Statistical Learning, LAB 1

Gustav Felländer(gufe0008), Axel Eriksson(axer0005)

2023-11-16

Task 1

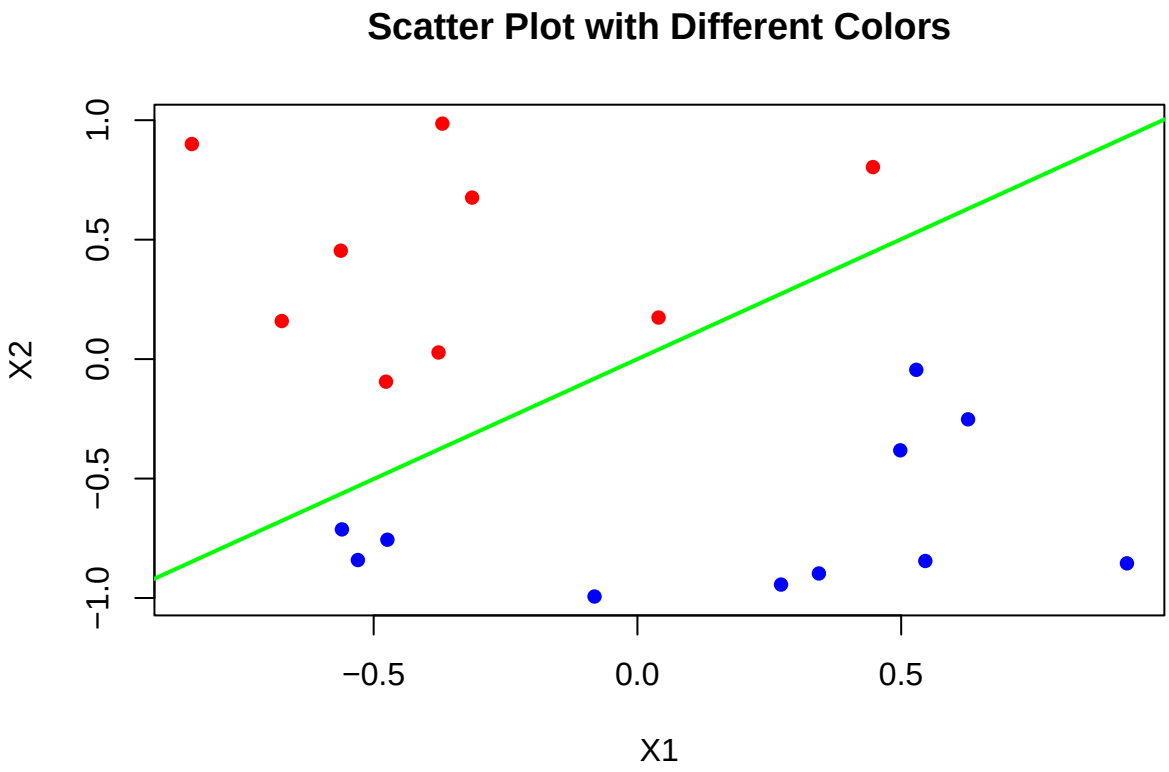
In this task we have implemented the Perceptron algorithm in R. The algorithm has been used determine a decision boundary from data generated randomly between -1 and 1. The ensure the performance of the algorithm a visualization of the decision boundary has been made.

```
rm(list=ls())
set.seed(201606)
N <- 20
x1 <- runif(N,-1,1)
x2 <- runif(N,-1,1)
X <- cbind(x1,x2)
y <- ifelse(x2>x1,-1,1)

#Plot the two different groups
colors <- ifelse(y == 1, "blue", "red")
plot(X[, 1], X[, 2], col = colors, pch = 16,
     main = "Scatter Plot with Different Colors", xlab = "X1", ylab = "X2")

#Add ones
X <- cbind(1,x1,x2)

#Initial guess for weights
w <- c(-10,11,21)

max_iter = 100
for(j in 1:max_iter){
  misclassified <- 0
  # Loop over the dataset
  for(i in 1:N){

    # Checks for misclassification
    if(sign(w%*%X[i,]) != y[i]) {

      # Update weights!! w
      w <- w + y[i]%*% X[i,]
      misclassified <- misclassified + 1
    }
  }
  # Checks if the algorithm has converged
  if(misclassified == 0) {
    cat("Converged in ",j)
```

```
    break
  }
}
```

```
## Converged in  8
```

```
#Verify that the obtained weights defines a good decision boundary
x_values <- seq(-1, 1, length.out = 100)
y_values <- (-w[1] - w[2] * x_values) / w[3]
line = lines(x_values, y_values, col = "green", lwd = 2)
```



Task 2

In this task we have implemented Ridge Regression and trained a predictive model with the Boston data. We have tested different values on the tuning parameter and with the help of a 10-fold-cross validation determined the performance of the parameters.

Extract the training and testing data

```
library(MASS)
data(Boston)
set.seed(2023)

# Prepare the data in training and test sets
```

```r
train_indices <- sample(1:nrow(Boston), 400)

train_data <- Boston[train_indices, ]
test_data <- Boston[-train_indices, ]

trainY <- train_data$medv
trainX <- train_data[, !(names(train_data) %in% c("medv"))]

testY <- test_data$medv
testX <- test_data[,!(names(test_data) %in% c("medv"))]

trainX <- as.matrix(trainX)
testX <- as.matrix(testX)
```

Implementation of Ridge Regression. With the inputs, trainX: Feature variables trainY: Target variable lambda: The parameter

```r
# Solve the LS problem for the optimal weights
ridge_reg <- function(trainX, trainY, lambda) {
  p <- ncol(trainX)
  I <- diag(p)
  w <- solve(t(trainX) %*% trainX + lambda * I) %*% t(trainX) %*% trainY
  return(w)
}
```

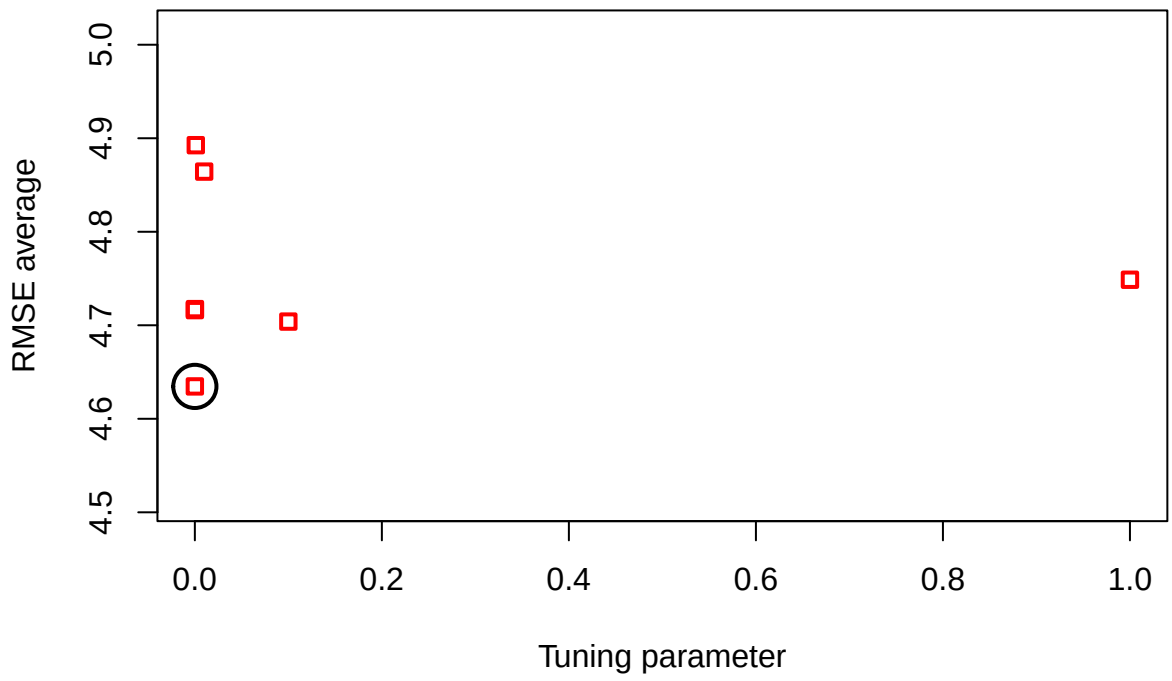
Implementation of a function applying k-fold-cross-validation. trainX: Feature variables trainY: Target variable k: How many times one wants to divied the training set into.

```r
# Function for doing k-fold cross-validation
k_fold <- function(trainX, trainY, k) {

  order <- sample(1:nrow(trainX))
  num_points <- round(nrow(trainX)/k)
  rmse <- numeric(k)

  for (fold in 0:(k-1)) {
    # Determine the amount of folds to use
    start_ind = fold*num_points+1
    end_ind <- (fold+1)*num_points

    testing_idx <- order[start_ind:end_ind]

    # Obtain the sets of features
    test_setX <- trainX[testing_idx,]
    training_setX <- trainX[-testing_idx,]

    # Obtain the sets of targets
    test_setY <- trainY[testing_idx]
    training_setY <- trainY[-testing_idx]

    # Evaluate the model
    predictions <- test_setX%*%ridge_reg(training_setX, training_setY, tuning_param[i])
    # Calculate the root mean squared error
```

```
    rmse[fold] <- sqrt(mean((predictions - test_setY)^2))
  }
  return(rmse)
}
```

Determine which tuning parameter to use to obtain the best performance. Plot of the RMSE average vs tuning parameters. The tuning parameter with the best performance has a circle around it. Finally we have a estimated the performance of the model on the actual testing set.

```
# Set the tuning parameters to loop through
tuning_param = c(0, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1)
tuning_result_rmse <- numeric(length(tuning_param))

# Loop over the tuning parameters
for (i in 1:length(tuning_param)) {
  rmse <- k_fold(trainX, trainY, 10)
  tuning_result_rmse[i] <- mean(rmse)
}

# Show the result
plot(c(tuning_param),tuning_result_rmse,
     pch=0,col="red",cex=1,lwd=2,asp=1,
     xlab="Tuning parameter",ylab="RMSE average")
best_res <- which.min(tuning_result_rmse)
points(tuning_param[best_res], tuning_result_rmse[best_res], pch=1, lwd = 2, cex = 3)
```



```r
# Choose best parameter
best_lambda <- tuning_param[best_res]
cat('The best lambda ',best_lambda)
```

```
## The best lambda  0
```

```r
# Estimate the model performance on the actual testing set
w_reg <- ridge_reg(trainX, trainY, best_lambda)
predictions <- testX%*%w_reg
rmse <- sqrt(mean((predictions - testY)^2))
cat("The performance of the model using the best lambda is:", rmse, "\n")
```

```
## The performance of the model using the best lambda is: 4.264456
```

##Task3

For the last task, our objective was to train a logistic regression model using a LASSO penalty. This was done on some gene expression data of patents that had brain cancer in order to predict their sub type of cancer.

First, we loaded the data set from supplied files and the required packages. After doing this, we drew a random sample of 181 patients in the data set to use as training data for the model.

```r
library(MASS)
library(glmnet)
library(caret)
# ----- Loading ----- #
# Load the data from files
gene_data <- read.table("GeneExpressionData.txt")
meta_data <- read.table("MetaData.txt")

# Draw 181 random samples from the gene data
random_idx <- sample(1:ncol(gene_data), 181)
```

Then, we prepared the data sets by splitting the randomly selected patients into a training set and the rest into a testing set.

```r
# ----- Feature data ----- #

# Training data
train_dataX <- as.matrix(gene_data[2:nrow(gene_data),random_idx])
train_dataX <- apply(train_dataX, c(1, 2), as.numeric)

# Testing data
test_dataX <- as.matrix(gene_data[2:nrow(gene_data),-random_idx])
test_dataX <- apply(test_dataX, c(1, 2), as.numeric)
```

This, was also done in a similar way for the target data, i.e. the data set containing the subtype of cancer of the patients. Also, for simplification we chose to make the types binary instead, so that "IDHmut-non-codel" is represented by a 0, and "IDHmut-codel" are 1's.

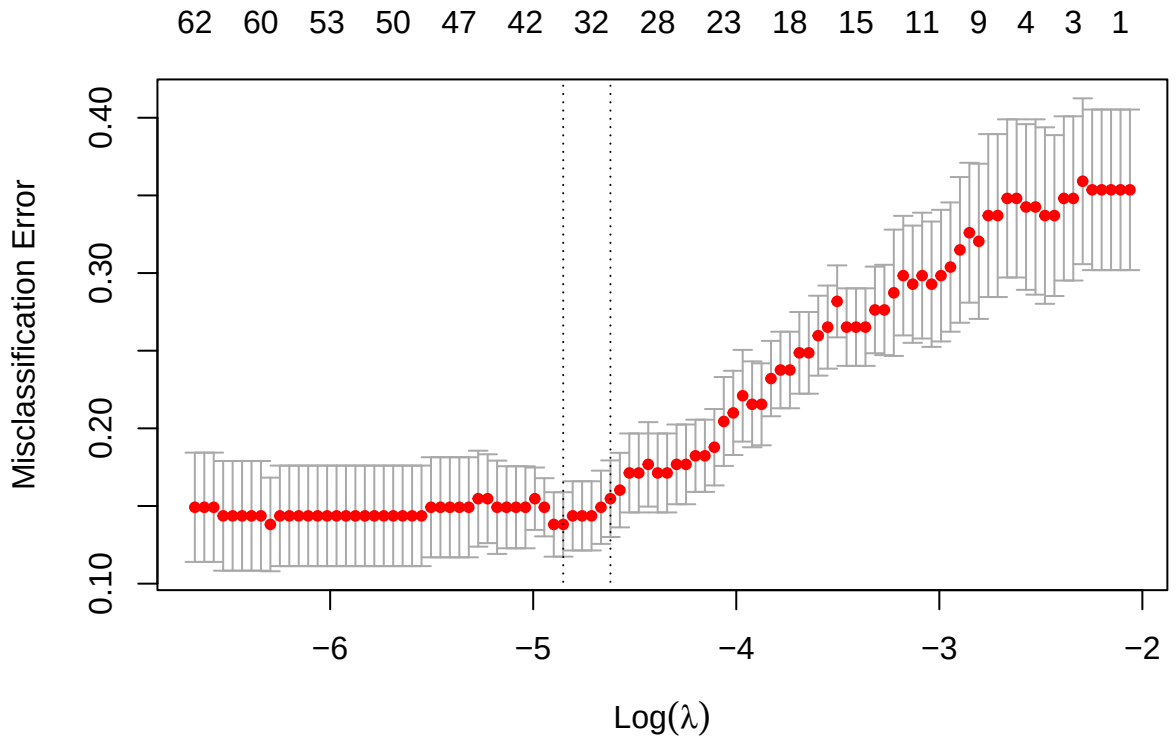
```r
# ----- Targets ----- #

# Training data
text_train_dataY <- meta_data$V31[random_idx]
train_dataY <- ifelse(text_train_dataY == "IDHmut-non-codel", 0, 1)

# Testing data
text_test_dataY <- meta_data$V31[-random_idx]
test_dataY <- ifelse(text_test_dataY == "IDHmut-non-codel", 0, 1)
```

To train the model, we used the glmnet-package. Here, we used the cv.glmnet function in order to enable using 10-fold cross validation for training and deciding a specified best hyperparameter λ for the model to use as LASSO-penalty.

```r
# ----- Training ----- #

# Train the model
model <- cv.glmnet(x = t(train_dataX),
                   y = train_dataY,
                   family = "binomial",
                   type.measure = "class",
                   alpha = 1,
                   nfolds = 10)
```

Lastly, we show the results from the model. This includes a plot over the misclassification rate versus λ, the genes with non-zero coefficients, i.e. the genes that are used by the model, and a confusion matrix along with the kappa statistic.

```r
# ----- Evaluation ----- #
# Show the plot over different lambdas
plot(model)
```




```
c <- coef(model, s = model$lambda.min)
print(cbind(c@i,c@x))
```

```
##       [,1]         [,2]
##  [1,]    0 -7.417394641
##  [2,]    2  1.481898291
##  [3,]    3 -0.024720927
##  [4,]    4  0.002033150
##  [5,]    6  0.110324132
##  [6,]    9  0.150541626
##  [7,]   12  0.034050509
##  [8,]   14  0.036700165
##  [9,]   15  0.227985506
## [10,]   17 -0.033042973
## [11,]   23  0.006722703
## [12,]   26  0.037318897
## [13,]   27 -0.154874382
## [14,]   32  0.057691163
## [15,]   35 -0.037792400
## [16,]   38  0.007845308
## [17,]   40 -0.079899472
## [18,]   41  0.090296850
## [19,]   46 -0.056668753
## [20,]   48  0.087134576
## [21,]   51 -0.084273333
```

```
## [22,]   57  0.087007891
## [23,]   61 -0.030199648
## [24,]   63  0.195243483
## [25,]   64 -0.137212918
## [26,]   74  0.180588678
## [27,]   76 -0.148624851
## [28,]   86  0.069624394
## [29,]   87 -0.029839358
## [30,]  151  0.383678310
## [31,]  152 -0.042464735
## [32,]  160 -0.235326717
## [33,]  192  0.037077713
## [34,]  205 -0.073144517
## [35,]  240 -0.104706484
## [36,]  295  0.053726030
```

```r
# Make predictions on the test set
predictions <- as.numeric(predict(model, t(test_dataX),
                                  s = "lambda.min", type = "class"))
# Evaluate the final model
confusionMatrix(as.factor(predictions), as.factor(test_dataY))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 26  3
##          1  4 13
##
##                Accuracy : 0.8478
##                  95% CI : (0.7113, 0.9366)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 0.002737
##
##                   Kappa : 0.6694
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8667
##             Specificity : 0.8125
##          Pos Pred Value : 0.8966
##          Neg Pred Value : 0.7647
##              Prevalence : 0.6522
##          Detection Rate : 0.5652
##    Detection Prevalence : 0.6304
##       Balanced Accuracy : 0.8396
##
##        'Positive' Class : 0
##
```