Group_A8

Ayesha Gamage-(ayega981)/Muditha Cherangani(mudch175)/Chandika Nishshanka(chache114)

2023-11-19

Group agreement:

Assignment 1, code mainly contributed by Muditha Cherangani(mudch175).

Assignment 2, coded by Ayesha Gamage-(ayega981)

Assignment 2,coded Chandika Nishshanka(chache114) and documentation part done by together.

Assignment 1. Handwritten digit recognition with Knearest neighbors.

1.2

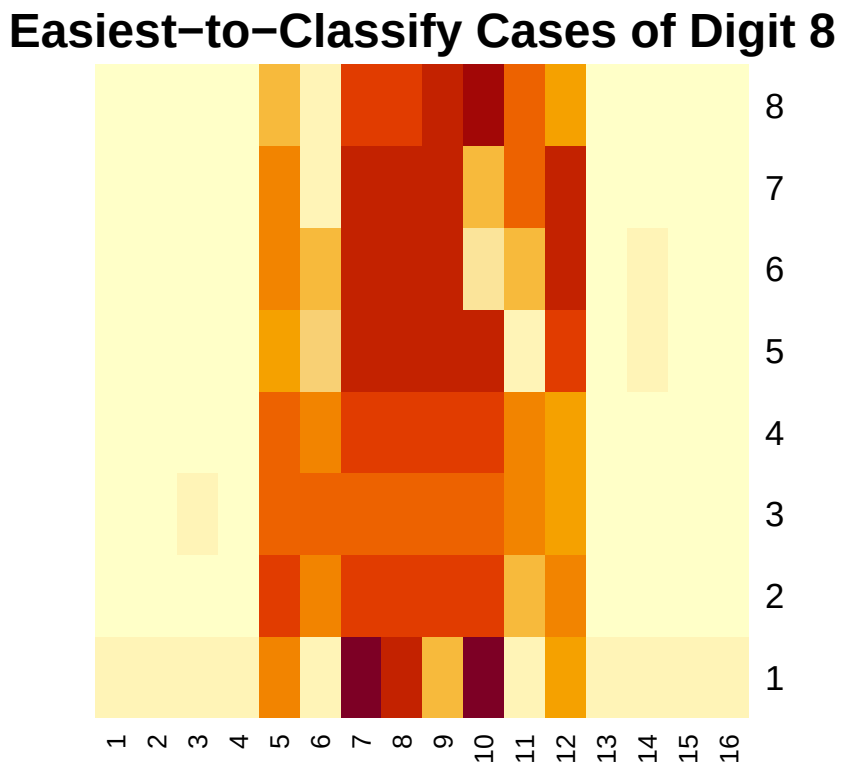
```
##    train_preds
##       0   1   2   3   4   5   6   7   8   9
##   0 202   0   0   0   0   0   0   0   0   0
##   1   0 179  11   0   0   0   0   1   1   3
##   2   0   1 190   0   0   0   0   1   0   0
##   3   0   0   0 185   0   1   0   1   0   1
##   4   1   3   0   0 159   0   0   7   1   4
##   5   0   0   0   1   0 171   0   1   0   8
##   6   0   2   0   0   0   0 190   0   0   0
##   7   0   3   0   0   0   0   0 178   1   0
##   8   0  10   0   2   0   0   2   0 188   2
##   9   1   3   0   5   2   0   0   3   3 183
```

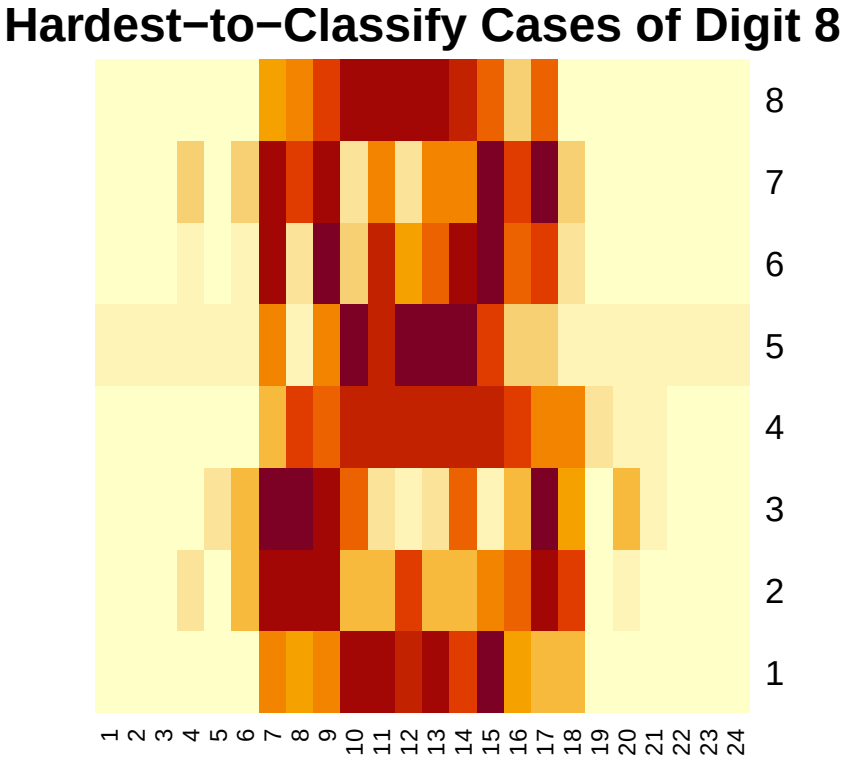
```
##    test_preds
##       0   1   2   3   4   5   6   7   8   9
##   0  77   0   0   0   1   0   0   0   0   0
##   1   0  81   2   0   0   0   0   0   0   3
##   2   0   0  98   0   0   0   0   0   3   0
##   3   0   0   0 107   0   2   0   0   1   1
##   4   0   0   0   0  94   0   2   6   2   5
##   5   0   1   1   0   0  93   2   1   0   5
##   6   0   0   0   0   0   0  90   0   0   0
##   7   0   0   0   1   0   0   0 111   0   0
##   8   0   7   0   1   0   0   0   0  70   0
##   9   0   1   1   1   0   0   0   1   0  85
```

```
## Training Misclassification Error: 0.04500262
```

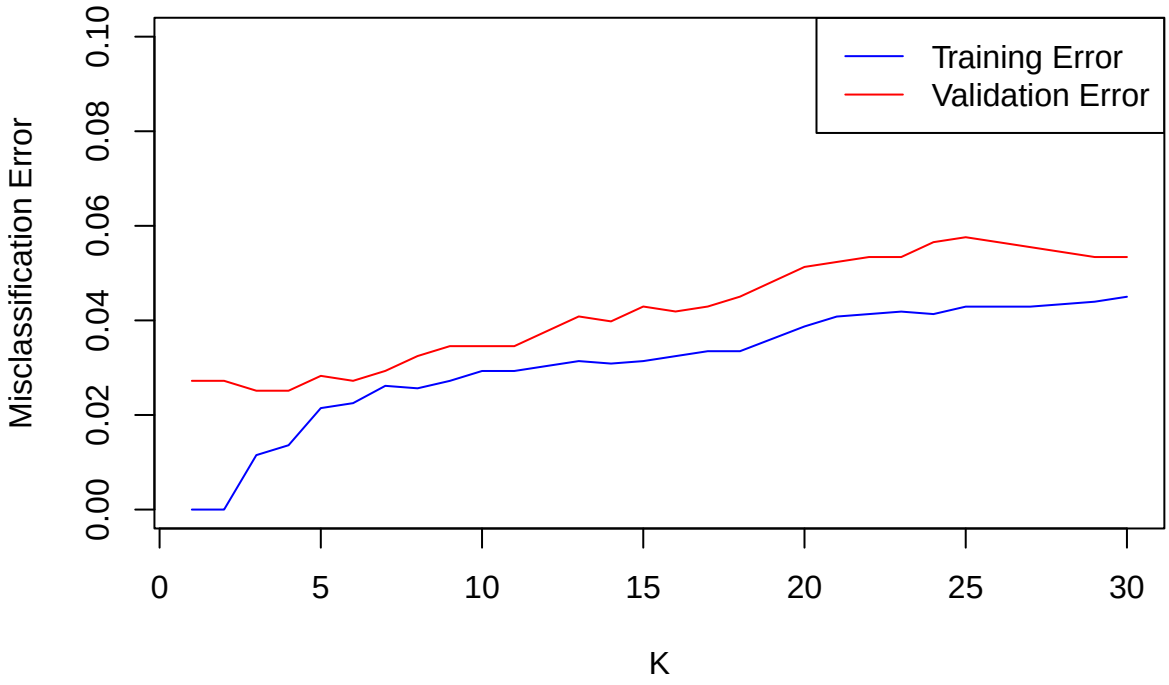
```
## Test Misclassification Error: 0.05329154
```

1.3


Easiest−to−Classify Cases of Digit 8
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16


Hardest−to−Classify Cases of Digit 8
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1.4




```
## Optimal K: 3
```

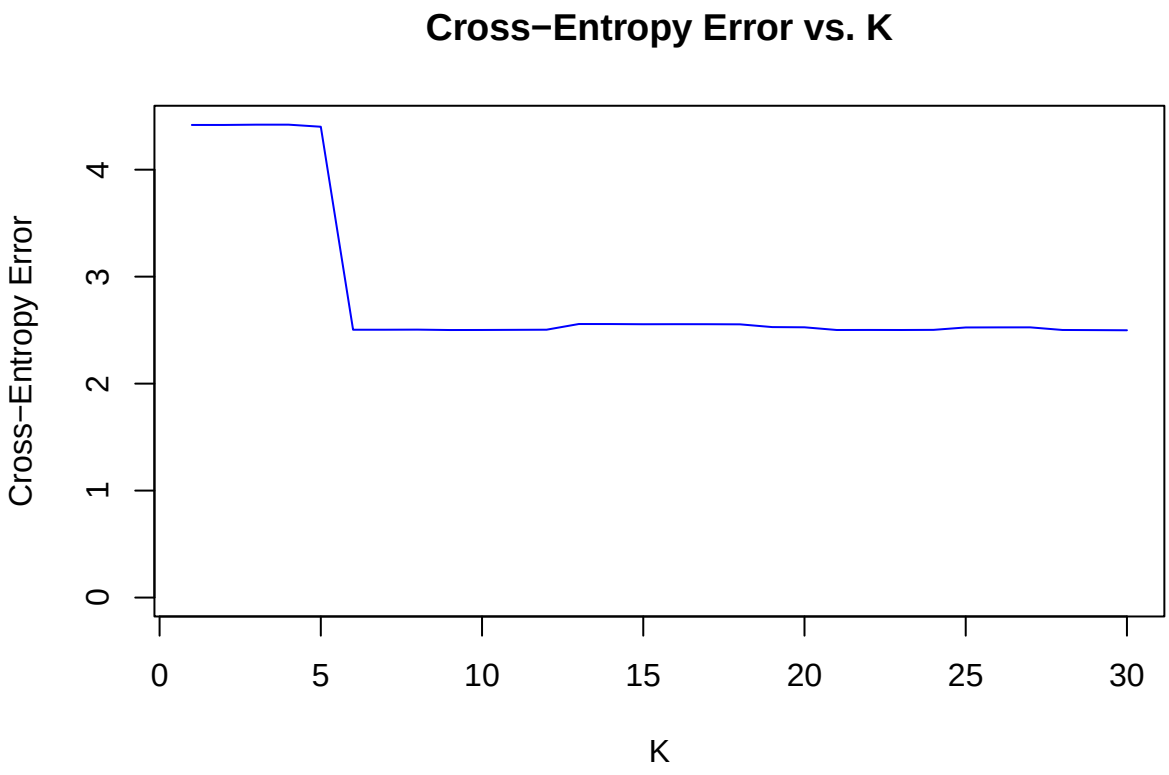
```
## Training Misclassification Error (Optimal K): 0.0115123
```

```
## Validation Misclassification Error (Optimal K): 0.02513089
```

```
## Test Misclassification Error (Optimal K): 0.02403344
```

In k-nearest neighbors (KNN), the model complexity is inversely related to the value of K. As K increases, the model becomes less complex, and as K decreases, the model becomes more complex. The optimal K is selected based on the minimum validation error.According to this result optimal K = 3. Validation Misclassification Error (Optimal K) is 0.02513089.

1.5




```
## Optimal K (Cross-Entropy): 30
```

The optimal K is selected based on the minimum cross-entropy error.According to this Optimal K (Cross-Entropy)is 30 .

Cross-entropy is preferred over misclassification error for multinomial classification due to its sensitivity to probabilities, provision of gradient information for optimization, and its ability to calibrate the model's output probabilities, offering more information about the certainty of predictions.

Assignment 2. Linear regression and ridge regression.

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

2.1

```
##
## Call:
## lm(formula = motor_UPDRS ~ -1 + Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 +
##     Jitter.DDP + Shimmer + Shimmer.dB. + Shimmer.APQ3 + Shimmer.APQ5 +
##     Shimmer.APQ11 + Shimmer.DDA + NHR + HNR + RPDE + DFA + PPE,
##     data = train_dataS)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9131 -0.7310 -0.1163  0.7355  2.1948
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## Jitter.Abs.    -0.153692   0.038770  -3.964 7.51e-05 ***
## Jitter.RAP     -5.394958  18.835401  -0.286 0.774568
## Jitter.PPQ5    -0.017627   0.075022  -0.235 0.814251
## Jitter.DDP      5.482775  18.838109   0.291 0.771033
## Shimmer         0.594098   0.205993   2.884 0.003950 **
## Shimmer.dB.    -0.151655   0.138310  -1.096 0.272943
## Shimmer.APQ3   30.832745  77.159061   0.400 0.689476
## Shimmer.APQ5   -0.393637   0.113693  -3.462 0.000542 ***
## Shimmer.APQ11   0.310079   0.061133   5.072 4.14e-07 ***
## Shimmer.DDA   -31.169459  77.158841  -0.404 0.686263
## NHR            -0.181120   0.045443  -3.986 6.87e-05 ***
## HNR            -0.233881   0.036207  -6.460 1.19e-10 ***
## RPDE            0.006219   0.022600   0.275 0.783206
## DFA            -0.280420   0.020137 -13.925  < 2e-16 ***
## PPE             0.236656   0.031857   7.429 1.37e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9395 on 3510 degrees of freedom
## Multiple R-squared:  0.1208, Adjusted R-squared:  0.1171
## F-statistic: 32.16 on 15 and 3510 DF,  p-value: < 2.2e-16
```

```
## Training MSE : 0.8789342
```

```
## Test MSE : 0.9367006
```

2.2

Training MSE : 0.8789342 Test MSE : 0.9367006

Significantly contributed variables found by examine the summery of lm model. In the summery Significant codes given which variable significantly contribute to model. As well as we can get idea about significantly contributed variables by examine the coefficients and their associated p-values in the summary of model. The p-value indicates the statistical significance of each variable and low p-value (less than 0.05) are considered statistically contributed.

In this example following variables are significantly contributed to prediction, Jitter.Abs. ,Shimmer,Shimmer.APQ5,Shimmer.APQ11 ,NHR ,HNR ,DFA,PPE

2.3

```
loglikelihood  = function(teta,sigma,data){
        formula = motor_UPDRS ~-1+ Jitter.Abs.+Jitter.RAP+Jitter.PPQ5+Jitter.DDP+Shimmer+Shimmer.dB.+Sh
        x_matrix = model.matrix(formula,data)
        #x_matrix = as.matrix(data[,7:21])
        t<-(as.matrix(teta))
        return ((-(n/2)*log(2*pi*sigma^2))-((1/2*sigma^2)*sum(((x_matrix %*% t) -data$motor_UPDRS)^2)))
}

Ridge <- function(par, data, lambda) {
       theta = par[1:length(par)-1]
       sigma = par[length(par)]
      #get negative log-likelihood from likelyhood function
       neg_likelihood <- -loglikelihood(theta, sigma, data)
       ridge_penalty <- lambda * sum(theta^2)
       total_neg_likelihood <- neg_likelihood + ridge_penalty
       return(total_neg_likelihood)
}

RidgeOpt = function(lambda){
      x_matrix = as.matrix(train_dataS[,7:21])
      init = c(rep(0, ncol(x_matrix)),0.1)
      result = optim(par = init,fn = Ridge, data = train_dataS, lambda = lambda,method = "BFGS")
      return(result)
}

 find_MSE = function(parameters ,datafm){
       parama = as.matrix(parameters$par[1:length(parameters$par)-1])
       predct_train = as.matrix(datafm[,7:21])
       pred = predct_train%*% parama
       return(mean((datafm$motor_UPDRS - pred)^2))
 }


 DF = function(lambda) {
      opt_result = RidgeOpt(lambda)
      theta =  as.matrix(opt_result$par[1:length(opt_result$par)-1])
      X = as.matrix(train_dataS[,7:21])
      H <- X %*% solve(t(X) %*% X + lambda * diag(length(theta))) %*% t(X)
      df <- sum(diag(H))
      return(df)
 }
```

Using optimal θ parameters from RidgeOptim function and $\lambda = 1$, λ =100 and $\lambda = 1000$ used to predict the motor_UPDRS values for training and test data. And compute the training and test MSE values. The optimal λ is the one that gives the minimum MSE on the test data set. According to this example 100 is the optimal value for λ.

```
DF = function(lambda) {
  opt_result = RidgeOpt(lambda)

  theta =  as.matrix(opt_result$par[1:length(opt_result$par)-1])
```

```
  X = as.matrix(train_dataS[,7:21])
  H <- X %*% solve(t(X) %*% X + lambda * diag(length(theta))) %*% t(X)
  df <- sum(diag(H))

  return(df)
}
```

2.4

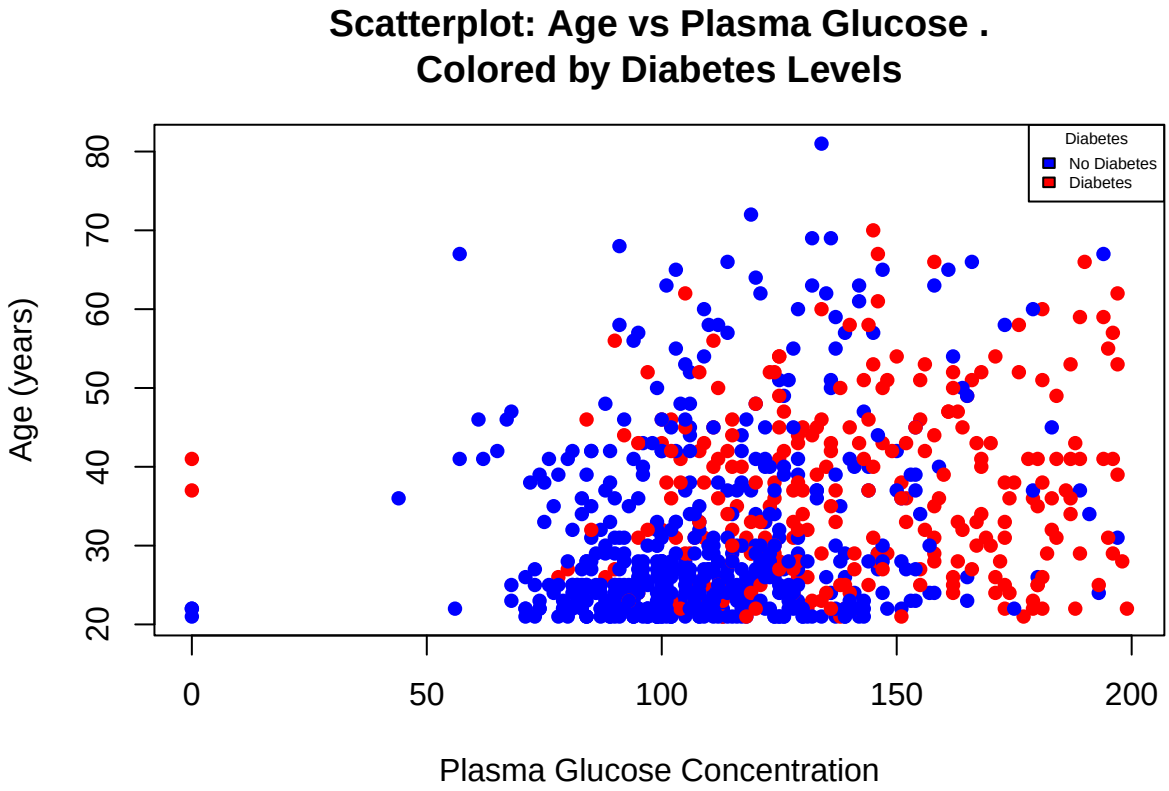
```
## Following table shows the computed values,
```

Lambda	Training_MSE	Test_MSE	DF
1	0.9997163	1.015544	12.865297
100	0.9958352	1.013826	9.086467
1000	0.9997246	1.015549	5.179486

Higher degrees of freedom allow the model to fit the training data more closely, potentially leading to overfitting. Therefore, we want to strike a balance between degrees of freedom and test MSE. The above table shows how the degree of freedom varies with λ. By comparison, we can conclude when λ increases, the value of DF decreases. When considering MSE and df,we can get 100 for optimal λ.

Assignment 3. Logistic regression and basis function expansion

3.1



* Logistic regression is used when the dependent variable is binary(0/1, True/False, Yes/No) in nature

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class.It is used for predicting the categorical dependent variable using a given set of independent variables.According to the question independent variable are Plasma glucose concentration a 2 hours in an oral glucose tolerance test & Age (years) and categorical dependent variable is Diabetes (0=no or 1=yes).Logistic regression predicts the output of a categorical dependent variable(Diabetes). Therefore the outcome must be a categorical or discrete value(In this if the diabetes having or not having like Yes or No).According to evaluation of the data & we can desire Diabetes is easy to classify by a standard logistic regression model that uses these two variables as features.

3.2

The probabilistic equation for the logistic regression model with two features,x_1 (Plasma Glucose Concentration) and x_2 (Age), and the target variable y (Diabetes), can be expressed as follows:

Probabilistic equation:

$$P(y = 1|x_1, x_2) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}}$$

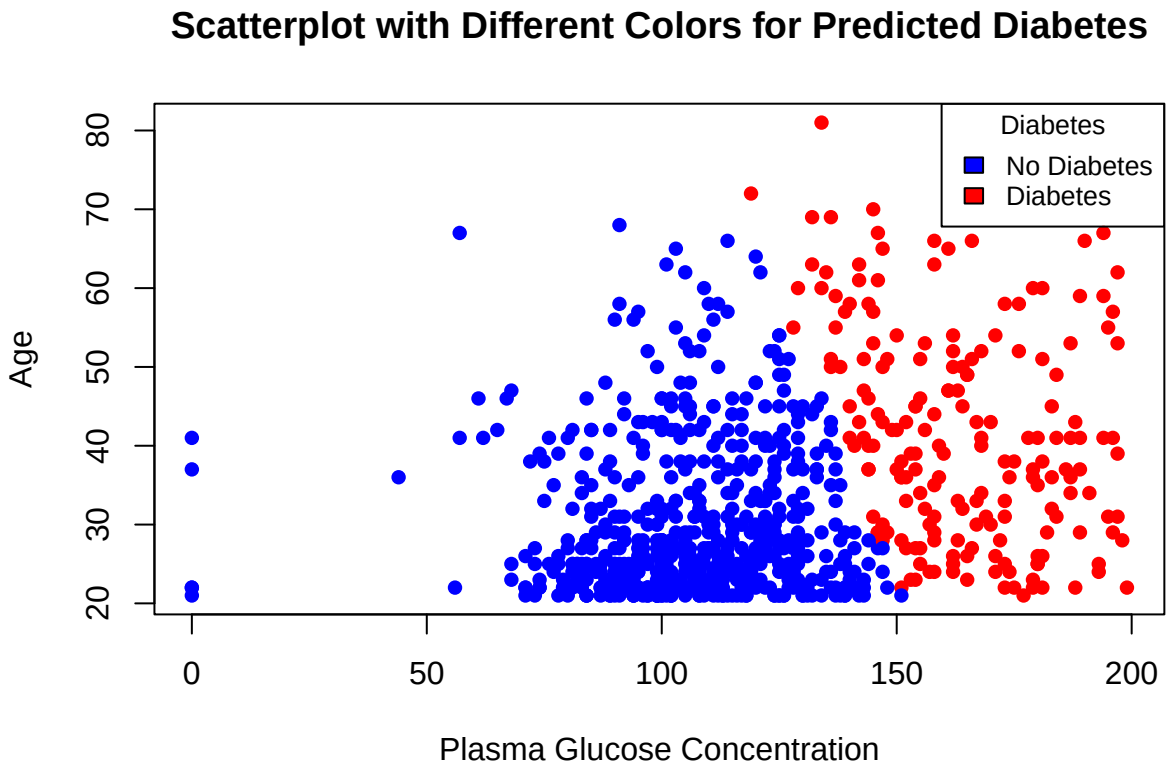
- $P(y = 1|x_1, x_2)$ is the probability that the target variable y is 1 (indicating diabetes) given the values of the predictor variables x_1 and x_2.

- e is the base of the natural logarithm.
- β_0, β_1, and β_2 are the coefficients corresponding to the predictor variables x_1 and x_2 respectively.

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
## Training Misclassification Error: 0.2659713
```



3.3

The decision boundary for a logistic regression model is determined by the coefficients of the model. Given the logistic regression equation:

$$\log\left(\frac{1-p}{p}\right) = \beta_0 + \beta_1 \times \text{PG concentration} + \beta_2 \times \text{Age}$$

where:

p is the probability of belonging to class 1,
β_0 is the intercept,
β_1 is the coefficient for PG concentration,
β_2 is the coefficient for Age.

According to estimated logistic regression model:

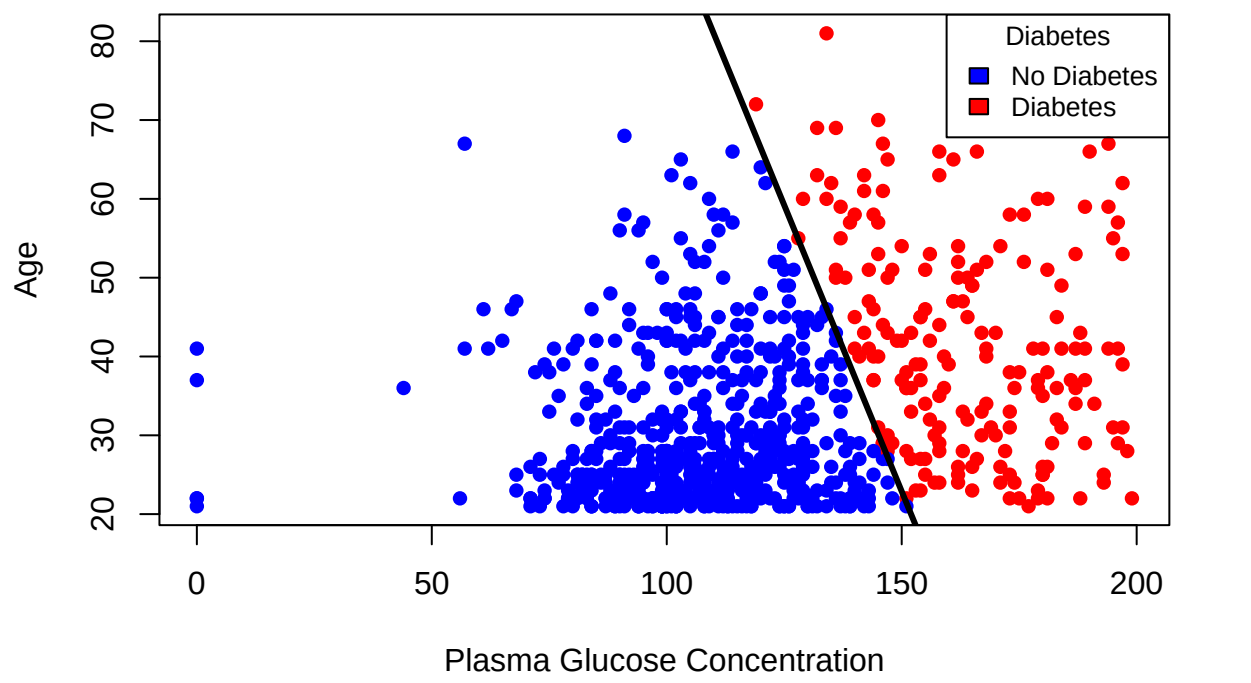
$$\begin{aligned} \text{Intercept} &= -5.89786 \\ \text{PG concentration coefficient} &= 0.03558 \\ \text{Age coefficient} &= 0.02450 \end{aligned}$$

The decision boundary occurs where the log-odds ($log\left(\frac{1-p}{p}\right)$log is zero. Therefore, set the equation to zero and solve for the predictors:

$$\beta_0 + \beta_1 \times \text{PG concentration} + \beta_2 \times \text{Age} = 0$$

This equation represents the decision boundary between the two classes(Age & PG concentration) in this logistic regression model then we can rearrange this equation to solve for one of the predictor(Age) express it explicitly in terms of the other(PG concentration).

$$\text{Age} = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2} \times \text{PG concentration}$$



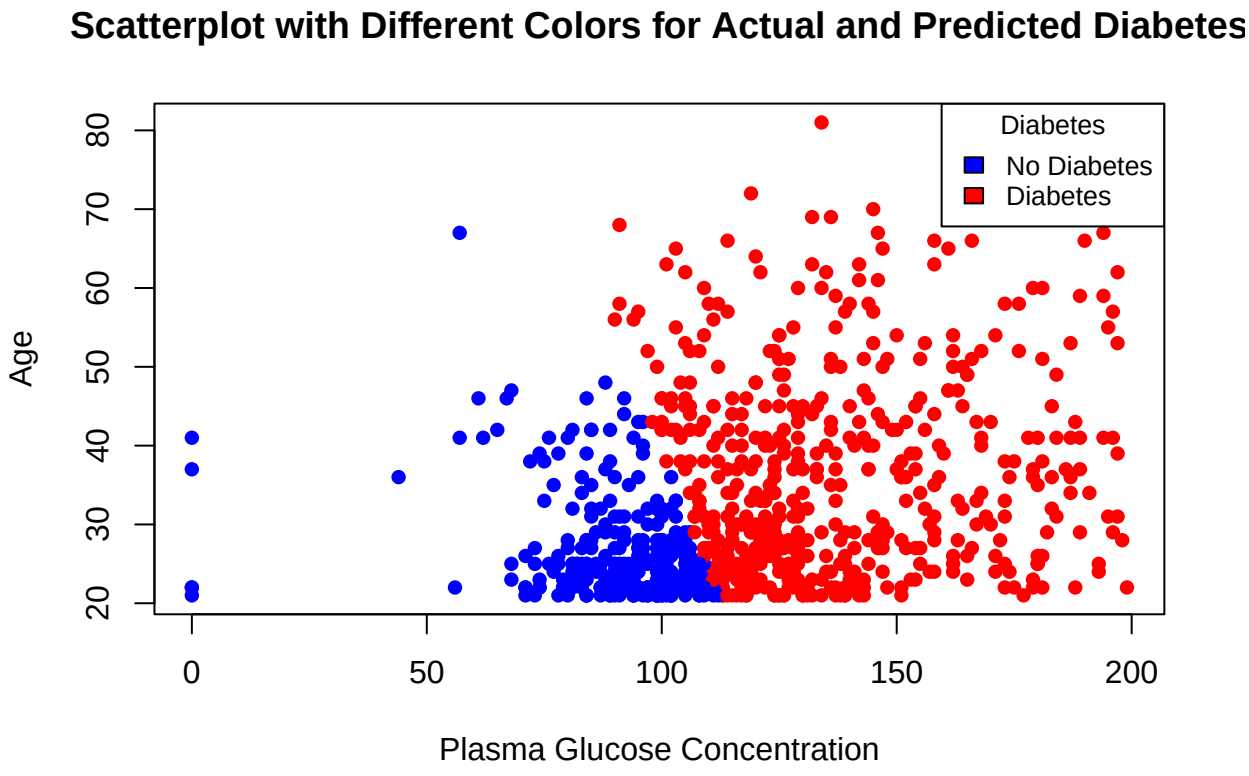
The decision boundary in the scatter plot adeptly aligns with the underlying distribution of data points, effectively delineating the separation between distinct classes. This alignment underscores the model's ability to discern patterns and make accurate classifications based on the provided features.

3.4

Changing the threshold value in logistic regression affects the classification of instances based on their predicted probabilities. Logistic regression produces probability scores for each observation, and a threshold is applied to these scores to determine the predicted class.

When threshold r = 0.2

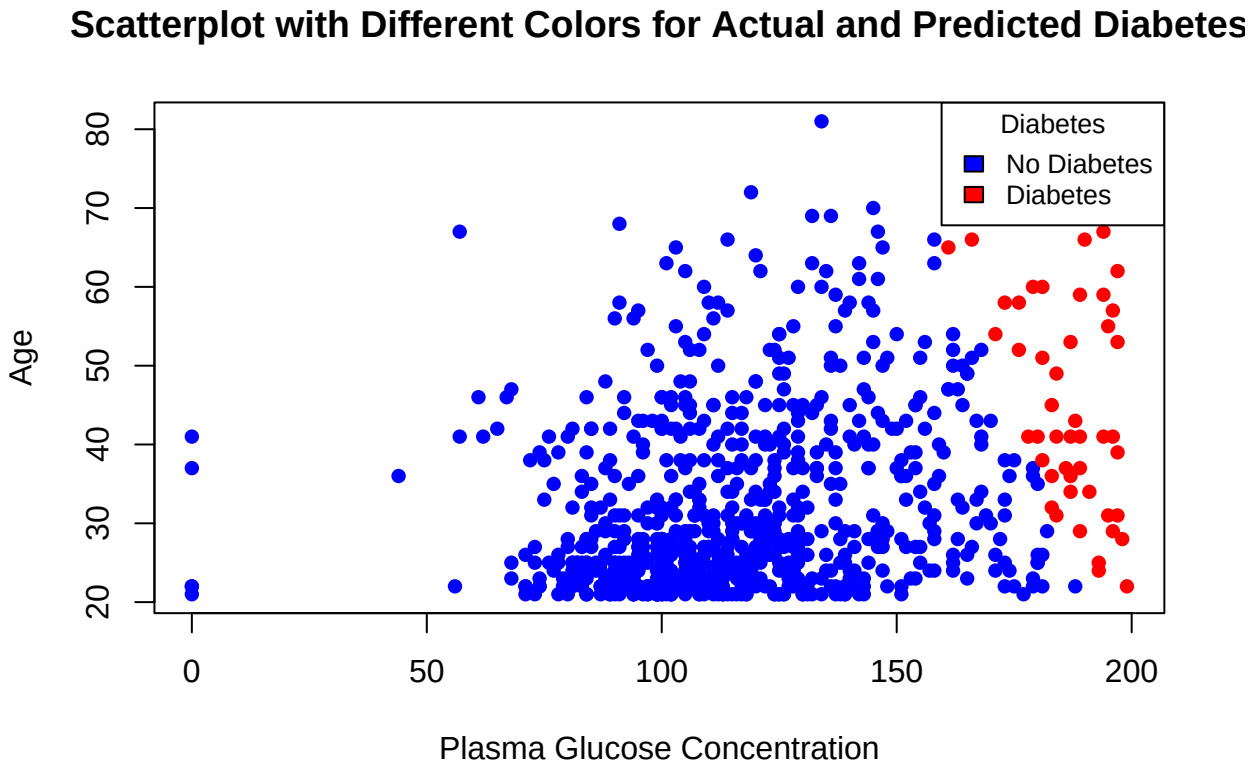
```
## Training Misclassification Error: 0.3741851
```



When we study the $r = 0.2$ scatter plot more cases are classified as diabetes class because a lower threshold ($r = 0.2$) allows less predictable cases to be classified as diabetes as well as the discrete predicted number of non-diabetics.The misclassification error is high compared to the misclassification error at the threshold value of $r = 0.5$.That means more deviations occur with actual and predicted diabetes.

When threshold r = 0.8

```
## Training Misclassification Error: 0.3142112
```



When we study the $r = 0.8$ scatter plot Fewer instances are classified as the diabetes class because a higher threshold requires higher predicted probabilities for diabetes classification as well as the discrete predicted number of diabetics.The misclassification error is high compared to the misclassification error at the threshold value of $r = 0.5$.That means more deviations occur with actual and predicted diabetes.

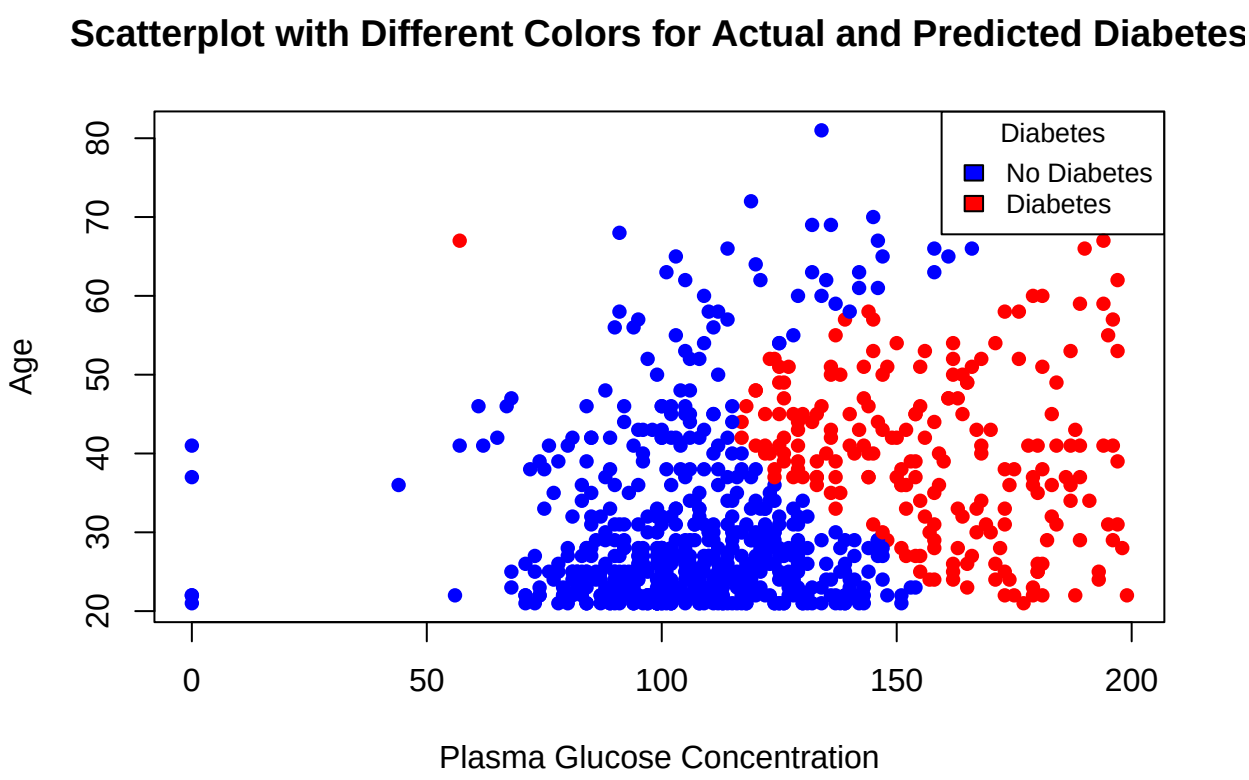
3.5

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Training Misclassification Error: 0.2464146
```



Comparing the misclassification rate of this model with the previous logistic regression model this module has a low misclassification rate. A lower misclassification rate generally indicates a better-performing model.

After adding new features in the model $(z1, z2, z3, z4, z5)$ the decision boundary behaves more like a curve than a straight line in the initial model.

Appendix: All r code for this report

```
knitr::opts_chunk$set(echo = TRUE)
##################code for Assignment 1 ######################
library(kknn)
data <- read.csv("optdigits.csv", header=FALSE)

colnames(data)[65] <- "digit"

n <- dim(data)[1]
set.seed(12345)
id <- sample(1:n, floor(n * 0.5))
train_data <- data[id,]

id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n * 0.25))
valid_data <- data[id2,]
```

```r
id3 <- setdiff(id1, id2)
test_data <- data[id3,]


# Train the model
k <- 30
model_train <- kknn(as.factor(digit) ~ ., train_data, train_data, k = k, kernel = "rectangular")

# Make predictions on training and test data
train_preds <- model_train$fitted.values
model_test <- kknn(as.factor(digit) ~ ., train_data, test_data, k = k, kernel = "rectangular")
test_preds <- model_test$fitted.values

# Calculate confusion matrices
train_confusion_matrix <- table(train_data$digit, train_preds)
test_confusion_matrix <- table(test_data$digit, test_preds)

print(train_confusion_matrix)
print(test_confusion_matrix)

# Calculate misclassification errors
train_error <- 1 - sum(diag(train_confusion_matrix)) / sum(train_confusion_matrix)
test_error <- 1 - sum(diag(test_confusion_matrix)) / sum(test_confusion_matrix)

# Print results
cat("Training Misclassification Error:", train_error, "\n")
cat("Test Misclassification Error:", test_error, "\n")


actual_digit_8_indices <- which(train_data$digit == 8)

# Get class probabilities for digit "8" in the training set
probs <- predict(model_train, newdata = train_data, type = "prob")[,"8"][actual_digit_8_indices]


easiest_indices <- head(order(probs), 2)
hardest_indices <- tail(order(probs), 3)

easiest_cases <- train_data[actual_digit_8_indices[easiest_indices], ]
hardest_cases <- train_data[actual_digit_8_indices[hardest_indices], ]

easiest_features <- as.matrix(easiest_cases[, 1:64])
hardest_features <- as.matrix(hardest_cases[, 1:64])

easiest_matrices <- matrix(as.numeric(unlist(easiest_features)), nrow = 8, byrow = TRUE)
hardest_matrices <- matrix(as.numeric(unlist(hardest_features)), nrow = 8, byrow = TRUE)

heatmap(easiest_matrices, Colv = NA, Rowv = NA, main = "Easiest-to-Classify Cases of Digit 8")
heatmap(hardest_matrices, Colv = NA, Rowv = NA, main = "Hardest-to-Classify Cases of Digit 8")

# Function to calculate misclassification error
calculate_error <- function(actual, predicted) {
```

```r
  return (1 - sum(diag(table(actual, predicted))) / sum(table(actual)))
}

# Function to fit kknn model and calculate error for different k values
fit_kknn_and_calculate_error <- function(train_data, valid_data, k) {
  train_errors <- numeric(length = length(k))
  valid_errors <- numeric(length = length(k))

  for (i in seq_along(k)) {
    model <- kknn(as.factor(digit) ~ ., train_data, train_data, k = k[i], kernel = "rectangular")
    train_preds <- model$fitted.values

    model_valid <- kknn(as.factor(digit) ~ ., train_data, valid_data, k = k[i], kernel = "rectangular")
    valid_preds <- model_valid$fitted.values

    train_errors[i] <- calculate_error(train_data$digit, train_preds)
    valid_errors[i] <- calculate_error(valid_data$digit, valid_preds)

  }

  return(list(train_errors = train_errors, valid_errors = valid_errors))

}


k_values <- 1:30

# Fit Kknn models and calculate errors
errors <- fit_kknn_and_calculate_error(train_data, valid_data, k_values)


plot(k_values, errors$train_errors, type = "l", col = "blue", xlab = "K", ylab = "Misclassification Erro
lines(k_values, errors$valid_errors, type = "l", col = "red")
legend("topright", legend = c("Training Error", "Validation Error"), col = c("blue", "red"), lty = 1)

# Find the optimal K based on the validation error
optimal_k <- k_values[which.min(errors$valid_errors)]
cat("Optimal K:", optimal_k, "\n")

# Estimate test error for the model with optimal K
model_test_optimal_k <- kknn(as.factor(digit) ~ ., train_data, test_data, k = optimal_k, kernel = "recta
test_preds_optimal_k <- model_test_optimal_k$fitted.values
test_error_optimal_k <- calculate_error(test_data$digit, test_preds_optimal_k)


cat("Training Misclassification Error (Optimal K):", errors$train_errors[which(k_values == optimal_k)],
cat("Validation Misclassification Error (Optimal K):", errors$valid_errors[which(k_values == optimal_k)]
cat("Test Misclassification Error (Optimal K):", test_error_optimal_k, "\n")


# Function to calculate cross-entropy error
calculate_cross_entropy <- function(actual, predicted_probs) {
  epsilon <- 1e-15
```

```r
  predicted_probs <- as.numeric(as.character(predicted_probs))
  predicted_probs <- pmax(predicted_probs, epsilon)

  # Calculate cross-entropy
  log_probs <- -log(predicted_probs[cbind(1:length(actual), actual)])
  return(mean(log_probs))
}


# Function to fit knn model and calculate cross-entropy error for different k values
fit_kknn_and_calculate_cross_entropy <- function(train_data, valid_data, k) {
  valid_errors <- numeric(length = length(k))

  for (i in seq_along(k)) {
    model_valid <- kknn(as.factor(digit) ~ ., train_data, valid_data, k = k[i], kernel = "rectangular")
    valid_preds <- model_valid$fitted.values

    valid_errors[i] <- calculate_cross_entropy(valid_data$digit, valid_preds)
  }
  return(valid_errors)
}
k_values <- 1:30

cross_entropy_errors <- fit_kknn_and_calculate_cross_entropy(train_data, valid_data, k_values)

plot(k_values, cross_entropy_errors, type = "l", col = "blue", xlab = "K", ylab = "Cross-Entropy Error"
title(main = "Cross-Entropy Error vs. K")

# Find the optimal K based on the minimum cross-entropy error
optimal_k <- k_values[which.min(cross_entropy_errors)]
cat("Optimal K (Cross-Entropy):", optimal_k, "\n")

##################End code for Assignment 1 ######################

##################code for Assignment 2 ######################
      library(caret)
      setwd("D:/MSC/SEMESTER1/732A99 Maching lerning/Exercixe")
      all_data = read.csv("parkinsons.csv")
        #partition data to train and test (60/40)
        set.seed(12345)
        n = dim(all_data)[1]
        id = sample(1:n, floor(n*0.6))
        train_data = all_data[id,]
        test_data = all_data[-id,]

        #data scalling
        datapre = preProcess(train_data, method=c("center", "scale"))
        train_dataS = predict(datapre, train_data)
        test_dataS = predict(datapre,test_data)

#linear regression model from the training data
        lm_model = lm(motor_UPDRS ~ -1+Jitter.Abs.+Jitter.RAP+Jitter.PPQ5+Jitter.DDP+Shimmer+Shimmer.dB
        summary(lm_model)
```

```
        para_vector = as.vector(coef(lm_model))

 # predict test data and training data, fit training data to model
        train_predictions <- predict(lm_model, newdata = train_dataS)
        test_predictions <- predict(lm_model, newdata = test_dataS)


#find MSE training data and test data
        tran_MSE <- mean((train_dataS$motor_UPDRS - train_predictions)^2)
        test_MSE <- mean((test_dataS$motor_UPDRS - test_predictions)^2)
        cat("Training MSE :",tran_MSE,"\n")
        cat("Test MSE :",test_MSE)
loglikelihood  = function(teta,sigma,data){
        formula = motor_UPDRS ~-1+ Jitter.Abs.+Jitter.RAP+Jitter.PPQ5+Jitter.DDP+Shimmer+Shimmer.dB.+Sh
        x_matrix = model.matrix(formula,data)
        #x_matrix = as.matrix(data[,7:21])
        t<-(as.matrix(teta))
        return ((-(n/2)*log(2*pi*sigma^2))-((1/2*sigma^2)*sum(((x_matrix %*% t) -data$motor_UPDRS)^2)))
}

Ridge <- function(par, data, lambda) {
       theta = par[1:length(par)-1]
       sigma = par[length(par)]
      #get negative log-likelihood from likelyhood function
       neg_likelihood <- -loglikelihood(theta, sigma, data)
       ridge_penalty <- lambda * sum(theta^2)
       total_neg_likelihood <- neg_likelihood + ridge_penalty
       return(total_neg_likelihood)
}

RidgeOpt = function(lambda){
      x_matrix = as.matrix(train_dataS[,7:21])
      init = c(rep(0, ncol(x_matrix)),0.1)
      result = optim(par = init,fn = Ridge, data = train_dataS, lambda = lambda,method = "BFGS")
      return(result)
}

 find_MSE = function(parameters ,datafm){
       parama = as.matrix(parameters$par[1:length(parameters$par)-1])
       predct_train = as.matrix(datafm[,7:21])
       pred = predct_train%*% parama
       return(mean((datafm$motor_UPDRS - pred)^2))
 }


 DF = function(lambda) {
      opt_result = RidgeOpt(lambda)
      theta =  as.matrix(opt_result$par[1:length(opt_result$par)-1])
      X = as.matrix(train_dataS[,7:21])
      H <- X %*% solve(t(X) %*% X + lambda * diag(length(theta))) %*% t(X)
      df <- sum(diag(H))
      return(df)
 }
```

```r
DF = function(lambda) {
  opt_result = RidgeOpt(lambda)

  theta =  as.matrix(opt_result$par[1:length(opt_result$par)-1])
  X = as.matrix(train_dataS[,7:21])
  H <- X %*% solve(t(X) %*% X + lambda * diag(length(theta))) %*% t(X)
  df <- sum(diag(H))

  return(df)
}


 opt_result1 = RidgeOpt(1)
 opt_result100 = RidgeOpt(100)
 opt_result1000 = RidgeOpt(1000)

 train_MSE_1 = find_MSE(opt_result1,train_dataS)
 train_MSE_100 =find_MSE(opt_result100,train_dataS)
 train_MSE_1000 =find_MSE(opt_result1000,train_dataS)

 test_MSE_1 = find_MSE(opt_result1,test_dataS)
 test_MSE_100 =find_MSE(opt_result100,test_dataS)
 test_MSE_1000 =find_MSE(opt_result1000,test_dataS)

 cat("Following table shows the computed values,\n")

df_1=DF(1)
df_100=DF(100)
df_1000=DF(1000)
data <- data.frame(
  Lambda = c(1, 100, 1000),
  Training_MSE = c(train_MSE_1, train_MSE_100, train_MSE_1000),
  Test_MSE = c(test_MSE_1, test_MSE_100, test_MSE_1000),
  DF = c(df_1, df_100, df_1000)
)
knitr::kable(data)
 ##################End code for Assignment 2 ######################

################## code for Assignment 3 ######################
pi_diabetes <-read.csv("pima-indians-diabetes.csv")

colnames(pi_diabetes)=c("No.time pregnant",
                        "PGconcentration",
                        "Diastolic BP",
                        "TSthickness",
                        "serum insulin",
                        "BM index",
                        "DP function",
                        "Age","Diabetes")

diabetes_colors <-ifelse(pi_diabetes$Diabetes == 1, "red", "blue")
```

```r
plot(pi_diabetes$PGconcentration,
     pi_diabetes$Age,
     col=diabetes_colors,
     pch=16,
     xlab = "Plasma Glucose Concentration",
     ylab = "Age (years)",
     main = "Scatterplot: Age vs Plasma Glucose .\nColored by Diabetes Levels"
     )

legend("topright", legend = c("No Diabetes", "Diabetes"),fill = c("blue", "red"), title = "Diabetes",ce


library(glmnet)

x <- c("PGconcentration", "Age")
y <- "Diabetes"

logistic_model <- glm(as.formula(paste(y, "~", paste(x, collapse = " + "))),
                      data = pi_diabetes,
                      family = "binomial")

predicted_probabilities <- predict(logistic_model, newdata = pi_diabetes, type = "response")

predicted_labels <- ifelse(predicted_probabilities >= 0.5, 1, 0)

pi_diabetes$Predicted_Diabetes <- predicted_labels

misclassification_error <- mean(pi_diabetes$Diabetes != pi_diabetes$Predicted_Diabetes)

cat("Training Misclassification Error:", misclassification_error, "\n")

diabetes_colors <-ifelse(pi_diabetes$Predicted_Diabetes == 1, "red", "blue")

plot(pi_diabetes$PGconcentration,pi_diabetes$Age, col =diabetes_colors,
     pch = 16, main = "Scatterplot with Different Colors for Predicted Diabetes",
     xlab = "Plasma Glucose Concentration", ylab = "Age")
legend("topright", legend = c("No Diabetes", "Diabetes"), fill = c("blue", "red"), title = "Diabetes",c

intercept <- -5.89786
beta_PG <- 0.03558
beta_Age <- 0.02450

decision_boundary <- function(PGconcentration) {
  return (- (intercept + beta_PG * PGconcentration) / beta_Age)
}

PG_values <- seq(min(pi_diabetes$PGconcentration), max(pi_diabetes$PGconcentration), length.out = 100)

decision_boundary_values <- decision_boundary(PG_values)

diabetes_colors <-ifelse(pi_diabetes$Predicted_Diabetes == 1, "red", "blue")

plot(pi_diabetes$PGconcentration,pi_diabetes$Age, col =diabetes_colors,
```

```
     pch = 16, main = "Different Colors for Actual and Predicted Diabetes with Desicion Boundary",
     xlab = "Plasma Glucose Concentration", ylab = "Age")
legend("topright", legend = c("No Diabetes", "Diabetes"), fill = c("blue", "red"), title = "Diabetes",c

lines(PG_values, decision_boundary_values, col = "black",lwd = 3)

library(glmnet)

x <- c("PGconcentration", "Age")
y <- "Diabetes"

logistic_model <- glm(as.formula(paste(y, "~", paste(x, collapse = " + "))),
                      data = pi_diabetes,
                      family = "binomial")

predicted_probabilities <- predict(logistic_model, newdata = pi_diabetes, type = "response")

predicted_labels <- ifelse(predicted_probabilities >= 0.2, 1, 0)

pi_diabetes$Predicted_Diabetes <- predicted_labels

misclassification_error <- mean(pi_diabetes$Diabetes != pi_diabetes$Predicted_Diabetes)

cat("Training Misclassification Error:", misclassification_error, "\n")

diabetes_colors <-ifelse(pi_diabetes$Predicted_Diabetes == 1, "red", "blue")

plot(pi_diabetes$PGconcentration,pi_diabetes$Age, col =diabetes_colors,
     pch = 16, main = "Scatterplot with Different Colors for Actual and Predicted Diabetes",
     xlab = "Plasma Glucose Concentration", ylab = "Age")
legend("topright", legend = c("No Diabetes", "Diabetes"), fill = c("blue", "red"), title = "Diabetes",c

library(glmnet)

x <- c("PGconcentration", "Age")
y <- "Diabetes"

logistic_model <- glm(as.formula(paste(y, "~", paste(x, collapse = " + "))),
                      data = pi_diabetes,
                      family = "binomial")

predicted_probabilities <- predict(logistic_model, newdata = pi_diabetes, type = "response")

predicted_labels <- ifelse(predicted_probabilities >= 0.8, 1, 0)

pi_diabetes$Predicted_Diabetes <- predicted_labels

misclassification_error <- mean(pi_diabetes$Diabetes != pi_diabetes$Predicted_Diabetes)

cat("Training Misclassification Error:", misclassification_error, "\n")

diabetes_colors <-ifelse(pi_diabetes$Predicted_Diabetes == 1, "red", "blue")
```

```r
plot(pi_diabetes$PGconcentration,pi_diabetes$Age, col =diabetes_colors,
     pch = 16, main = "Scatterplot with Different Colors for Actual and Predicted Diabetes",
     xlab = "Plasma Glucose Concentration", ylab = "Age")
legend("topright", legend = c("No Diabetes", "Diabetes"), fill = c("blue", "red"), title = "Diabetes",c


library(dplyr)
library(glmnet)

pi_diabetes$z1=pi_diabetes$PGconcentration^4
pi_diabetes$z2=pi_diabetes$PGconcentration^3*pi_diabetes$Age^1
pi_diabetes$z3=pi_diabetes$PGconcentration^2*pi_diabetes$Age^2
pi_diabetes$z4=pi_diabetes$PGconcentration^1*pi_diabetes$Age^3
pi_diabetes$z5=pi_diabetes$Age^4


new_logistic_model<-glm(Diabetes~PGconcentration+Age+z1+z2+z3+z4+z5,
                        data = pi_diabetes,
                        family = "binomial")

new_predicted_probabilities <- predict(new_logistic_model, newdata = pi_diabetes, type = "response")

new_predicted_labels <- ifelse(new_predicted_probabilities >= 0.5,1,0)

pi_diabetes$new_Predicted_Diabetes <- new_predicted_labels

misclassification_error <- mean(pi_diabetes$Diabetes != pi_diabetes$new_Predicted_Diabetes)

cat("Training Misclassification Error:", misclassification_error, "\n")

new_diabetes_colors <-ifelse(pi_diabetes$new_Predicted_Diabetes == 1, "red", "blue")

plot(pi_diabetes$PGconcentration,pi_diabetes$Age, col =new_diabetes_colors,
     pch = 16, main = "Scatterplot with Different Colors for Actual and Predicted Diabetes",
     xlab = "Plasma Glucose Concentration", ylab = "Age")

legend("topright", legend = c("No Diabetes", "Diabetes"), fill = c("blue", "red"), title = "Diabetes",c

##################End code for Assignment 3 ######################
```