knitr::kable(df, row.names = TRUE, booktabs=T, caption = "Training and test errors", digits = 4) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 1: Training and test errors

	MSE
Train	0.0057
Test	722.4294

Question:

Comment on the quality of fit and prediction and therefore on the quality of model.

Answer:

The MSE for the training data is very low, 0.0057 and for the test data the MSE is high, approximately 722. This means that the model is very overfitted from the training data and therefor the quality for the model is bad. This is expected, due to n (observations) (107) is almost equal to p (features) (100).

1.2 Question 1.2

Question:

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.

Answer:

Cost function:

$$\hat{\theta}^{lasso} = argmin \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_{1i} - ... - \theta_{100} x_{100i})^2 + \lambda \sum_{j=1}^{p} |\theta_j| \right\}$$

- n is the number of observations
- y_i is the observed value for the i-th observation.
- x_{ij} is the j-th feature (channel) value the i-th observation.
- θ_0 is the intercept term.
- θ_j is the regression coefficient for the j-th feature
- $\lambda > 0$ is penalty factor

1.3 Question 1.3

Question:

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?

Answer:

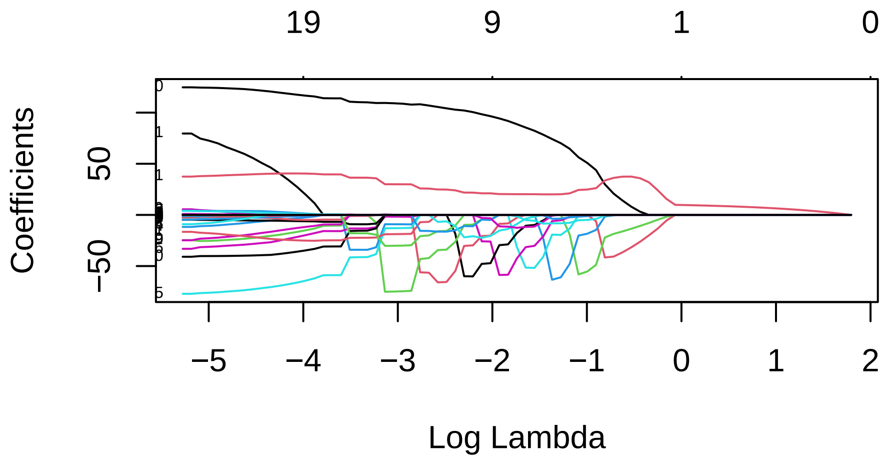


Figure 1: LASSO: regression coefficients depend on the log of penalty factor

In figure 1, the penalty factor $log(\lambda)$ and the coefficients for the features are plotted. As $log(\lambda)$ gets higher the more coefficients reach to zero. If the penalty factor $log(\lambda)$ is higher than 0, then the model select only one feature. If we want to select a model with only three features, then we would have a $log(\lambda)$ equal to around -0.3, then the λ value would be $exp(-0.3) = 0.74$.

1.4 Question 1.4

Question:

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

Answer:

When comparing this plot with the plot from step 3, none of the coefficients from the ridge features reach to zero, as it does in LASSO. Instead, the coefficients from the ridge converge to zero.

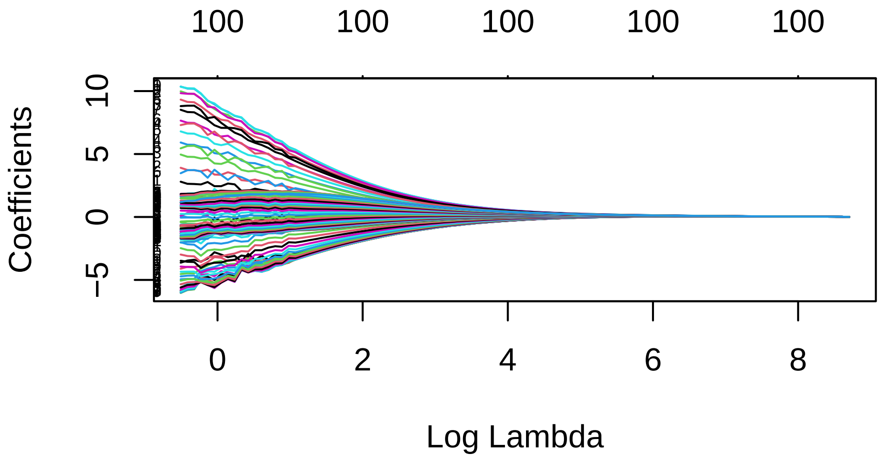


Figure 2: Ridge: regression coefficients depend on the log of penalty factor

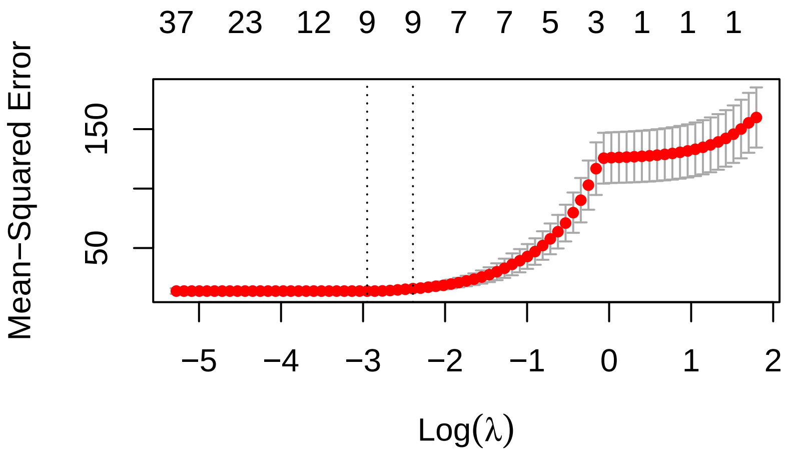
1.5 Question 1.5

Question:

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 λ value results in a statistically significantly better prediction than log λ = -4?

Answer:

```r
model_lasso_cv <- cv.glmnet(as.matrix(covariates),response, alpha = 1,
                   family = "gaussian")
plot(model_lasso_cv)
```



```r
coef_cv <- coef(model_lasso_cv, s="lambda.min")
```

```r
index <- coef_cv@i
variables <- coef_cv@Dimnames[[1]][index+1]
value <- coef_cv@x

df <- data.frame("Variables" = variables, "Coefficient" = value)

knitr::kable(df, row.names = FALSE, booktabs=T,
             caption = "Variables that were chosen in this LASSO model",digits = 5)%>%
      kable_styling(latex_options = "HOLD_position")
```

Table 2: Variables that were chosen in this LASSO model

Variables	Coefficient
(Intercept)	29.77500
Channel13	-74.65858
Channel14	-9.15842
Channel15	-12.86548
Channel16	-1.71411
Channel40	108.71143
Channel41	29.93592
Channel50	-0.00023
Channel51	-18.67084
Channel52	-30.02545

```
model_lasso_cv$lambda.min
```

```
## [1] 0.05234206
```

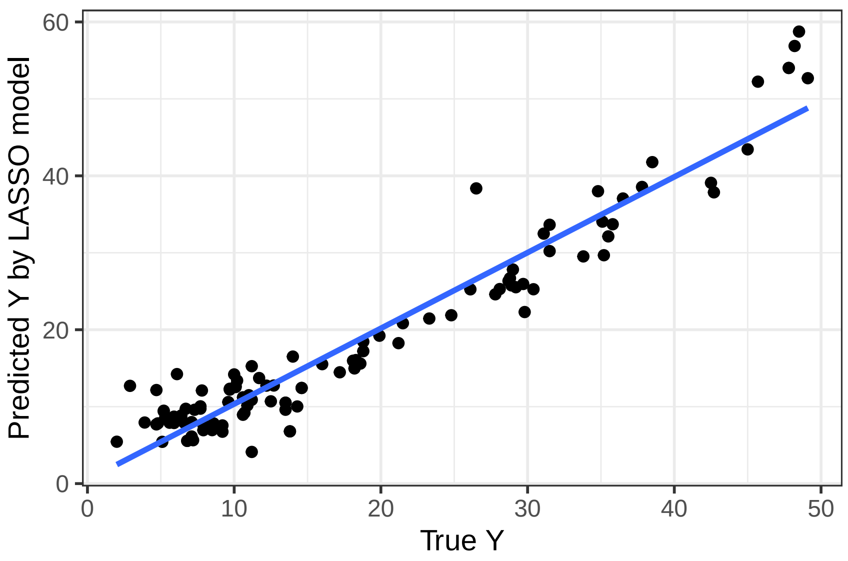
For $log(\lambda)$ values ranging from -5 to -2.5, the Mean Squared Error (MSE) remains constant. However, there is a notable increase in MSE when $log(\lambda)$ reaches 0. Then, as $log(\lambda)$ increases further, the MSE gradual increase. Table X displays the selected variables for this model by its corresponding coefficient. In total, eight variables were included. The optimal value for λ is determined to be $log(-2.857) = 0.06$. The plot suggest that the optimal λ value does not results in a statistically significantly better prediction than $log(\lambda)$ = -4, because they have around the same MSE.

Question:

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.

Answer:

```
library(ggplot2)

y <- tecator_test[,"Fat"]
ynew <- predict(model_lasso_cv, newx = as.matrix(tecator_test[, 2:101]),
                type = "response")


ggplot(data.frame(y,ynew), aes(y, lambda.1se)) + geom_point() + theme_bw() +
  labs(x = "True Y",
       y = "Predicted Y by LASSO model") +
  geom_smooth(method='lm',formula = y ~ x,se = FALSE )
```



Answer

The model predictions seems fairly good, because the point does not differ so much from the blue line. Perfect predictions, without any errors, would align the points precisely on the blue line.

2 Assignment 2. Decision trees and logistic regression for bank marketing.

The data in this assignment is related with direct marketing campaigns of a Portuguese banking institution. Data consists of 21 variables, where 20 are input variables about the clients and the output variable is if the client has a term deposit (yes/no).

2.1 Question 2.1

Question: 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.

Answer: Data was loaded into R, where all character variables were made into factor variables, and the variable duration was removed.

```r
# Index 12 is the variable duration
data <- read.csv2("bank-full.csv", stringsAsFactors = TRUE)[, -12]

# Data partitioning (40/30/30)
# Training data
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]

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

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

After the splitting the number of observations in the data sets are as follows:

- Training: 18 084 observations.
- Validation: 13 563 observations.
- Test: 13 563 observations.

2.2 Question 2.2

Question: 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.

Answer:

The default setting in tree for smallest allowed node size is 10 and the default for minimum deviance is 0.01. The three different trees were fitted and training error, validation error and number of leaves are presented in table 3.

```
# Tree a: default setting
tree_a <- tree(y ~., data=train)
# Misclassification for training data for tree a
pred_train <- predict(tree_a, newdata=train, type="class")
table_train <- table(train$y, pred_train)
miss_train <- 1 - (sum(diag(table_train)) / sum(table_train))
# Misclassification for validation data for tree a
pred_valid <- predict(tree_a, newdata=valid, type="class")
table_valid <- table(valid$y, pred_valid)
miss_valid <- 1 - (sum(diag(table_valid)) / sum(table_valid))
# Number of leaves
leaf <- sum(tree_a$frame$var == "<leaf>")
# Misclassification for tree a
model_a <- c(miss_train, miss_valid, leaf)



# Tree b: smallest size changed from 10 (default) to 7 000
tree_b <- tree(y ~., data=train, control=tree.control(nobs = nrow(train),
                                                      minsize = 7000))
# Misclassification for training data for tree b
pred_train <- predict(tree_b, newdata=train, type="class")
table_train <- table(train$y, pred_train)
miss_train <- 1 - (sum(diag(table_train)) / sum(table_train))
# Misclassification for validation data for tree b
pred_valid <- predict(tree_b, newdata=valid, type="class")
table_valid <- table(valid$y, pred_valid)
miss_valid <- 1 - (sum(diag(table_valid)) / sum(table_valid))
# Number of leaves
leaf <- sum(tree_b$frame$var == "<leaf>")
# Misclassification for tree b
model_b <- c(miss_train, miss_valid, leaf)


# Tree c: mindev changed from 0.01 (default) to 0.0005
```

```r
tree_c <- tree(y ~., data=train, control=tree.control(nobs = nrow(train),
                                                      mindev = 0.0005))
# Misclassification for training data for tree c
pred_train <- predict(tree_c, newdata=train, type="class")
table_train <- table(train$y, pred_train)
miss_train <- 1 - (sum(diag(table_train)) / sum(table_train))
# Misclassification for validation data for tree c
pred_valid <- predict(tree_c, newdata=valid, type="class")
table_valid <- table(valid$y, pred_valid)
miss_valid <- 1 - (sum(diag(table_valid)) / sum(table_valid))
# Number of leaves
leaf <- sum(tree_c$frame$var == "<leaf>")
# Misclassification for tree c
model_c <- c(miss_train, miss_valid, leaf)
```

Table 3: Misclassification error for the three trees on training and validation data.

	Training error	Validation error	Number of leaves
Tree a: default setting	0.104844	0.109268	6
Tree b: smallest node 7000	0.104844	0.109268	5
Tree c: min deviance 0.0005	0.094006	0.111922	122

From table 3, the training error and validation error for tree a and b are the same, however there is a difference in number of leaves where tree a has one more leaf. In tree b, the minimum node size is changed to 7 000 from 10 which forces the tree to have least 7 000 observations in each node. The increase in minimum node size makes that the tree can not split nodes smaller than 7 000 and thus the tree will have less leaves.

Training error is lowest for tree c, but the validation error is also highest for tree c. This indicates that tree c is more overfitted on training data compared to the tree a and b. The difference for tree c is that minimum deviance is changed from 0.01 to 0.0005. Minimum deviance is how much within-node deviance a node have to have compared to the root node to be able to split. By having a smaller number, the tree is allowed to split nodes with smaller deviance, which in turns allows the tree to grow and have smaller nodes than before.

Between tree a, b, and c the tree b is best since it has lowest validation error but also has the least amounts of leaves. By having less leaves the model is less complex and is easier to interpret.

2.3 Question 2.3

Question: 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).

Answer: Tree c is used where the tree is post-pruned to trees with between 2 and 50 leaves. The deviance for training data and validation data are calculated and the code is as follows:

```
fit <- tree(y ~., data=train,
            control=tree.control(nobs = nrow(train),
                                 mindev = 0.0005))
trainScore=rep(0,50)
validScore=rep(0,50)
for(i in 2:50){
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  validScore[i]=deviance(pred)
}
```

The tree with the lowest validation error had 22 leaves.

```
# Finds min, add +1 since index 1 is tree with 2 leaves.
which(min(validScore[2:50]) == validScore[2:50])+1
```

```
## [1] 22
```

The dependence of deviances for training and validation data for different numbers of leaves is presented in figure 3.

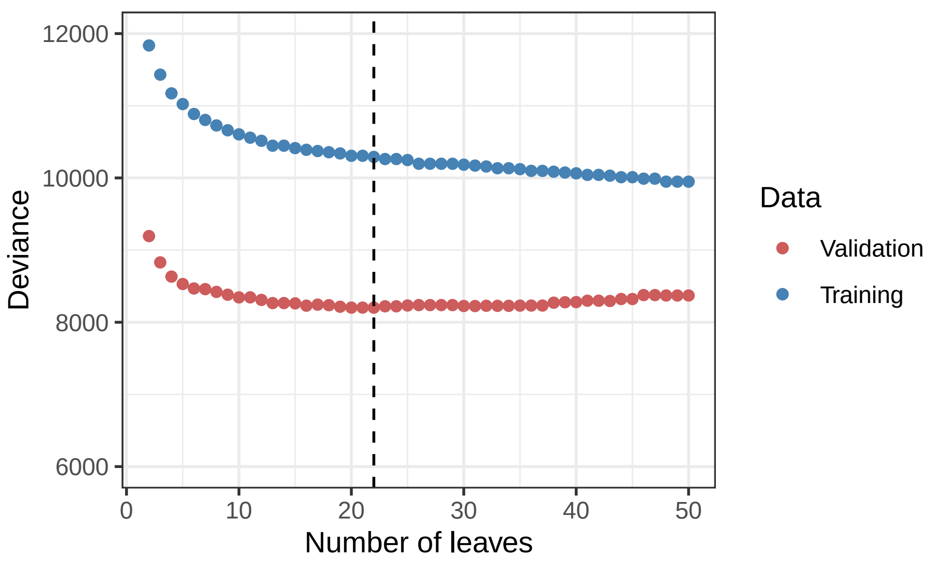


Figure 3: Dependence of deviances for training- and validationdata.

In figure 3 the dashed line shows where the tree with the lowest deviance for testdata is located. For tree models, the complexity increases when the number of leaves increase. In terms of bias-variance tradeoff a model with low complexity (few leaves) have high bias but low variance and is considered to be underfitting. In contrast a model with high complexity have low bias but high variance and is considered to be overfitting. In figure 3, the tree with 22 leaves (dashed line) is considered to be the tree where we have a balance between low/high values for bias and variance which gives us that the tree has the optimal fit since it is not underfitting or overfitting.

The tree with 22 leaves is considered to be optimal and is presented in figure 4.

```
best_fit <- prune.tree(fit, best=22)
plot(best_fit)
text(best_fit, pretty=0)
```

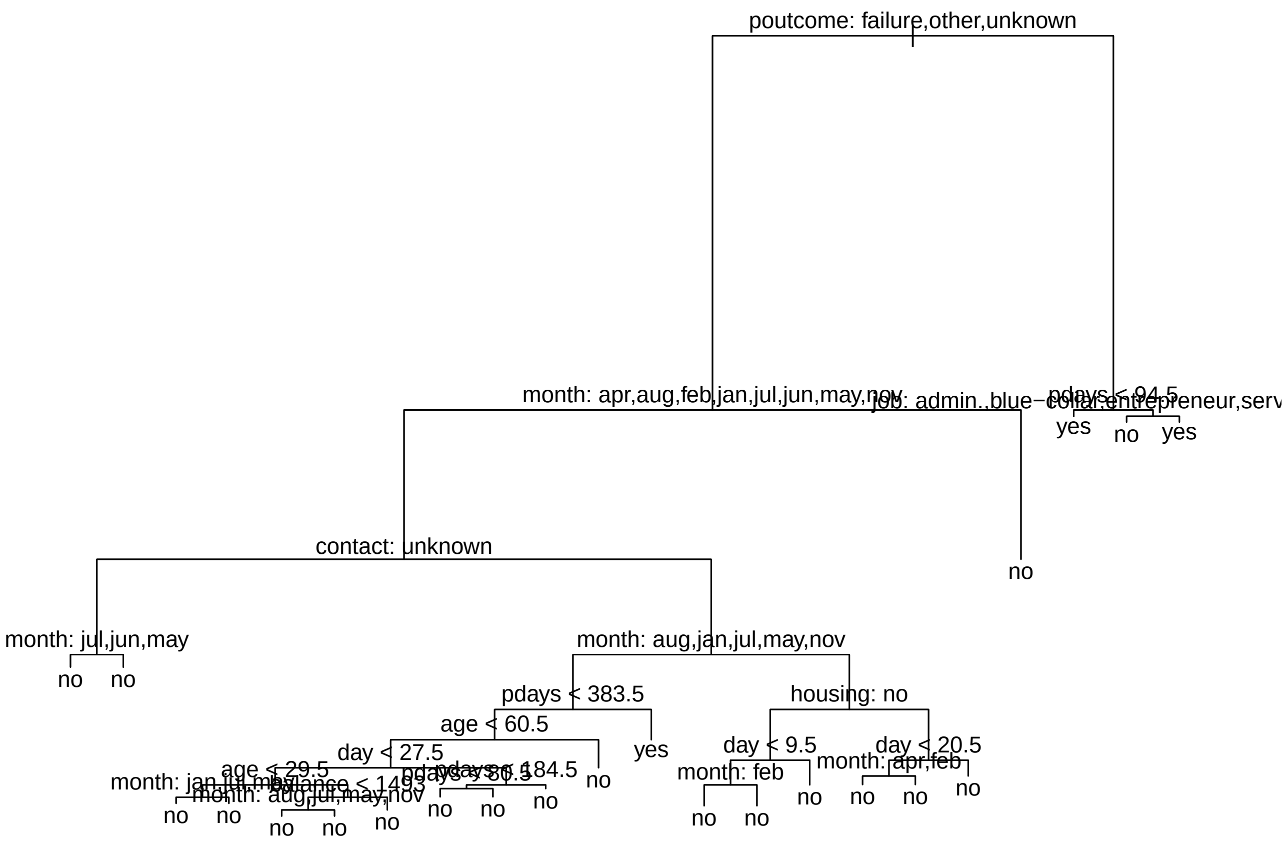


Figure 4: Tree with 22 leaves.

In figure 4 most of the leaves have the prediction "no". The leaves with the predictions "yes" used the variables: poutcome, month, contact, pdays, and job. These 5 variables seems to be the most important for decision making for this tree. The tree structure is presented in the following output:

```
best_fit
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##    1) root 18084 12850.00 no ( 0.88576 0.11424 )
##      2) poutcome: failure,other,unknown 17468 11030.00 no ( 0.90422 0.09578 )
##        4) month: apr,aug,feb,jan,jul,jun,may,nov 16828  9772.00 no ( 0.91520 0.08480 )
##          8) contact: unknown 5130  1599.00 no ( 0.96374 0.03626 )
##           16) month: jul,jun,may 5074  1502.00 no ( 0.96610 0.03390 ) *
##           17) month: apr,aug,feb,jan,nov 56    62.98 no ( 0.75000 0.25000 ) *
##          9) contact: cellular,telephone 11698  7914.00 no ( 0.89391 0.10609 )
##           18) month: aug,jan,jul,may,nov 9284  5503.00 no ( 0.91265 0.08735 )
##             36) pdays < 383.5 9246  5373.00 no ( 0.91510 0.08490 )
##               72) age < 60.5 9097  5107.00 no ( 0.91920 0.08080 )
##                144) day < 27.5 7670  4588.00 no ( 0.91147 0.08853 )
##                  288) age < 29.5 754   637.10 no ( 0.85013 0.14987 )
##                    576) month: jan,jul,may 681   528.00 no ( 0.86931 0.13069 ) *
##                    577) month: aug,nov 73    92.46 no ( 0.67123 0.32877 ) *
##                  289) age > 29.5 6916  3918.00 no ( 0.91816 0.08184 )
##                    578) balance < 1493 5180  2635.00 no ( 0.92973 0.07027 )
##                     1156) month: aug,jul,may,nov 5164  2596.00 no ( 0.93087 0.06913 ) *
##                     1157) month: jan 16    21.93 no ( 0.56250 0.43750 ) *
##                    579) balance > 1493 1736  1249.00 no ( 0.88364 0.11636 ) *
##                145) day > 27.5 1427   472.40 no ( 0.96076 0.03924 )
##                  290) pdays < 184.5 1308   462.50 no ( 0.95719 0.04281 )
##                    580) pdays < 80.5 1277   402.60 no ( 0.96319 0.03681 ) *
##                    581) pdays > 80.5 31    37.35 no ( 0.70968 0.29032 ) *
##                  291) pdays > 184.5 119     0.00 no ( 1.00000 0.00000 ) *
##               73) age > 60.5 149   190.10 no ( 0.66443 0.33557 ) *
##             37) pdays > 383.5 38    47.40 yes ( 0.31579 0.68421 ) *
##           19) month: apr,feb,jun 2414  2262.00 no ( 0.82187 0.17813 )
##             38) housing: no 1123  1321.00 no ( 0.72484 0.27516 )
##               76) day < 9.5 691   668.20 no ( 0.81187 0.18813 )
##                152) month: feb 505   364.20 no ( 0.88317 0.11683 ) *
##                153) month: apr,jun 186   247.30 no ( 0.61828 0.38172 ) *
##               77) day > 9.5 432   586.10 no ( 0.58565 0.41435 ) *
##             39) housing: yes 1291   803.20 no ( 0.90627 0.09373 )
##               78) day < 20.5 1210   655.90 no ( 0.92314 0.07686 )
##                156) month: apr,feb 1154   565.70 no ( 0.93328 0.06672 ) *
##                157) month: jun 56    67.01 no ( 0.71429 0.28571 ) *
##               79) day > 20.5 81   104.40 no ( 0.65432 0.34568 ) *
##        5) month: dec,mar,oct,sep 640   852.70 no ( 0.61562 0.38438 ) *
##      3) poutcome: success 616   806.40 yes ( 0.36201 0.63799 )
##        6) pdays < 94.5 170   185.50 yes ( 0.23529 0.76471 ) *
##        7) pdays > 94.5 446   603.90 yes ( 0.41031 0.58969 )
##         14) job: admin.,blue-collar,entrepreneur,services,technician 213   295.20 no ( 0.50704 0.4929
##         15) job: housemaid,management,retired,self-employed,student,unemployed,unknown 233   292.80 y
```

From the output the three leaves that predicts “yes” does it with the probabilities 0.68421, 0.76471, and 0.67811(last row). In the second to last row, the leaf predicts no with a probability of 50.7%. On the row starting with “291)”, we have that the model predicts no with 100% probability for 119 observations.

2.4 Question 2.4

Question: 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.

Answer: The confusion matrix is estimated for the test data and is presented in the following output:

```r
# Confusion matrix
pred_test <- predict(best_fit, newdata=test, type="class")
conf_mat_tree <- table(True = test$y, Predicton = pred_test)
conf_mat_tree
```

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

The accuracy of the model on test data is around 89%.

```r
# Accuracy
acc_test <- sum(diag(conf_mat_tree)) / sum(conf_mat_tree)
acc_test
```

```
## [1] 0.8910351
```

The F1-score test data is around 0.2246.

```r
# F1-score
# True positive
tp <- conf_mat_tree[2,2]
# False positive
fp <- conf_mat_tree[1,2]
# False negative
fn <- conf_mat_tree[2,1]

precision <- tp / (tp+fp)
recall <- tp / (tp+fn)
# F-score is between 0 and 1, where the closer to 1, the better.
f1_score <- 2 * (precision*recall) / (precision + recall)
f1_score
```

```
## [1] 0.224554
```

The accuracy of the model is around 89%, which looks high. But examining the class “yes”, the model only predicted around 13.5% (214/(214+1371)) of “yes” observations as “yes. Confusion matrix indicate that we have unbalanced data, which makes the model able to predict”no” (the dominant class) better than “yes”. F1-score focuses more on the prediction “yes” compared to accuracy. In this case we have really unbalanced classes and F1-score is therefore a better measurement. F1-score is between 0 and 1, where the closer F1-score is to 1 the better. The model has a F1-score of 0.22 and is not considered to have good predictive power.

2.5 Question 2.5

Question: Perform a decision tree classification of the test data with the following loss matrix,

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

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.

Answer: From the slides from the lectures we get the following from the loss matrix:

$E(y = no, \hat{y} = yes) = 1$
$E(y = yes, \hat{y} = no) = 5$

We can use this for classification of the model with:

$\frac{p(y=no|x)}{p(y=yes|x)} > 5 \rightarrow classify\ as\ no.$

The code used in R for this is as follows:

```
# Predicts the probability of each class.
pred_best <- predict(best_fit, test, type="vector")
prediction <- as.vector(ifelse(pred_best[,1]/pred_best[,2] > 5, "no", "yes"))
table(True = test$y, Prediction = prediction)
```

```
##      Prediction
## True     no   yes
##   no  11030   949
##   yes   771   814
```

Compared to the confusion matrix in 2.4, the model has now more predictions for the class “yes”. The loss matrix for the model now says that a wrong prediction on a class that is “yes” is 5 times more severe than a wrong prediction on a class that is “no”. Before the model predicted “yes” if the probability was above 50%, now the model predicts “yes” if the probability is above 16.7% (100/6).

2.6 Question 2.6

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

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

where $\pi = 0.05, 0.1, 0.15, ..., 0.9, 0.95$. 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?

Answer: True positive rate (TPR) and false positive rate (FPR) were calculated and the ROC curves for tree and logistic regression model are presented in figure 5.

```
# ROC for tree
d <- data.frame(TPR = 1, FPR = 1)
roc_logistic <- d[FALSE, ]
roc_tree <- d[FALSE, ]

pred_best <- predict(best_fit, test, type="vector")
```

```r
probs <- seq(from=0.05, to=0.95, by=0.05)
for(i in 1:length(probs)){
  prediction <- ifelse(pred_best[,2] > probs[i], "yes", "no")
  conf_mat <- table(test$y, prediction)
  if(dim(conf_mat)[2] == 1){
    TP <- 0
    FP <- 0
  } else {
    TP <- conf_mat[2,2]
    FP <- conf_mat[1,2]
  }
  TN <- conf_mat[1,1]
  FN <- conf_mat[2,1]

  # TPR, true positive rate
  roc_tree[i,1] <- TP / (TP+FN)
  # FPR, false positive rate
  roc_tree[i,2] <- FP / (FP+TN)
}
plot_data <- data.frame(roc_tree)

# ROC for logistic
model_logistic <- glm(y~., family="binomial", train)
pred_logistic <- predict(model_logistic, test, type="response")
d <- data.frame(TPR = 1, FPR = 1)
roc_logistic <- d[FALSE, ]
probs <- seq(from=0.05, to=0.95, by=0.05)
roc_logistic <- d[FALSE, ]

for(i in 1:length(probs)){
  pred <- ifelse(pred_logistic > probs[i], "yes", "no")
  conf_mat <- table(test$y, pred)
  TP <- conf_mat[2,2]
  TN <- conf_mat[1,1]
  FN <- conf_mat[2,1]
  FP <- conf_mat[1,2]

  # TPR, true positive rate
  roc_logistic[i,1] <- TP / (TP+FN)
  # FPR, false positive rate
  roc_logistic[i,2] <- FP / (FP+TN)
}
plot_data_logistic <- data.frame(roc_logistic)
```

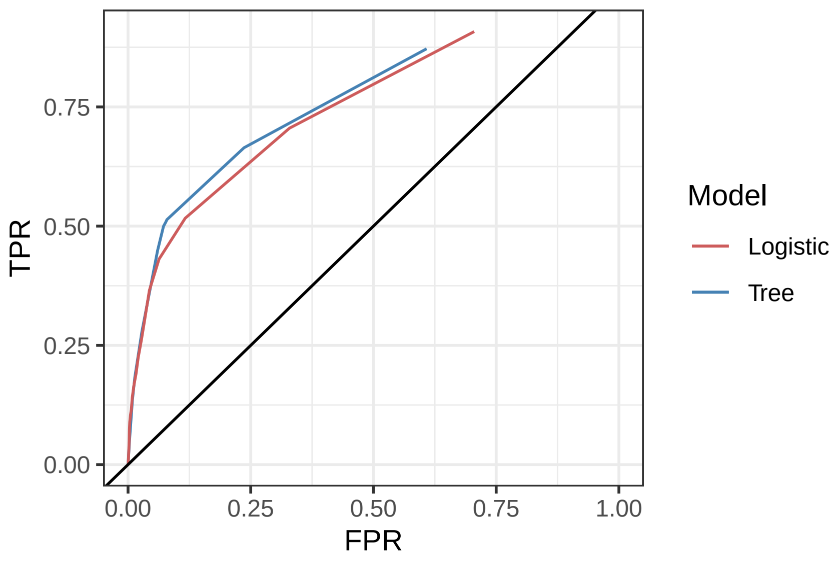


Figure 5: Roc curves for tree and logistic model.

In figure 5 the black line represents the theoretical value for a random classifier. A model with a ROC curve further away from this line towards upper left part of the plot is considered better. The tree model is therefore considered to be better than the logistic model.

In precision-recall curve, recall is defined the same as TPR in ROC curve and precision is defined as: $precision = \frac{TP}{TP+FP}$, where TP is true positive and FP is false positive. Since we have unbalanced data, where the class “yes” is a lot smaller, a precision-recall curve can give us better information on how well the models predicts the class “yes”.

3 Question 3: Principal Components and Implicit Regularization

The data file communities.csv contains the results of studies of the crime level in the united states based on various characteristics of the given location. The main variable that is studied is ViolentCrimesPerPop which represents the total number of violent crimes per 100K population. The meaning of other variables can be found at: https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime

```
df3 <- read.csv("communities.csv")
```

3.1 part 1)

Question: Scale all variables except for 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?
Answer:

```
# Scale the data expect for the predictor column
scaler <- preProcess(df3[, -which(names(df3) == "ViolentCrimesPerPop")])
df3_S <- predict(scaler, df3)
cat("row number:", dim(df3)[1], ", column number:", dim(df3)[2])
```

```
## row number: 1994 , column number: 101
```

```
# Implement PCA
X <- as.matrix(df3_S)
n <- nrow(df3_S)
S <- (t(X) %*% X) / n
eigenval <- eigen(S)$values

sorted_proportions <- sort(eigenval / sum(eigenval), decreasing = TRUE)
variance_explained <- cumsum(sorted_proportions) / sum(sorted_proportions)

cat("Number of components to explain 95% variance:", which(variance_explained >= 0.95)[1], "\n",
    "Proportion of variation explained by PC1:",sorted_proportions[1], "\n",
    "Proportion of variation explained by PC2:",sorted_proportions[2], "\n")
```

```
## Number of components to explain 95% variance: 35
##  Proportion of variation explained by PC1: 0.2501082
##  Proportion of variation explained by PC2: 0.1692125
```

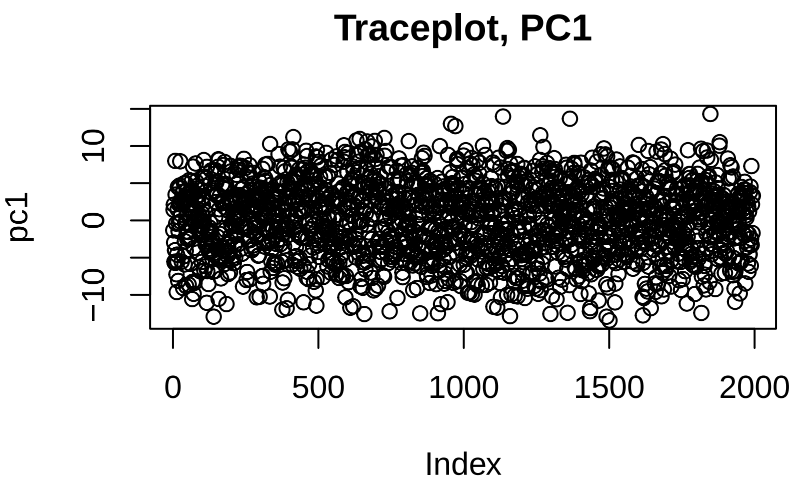
35 components are need to explain 95% variance. Proportion of the variance explained by first and second principal component are 0.250108 and 0.1692125 relatively.

3.2 part 2)

Question: 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 feature 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)

Answer: Trace plot of the first principle component can be seen below. Many features have a notable contribution since this component captures the most variance in the data.

```
pca_result <- princomp(df3_S)
pc1 <- pca_result$scores[,1]
plot(pc1, main = 'Traceplot, PC1')
```



```
loadings_pc1 <- pca_result$loadings[, 1]
top_5_features_pc1 <- names(sort(abs(loadings_pc1), decreasing = TRUE))[1:5]
cat("The top 5 features contribute mostly by the absolute value in PC1: \n",
    top_5_features_pc1[1], "\n",
    top_5_features_pc1[2], "\n",
    top_5_features_pc1[3], "\n",
    top_5_features_pc1[4], "\n",
    top_5_features_pc1[5], "\n")
```

```
## The top 5 features contribute mostly by the absolute value in PC1:
##  medFamInc
##  medIncome
##  PctKids2Par
##  pctWInvInc
##  PctPopUnderPov
```

The meanings of the variables are as follows:

- ViolentCrimesPerPop: total number of violent crimes per 100K popuation (numeric - decimal)
- medFamInc: median family income (differs from household income for non-family households) (numeric - decimal)
- medIncome: median household income (numeric - decimal)
- PctKids2Par: percentage of kids in family housing with two parents (numeric - decimal)
- pctWInvInc: percentage of households with investment / rent income in 1989 (numeric - decimal)
- PctPopUnderPov: percentage of people under the poverty level (numeric - decimal)

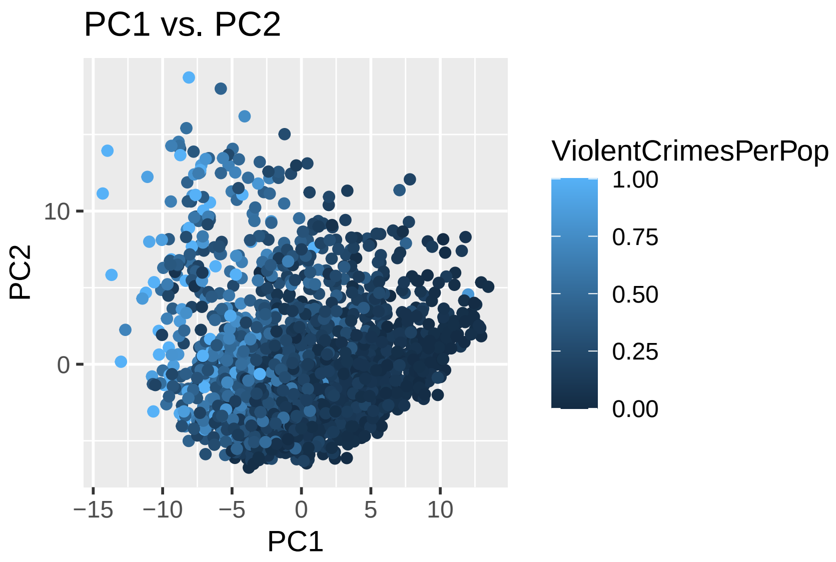
“PctPopUnderPov” and “medIncome” might be highly correlated because poverty level is generally related to income. Also percentage of kids in family housing with two parents (PctKids2Par) might be highly correlated to “medIncome” as well because if two of the parents are working, it might increase the median income. Almost all of the variables are about the economic status of the region and there’s a high possibility that these predictor variables are correlated to each other.

“medFamInc”, “medIncome”, ” PctKids2Par”, “pctWInvInc” have a negative, “PctPopUnderPov” has a positive correlation with “ViolentCrimesPerPop”. In general, it can be observed that as the income level increases, crime rate decreases.

```r
loadings_pc2 <- pca_result$loadings[, 2]
top_5_features_pc2 <- names(sort(abs(loadings_pc2), decreasing = TRUE))[1:5]
cat("The top 5 features contribute mostly by the absolute value in PC2: \n",
    top_5_features_pc2[1], "\n",
    top_5_features_pc2[2], "\n",
    top_5_features_pc2[3], "\n",
    top_5_features_pc2[4], "\n",
    top_5_features_pc2[5], "\n")
```

```
## The top 5 features contribute mostly by the absolute value in PC2:
##  PctRecImmig10
##  PctRecImmig8
##  PctRecImmig5
##  PctRecentImmig
##  PctForeignBorn
```

```r
pc_scores <- as.data.frame(pca_result$scores[, 1:2])
pc_scores$ViolentCrimesPerPop <- df3_S$ViolentCrimesPerPop

ggplot(pc_scores, aes(x = -Comp.1, y = -Comp.2, color = ViolentCrimesPerPop)) +
  geom_point() +
  labs(x = "PC1", y = "PC2", color = "ViolentCrimesPerPop") +
  ggtitle("PC1 vs. PC2")
```



Plot of the PC scores in the coordinates (PC1, PC2) can be seen above. The sign or direction of the principal components in the PCA is arbitrary because it only cares about capturing the maximum variance in the data hence signs of the principal components are flipped to interpret the results in a more meaningful way.

4 variables of out 5 that contribute mostly to the PC1 has negative correlation with the target variable "ViolentCrimesPerPop". PC1 is related to economic status of the region and it can observed that ViolentCrimesPerPop is highest (almost 1) when PC1 is low. PC2 is about immigration rates and from the graph it can be observed that" ViolentCrimesPerPop" is almost zero when PC2 is low but there are points where "ViolentCrimesPerPop" is low even though PC2 is high and these points are in the region where PC1 is positive. It can be observed that PC1 is more dominant which means despite the high immigration rate is high, crime rate is low in the states where income is high.

3.3 part 3)

Question: 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 the model.

Answer:

```
library(caret)
# Split the data
n <- dim(df3)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train_data <- df3[id,]
test_data <- df3[-id,]

# Scale the data
scaler <- preProcess(train_data)
trainS <- predict(scaler, train_data)
testS <- predict(scaler, test_data)
```

```
# Fit the lm model
lm_model <- lm(ViolentCrimesPerPop ~ ., data = trainS)

# Predictions on the train & test data
predictor_cols <- setdiff(names(train_data), "ViolentCrimesPerPop")
trainS_x <- trainS[, predictor_cols]
testS_x <- testS[, predictor_cols]
y_train <- trainS$ViolentCrimesPerPop
y_test <- testS$ViolentCrimesPerPop

predS_train <- predict(lm_model, newdata = trainS_x)
predS_test <- predict(lm_model, newdata = testS_x)

mse_train <- mean((y_train - predS_train)^2)
mse_test <- mean((y_test - predS_test)^2)

cat("Mean Squared Error (MSE) on the training data:", mse_train, "\n")
```

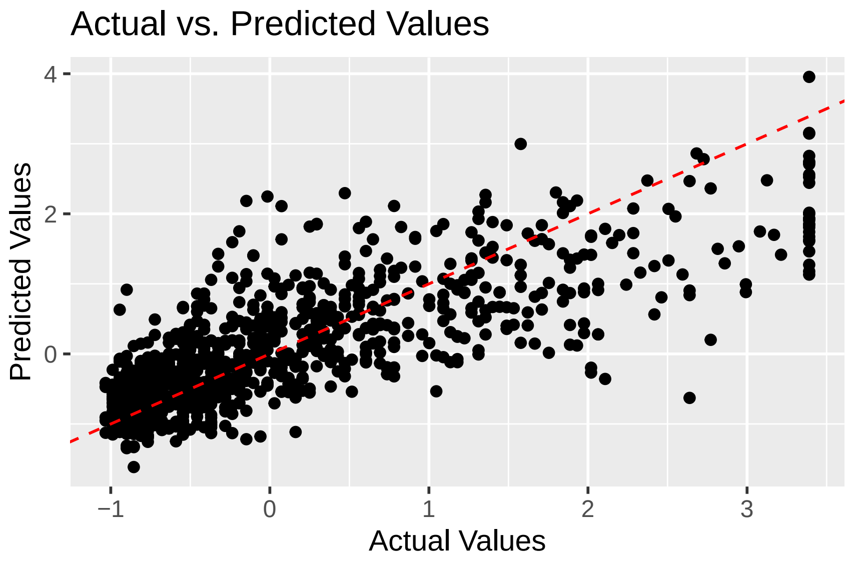
```
## Mean Squared Error (MSE) on the training data: 0.2752071
```

```
cat("Mean Squared Error (MSE) on the test data:", mse_test, "\n")
```

```
## Mean Squared Error (MSE) on the test data: 0.4248011
```

By looking at the graph below, it can be observed that the quality of the model is not good. The mean of the scaled predictor variable on the test data is -0.01699304 and MSE on the test data is 0.4248011. MSE is quite high, approximately 25 times of the mean of the predictor variable.

```
library(ggplot2)
data <- data.frame(Actual = y_test, Predicted = predS_test)
ggplot(data, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(x = "Actual Values", y = "Predicted Values") +
  ggtitle("Actual vs. Predicted Values")
```



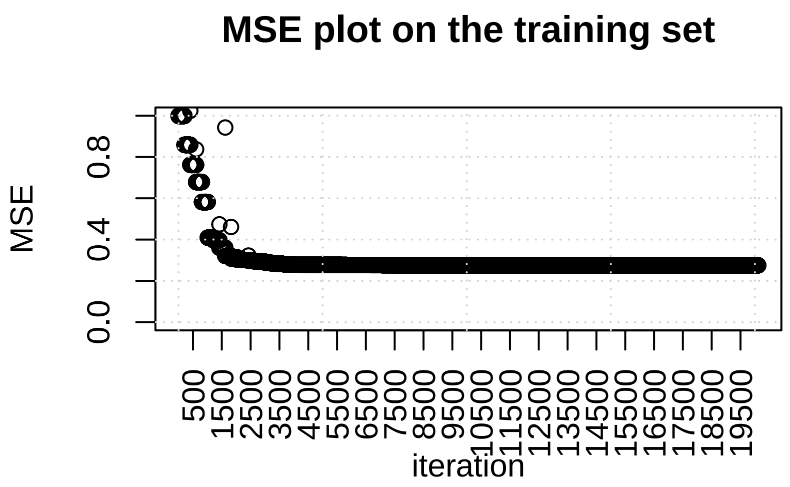
3.4 part 4)

Question: 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 (optim() function without gradient specified) to optimize cost with starting point $\theta^0 = 0$ 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 tho the early stopping criterion. Compute the training and test error in the optimal model, compare them with results in step 3 and make conclusions.

Answer:

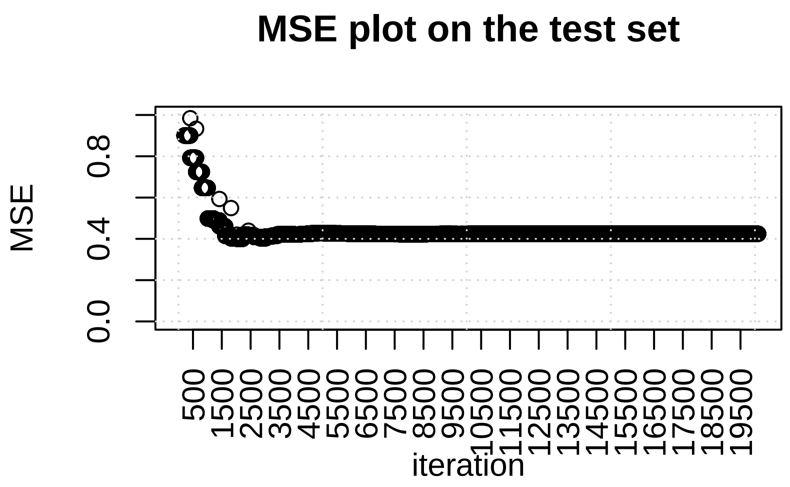
```
train_mse <- list()
Params <- list()
test_mse <- list()
k <- 0
cost_function <- function(theta){
  y_hat_train <- as.matrix(trainS_x) %*% as.matrix(theta)
  f <- mean((y_train - y_hat_train)^2)
  k <<- k+1
  train_mse[[k]] <<- f
  Params[[k]] <<- theta

  y_hat_test <- as.matrix(testS_x) %*% as.matrix(theta)
  test_mse[[k]] <<- mean((y_test - y_hat_test)^2)
  return(f)
}

theta_init <- c(rep(0, ncol(trainS_x)))
result <- optim(theta_init, fn=cost_function, method="BFGS")
```

```
mse_val_train <- train_mse[1:length(train_mse)]
iter <- seq(1,length(train_mse))
plot(iter, mse_val_train, main = "MSE plot on the training set", xlab = "iteration",
     ylab = "MSE", ylim = c(0,1), xaxt="n")
axis(1, at = seq(500, 20000, by = 1000), las=2)
grid()
```



Approximately 1500 iteration is optimal according tho the early stopping criterion. After 1500, there is no significant decrease in the MSE.

```
mse_val_test <- test_mse[1:length(test_mse)]
iter <- seq(1,length(test_mse))
plot(iter, mse_val_test, main = "MSE plot on the test set",  xlab = "iteration",
     ylab = "MSE", ylim = c(0,1), xaxt="n")
axis(1, at = seq(500, 20000, by = 1000), las=2)
grid()
```



```r
# Get the results from early stopping condition
cat("MSE on the training data:", train_mse[[1500]], "\n")
```

```
## MSE on the training data: 0.3607042
```

```r
cat("MSE on the test data:", test_mse[[1500]],  "\n")
```

```
## MSE on the test data: 0.4611746
```

The MSE on the test data with the parameters at iteration 1500 is 0.4611746, it is slightly high compared to step 3. Finding the optimal parameters with BFGS method did not give a better result. For the linear regression, there is a closed formula to find the optimal parameters which is $\hat{\theta} = (X^T X)^{-1} X^T y$ hence solving this problem with this approach is not efficient. Finding the optimal parameters iteratively is generally being used in the problems where cost function is nonlinear.

4 Statement of Contribution

We worked on the assignment individually for the computer labs (to be more efficient when asking questions), William on task 1, Duc on task 2, and Simge on task 3. We later solved all assignment individually and compared and discussed our solutions before dividing the task of writing the laboration report.

4.1 Question 1

Text written by William.

4.2 Question 2

Text written by Duc.

4.3 Question 3

Text written by Simge.

5 Appendix

The code used in this laboration report are summarised in the code as follows:

```r
library(ggplot2)
library(tree)
library(knitr)
library(dplyr)
library(kableExtra)
knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 4.5,
  fig.height = 3)
set.seed(12345)


# Read in data
tecator <- read.csv("tecator.csv")


# Partitioning training data (50%)
n <- dim(tecator)[1]
id <- sample(1:n, floor(n*0.5))
tecator_train <- tecator[id,]


# Partitioning test data (50%)
tecator_test <- tecator[-id,]



# column 1, 103 and 104 are sample, moisture and protein and should
# not be used as features
lm_model <- lm(Fat ~ ., data = tecator_train[,-c(1,103:104)])



pred_train <- predict(lm_model, newdata = tecator_train)
pred_test <- predict(lm_model, newdata = tecator_test)

mse_train <- mean((tecator_train$Fat - pred_train)^2)
mse_test <- mean((tecator_test$Fat - pred_test)^2)


df <- data.frame("MSE" = c(mse_train, mse_test))
rownames(df) <- c("Train","Test")

knitr::kable(df, row.names = TRUE, booktabs=T, caption = "Training and test errors", digits = 4) %>%
    kable_styling(latex_options = "HOLD_position")

library(caret)
library(glmnet)

set.seed(12345)

# Asked lab assistents if we would scale the data
# They said no

# scaler <- preProcess(tecator_train)
# data1 <- predict(scaler, tecator_train)
covariates <- tecator_train[,2:101]
response <- tecator_train[, 102]

model_lasso <- glmnet(as.matrix(covariates),response, alpha = 1,
                      family="gaussian")

plot(model_lasso, xvar="lambda", label=TRUE)






model_ridge <- glmnet(as.matrix(covariates), response, alpha = 0,
                      family = "gaussian")

plot(model_ridge, xvar="lambda", label=TRUE)


model_lasso_cv <- cv.glmnet(as.matrix(covariates),response, alpha = 1,
                   family = "gaussian")
plot(model_lasso_cv)
```

```r
coef_cv <- coef(model_lasso_cv, s="lambda.min")



index <- coef_cv@i
variables <- coef_cv@Dimnames[[1]][index+1]
value <- coef_cv@x

df <- data.frame("Variables" = variables, "Coefficient" = value)

knitr::kable(df, row.names = FALSE, booktabs=T,
             caption = "Variables that were chosen in this LASSO model",digits = 5)%>%
      kable_styling(latex_options = "HOLD_position")

model_lasso_cv$lambda.min


library(ggplot2)

y <- tecator_test[,"Fat"]
ynew <- predict(model_lasso_cv, newx = as.matrix(tecator_test[, 2:101]),
                type = "response")


ggplot(data.frame(y,ynew), aes(y, lambda.1se)) + geom_point() + theme_bw() +
  labs(x = "True Y",
       y = "Predicted Y by LASSO model") +
  geom_smooth(method='lm',formula = y ~ x,se = FALSE )


# Index 12 is the variable duration
data <- read.csv2("bank-full.csv", stringsAsFactors = TRUE)[, -12]

# Data partitioning (40/30/30)
# Training data
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]

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

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

```r
# Tree a: default setting
tree_a <- tree(y ~., data=train)
# Misclassification for training data for tree a
pred_train <- predict(tree_a, newdata=train, type="class")
table_train <- table(train$y, pred_train)
miss_train <- 1 - (sum(diag(table_train)) / sum(table_train))
# Misclassification for validation data for tree a
pred_valid <- predict(tree_a, newdata=valid, type="class")
table_valid <- table(valid$y, pred_valid)
miss_valid <- 1 - (sum(diag(table_valid)) / sum(table_valid))
# Number of leaves
leaf <- sum(tree_a$frame$var == "<leaf>")
# Misclassification for tree a
model_a <- c(miss_train, miss_valid, leaf)



# Tree b: smallest size changed from 10 (default) to 7 000
tree_b <- tree(y ~., data=train, control=tree.control(nobs = nrow(train),
                                                      minsize = 7000))
# Misclassification for training data for tree b
pred_train <- predict(tree_b, newdata=train, type="class")
table_train <- table(train$y, pred_train)
miss_train <- 1 - (sum(diag(table_train)) / sum(table_train))
# Misclassification for validation data for tree b
pred_valid <- predict(tree_b, newdata=valid, type="class")
table_valid <- table(valid$y, pred_valid)
miss_valid <- 1 - (sum(diag(table_valid)) / sum(table_valid))
# Number of leaves
leaf <- sum(tree_b$frame$var == "<leaf>")
# Misclassification for tree b
model_b <- c(miss_train, miss_valid, leaf)


# Tree c: mindev changed from 0.01 (default) to 0.0005
tree_c <- tree(y ~., data=train, control=tree.control(nobs = nrow(train),
                                                      mindev = 0.0005))
# Misclassification for training data for tree c
pred_train <- predict(tree_c, newdata=train, type="class")
table_train <- table(train$y, pred_train)
miss_train <- 1 - (sum(diag(table_train)) / sum(table_train))
# Misclassification for validation data for tree c
pred_valid <- predict(tree_c, newdata=valid, type="class")
table_valid <- table(valid$y, pred_valid)
miss_valid <- 1 - (sum(diag(table_valid)) / sum(table_valid))
# Number of leaves
leaf <- sum(tree_c$frame$var == "<leaf>")
# Misclassification for tree c
model_c <- c(miss_train, miss_valid, leaf)
```

```r
# Summarised results
table <- data.frame(rbind(model_a, model_b, model_c))
colnames(table) <- c("Training error", "Validation error", "Number of leaves")
rownames(table) <- c("Tree a: default setting", "Tree b: smallest node 7000",
                     "Tree c: min deviance 0.0005")

kable(table, digits=6, booktabs=T,
      caption="Misclassification error for the three trees on training and validation data.") %>%
   kable_styling(latex_options = "HOLD_position")
fit <- tree(y ~., data=train,
            control=tree.control(nobs = nrow(train),
                                 mindev = 0.0005))
trainScore=rep(0,50)
validScore=rep(0,50)
for(i in 2:50){
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  validScore[i]=deviance(pred)
}
# Finds min, add +1 since index 1 is tree with 2 leaves.
which(min(validScore[2:50]) == validScore[2:50])+1
plot_data <- data.frame(training = trainScore[-1], valid = validScore[-1])
ggplot(plot_data, aes(x=2:50)) +
  geom_point(aes(y=training, color="training")) +
  geom_point(aes(y=valid, color="test")) +
  geom_vline(xintercept=22, linetype="dashed") +
  ylim(6000, 12000) +
  labs(x="Number of leaves", y="Deviance") +
  scale_colour_manual(name="Data",
                      values=c("indianred", "steelblue"),
                      labels=c("Validation", "Training")) +
  theme_bw()


best_fit <- prune.tree(fit, best=22)
plot(best_fit)
text(best_fit, pretty=0)
best_fit
# Confusion matrix
pred_test <- predict(best_fit, newdata=test, type="class")
conf_mat_tree <- table(True = test$y, Predicton = pred_test)
conf_mat_tree

# Accuracy
acc_test <- sum(diag(conf_mat_tree)) / sum(conf_mat_tree)
acc_test
# F1-score
# True positive
tp <- conf_mat_tree[2,2]
# False positive
fp <- conf_mat_tree[1,2]
# False negative
fn <- conf_mat_tree[2,1]

precision <- tp / (tp+fp)
recall <- tp / (tp+fn)
# F-score is between 0 and 1, where the closer to 1, the better.
f1_score <- 2 * (precision*recall) / (precision + recall)
f1_score
# Predicts the probability of each class.
pred_best <- predict(best_fit, test, type="vector")
prediction <- as.vector(ifelse(pred_best[,1]/pred_best[,2] > 5, "no", "yes"))
table(True = test$y, Prediction = prediction)
# ROC for tree
d <- data.frame(TPR = 1, FPR = 1)
roc_logistic <- d[FALSE, ]
roc_tree <- d[FALSE, ]

pred_best <- predict(best_fit, test, type="vector")
probs <- seq(from=0.05, to=0.95, by=0.05)
for(i in 1:length(probs)){
  prediction <- ifelse(pred_best[,2] > probs[i], "yes", "no")
  conf_mat <- table(test$y, prediction)
  if(dim(conf_mat)[2] == 1){
    TP <- 0
    FP <- 0
  } else {
    TP <- conf_mat[2,2]
    FP <- conf_mat[1,2]
  }
  TN <- conf_mat[1,1]
  FN <- conf_mat[2,1]

  # TPR, true positive rate
  roc_tree[i,1] <- TP / (TP+FN)
  # FPR, false positive rate
  roc_tree[i,2] <- FP / (FP+TN)
}
plot_data <- data.frame(roc_tree)

# ROC for logistic
model_logistic <- glm(y~., family="binomial", train)
pred_logistic <- predict(model_logistic, test, type="response")
d <- data.frame(TPR = 1, FPR = 1)
roc_logistic <- d[FALSE, ]
probs <- seq(from=0.05, to=0.95, by=0.05)
```

```r
roc_logistic <- d[FALSE, ]

for(i in 1:length(probs)){
  pred <- ifelse(pred_logistic > probs[i], "yes", "no")
  conf_mat <- table(test$y, pred)
  TP <- conf_mat[2,2]
  TN <- conf_mat[1,1]
  FN <- conf_mat[2,1]
  FP <- conf_mat[1,2]

  # TPR, true positive rate
  roc_logistic[i,1] <- TP / (TP+FN)
  # FPR, false positive rate
  roc_logistic[i,2] <- FP / (FP+TN)
}
plot_data_logistic <- data.frame(roc_logistic)
ggplot(plot_data) +
  geom_line(aes(x=FPR, y=TPR, color="Tree")) +
  geom_line(data=plot_data_logistic, aes(x=FPR, y=TPR, color="Logistic")) +
  scale_x_continuous(limits=c(0,1)) +
  geom_abline(intercept = 0, slope = 1) +
  scale_color_manual(name="Model",
                     values=c("indianred", "steelblue"),
                     labels=c("Logistic", "Tree")) +
  theme_bw()

df3 <- read.csv("communities.csv")
# Scale the data expect for the predictor column
scaler <- preProcess(df3[, -which(names(df3) == "ViolentCrimesPerPop")])
df3_S <- predict(scaler, df3)
cat("row number:", dim(df3)[1], ", column number:", dim(df3)[2])
# Implement PCA
X <- as.matrix(df3_S)
n <- nrow(df3_S)
S <- (t(X) %*% X) / n
eigenval <- eigen(S)$values

sorted_proportions <- sort(eigenval / sum(eigenval), decreasing = TRUE)
variance_explained <- cumsum(sorted_proportions) / sum(sorted_proportions)

cat("Number of components to explain 95% variance:", which(variance_explained >= 0.95)[1], "\n",
    "Proportion of variation explained by PC1:",sorted_proportions[1], "\n",
    "Proportion of variation explained by PC2:",sorted_proportions[2], "\n")
pca_result <- princomp(df3_S)
pc1 <- pca_result$scores[,1]
plot(pc1, main = 'Traceplot, PC1')
loadings_pc1 <- pca_result$loadings[, 1]
top_5_features_pc1 <- names(sort(abs(loadings_pc1), decreasing = TRUE))[1:5]
cat("The top 5 features contribute mostly by the absolute value in PC1: \n",
    top_5_features_pc1[1], "\n",
    top_5_features_pc1[2], "\n",
    top_5_features_pc1[3], "\n",
    top_5_features_pc1[4], "\n",
    top_5_features_pc1[5], "\n")
loadings_pc2 <- pca_result$loadings[, 2]
top_5_features_pc2 <- names(sort(abs(loadings_pc2), decreasing = TRUE))[1:5]
cat("The top 5 features contribute mostly by the absolute value in PC2: \n",
    top_5_features_pc2[1], "\n",
    top_5_features_pc2[2], "\n",
    top_5_features_pc2[3], "\n",
    top_5_features_pc2[4], "\n",
    top_5_features_pc2[5], "\n")
pc_scores <- as.data.frame(pca_result$scores[, 1:2])
pc_scores$ViolentCrimesPerPop <- df3_S$ViolentCrimesPerPop

ggplot(pc_scores, aes(x = -Comp.1, y = -Comp.2, color = ViolentCrimesPerPop)) +
  geom_point() +
  labs(x = "PC1", y = "PC2", color = "ViolentCrimesPerPop") +
  ggtitle("PC1 vs. PC2")
library(caret)
# Split the data
n <- dim(df3)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train_data <- df3[id,]
test_data <- df3[-id,]

# Scale the data
scaler <- preProcess(train_data)
trainS <- predict(scaler, train_data)
testS <- predict(scaler, test_data)
# Fit the lm model
lm_model <- lm(ViolentCrimesPerPop ~ ., data = trainS)

# Predictions on the train & test data
predictor_cols <- setdiff(names(train_data), "ViolentCrimesPerPop")
trainS_x <- trainS[, predictor_cols]
testS_x <- testS[, predictor_cols]
y_train <- trainS$ViolentCrimesPerPop
y_test <- testS$ViolentCrimesPerPop

predS_train <- predict(lm_model, newdata = trainS_x)
predS_test <- predict(lm_model, newdata = testS_x)

mse_train <- mean((y_train - predS_train)^2)
mse_test <- mean((y_test - predS_test)^2)

cat("Mean Squared Error (MSE) on the training data:", mse_train, "\n")
cat("Mean Squared Error (MSE) on the test data:", mse_test, "\n")
library(ggplot2)
data <- data.frame(Actual = y_test, Predicted = predS_test)
ggplot(data, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(x = "Actual Values", y = "Predicted Values") +
  ggtitle("Actual vs. Predicted Values")
train_mse <- list()
Params <- list()
test_mse <- list()
k <- 0
cost_function <- function(theta){
  y_hat_train <- as.matrix(trainS_x) %*% as.matrix(theta)
  f <- mean((y_train - y_hat_train)^2)
  k <<- k+1
  train_mse[[k]] <<- f
  Params[[k]] <<- theta

  y_hat_test <- as.matrix(testS_x) %*% as.matrix(theta)
  test_mse[[k]] <<- mean((y_test - y_hat_test)^2)
  return(f)
}

theta_init <- c(rep(0, ncol(trainS_x)))
result <- optim(theta_init, fn=cost_function, method="BFGS")
mse_val_train <- train_mse[1:length(train_mse)]
iter <- seq(1,length(train_mse))
plot(iter, mse_val_train, main = "MSE plot on the training set", xlab = "iteration",
     ylab = "MSE", ylim = c(0,1), xaxt="n")
axis(1, at = seq(500, 20000, by = 1000), las=2)
grid()
mse_val_test <- test_mse[1:length(test_mse)]
iter <- seq(1,length(test_mse))
plot(iter, mse_val_test, main = "MSE plot on the test set",  xlab = "iteration",
     ylab = "MSE", ylim = c(0,1), xaxt="n")
axis(1, at = seq(500, 20000, by = 1000), las=2)
grid()
# Get the results from early stopping condition
cat("MSE on the training data:", train_mse[[1500]], "\n")
cat("MSE on the test data:", test_mse[[1500]],  "\n")
```