1 Assignment 1 - Explicit regularization

1.1

The underlying probabilistic model for linear regression is $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_p x_p$. In our case x describes the value of each channel and θ is the coefficients calculated using the least squares method. Creating a linear regression model and calculating the MSE for the fitted values and the MSE for predicted values on the testing data gave us the following:

	MSE
Train	0.005709117
Test	722.4294

Here we can see that the MSE for the training data is really low and is substantially larger for the testing data. The large difference in MSE shows that the model is really good for the training data but is much too overfitted. This means that the model is of low quality overall since we want the model to accurately predict new data, and we don't care as much about how well it predicts the training data.

1.2

With LASSO regression we add a penalty term $\lambda||\theta||_1$ to the cost function of linear regression, giving us the cost function to be optimized as $\hat{\theta} = arg\ \underset{\theta}{min} \frac{1}{n}||X\theta - y||_2^2 + \lambda||\theta||_1$. In this case X is a vector containing the values of the channels, θ is the coefficients calculated in our model, and y is the value for the level of fat.

1.3

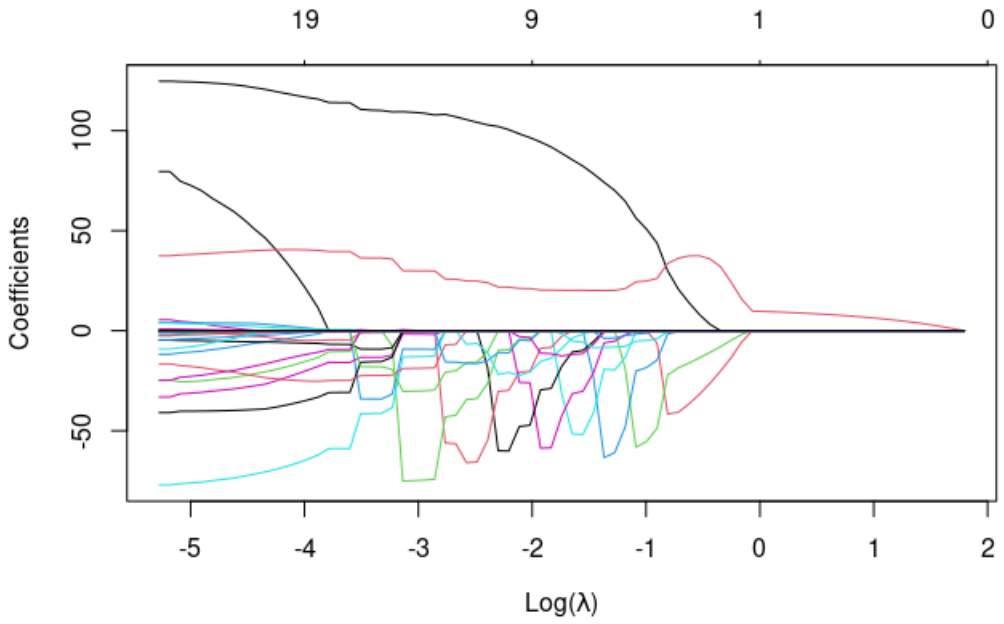

Figure 1: Plot over model coefficients dependence on log λ when using LASSO regression.

From the plot in Figure 1 we can see that even with a small penalty of log $\lambda = -4$ a large majority of the coefficients (θ) are zero. We have a total of 100 variables to build or model on but only 19 features are active at that point, i.e. we have 19 coefficients which are non-zero. When increasing the value of λ, different features activate but the the total active features trend toward zero. At log $\lambda = 2$ all the features in the model are inactive, which means all the coefficients are zero. We can also see that choosing a value somewhere between log $\lambda = 0$ and log $\lambda = -0.4$ gives us a model with only three features, as only three coefficients are non-zero. Retrieving the values of lambda when three coefficients are active from our model gives us the values $\lambda = 0.8530452$, $\lambda = 0.7772630$, and $\lambda = 0.7082131$.

1.4

Using ridge regression with our model and plotting the coefficients dependency on log λ gives us the following plot:

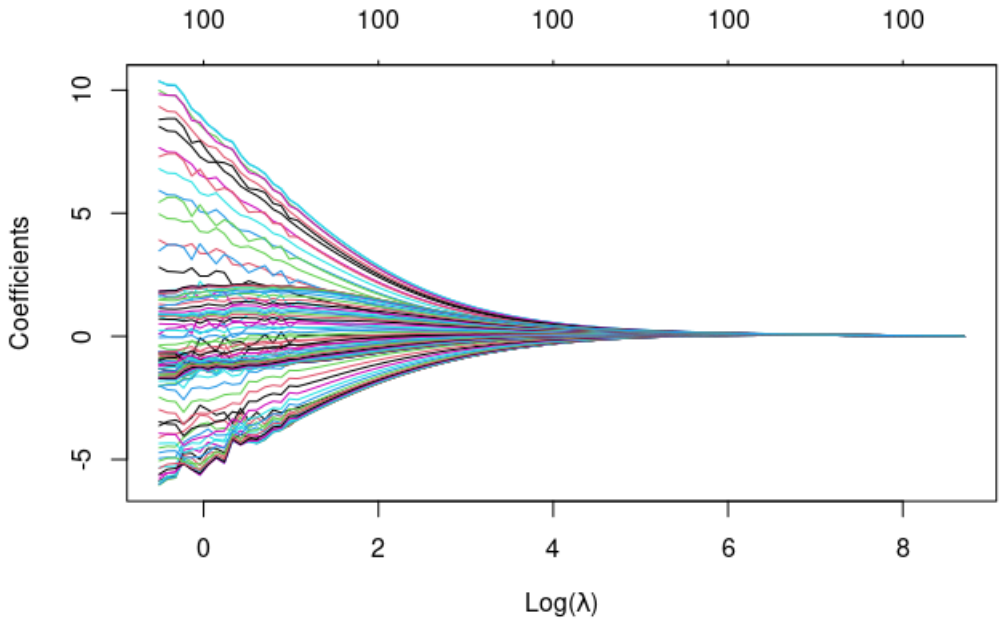

Figure 2: Plot over model coefficients dependence on log λ when using ridge regression.

With ridge regression we can see that the model penalizes all the coefficients similarly, causing all the coefficients to trend toward zero at a relatively similar rate when λ increases. However, the active features are always 100 since ridge regression can not set a coefficient to zero, only asymptotically close to zero. We can see this by looking at the numbers at the top of the plot which shows constantly show 100. Using a model with as many features as this it might be better to use LASSO regression instead of ridge regression as there are probably some features that are useless when making a prediction. That is, unless we know most features are useful for the prediction.

1.5

Computing the MSE for different log λ values gives us the following plot:

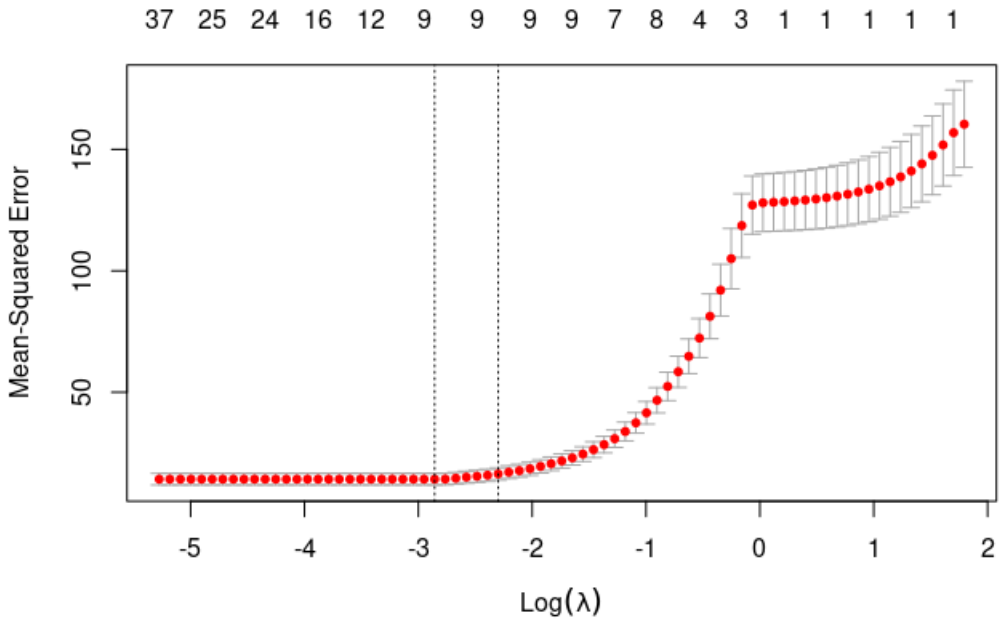

Figure 3: Plot over CV score for different log λ values.

In this plot we can see that using larger penalties increases the MSE. The MSE stays about the same up until our optimal log λ value. From the cross validation we get an optimal log λ value of -2.856921 which seems to be the left vertical dotted line in the plot. This model has 9 chosen variables which we can see from the number at the top of the plot. Looking at just the plot shows no indication that log $\lambda = -2.856921$ is better than log $\lambda = -4$ but by using less active features we can minimize the impact of unimportant factors. Using the optimal lambda value to make predictions gave us a much better result than the original linear regression model.

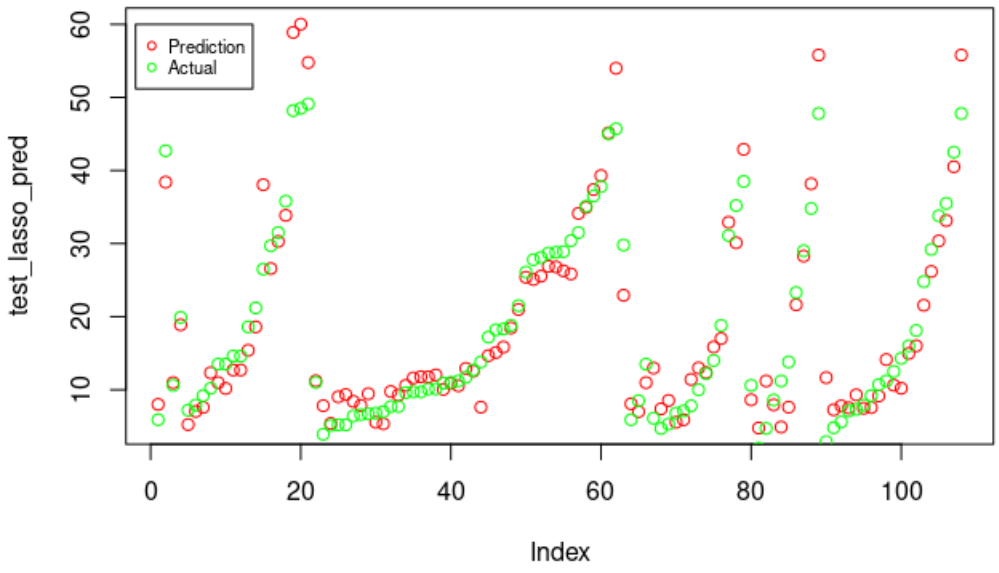

Figure 4: Scatter plot over actual and predicted values using LASSO regression with optimal λ.

The scatter plot in Figure 4 shows the predicted values in red and the actual values in green. From this we can see that the predictions are close to the actual values, indicating that the LASSO model and the predictions are good.

2 Assignment 2

2.1 Dividing and importing data

```r
#str(optdigit)
data$duration <- NULL
#Divide data to 40/30/30 <- training/val/test

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train = data[id,]
id1 = setdiff(1:n,id)
set.seed(12345)
id2 = sample(id1, floor(n*0.3))
valid = data[id2,]
id3 = setdiff(id1, id2)
test = data[id3,]
missclassific=function(c_matrix, fit_matrix){
  n = length(fit_matrix[,1])
  return(1-sum(diag(c_matrix))/n)
}
```

Figure 5: Dividing data.

2.2 Making 3 different trees

First a tree with no limitations was created:

```
fit=tree(y ~., data=train)
```

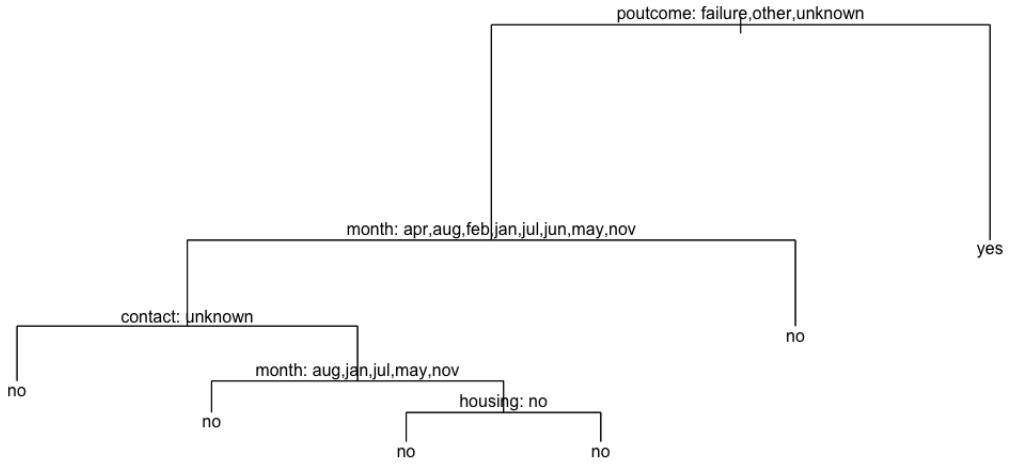

Figure 6: Tree 1 with no constraints.

The next tree had a minimum value for node size which should be greater than or equal to 7000.

```
fit2=tree(y~., data=train, control=tree.control(nobs= dim(train)[1], minsize=7000))
```

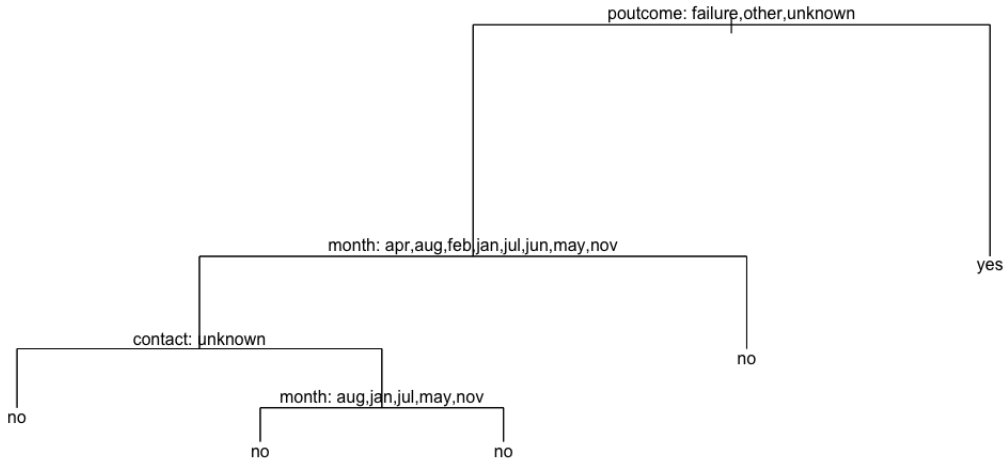

Figure 7: Tree 2 with minimum node size of 7000.

The last tree had to have a minimum deviance of 0.0005. Removed the labels for leafs and decision points, due to cluttering.

```
fit3=tree(y ~.,data=train, mindev=0.0005)
```

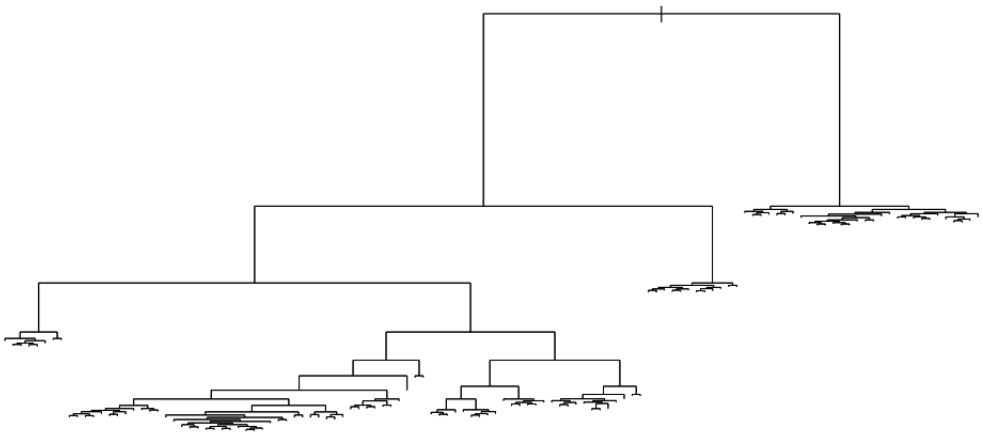

Figure 8: Tree 3 with a minimum deviance of 0.0005.

The first tree just uses the standard settings of the tree package in r. The second tree had a minimum node size of 7000 this removed one node split from Tree 1 due to the node having a size lower than 7000. In this case it doesn't affect either our Accuracy or Missclassification due to the extra node in Tree 1 having the same y for both leaves. The last tree had a minimum deviance requirement which was lowered drastically (from 0.01 to 0.0005) this means that nodes that earlier met the stop criteria as being impure gets let through and leads to an overfitted model. According to the misclassification error it seems that the training data of tree 3 is generally better but the misclassification for the validation data is worse. Tree 1 and 2 have the same misclass for both training and validation.

Misclass	Tree1	Tree2	Tree3
Train	0.1048	0.1048	0.09362
Valid	0.1093	0.1093	0.1118484

2.3 Studying optimal tree depth

Here we take the tree number 3 with the minimum deviance constraint and check what depth gives us the best test-deviance value. To do this we loop over the tree to test all possible prunes from 2-50 and check the one with the lowest test-deviance.

```
trainScore=rep(0,50)
testScore=rep(0,50)
for(i in 2:50) {
  prunedTree=prune.tree(fit3,newdata=train, best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
  }
```

Figure 9: Pruning algorithm.

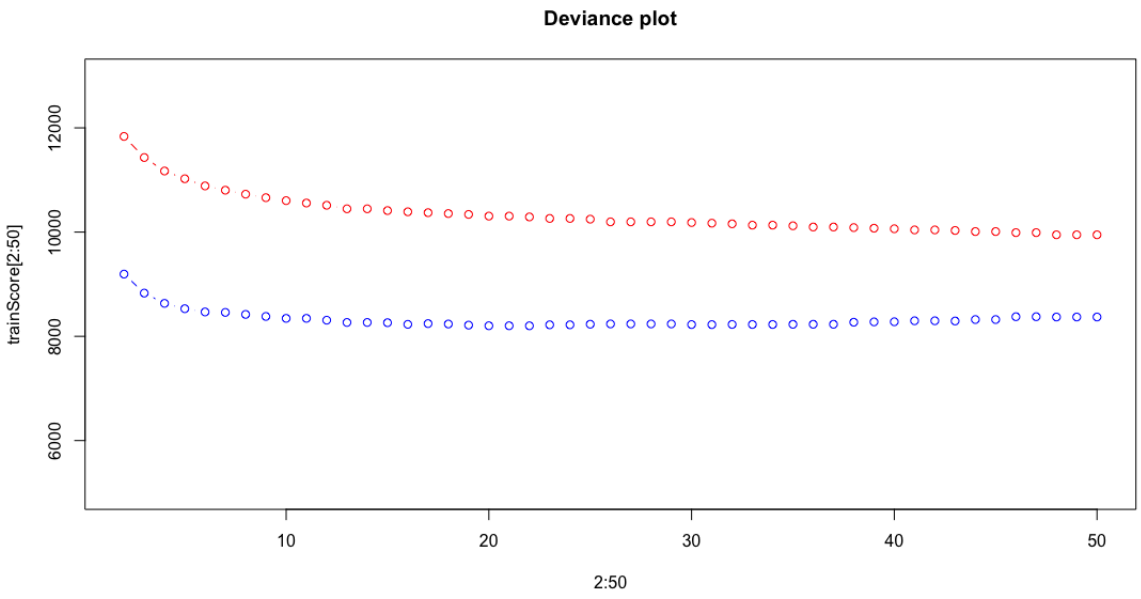

Figure 10: Pruning plot

After checking the value with the lowest test score it ended up being the tree with 22 leaves. The most important variables seem to be, poutcome which is the the first dividing varibale, pdays due to it being the second deciding variable at value ¡383.5 Month seems to play an important role here also due to it being in the tree twice before the deciding variable pdays. Job is also one of the deciding factors in the pruned tree.

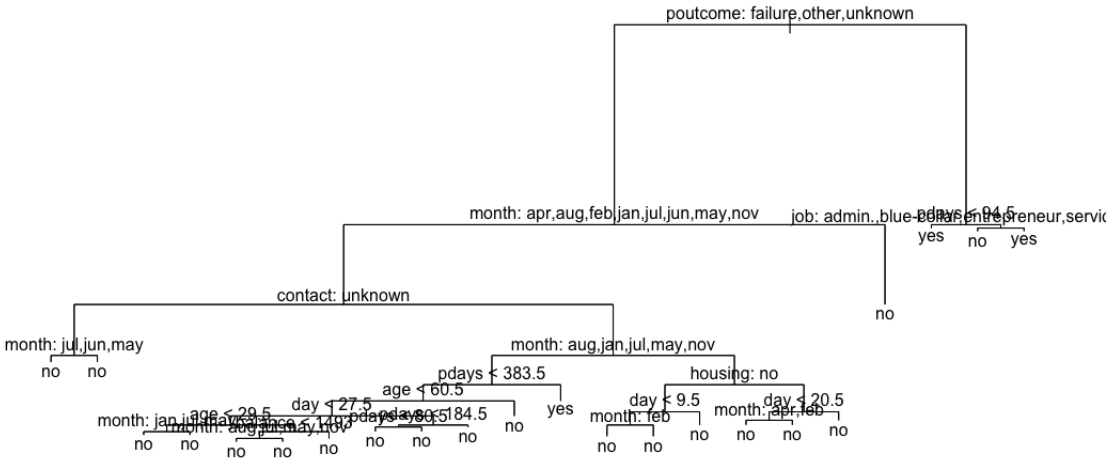

Figure 11: Pruned tree

2.4 Confusion matrix, Accuracy and F1

For this optimal tree the following models were applied.

2.4.1 Confusion matrix

	No	Yes
No	11818	107
Yes	1417	221

2.4.2 Accuracy

Accuracy is calculated as the total amount of correct predictions both No and Yes divided by the Total amount of predictions. In our case the Accuracy value was: 0.8876

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Figure 12: Formula for Accuracy.

2.4.3 F1

The F1 model is made up of recall and precision. Formulas can be seen below. The F1 score came out to 0.2248. Due to the high number of No's in the confusion matrix F1 does a better job of accurately portraying the effectiveness of the model. This is due to it taking both Precision and Recall into account which gives a better understanding of the data balance when there is an uneven class distribution than Accuracy would.

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Figure 13: Formula for F1.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figure 14: Formula for Recall.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Figure 15: Formula for Precision.

2.4.4 Code for the models

```
accuracy=((cmatrixB[2,2]+cmatrixB[1,1])/(cmatrixB[1,2]+cmatrixB[1,1]+cmatrixB[2,1]+cmatrixB[2,2]))

print(accuracy)

precision=(cmatrixB[2,2]/(cmatrixB[1,2]+cmatrixB[2,2]))

print(precision)

recall=(cmatrixB[2,2]/(cmatrixB[2,1]+cmatrixB[2,2]))
print(recall)

F1=2*(precision*recall/(precision+recall))
print(F1)
```

Figure 16: Code.

2.5 Performing a tree classification with loss matrix

Here we apply the following loss matrix to our formula. This means that the prediction must be 5 times more sure to predict a yes.

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

Figure 17: Loss matrix.

The implementation of the code.

```r
fit5 = tree(y ~., data=train, control=tree.control(nobs=dim(train)[1],mindev=0.0005))
#lägga till pruned tree här? Jag säger nej!

fit5pred <- predict(fit5, newdata=test, type="vector")

conf5table=table(test$y, ifelse(fit5pred[,2]/fit5pred[,1]>5, "Yes", "No"))
print(conf5table)

misc5=missclassific(conf5table,test)
print(misc5)

accuracy5=((conf5table[2,2]+conf5table[1,1])/(conf5table[1,2]+conf5table[1,1]+conf5table[2,1]+conf5table[2,2]))

print(accuracy5)

precision5=(conf5table[2,2]/(conf5table[1,2]+conf5table[2,2]))

print(precision5)

recall5=(conf5table[2,2]/(conf5table[2,1]+conf5table[2,2]))
print(recall5)

F1_5=2*(precision5*recall5/(precision5+recall5))
print(F1_5)
```

Figure 18: Lossmatrix Code.

Our confusion matrixes show that applying this loss matrix the misclassification rate goes up slightly. This also affects the accuracy of the model. The same can be said for the F1 value. Due to the loss function we can see that much lower amount of yeses are being predicted. The chance of getting a false positive yes is half the probability after the loss matrix however the ratio correct yeses over false yeses seems to be almost identical.

	yhat_predB	
	no	yes
no	11818	107
yes	1417	221

Figure 19: Confusion matrix no Loss matrix.

	No	Yes
no	11930	49
yes	1485	100

Figure 20: Confusion matrix Loss matrix.

	Misclass	Accuracy	F1
Optimal Tree	0.1123645	0.8876	0.2248
Loss matrix	0.1130935	0.88691	0.1153403

2.6 ROC estimation with Logistic regression

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

$$\hat{Y} = 1 \; if \; p(Y = 'good'|X) > \pi, otherwise \; \hat{Y} = 0$$

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?

Figure 21: Task 6.

Here we compute the Reciever Operating Characteristic (ROC) curve for both our optimal tree which was derived from task 3 and also for a logistic regression function. The ROC curve plots two parameters: True Positive Rate (TPR) and False Positive Rate (FPR).

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Figure 22: TPR FPR calculations

THe TPR and FPR values were derived from our Optimal tree as follows:

```
fit6 = tree(y ~., data=train, control=tree.control(nobs=dim(train)[1],mindev=0.0005))
opt6 = prune.tree(fit6, best=22)
fit6pred <- predict(fit6, newdata=test, type="vector")
pi <- seq(0.00,1,0.05)

fpr <- c(rep(0,length(pi)))
tpr <- c(rep(0,length(pi))) #Make matrix with 0's len pi.




for(i in 1:length(pi)){
  fit6pred.tree <- ifelse(fit6pred[,2]>pi[i], 1,0)
  fit6c = table(test$y,fit6pred.tree)

  if (is.na(table(fit6pred.tree)[2])) {
    if (colnames(fit6c)[1] == "yes") {
      fit6c = cbind(c(0,0), fit6c)
    } else {
      fit6c = cbind(fit6c, c(0,0))
    }
  }

  fpr[i]= fit6c[1,2]/(fit6c[1,2]+fit6c[1,1])
  tpr[i]= fit6c[2,2]/(fit6c[2,1]+fit6c[2,2])
  print(tpr[i])
  }
```

Figure 23: ROC tree code

```r
#Logistic regression
library(nnet)
fprlog <- c(rep(0,length(pi)))
tprlog <- c(rep(0,length(pi))) #Make matrix with 0's len pi.

logistic2 <- multinom(test$y ~ ., data = test, Hess=T)
p2 <- predict(logistic2, data= test, type = 'probs')
for(i in 1:length(pi)){
  prediction4 <- ifelse(p2>pi[i],'yes','no')
  table2 <- table(test$y,prediction4)
  if (is.na(table(prediction4)[2])) {
    if (colnames(table2)[1] == "yes") {
      table2 = cbind(c(0,0), table2)
    } else {
      table2 = cbind(table2, c(0,0))
    }
  }
  fprlog[i]= table2[1,2]/(table2[1,2]+table2[1,1])
  tprlog[i]= table2[2,2]/(table2[2,1]+table2[2,2])
  print(tprlog[i])
}
```

Figure 24: ROC logistics

In the ROC curves we can see that the AUC differs very little, but I would argue that a Precision-Recall curve would be better suited due to the class imbalance in the observations. With the amount of No's being almost 10x the amount of yeses ROC gives an incorrect showing of the skill of the algorithm. ROC is a good way to see that the algorithms performance is better than a cointoss but not much more than that.

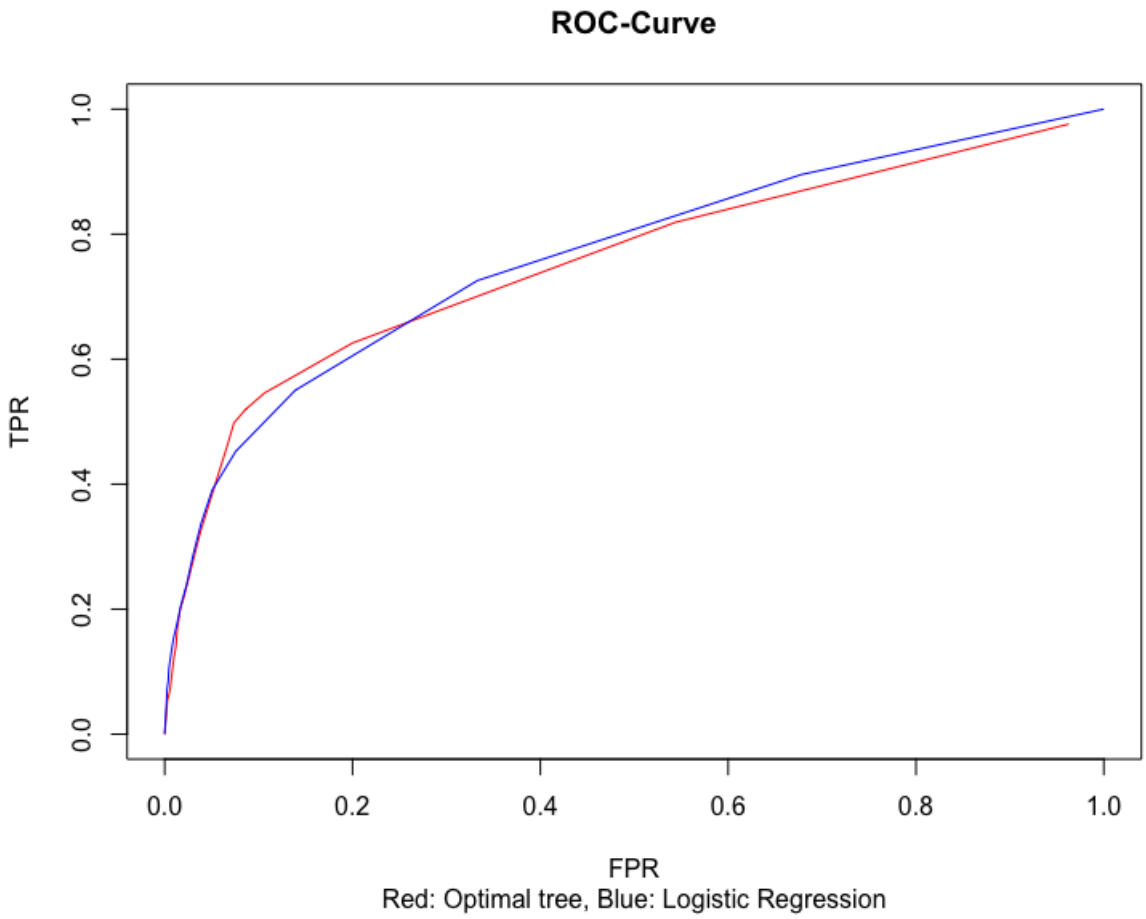

Figure 25: Plot of ROC curves

3 Assignment 3 - Principal components and implicit regularization

Assignment 3 contained a data set regarding American communities and their rate of crimes per inhabitant. Together with crime rate were also 100 different variables. This assignment was divided by one sub-section for each sub-task.

3.1

To obtain 95 % of variance for the data, 33 features need to be considered. This was performed by first calculating the covariance matrix for the data and then obtaining the eigenvalues for the covariance matrix.

```
sum(pcVar[1:33])/sum(pcVar) #resulting in 0.9519715
```

The first two principal components had the following proportion of variation.

	Proportion of variation
PC1	0.2639071
PC2	0.1879254

These two PCAs covers much more than any two single variables, this is the benefit of PCA.

3.2

In my opinion, many features/variables have a noticeable contribution to this component. No single value has greatly larger absolute value than any other values. This makes the data set suitable for PCA. The top 5 features are PctHousNoPhone, PctPoPUnderPov, RentHighQ, medIncome and PctYoungKids2Par, in said order. The common factor in value 2, 3 and 4 is poverty level. PctnoPhone might be an affect on poverty level as well as amount of children, but not as connected as the others in my opinion. When plotting all communities over our first two principle components PC1 and PC2 when can much clearer see a grouping of high and low (mostly low) violent crimes per pop in figure 26.

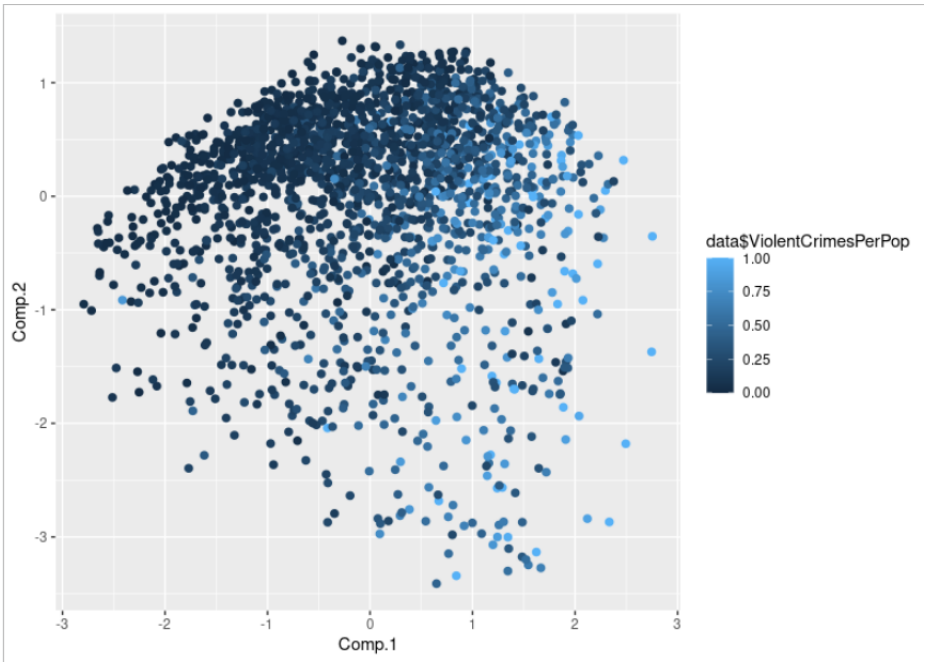

Figure 26: 2D Scoreplot on all communities over PC1 and PC2.

By using PCA we can observe/plot a high dimensional data set (100 variables in this case) and plot it in the same manner as a two dimensional dataset. Instead of having two variables on the x and y axis we now have Principle Components connected to many variables.

3.3

Scaling was enabled with a simple function

```
normalize <- function(x) {
    return ((x - min(x)) / (max(x) - min(x)))
}
```

and normalized each column at a time, except for the output variable column. This column was already normalized in the range 0-1. Splitting was done in the same manner as for Laboration 1. See appendix for further details. Linear regression was made through the following line,

```
model <- lm(ViolentCrimesPerPop~., data = training).
```

Generating Mean Squared Error loss for training and test data gave the following result.

	Model 1
	MSE
Train	0.01406865
Test	0.02171593

The test values from MSE seems to be low, which means the model is good at predicting new output values. But the model is getting a better score on the training data is was trained on, compared to the test data. This indicates that it may be over-fitted and this is not wanted.

3.4

The generated cost function is the MSE of the predicted $\hat{Y}$ using

$$\hat{Y} = X\theta,$$

where X is the test data and θ is the parameter we are altering for each iteration. Using this cost function and starting with $\theta_i = 0, \forall i \in [1, 100]$. In figure 27 we can see that more iterations reduces the MSE for the test data set. But the MSE for training data is lowest around iteration 1200-1400.

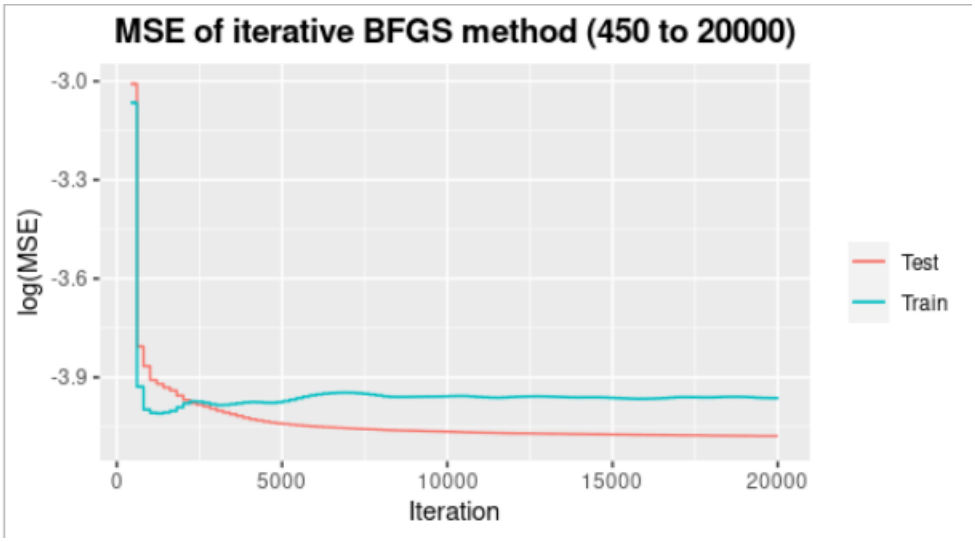

Figure 27: Improvement (reduction) of Mean Square Error over multiple iterations of BFGS method. A zoomed-in version can be seen in figure 28.

According to the early stopping criteria the optimal θ value would be chosen at iteration 1400. This iteration value comes from when the MSE for training data starts to increase instead of

decrease. This value is easier to spot in the zoomed in version in figure 28. Early stopping criteria states that further iterations after 1400 is over-fitting.

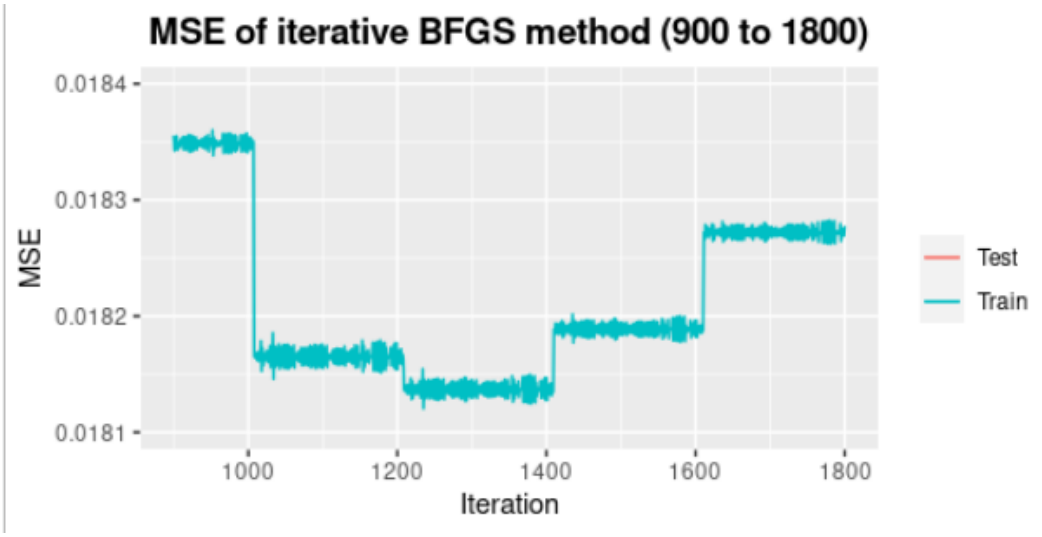

Figure 28: Tipping point for MSE. Early stopping criteria wants to stop at iteration 1400.

Using the optimal θ provided from *optim*(), generated from

```
result = optim(.....)
theta = result$par
```

on the training and test data, together with simple matrix multiplication and calculating MSE we get the second column in the table below. Firstly, the θ in model 2 was not obtained by using the early stopping criteria, but instead the default optimal value from *optim*(). I am unsure of which θ was asked to use, so both can be found in the table. Secondly, it is important to note that θ in column 2 and 3 (named model 2 and 3) was trained on the *test* data and not the training data like the method described in 3.3 (same as 1st column in table below).

	Model 1	**Model 2**	**Model 3**
	MSE with θ from *lm*()	MSE with θ from *optim*()	MSE with θ from early stop
Train	0.01406865	0.01899532	0.01814695
Test	0.02171593	0.01692447	0.01983653

Table 3.4: MSE on training- and test data of the three models

Model 3 in table 3.4 is the model containing a θ generated from the early stopping condition using *optim*(). Namely from iteration 1400 in said function.

I think it is very interesting that even though model 3 was trained on the test data it gave better score for training data than test data. With early stopping criteria we can know for sure that the model is not over fitting which is a valuable feature. I would also argue that model 3 is the best model of these three. Namely from having such a low MSE from new data. This model will perform the best on new data out of the three models. Evaluating the MSE from data the model has been trained on is not as valuable as MSE on new data.

4 Appendix

4.1 Code for Assignment 1

```
#Library for regression
library(glmnet)

#Read in data
absorbance_data=read.csv("tecator.csv",stringsAsFactors = FALSE)

#Divide into training and test data, 50-50
n=dim(absorbance_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=absorbance_data[id,]
test=absorbance_data[-id,]

#Task 1
####################################################################

#Create new dataset with only fat and channels. Divide into train/test, x/y
fat_train <- train[,2:102]
y_fat_train <- fat_train$Fat
x_fat_train = fat_train[-101]
fat_test <- test[,2:102]
y_fat_test <- fat_test$Fat
x_fat_test = fat_test[-length(fat_test)]

#Linear regession model on Fat. MSE for test and train
model <- lm(Fat~., data=fat_train)
MSE_train <- mean((fitted(model)-y_fat_train)^2)
pred_test = predict(model, newdata = fat_test)
MSE_test <- mean((pred_test-y_fat_test)^2)

#Task 2
####################################################################
#Report only

#Task 3
####################################################################
#Lasso regression model
x_model <- as.matrix(x_fat_train)
lambda.lasso <- glmnet(x=x_model, y=y_fat_train, alpha = 1)

#Plot over coefficients dependent on log lambda
plot(lambda.lasso, "lambda", label= FALSE, xlab = expression(paste("Log(",lambda,")")))
```

```
#Lambda values for exactly three nonzero coefficients
lambda_3 = lambda.lasso$lambda[lambda.lasso$df==3]

#Task 4
#########################################################
#Ridge regression model
lambda.ridge <- glmnet(x=x_model, y=y_fat_train, alpha = 0)
plot(lambda.ridge, "lambda", label= FALSE, xlab = expression(paste("Log(",lambda,")")))

#Task 5
#########################################################

#Plot over CV score for log lambda
cv.lambda.lasso <- cv.glmnet(x=x_model, y=y_fat_train, alpha = 1)
plot(cv.lambda.lasso)

#Calculate optimal lambda and make prediction
opt.lambda.lasso <- glmnet(x=x_model, y=y_fat_train,lambda = cv.lambda.lasso$lambda.min, alpha = 1)
opt.log.lambda = log(opt.lambda.lasso$lambda)
opt.lambda = opt.lambda.lasso$lambda
test_lasso_pred = predict(opt.lambda.lasso, newx = as.matrix(x_fat_test))

#Scatterplot over original vs predicted test values with optimal lambda
plot(test_lasso_pred,col="red")
points(y_fat_test, col = "green")
```

4.2 Code for Assignment 2

```
setwd("/home/")
data <- read.csv2(file = "~/Downloads/bank-full.csv", header=TRUE, stringsAsFactors = TRUE)

#str(optdigit)
data$duration <- NULL
#Divide data to 40/30/30 <- training/val/test

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train = data[id,]
id1 = setdiff(1:n,id)
set.seed(12345)
id2 = sample(id1, floor(n*0.3))
valid = data[id2,]
id3 = setdiff(id1, id2)
```

```
test = data[id3,]
missclassific=function(c_matrix, fit_matrix){
  n = length(fit_matrix[,1])
  return(1-sum(diag(c_matrix))/n)
}

library(tree)

#Task 2
fit=tree(y ~., data=train)
plot(fit)
text(fit, pretty=0)
fit
summary(fit)
yhat_pred1 <- predict(fit,newdata=valid,type="class")
cmatrix1 = table(valid$y,yhat_pred1)
misclass1 = missclassific(cmatrix1, valid)

print(cmatrix1)
print(misclass1)


fit2=tree(y~., data=train, control=tree.control(nobs= dim(train)[1], minsize=7000))

plot(fit2)
text(fit2, pretty=0)
fit2
summary(fit2)
yhat_pred2 <- predict(fit2,newdata=valid,type="class")
cmatrix2 = table(valid$y,yhat_pred2) #conf matrix
misclass2 = missclassific(cmatrix2, valid)

print(cmatrix2)
print(misclass2)


fit3=tree(y ~.,data=train, mindev=0.0005)
plot(fit3)

sumfit3 <- summary(fit3)

yhat_pred3 <- predict(fit3,newdata=valid,type="class")

cmatrix3 = table(valid$y,yhat_pred3)
misclass3 = missclassific(cmatrix3, valid)
```

```
print(cmatrix3)
print(misclass3)

#Task 3
#Checking every train and testscore for pruned trees from 2-50. Saving into vectors.
trainScore=rep(0,50)
testScore=rep(0,50)
for(i in 2:50) {
  prunedTree=prune.tree(fit3,newdata=train, best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
  }
plot(2:50, trainScore[2:50], type="b", col="red",
     ylim=c(5000,12999), main="Deviance plot")
points(2:50, testScore[2:50], type="b", col="blue")

mindev = min(testScore[2:50])
print(mindev)

for (i in 1:50) {
  print(i)
  print(testScore[i])
}

bestPrune = which(testScore[2:50] == mindev)
print(bestPrune)
bestTree=prune.tree(fit3, best = 22)
plot(bestTree)
text(bestTree, pretty=0)


yhat_predB <- predict(bestTree,newdata=valid,type="class")
#F1,Misc,Accuracy.
cmatrixB = table(valid$y,yhat_predB)
misclassB = missclassific(cmatrixB, valid)
print(cmatrixB)
print(misclassB)


accuracy=((cmatrixB[2,2]+cmatrixB[1,1])/(cmatrixB[1,2]+cmatrixB[1,1]+cmatrixB[2,1]+cmatrixB[2,2]))

print(accuracy)

precision=(cmatrixB[2,2]/(cmatrixB[1,2]+cmatrixB[2,2]))
```

```
print(precision)

recall=(cmatrixB[2,2]/(cmatrixB[2,1]+cmatrixB[2,2]))
print(recall)

F1=2*(precision*recall/(precision+recall))
print(F1)


#TASK 5

fit5 = tree(y ~., data=train, control=tree.control(nobs=dim(train)[1],mindev=0.0005))
#lägga till pruned tree här? Jag säger nej!

fit5pred <- predict(fit5, newdata=test, type="vector")

conf5table=table(test$y, ifelse(fit5pred[,2]/fit5pred[,1]>5, "Yes", "No")) #Adding the loss matrix
print(conf5table)
#F1,Misc,Accuracy.
misc5=missclassific(conf5table,test)
print(misc5)

accuracy5=((conf5table[2,2]+conf5table[1,1])/(conf5table[1,2]+conf5table[1,1]+conf5table[2,1]+conf5t

print(accuracy5)

precision5=(conf5table[2,2]/(conf5table[1,2]+conf5table[2,2]))

print(precision5)

recall5=(conf5table[2,2]/(conf5table[2,1]+conf5table[2,2]))
print(recall5)

F1_5=2*(precision5*recall5/(precision5+recall5))
print(F1_5)

#TASK6
#Optimal tree

fit6 = tree(y ~., data=train, control=tree.control(nobs=dim(train)[1],mindev=0.0005))
opt6 = prune.tree(fit6, best=22)
fit6pred <- predict(fit6, newdata=test, type="vector")
pi <- seq(0.00,1,0.05)

fpr <- c(rep(0,length(pi)))
tpr <- c(rep(0,length(pi))) #Make matrix with 0's len pi.
```

```
for(i in 1:length(pi)){
  fit6pred.tree <- ifelse(fit6pred[,2]>pi[i], 1,0) #Applying the function to our predicted tree from
  fit6c = table(test$y,fit6pred.tree)

  if (is.na(table(fit6pred.tree)[2])) {
    if (colnames(fit6c)[1] == "yes") {#Making the matrix 2x2
      fit6c = cbind(c(0,0), fit6c)
    } else {
      fit6c = cbind(fit6c, c(0,0))
    }
  }

  fpr[i]= fit6c[1,2]/(fit6c[1,2]+fit6c[1,1]) #Saving FPR and TPR values
  tpr[i]= fit6c[2,2]/(fit6c[2,1]+fit6c[2,2])
  print(tpr[i])
  }

#Logistic regression
library(nnet)
fprlog <- c(rep(0,length(pi)))
tprlog <- c(rep(0,length(pi))) #Make matrix with 0's len pi.

logistic2 <- multinom(test$y ~ ., data = test, Hess=T)
p2 <- predict(logistic2, data= test, type = 'probs')
for(i in 1:length(pi)){
  prediction4 <- ifelse(p2>pi[i],'yes','no')
  table2 <- table(test$y,prediction4)
  if (is.na(table(prediction4)[2])) {
    if (colnames(table2)[1] == "yes") { #Making the matrix 2x2
      table2 = cbind(c(0,0), table2)
    } else {
      table2 = cbind(table2, c(0,0))
    }
  }
  fprlog[i]= table2[1,2]/(table2[1,2]+table2[1,1])
  tprlog[i]= table2[2,2]/(table2[2,1]+table2[2,2])
  print(tprlog[i])
}

#Plotting ROC
plot(fpr,tpr, xlab="FPR", ylab="TPR", main="ROC-Curve",sub="Red: Optimal tree, Blue: Logistic Regres
points(fprlog, tprlog, type="l", col="blue")
```

4.3 Code for Assignment 3

```
#Assignment 3
 communities <- read.csv("communities.csv",stringsAsFactors =
FALSE) set.seed(12345)
### Task 1
#normalizing
normalize <- function(x) {

  return ((x - min(x)) / (max(x) - min(x)))
}
data = communities
for (i in 1:100){
  data[i] = normalize(communities[i])
}

#manual PCA with eigen()
n = length(data[,2]) #sample size
covdata = cov(data)
eig = eigen(covdata)
pcVar = eig$values / (n-1)
sum(pcVar[2])/sum(pcVar)

### Task 2
res = princomp(data)

#(((maybe not necessary
lambda = res$sdev^2
a = as.double(sprintf("%2.3f",lambda/sum(lambda)*100))
#)))

#PCA values by order of importance
res$loadings

#PCA traceplot for first component only
plot(res$loadings[,1])

#the 5 most important variables for PC1
sort(abs(res$loadings[,1]), decreasing = TRUE)
#PctHousNoPhone: percent of occupied housing units without phone (+)
```

```
#(in 1990, this was rare!)
#PctPopUnderPov: percentage of people under the poverty level (+)
#RentHighQ: rental housing - upper quartile rent (-)
#medIncome: median household income (-)
#PctYoungKids2Par: percent of kids 4 and under in two parent households (-)

#variance = 0
#for (i in 1:100){
#  variance = variance + a[i]
#  if(variance > 95){
#    print(i) #answer is 33
#    break
#  }
#}

#PC score plot with colors of voilentCrimesPerPop
library(ggplot2)
ggplot(as.data.frame(res$scores), aes(x=Comp.1, y=Comp.2, colour=data$ViolentCrimesPerPop)) +
  geom_point(size=2)

### Task 3
set.seed(12345)
n=dim(data)[1]
id = sample(1:n, floor(n*0.5)) #for 0-50
training = data[id,]
test = data[-id,]

# linear regression model
model <- lm(ViolentCrimesPerPop~., data = training)
pred_train = fitted(model)
pred_test = predict(model, newdata=test)

answer_train = training$ViolentCrimesPerPop
answer_test = test$ViolentCrimesPerPop

MSE_train = mean((answer_train - pred_train)^2)
MSE_test = mean((answer_test - pred_test)^2)
summary(model)

### Task 4 ###
# starting value of theta (0,0,0,0,..,0).
k = 0
test_MSE_list = list()
train_MSE_list = list()
myCostFunc <- function(theta, x_test, y_test, x_train, y_train){
  y_hat_test = x_test %*% theta
```

```
  y_hat_train = x_train %*% theta
  .GlobalEnv$k = .GlobalEnv$k + 1
  test_MSE = mean((y_hat_test-y_test)^2)
  train_MSE = mean((y_hat_train-y_train)^2)
  .GlobalEnv$test_MSE_list[k] = test_MSE
  .GlobalEnv$train_MSE_list[k] = train_MSE
  if(.GlobalEnv$k == 1400){ #1400 was found in plot as turning point
    .GlobalEnv$early_stop_theta = theta
  }
  return (test_MSE)
}
x_train = as.matrix(training[-101]) # variables
y_train = as.matrix(training[101])  # output answer

x_test = as.matrix(test[-101])  # variables
y_test = as.matrix(test[101])   # output answer


theta = as.matrix(rep(0,dim(x_test)[2]))
early_stop_theta = theta

result = optim(par = theta,fn = myCostFunc, x_test = x_test, y_test = y_test,
               x_train=x_train, y_train=y_train, method = "BFGS")
x1 = (unlist(test_MSE_list))
x2 = (unlist(train_MSE_list))
plot(x1, type="l", col="red",main = "MSE on test data over all iteration")
lines(x2, color="blue")
iteration = 1:k
df = data.frame(iteration,x1,x2)

#colors <- c("Test" = "cyan", "Training" = "orange")
ggplot(df,aes(x = iteration)) +
  geom_line(aes(y = x1, color = "Test")) +
  geom_line(aes(y = x2, color = "Train")) +
  labs(x = "Iteration",
       y = "MSE",
       title = "MSE of iterative BFGS method (900 to 1800)",
       color = " " #to remove legend title
       ) +
  #xlim(900,1800) +
  #ylim(0.0181,0.0184) +
  theme(plot.title = element_text(size=14, face="bold", hjust = 0.5))

#Comparing Linear model (lm) and BFGS iterative model (optim)
# Using theta optimal from optim() AND theta from early stop condition.
```

```
#Optimal theta hat from optim()
theta_hat_optim = result$par
y_hat_optim_test = x_test %*% theta_hat_optim
y_hat_optim_training = x_train %*% theta_hat_optim
mse_optim_test = mean((y_test - y_hat_optim_test)^2)
mse_optim_training = mean((y_train - y_hat_optim_training)^2)

#Optimal theta hat from early stop condition
# achieved from storing theta at iteration 1400.
# variable is called early_stop_theta
y_hat_early_stop_test = x_test %*% early_stop_theta
y_hat_early_stop_training = x_train %*% early_stop_theta
mse_early_stop_test = mean((y_test - y_hat_early_stop_test)^2)
mse_early_stop_training = mean((y_train - y_hat_early_stop_training)^2)

# optimal "theta" from lm() function. Now using predict()
# instead of normal matrix multiplication
y_hat_lm_test = predict(model,newdata = as.data.frame(x_test))
y_hat_lm_training = predict(model,newdata = as.data.frame(x_train))
# above line should be same as fitted(model). Wanted cohesion instead.
mse_lm_test = mean((y_test - y_hat_lm_test)^2)
mse_lm_training = mean((y_train - y_hat_lm_training)^2)
```