```
library(glmnet)
library(tree)
library(caret)
library(dplyr)

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

#Read in data from "tecator.csv" and store in a data frame called "tecator"
tecator = read.csv("tecator.csv", header = T)

# Get the number of rows in the data frame and store in a variable called "n"
n = dim(tecator)[1]

# Set the seed for the random number generator
set.seed(12345)

# Create a new data frame called "df" that contains all the columns from the
"tecator" data frame except for the first column
df = data.frame(tecator[c(2:102)])

# Generate a random sample of row indices from the "tecator" data frame
id=sample(1:n, floor(n*0.5))

# Use the "id" indices to select a subset of rows from the "df" data frame and
store them in a new data frame called "train"
train = df[id,]

# Use the "id" indices to select all rows except those in the "id" subset from
the "df" data frame and store them in a new data frame called "test"
test = df[-id,]

# Fit a linear regression model to the "train" data frame, with the "Fat"
column as the response variable and all the other columns as predictor
variables
fit = lm(Fat~ ., data = train)

# Use the "fit" model to make predictions on the training data and store the
predictions in a vector called "train_preds"
train_preds = predict(fit, train)

# Use the "fit" model to make predictions on the test data and store the
predictions in a vector called "test_preds"
test_preds = predict(fit, test)

# Generate a summary of the "fit" model and store it in an object called "sum"
sum = summary(fit)

# Calculate the mean squared error (MSE) of the model's predictions on the
training data and store in a variable called "MSE_train"
```

```
MSE_train=mean((train_preds - train$Fat)^2)

# Calculate the MSE of the model's predictions on the test data and store in a
variable called "MSE_test"
MSE_test=mean((test_preds - test$Fat)^2)

# Print "Test error" and the MSE of the model's predictions on the test data
print("Test error")
MSE_test

# Print "Train error" and the MSE of the model's predictions on the training
data
print("Train error")
MSE_train

 "
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 λλ) and interpret this plot. What value of the penalty factor can
be
chosen if we want to select a model with only three features?"

# Get the "Fat" column from the "train" data frame and store in a variable
called "y"
y = train$Fat

# Get the first 100 columns from the "train" data frame and store in a
variable called "x"
x = train[1:100]

# Fit a lasso regression model to the "x" and "y" data using the "glmnet"
function, with a Gaussian error distribution
model_lasso= glmnet(as.matrix(x), as.matrix(y), alpha=1,family="gaussian")

# Plot the model's coefficients as a function of the regularization parameter
"lambda"
plot(model_lasso, xvar = "lambda")

# Use the "model_lasso" model to make predictions on the "x" data and store
the predictions in a variable called "ynew"
ynew=predict(model_lasso, newx=as.matrix(x), type="response")

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

# Get the "Fat" column from the "train" data frame and store in a variable
called "y"
y = train$Fat

# Get the first 100 columns from the "train" data frame and store in a
variable called "x"
x = train[1:100]
```

```
# Fit a ridge regression model to the "x" and "y" data using the "glmnet"
function, with a Gaussian error distribution
model_lasso= glmnet(as.matrix(x), as.matrix(y), alpha=0,family="gaussian")

# Plot the model's coefficients as a function of the regularization parameter
"lambda"
plot(model_lasso, xvar = "lambda")

# Use the "model_lasso" model to make predictions on the "x" data and store
the predictions in a variable called "ynew"
ynew=predict(model_lasso, newx=as.matrix(x), type="response")
```

"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 $\lambda\lambda$ and comment how the CV score changes with log $\lambda\lambda$. Report the optimal $\lambda\lambda$ and how many variables were chosen in this model. Does the information displayed in the plot suggests that the optimal $\lambda\lambda$ value results in a statistically
significantly better prediction than log $\lambda\lambda$ = −4? Finally, create a scatter plot
of the original test versus predicted test values for the model corresponding to optimal lambda and comment whether the model predictions are good."

```
# Fit a lasso regression model to the "x" and "y" data using cross-validation
and the "cv.glmnet" function, with a Gaussian error distribution
model_lasso= cv.glmnet(as.matrix(x), as.matrix(y), alpha=1,family="gaussian")

# Get the value of the regularization parameter "lambda" that resulted in the
lowest cross-validation error
lambda_min = model_lasso$lambda.min

# Plot the model's coefficients as a function of the regularization parameter
"lambda"
plot(model_lasso, xvar = "lambda")

# Fit a lasso regression model to the "x" and "y" data using the "glmnet"
function and the "lambda_min" value, with a Gaussian error distribution
better_model = glmnet(as.matrix(x), as.matrix(y), lambda = lambda_min, alpha =
1, family = "gaussian")

# Use the "better_model" model to make predictions on the "x" data and store
the predictions in a variable called "ynew"
ynew=predict(better_model, newx=as.matrix(x), s = lambda_min ,
type="response")

# Plot the original "y" values against the predicted "ynew" values, with a red
color and a labeled x-axis and y-axis
plot(y, ynew, xlab = "Original", ylab = "Predicted",col = "red", main =
"Scatter plot")

# Add a line with slope 1 and intercept 0 to the plot
abline(0,1)
```

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

```
# Read in data from "bank-full.csv" and store in a data frame called "d"
d = read.csv("bank-full.csv", sep = ";", stringsAsFactors = TRUE)

# Create a copy of the "d" data frame called "data"
data = d

# Remove the "duration" column from the "data" data frame
data$duration = c()

# Get the "y" column from the "d" data frame and store in a variable called
"output"
output = d['y']

# Get the number of rows in the "data" data frame and store in a variable
called "n"
n = dim(data)[1]

# Set the seed for the random number generator
set.seed(12345)

# Generate a random sample of row indices from the "data" data frame
id=sample(1:n, floor(n*0.4))

# Use the "id" indices to select a subset of rows from the "data" data frame
and store them in a new data frame called "train"
train=data[id,]

# Get the indices of the rows in the "data" data frame that are not in the
"id" subset
id1=setdiff(1:n, id)

# Set the seed for the random number generator
set.seed(12345)

# Generate a random sample of row indices from the "id1" subset
id2=sample(id1, floor(n*0.3))

# Use the "id2" indices to select a subset of rows from the "data" data frame
and store them in a new data frame called "valid"
valid=data[id2,]

# Get the indices of the rows in the "id1" subset that are not in the "id2"
subset
id3=setdiff(id1,id2)
```

```
# Use the "id3" indices to select a subset of rows from the "data" data frame
and store them in a new data frame called "test"
test=data[id3,]

"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."

# Fit a decision tree model to the "train" data using the "tree" function,
using all of the columns except the "y" column as predictors
fit=tree(as.factor(y)~., data=train)

# Plot the decision tree model
plot(fit)

# Add text labels to the plot to show the splits at each node of the tree
text(fit, pretty=0)

# Fit a decision tree model to the "train" data using the "tree" function,
using all of the columns except the "y" column as predictors, with a minimum
size of 7000 for each leaf
fit2=tree(as.factor(y)~., data=train, minsize=7000)

# Plot the decision tree model
plot(fit2)

# Add text labels to the plot to show the splits at each node of the tree
text(fit2, pretty=0)

# Fit a decision tree model to the "train" data using the "tree" function,
using all of the columns except the "y" column as predictors, with a minimum
relative decrease in impurity of 0.0005 required to split a node
fit3=tree(as.factor(y)~., data=train, mindev=0.0005)

# Plot the decision tree model
plot(fit3)

# Add text labels to the plot to show the splits at each node of the tree
text(fit3, pretty=0)

  "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)."
```

```
# Make predictions on the "train" data using the first decision tree model and
store the predictions in a variable called "Yfit_t"
Yfit_t=predict(fit, newdata=train, type="class")

# Create a confusion matrix for the predictions on the "train" data using the
first decision tree model
t1<-table(train$y,Yfit_t)

# Calculate the misclassification rate for the predictions on the "train" data
using the first decision tree model
mis_t1 <- 1-sum(diag(t1))/sum(t1)

# Make predictions on the "train" data using the second decision tree model
and store the predictions in a variable called "Yfit_t2"
Yfit_t2=predict(fit2, newdata=train, type="class")

# Create a confusion matrix for the predictions on the "train" data using the
second decision tree model
t2<-table(train$y,Yfit_t2)

# Calculate the misclassification rate for the predictions on the "train" data
using the second decision tree model
mis_t2 <- 1-sum(diag(t2))/sum(t2)

# Make predictions on the "train" data using the third decision tree model and
store the predictions in a variable called "Yfit_t3"
Yfit_t3=predict(fit3, newdata=train, type="class")

# Create a confusion matrix for the predictions on the "train" data using the
third decision tree model
t3<-table(train$y,Yfit_t3)

# Calculate the misclassification rate for the predictions on the "train" data
using the third decision tree model
mis_t3 <- 1-sum(diag(t3))/sum(t3)

# Make predictions on the "valid" data using the first decision tree model and
store the predictions in a variable called "Yfit_v"
Yfit_v=predict(fit, newdata=valid, type="class")

# Create a confusion matrix for the predictions on the "valid" data using the
first decision tree model
v1<-table(valid$y,Yfit_v)

# Calculate the misclassification rate for the predictions on the "valid" data
using the first decision tree model
mis_v1<-1-sum(diag(v1))/sum(v1)

# Make predictions on the "valid" data using the second decision tree model
and store the predictions in a variable called "Yfit_v2"
Yfit_v2=predict(fit2, newdata=valid, type="class")

# Create a confusion matrix for the predictions on the "valid" data using the
second decision tree model
```

```
v2<-table(valid$y,Yfit_v2)

# Calculate the misclassification rate for the predictions on the "valid" data
using the second decision tree model
mis_v2<-1-sum(diag(v2))/sum(v2)

# Make predictions on the "valid" data using the third decision tree model and
store the predictions in a variable called "Yfit_v3"
Yfit_v3=predict(fit3, newdata=valid, type="class")

# Create a confusion matrix for the predictions on the "valid" data using the
third decision tree model
v3<-table(valid$y,Yfit_v3)

# Calculate the misclassification rate for the predictions on the "valid" data
using the third decision tree model
mis_v3<-1-sum(diag(v3))/sum(v3)

# Print the misclassification rates for the predictions on the "train" and
"valid" data using the first decision tree model
print("1) Training and validation")
print(mis_t1)
print(mis_v1)

# Print the misclassification rates for the predictions on the "train" and
"valid" data using the second decision tree model
print("2) Training and validation")
print(mis_t2)
print(mis_v2)

# Print the misclassification rates for the predictions on the "train" and
"valid" data using the third decision tree model
print("3) Training and validation")
print(mis_t3)
print(mis_v3)


"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."


# Create two vectors called "trainScore" and "testScore" with length 50 and
filled with 0s
trainScore=rep(0,50)
testScore=rep(0,50)

# Loop through the numbers 2 to 50
for(i in 2:50) {
  # Prune the third decision tree model to have "i" leaves
  prunedTree=prune.tree(fit3,best=i)
```

```
  # Make predictions on the "valid" data using the pruned decision tree model
and store the predictions in a variable called "pred"
  pred=predict(prunedTree, newdata=valid, type="tree")

  # Calculate the deviance for the pruned decision tree model on the "train"
data and store it in the "trainScore" vector
  trainScore[i]=deviance(prunedTree)

  # Calculate the deviance for the predictions on the "valid" data using the
pruned decision tree model and store it in the "testScore" vector
  testScore[i]=deviance(pred)
}

# Plot the deviances for the pruned decision tree models on the "train" data
(in red) and the deviances for the predictions on the "valid" data (in blue)
plot(2:50, trainScore[2:50], type="b", col="red", ylim=c(min(testScore[-1]),
max(trainScore[-1])))
points(2:50, testScore[2:50], type="b", col="blue")

# Find the number of leaves that gives the minimum deviance on the "valid"
data
optimal_leaves = which.min(testScore[2:50])

# Prune the third decision tree model to have the optimal number of leaves
finalTree=prune.tree(fit3, best=optimal_leaves)

# Make predictions on the "valid" data using the final pruned decision tree
model and store the predictions in a variable called "finalfit"
finalfit=predict(finalTree, newdata=valid, type="class")

# Create a confusion matrix for the predictions on the "valid" data using the
final pruned decision tree model
tab = table(valid$y,finalfit)

# Plot the final pruned decision tree model
plot(finalTree)

# Print the summary of the third decision tree model
summary(fit3)

# Print the summary of the final pruned decision tree model
summary(finalTree)
#text(fit3, pretty=0)

"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."

# Prune the decision tree model fit3 using the optimal number of leaves
opt_tree <- prune.tree(fit3, best=optimal_leaves)

# Make predictions on the test set using the pruned tree
ffitTest <- predict(opt_tree, newdata=test, type="class")
```

```
# Calculate the confusion matrix
c_m <- table(test$y, ffitTest)

# Print the confusion matrix
c_m

# Calculate the accuracy
acc <- sum(c_m[1], c_m[4])/sum(c_m[1:4])

# Calculate the precision
prec <- c_m[4] / sum(c_m[4], c_m[2])

# Calculate the recall
recall <- c_m[4] / sum(c_m[4], c_m[3])

# Calculate the F1 score
f1_score <- (2*(recall*prec)) / (recall + prec)

"Perform a decision tree classification of the test data with the following
loss
matrix,
732A99/732A68/ TDDE01 Machine Learning
Division of Statistics and Machine Learning
Department of Computer and Information Science
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."

# Make predictions on the test set using the pruned tree model
predtree5 <- predict(opt_tree, newdata=test, type="vector")

# Extract the probability of the positive class from the predictions
probY <- predtree5[,2]

# Extract the probability of the negative class from the predictions
probN <- predtree5[,1]

# Create a new vector of predictions based on the ratio of the probabilities
pred5 <- ifelse((probY/probN)>1/5, "yes", "no")

# Calculate the confusion matrix for the new predictions
c_m <- table(test$y, pred5)

# Print the confusion matrix
c_m

# Calculate the accuracy
acc <- sum(c_m[1], c_m[4])/sum(c_m[1:4])

# Calculate the precision
prec <- c_m[4] / sum(c_m[4], c_m[2])

# Calculate the recall
recall <- c_m[4] / sum(c_m[4], c_m[3])
```

```
# Calculate the F1 score
f1_score <- (2*(recall*prec)) / (recall + prec)


  "Use the optimal tree and a logistic regression model to classify the test
data by
using the following principle:
where ππ = 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 precisionrecall curve could be a better option here?"
# Train a decision tree model on the training data
optimalTree <- tree(as.factor(y)~., data=train, mindev=0.0005)

# Prune the tree using the optimal number of leaves
optimalTree <- prune.tree(optimalTree, best=21)

# Create a sequence of probability thresholds
pi <- seq(0.05, 0.95, 0.05)

# Train a logistic regression model on the training data
logic_model <- glm(as.factor(y)~.,data = train ,family="binomial")

# Make predictions on the test set using the logistic regression model
pred6_probY = predict(logic_model, newdata = test, type = "response")

# Calculate the probability of the negative class
pred6_probN = 1 -pred6_probY

# Make predictions on the test set using the pruned decision tree model
tree_pred = predict(optimalTree, newdata = test, type = "vector")

# Initialize vectors for false positive rate and true positive rate for both
models
fpr_1 <- c(1:length(pi))
tpr_1 <- c(1:length(pi))
fpr_2 <- c(1:length(pi))
tpr_2 <- c(1:length(pi))

# Initialize the predictions for the decision tree model
pred6 <- tree_pred[, 1]

# Loop through each probability threshold
for (i in 1:length(pi)){

  # Decision tree model
  tpr_1[i] = 0
  fpr_1[i] = 0

  # Create predictions for the decision tree model based on the current
threshold
  pred6 <- ifelse(tree_pred[,2]>pi[i], "yes", "no")
```

```
  # Calculate the confusion matrix for the decision tree model
  pred6_matrix <- table(test$y, pred6)

  # If the confusion matrix has more than one column, calculate TPR and FPR
  if(ncol(pred6_matrix) > 1){
    tpr_1[i] <- pred6_matrix[2,2] / (pred6_matrix[2,1]+pred6_matrix[2,2])
    fpr_1[i] <- pred6_matrix[1,2] / (pred6_matrix[1,1]+pred6_matrix[1,2])
  }

  # Logistic regression model
  tpr_2[i] = 0
  fpr_2[i] = 0

  # Create predictions for the logistic regression model based on the current
threshold
  pred6_logic <- ifelse(pred6_probY > pi[i], "yes", "no")

  # Calculate the confusion matrix for the logistic regression model
  pred6_logic_matrix <- table(test$y,pred6_logic)

  # Calculate TPR and FPR for the logistic regression model
  tpr_2[i] <- pred6_logic_matrix[2,2] /(pred6_logic_matrix[2,1]
+pred6_logic_matrix[2,2])
  fpr_2[i] <- (pred6_logic_matrix[1,2] / (pred6_logic_matrix[1,1]
                                          +pred6_logic_matrix[1,2]))
}

# Plot the TPR and FPR for the decision tree model
plot(fpr_1,tpr_1, type='l',xlim = c(0,1), ylim = c(0,1),
     xlab='FPR', ylab='TPR', col='red')

# Plot the TPR and FPR for the logistic regression model
lines(fpr_2,tpr_2, type='l',xlim = c(0,1),
      xlab='FPR', ylab='TPR', col='blue')

# Add a reference line to the plot
abline(0,1, lty = 5)

# Add a legend to the plot
legend(x = "bottomright" , col = c("red", "blue", "black"),
       legend = c("Tree","Logistic", "Reference"), lwd = 2,title = "Lines",
       lty = c(1,1,5))


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

  rm(list = ls(all = TRUE))
  graphics.off()
  shell("cls")
  # Read in the communities data from a CSV file
  data = read.csv(file = "communities.csv", header = TRUE)
```

```
  # Create a logical vector indicating which column is "ViolentCrimesPerPop"
  index <- names(data) %in% "ViolentCrimesPerPop"

  # Scale the data, excluding "ViolentCrimesPerPop"
  data.scaled <- scale(x = data[, !index], center = TRUE, scale = TRUE)

  # Calculate the eigenvalues and eigenvectors of the covariance matrix of the
original data
  e = eigen(cov(data[, -1]))

  # Calculate the eigenvalues and eigenvectors of the covariance matrix of the
scaled data
  e.scaled = eigen(cov(data.scaled))

  # Calculate the cumulative sum of the scaled eigenvalues
  cum_var = cumsum(e.scaled$values/sum(e.scaled$values))

  # Calculate the number of eigenvalues needed to explain 95% of the variance
  sum(cum_var<0.95)

  # Calculate the proportion of the variance explained by the first two
eigenvalues
  e.scaled$values[1:2]/sum(e.scaled$values)

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

  # Read in the communities data from a CSV file
  data = read.csv(file = "communities.csv", header = TRUE)

  # Create a logical vector indicating which column is "ViolentCrimesPerPop"
  index <- names(data) %in% "ViolentCrimesPerPop"

  # Scale the data, excluding "ViolentCrimesPerPop"
  data.scaled <- scale(x = data[, !index], center = TRUE, scale = TRUE)

  # Perform principal component analysis on the scaled data
  pr=princomp(data.scaled)

  # Calculate the eigenvalues
  lambda=pr$sdev^2

  # Calculate the proportion of variance explained by each eigenvalue
  var = sprintf("%2.3f",lambda/sum(lambda)*100)

  # Extract the loadings for the first principal component
  ev1 = pr$loadings[,1]
```

```
  # Find the top five absolute loadings for the first principal component
  ev1[order(abs(ev1),decreasing = TRUE)[1:5]]

  # Load the ggfortify library for plot visualization
  library(ggfortify)

  # Create a scatterplot matrix of the first two principal components, colored
by "ViolentCrimesPerPop"
  autoplot(pr, data = data, colour = "ViolentCrimesPerPop")

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

  # Read in the communities data from a CSV file
  df = read.csv("communities.csv")

  # Scale the data
  df = scale(df, TRUE, TRUE)

  # Set a seed for reproducibility
  set.seed(12345)

  # Split the data into a training set and a test set
  n <- dim(df)[1]
  id <- sample(1:n,floor(n*0.5))
  df_train <- data.frame(df[id,])
  df_test <- data.frame(df[-id,])

  # Fit a linear regression model to the training data
  lr = lm(ViolentCrimesPerPop ~ .,df_train)

  # Make predictions on the training and test sets
  train.pred = predict(lr, df_train)
  test.pred = predict(lr, df_test)

  # Calculate the mean squared error (MSE) for the training and test sets
  train_MSE = mean((train.pred - df_train$ViolentCrimesPerPop) ^ 2)
  test_MSE = mean((test.pred - df_test$ViolentCrimesPerPop) ^ 2)

  # Print the MSE for the training and test sets
  print("Train error")
  train_MSE
  print("Test error")
  test_MSE
"Implement a function that depends on parameter vector θθ and represents the
cost function for linear regression without intercept on the training data
set.
Afterwards, use BFGS method (optim() function without gradient specified)
```

to optimize this cost with starting point $\theta\theta 0$ = 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 to the early stopping criterion. Compute the training and test error in the optimal model, compare them with results in step 3 and make conclusions.
a. Hint 1: don't store parameters from each iteration (otherwise it will take a lot of memory), instead compute and store test errors directly.
b. Hint 2: discard some amount of initial iterations, like 500, in your plot to make the dependences visible."

```
  # Initialize empty vectors for storing training and test errors
  train_error <<- numeric(0)
  test_error <<- numeric(0)

  # Set the seed for reproducibility
  set.seed(12345)

  # Define the cost function
  cost <- function(theta, train, acc_train, test, acc_test){
    # Calculate predictions on the training and test data using theta
    pred_train = train %*% theta
    pred_test = test %*% theta

    # Calculate MSE of predictions on training and test data
    mse_train = mean((acc_train-pred_train)^2)
    mse_test = mean((acc_test - pred_test)^2)

    # Append MSEs to the train_error and test_error vectors
    train_error <<- append(train_error,mse_train)
    test_error <<- append(test_error,mse_test)

    # Return MSE on training data as cost
    return(mse_train)
  }

  # Create matrices of predictors and responses for the training and test data
  trainy = as.matrix(df_train[,1:(dim(df_train)[2]-1)])
  acc_train = as.matrix(df_train['ViolentCrimesPerPop'])
  testy = as.matrix(df_test[,1:(dim(df_test)[2]-1)])
  acc_test = as.matrix(df_test['ViolentCrimesPerPop'])

  # Initialize theta as a matrix of all 0s with the same number of columns as
trainy
  theta = numeric(dim(trainy)[2])
  theta = as.matrix(theta)

  # Use the optim function to find the optimal value of theta
  opt = optim(par=theta, fn=cost, train = trainy, acc_train = acc_train,
              test=testy, acc_test=acc_test, method = "BFGS")

  # Extract the optimal value of theta from the optim function output
  opt_theta = opt$par
```

```
  # Calculate the MSE of the predictions made using opt_theta on the training
and test data
  train_opt_error = opt$value
  test_opt_error = mean((acc_test - (testy %*% opt_theta))^2)

  # Print the MSEs
  print("calculated optimal train")
  train_opt_error
  print("Lm train error")
  train_MSE
  print("calculated optimal test")
  test_opt_error
  print("Lm test error")
  test_MSE

  # Create a logical vector indicating which elements of train_error and
test_error should be plotted
  excluded = c(TRUE,rep(FALSE,500))

  # Extract the relevant elements of train_error and test_error
  rest_train = train_error[excluded]
  rest_test = test_error[excluded]

  # Find the index of the minimum test error
  test_min_ind = which(test_error==min(test_error))

  print("Early stopping index and MSE")
  test_min_ind
  min(test_error)

  plot(rest_train, xlim=c(0,length(rest_train)), ylim=c(0,1.5), col = "blue")
  points(rest_test, col="red")

  # Add horizontal lines at the final MSE values for the training and test
data
  lines(c(0,1000), rep(train_MSE, 2), col="blue")
  lines(c(0,1000), rep(test_MSE, 2), col="red")

  # Print the early stopping index and minimum test error
  print("Early stopping index and MSE")
  test_min_ind
  min(test_error)
```