TDDE01 : Computer lab 2 report

Group B.15

Charlie Elf - chael086 - chael086@student.liu.se
Christoffer Gärdin - chrga084 - chrga084@student.liu.se
Felix Lindgren - felli675 - felli675@student.liu.se

October 26, 2022

Contents

1 Statement Of Contribution

In this lab the main responsibility of the 3 assignments was divided onto the group members. Assignment 1 was held responsible for by Charlie Elf. Assignment 2 was held responsible for by Christoffer Gärdin. Assignment 3 was held responsible for by Felix Lindgren. The load was in that way divided onto the group members. The assignment responsibility included for each group member making sure that code was working to solve the problem and the writing of each assignment in the lab report.

2 Assignment 1 - Explicit Regularization

In this assignment the level of fat in meat samples are to be predicted using non regularized linear regression, ridge regression and LASSO regression. To do this each sample has a 100 channel spectrum of absorbance working as input variables. In total there are 215 samples. These are divided randomly into training and test data with 50/50 proportion.

2.1 Linear regression fit and mean square error

Assuming that the fat content based on the channels as inputs can be modeled as a linear regression. The probabilistic model can be written as:

$$fat \sim N(\theta_0 + \sum_{n=1}^{100} \theta_i channel_i, \sigma^2) \tag{1}$$

A linear regression model is created from the training data with the 100 channels as input variables and the fat-value as output variable. The model is then fitted to the training data and the test data. Mean square error is calculated for the two to evaluate the linear regression model. The resulting errors is seen in Table 1.

Table 1: Table showing the mean square error for training and test data.

Train	Test
0.005709117	722.4294

In the table it is clear that the linear regression model is not a good model for predicting the level of fat as the test error is above 722 compared to training error of about 0.0057.

2.2 Cost function

Another way to model the fat levels could be a LASSO regression model in which a cost function is used in the regularization. The cost function which should be minimized used in this model is:

$$J(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} * \sum_{i=1}^{N} (y_i - \theta_0 - \theta_1 * x_{1j} - ... - \theta_p * x_{pj})^2 \tag{2}$$

2.3 LASSO regression

The LASSO regression was fitted to the training data using the glmnet() function. The penalty factor λ in a LASSO regression is used to limit how much the coefficients affect the prediction. To show how the model coefficients depend on the logarithm of the penalty factor λ changes a plot of $log(\lambda)$ is seen in Figure 1. $\lambda = 0.56$, $(log(\lambda) = -0.25)$

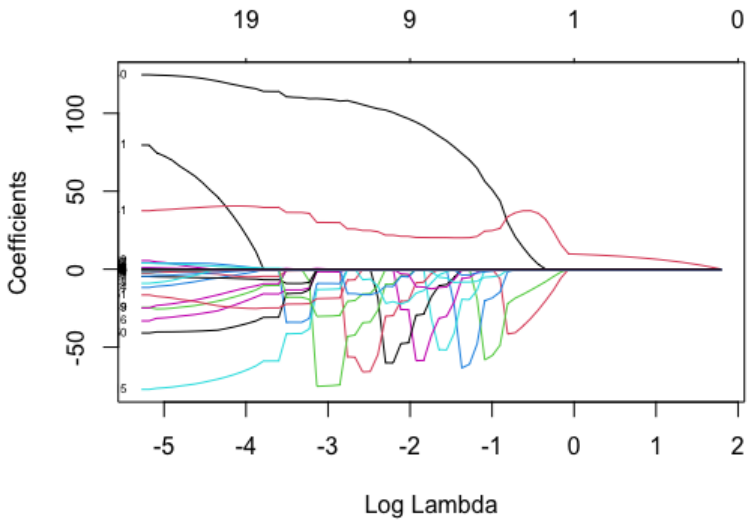

Figure 1: Coefficients dependency on λ for LASSO regression.

In Figure 1 we see that for low values on the penalty factor many coefficients are nonzero. This is reasonable since the penalty is meant to decrease the amount of nonzero coefficients as the penalty increases. For $log(\lambda) = -0.3$ $(\lambda = 0.56)$ the amount of coefficients is reduced to 3.

2.4 Ridge regression

The model is instead of LASSO now fitted using ridge regression which is a way of penalizing the model to gain smaller coefficients. Coefficients dependecy on the penalty factor λ is shown in Figure 2.

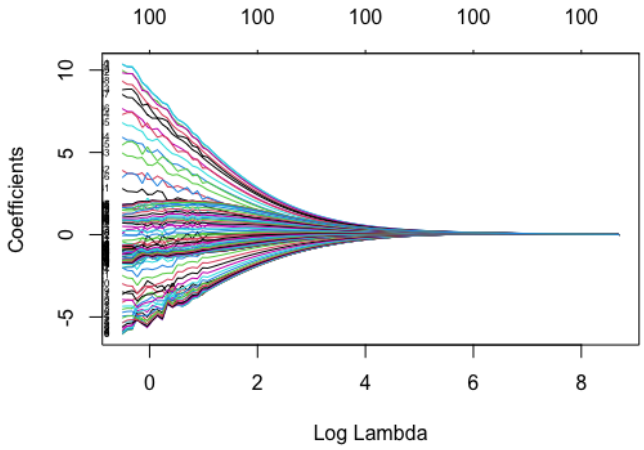

Figure 2: Coefficients dependency on λ for ridge regression

We see in 2 that the coefficients are decrease as λ increases. The Ridge regression makes it so that all the coefficients converges towards zero instead of as with LASSO regression where coefficients drop to zero over the whole spectrum of values for the penalty factor. It would be harder to choose a penalty factor value to gain a specific model than with the LASSO model.

2.5 Cross-validation

Below in Figure 3 is a cross-validation plot for the LASSO regression model.

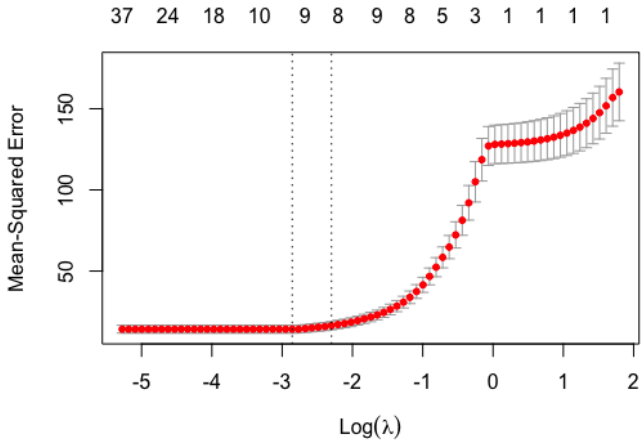

Figure 3: Cross-validation score for $log(\lambda)$.

In the cross-validation plot we see the cross-validation score (Mean-Squared Error) clearly start to increase from $log(\lambda)$. As the penalty factor increases the amount of nonzero coefficients decreases

which in turn can provide a model to simple and thus the cross-validation score increases. The left dashed vertical line in the plot is λ_{min} which is were the error is lowest and gives the optimal model. Here $\lambda_{min} = 0.05744535$ ($log(\lambda_{min}) = -1.24$) which gives 9 nonzero coefficients + intercept.
From Figure 3 it is not reasonable to say that for example $log(\lambda) = -4$ would be a worse choice since the cross-validation score and the variance are the same as for λ_{min}. A comparison between the predictions of the LASSO regression model against the true values are shown in Figure 4.

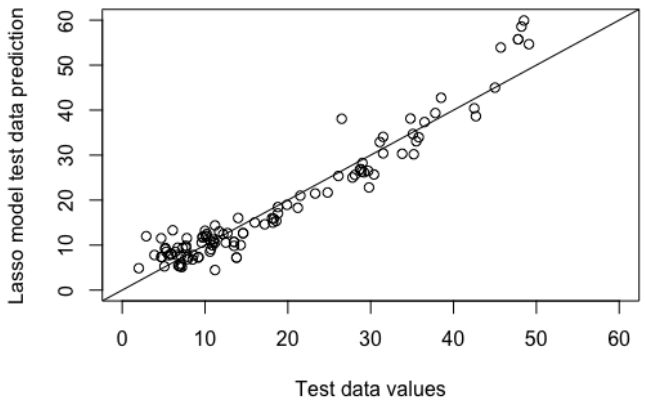

Figure 4: LASSO regression model predictions vs true values for the test data.

A straight line with slope = 1 is added to the plot to visualize the deviation of the predicted values. Perfect predictions are on the line. The model seems to work better for low values of fat content and fairly well overall except for some outliers which makes the model less good.

3 Assignment 2 - Decision trees and logistic regression for bank marketing

In this assignment, we go through a data set with direct marketing campaigns (through phone calls) of a Portuguese banking institution. By using the method decision trees with different settings we predict "yes" or "no" depending on if a customer subscribed to the product (bank term deposit) based on 19 different features.

3.1 Data pre-processing

The data was split into three different sets: training, validation, and test with a proportion of 40%, 30%, and 30% respectively. The feature variable *duration* was discarded as it is only meant for benchmark purposes and prevents a realistic predictive model. Since the features that were characters should be categorical variables, we needed to change these to factors.

3.2 Decision tree settings

In Table 2 we can observe the misclassification rates for the training and validation data based on three different decision trees: default settings, smallest allowed node size equal to 7000, and minimum deviance equal to 0.0005. The best model of these three based on misclassification rate is a decision tree with default settings and the smallest allowed node size equal to 7000. Both have a misclassification rate of 10.92679% on the validation data compared to the tree with minimum deviance which has a misclassification rate of 11.18484%. The best of these three can be argued is the minsize tree as it has the smallest amount of splits and low misclassification rate.

With an allowed node size of 7 000, a split is only made if the number of observations is at least 7 000 on both sides of the split, otherwise the node becomes a terminal node. We can observe in Table 2 that the smallest node size tree has one less terminal node and variable in its tree compared to the default tree. This is due to the fact that a variable did not meet the condition of a minimum size equal to 7 000 after a split and became a terminal node.

With a minimum deviance of 0.0005, all nodes must be at least 0.5 % of the root node deviance, otherwise a split will not occur and become a terminal node. In Table 2 we can see that the minimum deviance tree has 122 terminal nodes and 13 variables. Lowering the deviance limit means that we get a better fit (e.g. mindev = 0 and minsize = 2 will make the data fit perfectly if depth limit allow such a tree). This means that we get a larger tree with more nodes and a tree that fits the data better than the default tree. But at the same time it can get overfitted, in Table 2 we can see that it performs well on the training data but a lot worse on the validation data.

Table 2: Different tree settings.

	Terminal nodes	**Variables**	**Misclassification rate**
Default training	6	4	10.48441%
Default validation	6	4	10.92679%
Smallest node size training	5	3	10.48441%
Smallest node size validation	5	3	10.92679%
Minimum deviance training	122	13	9.328688%
Minimum deviance validation	122	13	11.18484%

3.3 Optimal tree depth

By looking at Figure 5 we can see that the optimal number of leafs is 22 as it gives the lowest deviance for the validation data. A tree with 22 leafs seem to find a good trade off between being able to generalize and fit the data well. We also see that the training data gets a lower and lower deviance the more we increase the number of leafs as it gets more fitted to the data. At the same time, we can see that more leafs than 22 for the validation data leads to an overfitted model as the deviance begins to increase again. The variables used to create the optimal tree were poutcome, month, contact, pdays, age, day, balance, housing, and job. The root node is poutcome which is the most improtant for decsicion making in this tree. Job and month are children to the root node and therefore second most important. The tree seems to have a lot more splits to the left side of the root node compared to the right side.

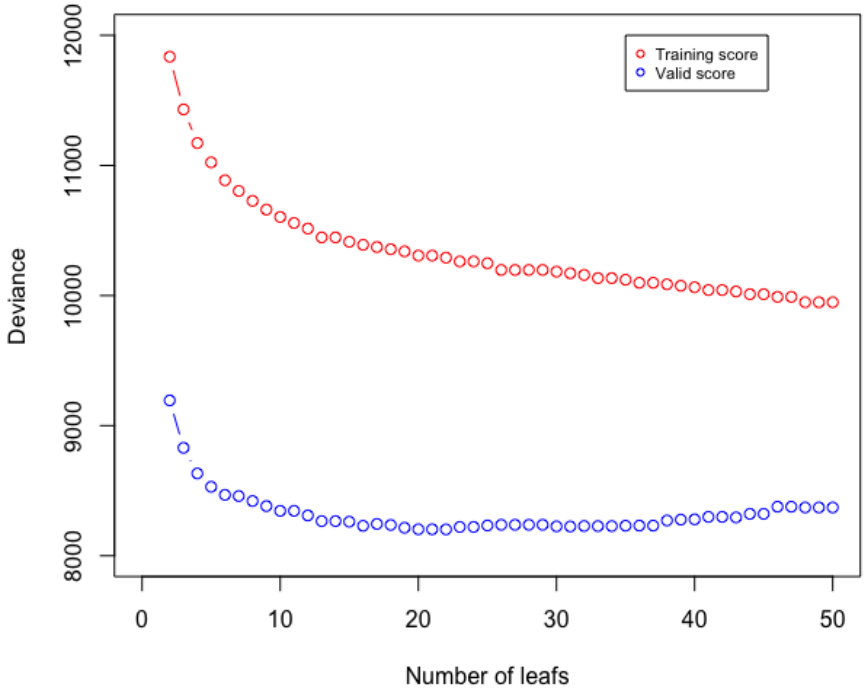

Figure 5: Deviance for training data and validation data based on the number of leafs.

3.4 Prediction power

F1-score may be a better choice than accuracy. That is because the data is quiet unbalanced, for example we can see that the model is very bad at predicting if their is an actual customer. If just looking at the accuracy we may think that we have a very good model. In Table 3 it is possible to see that 1371 actual customers would be predicted as not customers which leads to that the bank may loose many potential future customers. So, in overall the classifier seems to perform well but due to the relative high false negatives and low F1 score, we think the model does not seem to have

a very good predictive power for the purpose it is going to be deployed for.

Table 3: Confusion matrix for the optimal tree with the test data.

	no	yes
no	11872	107
yes	1371	214

Table 4: Different measures for the optimal tree with the test data.

	Accuracy	**Recall**	**Precision**	**F1-score**
Optimal tree	89.10351%	13.50158%	66.66667%	0.224554

3.5 Loss matrix

In the case of many false negatives that we want to avoid in order to not lose too many customers, a loss matrix is used which makes false negatives 5 times worse than false positives. We can see in Table 5 that the model makes more effort to find all positive cases as these have increased compared to negative cases. In this model, we can see that it is not as bad to guess wrong on false positives compared to false negatives. We also see in Table 6 that the F1 score is significantly better than before and accuracy has only decreased by about 2%.

Table 5: Confusion matrix for the optimal tree with loss matrix on the test data.

	no	yes
no	11030	949
yes	771	814

Table 6: Different measures of the optimal tree with loss matrix on the test data.

	Accuracy	**Recall**	**Precision**	**F1-score**
Optimal tree	87.31937%	51.35647%	46.1713%	0.4862605

3.6 ROC curves

It is possible in Figure 6 to note that the tree model seems to peform slightly better than the logistic regression model. We can see that the red line has higher TPR and lower FPR. The tree model had no negative class for high values of the threshold. When there is a large unbalance between the classes, it may be better to use a precision-recall curve. This gives us a better measurement for determining a good model as the false negatives and true negatives tend to be high in this data set. Such curve could get a better measure of the performance for the models which takes account that the model did not performed well on predicting if an customer would buy the product as we saw in Section 3.4.

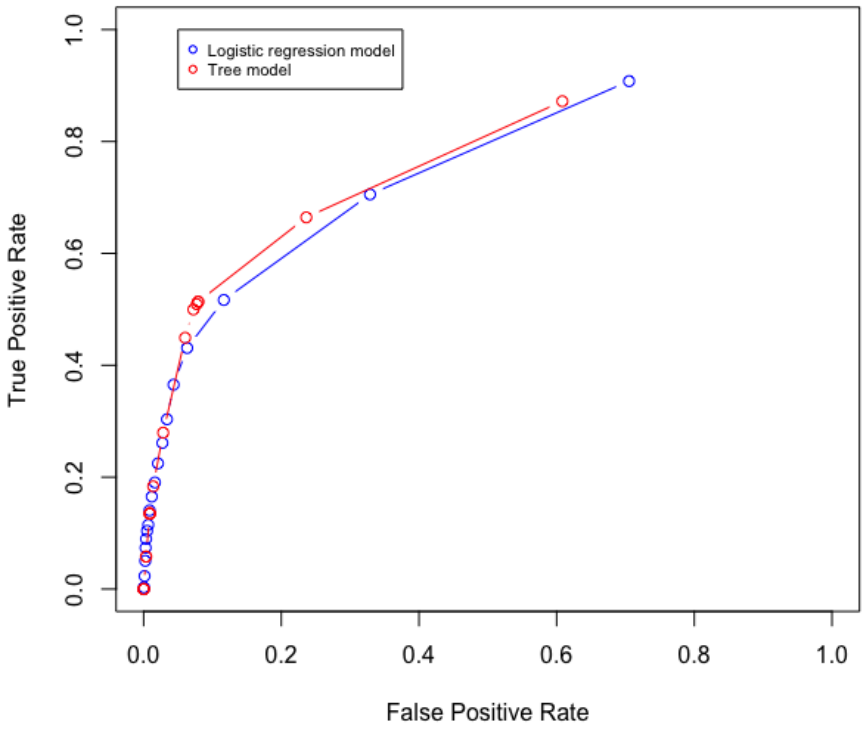

Figure 6: ROC curve for Tree model and Logistic regression model.

4 Assignment 3 - Principal components and implicit regularization

4.1 PCA with eigen

First all the data except for the ViolentCrimesPerCpt were scaled. Then the covariance matrix S was created from the scaled data. Using the built in *eigen* function on S the eigenvalues were computed for the components. From the eigenvalues we could see that the 35 largest components were required to reach 95% variance. The two largest components represented 25% and 16% respectively.

4.2 PCA with princomp

Next *princomp* was used to compute the principle components. In Figure 7 shows the trace plot of the first component. It shows that there are quite a few features contributing to the component. After that the five absolute largest contributing features of the first component were computed. They are show in Table 7 below. The table shows that four of the features are related income and the fifth is related to two parent households, these two areas both make logical sense as big contributing factors. Then the data is mapped onto the first two compnents PCA1 and PCA2,

which is shown in Figure8, the coloring shows the values of ViolentCrimesPerPop for the point. In this plot we can see the that points with high ViolentCrimesPerPop also seem to be positive in either the PCA1 or PCA2 axis. Conversely points with a low ViolentCrimesPerPop instead mostly negative in either PCA1 or PCA2.

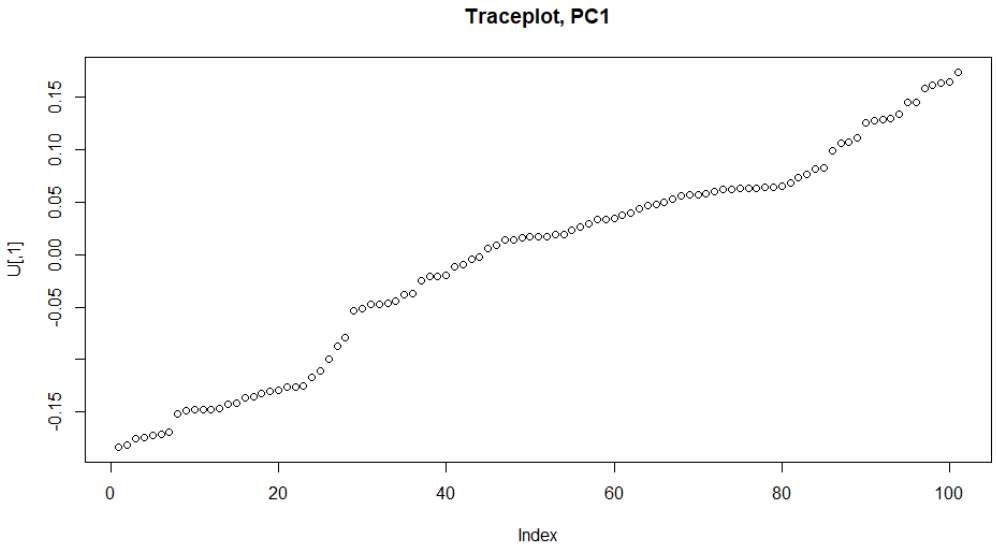

Figure 7: Trace plot of the first principle component.

Table 7: The five highest contributing features in first principle component

MedFamInc	MedIncome	PctKids2Par	PctWInvInc	PctPopUnderPov
-0.1831478	-0.1818171	-0.1755406	-0.1748076	0.1737183

4.3 Solving with linear regression

All features as well as the target, ViolentCrimesPerPop, were scaled and then split up evenly in training and test data. Then a linear regression model was trained and evaluated using MSE on the test data. The training and test error are displayed in Table 8 below. The model seems to perform alright, but suffers some due to overfitting to the training data.

Table 8: Error score for training and test of the linear regression

Training	Test
0.2591772	0.4000579

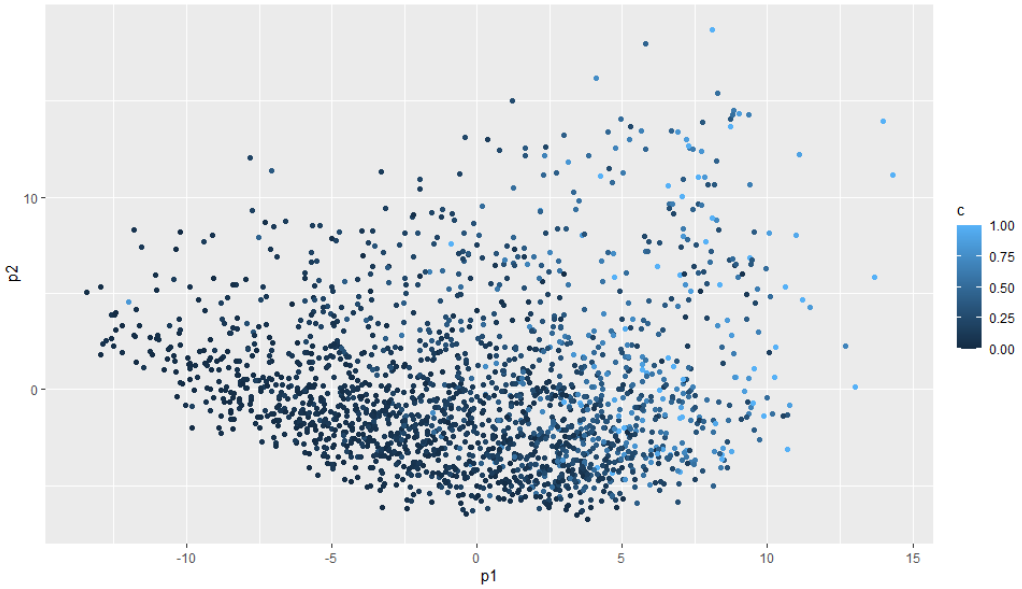

Figure 8: Score plot showing the first and second PC with the data plotted on them, coloring shows the ViolentCrimesPerPop for each point

4.4 Linear regression with early stopping

Here the linear model is instead obtained by using *optim*, the input parameters were all of the feature coefficients and the function to minimize was the set to the training loss of the linear model. During each iteration of the function the training and test MSE was saved which is plotted in Figure 9. This shows that while the training error is correctly decreasing, the test error varies wildly before settling. The test error shows that the lowest MSE is achieved around 2300 iterations, f the criterion of early stopping is used then that would be the chosen model. The scores from the optimal model chosen by early stopping is shown in Table 9 below. It shows that a lower test error was obtained by using early stopping at the cost of the training error which is higher in this model. This makes sense since the model is not trying to optimize for the error of the test set, that is being done retroactively.

Table 9: Shows the training and test error for the optimal early stopped model

Training	Test
0.2858249	0.3769468

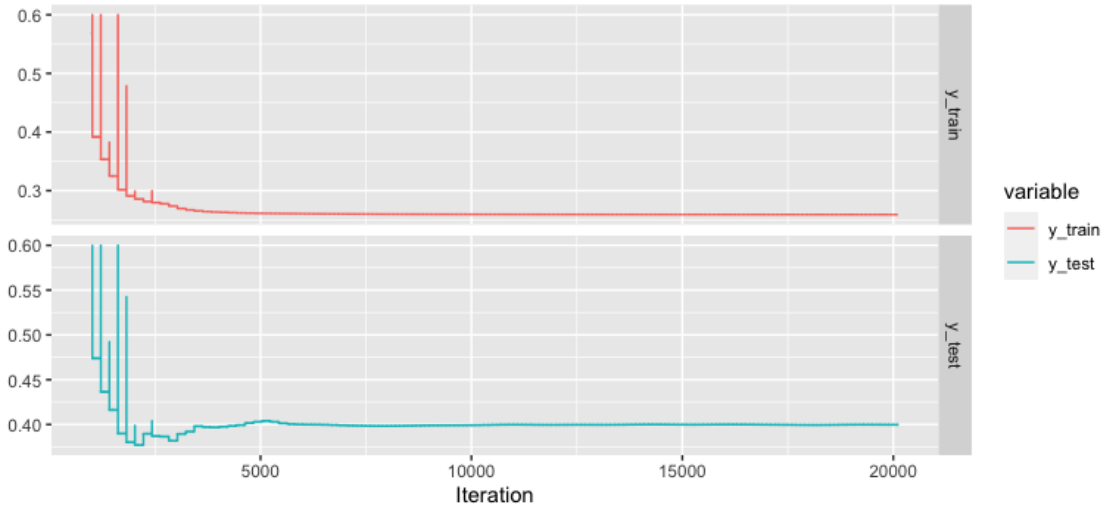

Figure 9: Shows the depence of the training and test error on the iteration of the optimization

A Source code

A.1 Assignment 1

```
#Lab 2
library("glmnet")
#Assignment 1
data=read.csv("tecator.csv")

#Divide into train and test
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

# 1.1
plot(density(train$Fat), main="Distribution", xlab="Fat", ylab="Density")
# Linear regression model
fat_model <- lm(Fat ~ ., train[,2:102])
pred.train = predict(fat_model, train[,2:101])
pred.test = predict(fat_model, test[,2:101])

# Calculating mean squared error training data
results.train <- cbind(pred.train, train$Fat)
colnames(results.train) <- c('Pred','Real')
results.train <- as.data.frame(results.train)
mse.train <- mean((results.train$Real - results.train$Pred)^2)
cat(paste("MSE training:"), mse.train,"\n") #Small error

# Calculating mean squared error test data
results.test <- cbind(pred.test, test$Fat)
colnames(results.test) <- c('Pred','Real')
results.test <- as.data.frame(results.test)
mse.test <- mean((results.test$Real - results.test$Pred)^2)
cat(paste("MSE test: "), mse.test,"\n")
```

```
33
34 # 1.2
35 # J(\theta, X, Y) = (1/n) * SUM(y_i - \theta_0 - \theta_1*x_{1j} -...- \theta_p*x_{
       pj})^2)
36 # j = 1:100   All channels as features
37 # Cost function should be minimized
38
39 # 1.3
40 # LASSO regression model
41 lasso_model = glmnet(as.matrix(train[2:101]), train$Fat, alpha = 1)
42 plot(lasso_model, xvar="lambda", label=TRUE)
43
44
45 # 1.4
46 # Ridge regression model
47 ridge_model = glmnet(as.matrix(train[2:101]), train$Fat, alpha = 0)
48 plot(ridge_model, xvar="lambda", label=TRUE)
49
50 # 1.5
51 # Cross-validation of LASSO regression model
52 lasso_cv = cv.glmnet(as.matrix(train[2:101]), train$Fat, alpha=1)
53 lasso_cv$lambda.min
54 plot(lasso_cv)
55 coef(lasso_cv, a0="lambda.min") #Optimal lambda
56 colSums(coef(lasso_cv) != 0) #Amount of nonzero elements
57
58 # Scatterplot with y=x line for comparison
59 lasso_pred = predict(lasso_cv, as.matrix(test[2:101]), s = lasso_cv$lambda.min)
60 plot(test$Fat, lasso_pred, xlim = c(0,60), ylim = c(0,60), xlab="Test data values",
       ylab="Lasso model test data prediction")
61 abline(a=0, b=1)
```

A.2 Assignment 2

```
1  library('tree')
2  library(xtable)
3  setwd("/Users/christoffergardin/Desktop/TDDE01/Lab 2")
4
5  #1.
6  #Data pre-processing, set as factor for the features that are categorical and drop
       duration column.
7  data <- read.csv('bank-full.csv', header = TRUE, sep = ";", stringsAsFactors = T)
8  df <- data[-12]
9  #Split into training/validation/test as 40/30/30
10 n = dim(df)[1]
11 set.seed(12345)
12 id = sample(1:n, floor(n*0.4))
13 train = df[id,]
14 id1 = setdiff(1:n, id)
15 set.seed(12345)
16 id2 = sample(id1, floor(n*0.3))
17 valid = df[id2,]
18 id3 = setdiff(id1,id2)
19 test = df[id3,]
20
21 #Misclassification function
22 missclass=function(X,X1){
23   n=length(X)
24   return(1-sum(diag(table(X,X1)))/n) }
25
26 #2a)
27 #Decision tree with default settings
28 tree.a <- tree(y ~ ., method = "class", data = train)
```

```
pred.valid_tree.a <- predict(tree.a,valid,type="class") #Prediction on valid data
    sample
pred.train_tree.a <- predict(tree.a,train,type="class") #Prediction on train data
    sample
#Misclassification rates
miss.valid_tree.a <- missclass(valid$y,pred.valid_tree.a)
miss.train_tree.a <- missclass(train$y,pred.train_tree.a)
cat(paste("train_tree.a misclassification rate: "), miss.train_tree.a ,"\n")
cat(paste("valid_tree.a misclassification rate: "), miss.valid_tree.a,"\n")

plot(tree.a)
text(tree.a, pretty = 0)
summary(tree.a)
tree.a

#2b)
#Decision tree with smallest allowed node size equal to 7 000
tree.b <- tree(y ~ ., method = "class", data = train, control = tree.control(nobs =
    nrow(train), minsize = 7000))
pred.valid_tree.b <- predict(tree.b,valid,type="class") #Prediction on valid data
    sample
pred.train_tree.b <- predict(tree.b,train,type="class") #Prediction on train data
    sample
#Misclassification rates
miss.valid_tree.b <- missclass(valid$y,pred.valid_tree.b)
miss.train_tree.b <- missclass(train$y,pred.train_tree.b)
cat(paste("train_tree.b misclassification rate: "), miss.train_tree.b ,"\n")
cat(paste("valid_tree.b misclassification rate: "), miss.valid_tree.b,"\n")

plot(tree.b)
text(tree.b, pretty = 0)
summary(tree.b)

#2c)
#Decision tree with minimum deviance 0.0005
tree.c <- tree(y ~ ., method = "class", data = train, control = tree.control(nobs =
    nrow(train), mindev = 0.0005))
pred.valid_tree.c <- predict(tree.c,valid,type="class") #Prediction on valid data
    sample
pred.train_tree.c <- predict(tree.c,train,type="class") #Prediction on train data
    sample
#Misclassification rates
miss.valid_tree.c <- missclass(valid$y,pred.valid_tree.c)
miss.train_tree.c <- missclass(train$y,pred.train_tree.c)
cat(paste("train_tree.b misclassification rate: "), miss.train_tree.c ,"\n")
cat(paste("valid_tree.b misclassification rate: "), miss.valid_tree.c,"\n")

plot(tree.c,type="uniform")
text(tree.c, pretty = 0)
summary(tree.c)

#3
#Init deviance score vector for training set and valid set
train.score <- rep(0,50)
valid.score <- rep(0,50)

#Pruning and studying the trees up to 50 leaves
for(i in 2:50) {
  pruned.tree <- prune.tree(tree.c, best = i)
  pred.v.tree <- predict(pruned.tree, valid, type = "tree")
  train.score[i] <- deviance(pruned.tree)
  valid.score[i] <- deviance(pred.v.tree)
}

```

```r
#Optimal nr. of leafs: 22 (2:50 has one less element than valid score hence 21 + 1)
opt.leaf <- (which.min(valid.score[2:50])) + 1
plot(2:50, train.score[2:50], type = "b", col = "red", xlim = c(0,50), ylim = c
    (8000,12000), xlab = "Number of leafs", ylab = "Deviance")
points(2:50, valid.score[2:50], type = "b", col = "blue")
legend(35, 12000, legend=c("Training score", "Valid score"),
       col=c( "red", "blue"), cex=0.7, pch = 1)

#Final tree with 22 leafs(terminal nodes)
final.tree <- prune.tree(tree.c, best = opt.leaf)
pred.final <- predict(final.tree, valid, type = "class")
table.final.valid <- table(valid$y,pred.final)
plot(final.tree)
text(final.tree, pretty = 0)
summary(final.tree)

#Task 4
#Prediction on the final tree with 22 leafs and the test data set
pred.final.test <- predict(final.tree, test, type = "class")
table.final <- table(test$y,pred.final.test)

acc <- (table.final[2,2]+table.final[1,1])/(sum(table.final)) #accuracy = (TP+TN)/(
    Total)
prec <- (table.final[2,2])/(table.final[2,2]+table.final[1,2]) #precision = (TP)/(TP
    +FP)
recall <- (table.final[2,2])/(table.final[2,2]+table.final[2,1]) #recall = (TP)/(TP+
    FN)
f1.score <- (2*prec*recall)/(prec+recall) # F1-score = (2*precision*recall)/(
    precision+recall)

cat(paste("Final tree accuracy: "), acc,"\n")
cat(paste("Final tree precision: "), prec,"\n")
cat(paste("Final recall: "), recall,"\n")
cat(paste("Final tree F1.score: "), f1.score,"\n")
print(table.final)

#Task 5
#Prediction on the final tree with 22 leafs, the test data, and a loss matrix
pred.loss_matrix <- predict(final.tree, test, type = "vector")
pred.loss_matrix_1 <- transform(pred.loss_matrix, value = no / yes)
pred.loss_matrix_2 <- ifelse(pred.loss_matrix_1[,3] > (5/1),"no","yes") #for example
     if we would had a loss matrix (0 5 2 0) we would take >= 5/2
table.loss_matrix <- table(test$y,pred.loss_matrix_2)
print(table.loss_matrix)

acc1 <- (table.loss_matrix[2,2]+table.loss_matrix[1,1])/(sum(table.loss_matrix)) #
    accuracy = (TP+TN)/(Total)
prec1 <- (table.loss_matrix[2,2])/(table.loss_matrix[2,2]+table.loss_matrix[1,2]) #
    precision = (TP)/(TP+FP)
recall1 <- (table.loss_matrix[2,2])/(table.loss_matrix[2,2]+table.loss_matrix[2,1])
    #recall = (TP)/(TP+FN)
f1.score1 <- (2*prec1*recall1)/(prec1+recall1) # F1-score = (2*precision*recall)/(
    precision+recall)

#Task 6
#Logistic regression model
glm.model <- glm(y ~., data = train, family = binomial)
glm.pred <- predict(glm.model, test, type = "response")

#Probabilistic values for the final tree
pred.opt.test <- predict(final.tree, test, type = "vector")

#Init vectors for TPR and FPR
tpr.tree <- rep(0,18)
```

```r
tpr.glm <- rep(0,18)
fpr.tree <- rep(0,18)
fpr.glm <- rep(0,18)

#Calculating FPR and TPR for both the logistic regression model and the tree model
i = 0.05
j = 1
while (i < 1) {
  glm.res <- ifelse(glm.pred > i,"yes","no")
  pred.final.test <- ifelse(pred.opt.test[,2] > i,"yes","no")
  tree.table <- table(test$y,pred.final.test)
  glm.table <- table(test$y,glm.res)

  #For large values on i the tree model does not have a "yes" column
  if(tree.table[1,1]+tree.table[2,1] == nrow(pred.opt.test)) {
    tpr.tree[j] <- 0
    tpr.glm[j] <- (glm.table[2,2])/(glm.table[2,1] + glm.table[2,2])
    fpr.tree[j] <- 0
    fpr.glm[j] <- (glm.table[1,2])/(glm.table[1,1] + glm.table[1,2])
  }
  else {
    tpr.tree[j] <- (tree.table[2,2])/(tree.table[2,1] + tree.table[2,2])
    tpr.glm[j] <- (glm.table[2,2])/(glm.table[2,1] + glm.table[2,2])
    fpr.tree[j] <- (tree.table[1,2])/(tree.table[1,1] + tree.table[1,2])
    fpr.glm[j] <- (glm.table[1,2])/(glm.table[1,1] + glm.table[1,2])
  }
  i = i + 0.05
  j = j + 1
}

plot(fpr.glm, tpr.glm, col = "blue", type = "b",xlim = c(0,1), ylim = c(0,1), xlab =
     "False Positive Rate", ylab = "True Positive Rate")
points(fpr.tree, tpr.tree, type = "b", col = "red")
legend(0.05, 1, legend=c("Logistic regression model", "Tree model"),
       col=c( "blue", "red"), cex=0.7, pch = 1)
```

A.3 Assignment 3

```r
library(ggplot2)
library(reshape2)
#3.1
data=read.csv2("communities.csv", dec=".", sep=",")
index <- names(data) %in% c('ViolentCrimesPerPop')
X <- scale(data[,!index], center=TRUE)
X <- cbind(X, ViolentCrimesPerPop=data[,index])
N <- length(data[,1])
S = 1/N * t(X) %*% X
d = eigen(S)
lambda = d$values
sprintf("%2.3f",lambda/sum(lambda)*100)

# First 35 components will sum up to to 95% variance
sum(lambda[1:35])

# Contribuitions of PCA 1 and 2
sprintf("PCA1: %2.3f | PCA2: %2.3f", lambda[1], lambda[2])

#3.2

res <- princomp(X)
# Plot the trace for the first component
re <- prcomp(X)
U <- re$rotation
```

```r
plot(sort(U[,1]), main="Traceplot, PC1", ylab="U[,1]")


df = data.frame(len=res$scores[,1])
pca1 = U[,1]
pca2 = U[,2]

# Five most contributive factors for PCA1
pca1[sort(abs(pca1), index.return=TRUE, decreasing = TRUE)$ix[1:5]]

#Plot scores for pca1 and pca2 with coloring that show Violentcrimes
df = data.frame(p1 = re$x[,1], p2=re$x[,2], c = data$ViolentCrimesPerPop)
ggplot(df, aes(x=p1, y=p2, color=c)) +
  geom_point()


# 3.3
n=dim(data)[1]
sd = as.data.frame(scale(data))
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=as.data.frame(sd[id,])
test=as.data.frame(sd[-id,])
model = lm(ViolentCrimesPerPop ~ ., data=train)
summary(model)
pred.train = predict(model, train)
pred.test = predict(model, test)
# Train and test errors for LM
mse.train = mean((train$ViolentCrimesPerPop - pred.train)^2)
mse.test = mean((test$ViolentCrimesPerPop - pred.test)^2)


#3.4
min.mse <- function(par){
  theta <- par[1:100]
  m_train <- as.matrix(train[,1:100]) %*% matrix(theta,length(theta),1)
  m_test <- as.matrix(test[,1:100]) %*% matrix(theta,length(theta),1)
  mse.train = mean((train$ViolentCrimesPerPop - m_train)^2)
  mse.test = mean((test$ViolentCrimesPerPop - rowSums(m_test))^2)
  .GlobalEnv$k= .GlobalEnv$k+1
  .GlobalEnv$test_score[[k]] = mse.test
  .GlobalEnv$train_score[[k]] = mse.train
  return(mse.train)
}

#run regression for
par = replicate(100, 0)
train_score = c()
test_score = c()
k = 0
r <- optim(par, min.mse, method = c("BFGS"))

#plot test and training error

y_test = unlist(test_score[500:k])
df = data.frame(y_train=pmin(unlist(train_score[1000:k]), 0.6), y_test=pmin(unlist(
    test_score[1000:k]),0.6), x_=c(1000:k))
dt.df <- melt(df, measure.vars = c("y_train", "y_test"))
dt$x_ <- x_= c(1000:k)
ggplot(data = dt.df, aes(x=x_, y=value))+
  geom_line(aes(color=variable))+
  facet_grid(variable ~ ., scales = "free_y") +
  labs(x="Iteration", y = "")
  theme(legend.position="none")
```

```
89
90
91 #minimum test and training values from early stop
92 min(unlist(train_score)) # task3 train: 0.2591772 task4: 0.2592247
93 min(unlist(test_score)) # task3 test: 0.4000579 task4: 0.3769468
94
95 st = sort(unlist(test_score), index.return=TRUE)
96 st$ix[1:10]
97 unlist(test_score)[2183]
98 unlist(train_score)[2183]
```