Lab1

liume102@student.liu.se
hanxi898@student.liu.se
xiali125@student.liu.se

12 Nov 2024

Contents

1 Statement of Contribution

In Assignment 1, Xiaochen Liu was mainly responsible for code writing while Liuxi Mei was responsible for the analyes. Assignment 2 was mainly contributed by Han Xia. In assignment 3, Liuxi Mei was responsible for code writing while Han Xia was responsible for the analysis. Assignment 4 was mainly contributed by Xiaochen Liu and Liuxi Mei. Results from all assignments have been discussed afterwards between Liuxi Mei, Xiaochen Liu and Han Xia and the group report was created based on this discussion.

2 Introduction

This is the first lab in the Machine Learning In this lab, contains the following tasks:1. Handwritten digit recognition with K-nearest neighbors.2. Linear regression and ridge regression.3. Logistic regression and basis function expansion.4. Theory

3 Assignment 1: Handwritten digit recognition with K-nearest neighbors

3.1 Load and check data

```
# Load packages
library('ggplot2') # visualization
library('ggthemes') # visualization
```

```
## Warning:  'ggthemes' R 4.4.2
```

```
library('scales') # visualization
library('dplyr') # data manipulation
library('randomForest') # classification algorithm
```

```
## Warning:  'randomForest' R 4.4.2
```

```
library('caret')
```

```
## Warning:  'caret' R 4.4.2
```

Now that our packages are loaded and we divide it into training, validation and test sets (50%/25%/25%)

```r
# do not use StringAsFact = FALSE
digitals <- read.csv('optdigits.csv',header = FALSE)
# change all the columns to factor
#digitals <- digitals %>% mutate_all(as.factor)
digitals$V65 <- as.factor(digitals$V65)
train_index <- createDataPartition(digitals$V65, p = 0.5, list = F)
train_digitals <- digitals[train_index,]
remainingData <- digitals[-train_index, ]
validationIndex <- createDataPartition(remainingData$V65, p = 0.5, list = FALSE)
valid_digitals <- remainingData[validationIndex, ]
test_digitals <- remainingData[-validationIndex, ]
cat("train length:", nrow(train_digitals),'\n')
```

```
## train length: 1914
```

```r
cat("test length:", nrow(valid_digitals),'\n')
```

```
## test length: 956
```

```r
cat("valid length:", nrow(test_digitals),'\n')
```

```
## valid length: 953
```

3.2 KNN to fit classification model using train data

```r
library(kknn)
formula <- V65~.
# if kenerl = 'rectangular' , so every point in the neighborhood is weighted equally
# both of the parameters of train and test use train_digital data
# if  your predict columns is continuous, kknn will recognized as a regression task
# under this situation, you can not get a probability of the prediction
knn_train_model <- kknn(formula, train_digitals, train_digitals,
                        kernel = 'rectangular',distance = 1,)
train_predictions <- fitted(knn_train_model)
print(length(train_predictions))
```

```
## [1] 1914
```

3.3 Predict on the test data

```r
knn_test_model <- kknn(formula, train = train_digitals, test = test_digitals,
                       k = 30, kernel = 'rectangular')
print(length(knn_test_model$fitted.values))
```

```
## [1] 953
```

3.4 Confusion matrices and Misclassification errors for train data and test data

```
train_confusion <- table(train_digitals$V65, train_predictions)
test_confusion <- table(test_digitals$V65, knn_test_model$fitted.values)
test_error_rate <- 1 - sum(diag(test_confusion)) / sum(test_confusion)
train_error_rate <- 1-  sum(diag(train_confusion)) / sum(train_confusion)
# only observer the top 10 rows
cat("Misclassification errors on train data:", train_error_rate, '\n')
```

```
## Misclassification errors on train data: 0.01776385
```

```
cat("train_confusion:")
```

```
## train_confusion:
```

```
table(train_digitals$V65, train_predictions)
```

```
##    train_predictions
##       0   1   2   3   4   5   6   7   8   9
##   0 188   0   0   0   0   0   0   0   0   0
##   1   0 193   1   0   0   0   0   0   0   1
##   2   0   1 188   0   0   0   0   1   0   0
##   3   0   0   0 190   0   0   0   2   1   2
##   4   0   0   0   0 192   0   2   0   0   0
##   5   0   0   0   1   0 185   0   0   0   2
##   6   1   0   0   0   0   0 188   0   0   0
##   7   0   2   0   1   0   0   0 191   0   0
##   8   0   7   0   2   0   0   0   0 180   1
##   9   0   1   0   1   0   0   0   3   1 185
```

```
cat("Misclassification errors on test data:", test_error_rate, '\n')
```

```
## Misclassification errors on test data: 0.05351522
```

```
cat("test confusion:")
```

```
## test confusion:
```

```
table(test_digitals$V65, knn_test_model$fitted.values)
```

```
##
##      0  1  2  3  4  5  6  7  8  9
##   0 93  0  0  0  1  0  0  0  0  0
##   1  0 92  1  1  0  0  1  1  0  1
##   2  0  0 91  0  0  0  0  2  2  0
##   3  0  0  0 94  0  0  0  2  0  1
##   4  0  1  0  0 90  0  1  3  0  1
##   5  0  0  0  2  0 83  0  2  0  7
##   6  0  1  0  0  0  0 93  0  0  0
##   7  0  0  0  0  0  0  0 96  0  0
##   8  0  6  0  1  0  0  1  0 84  3
##   9  1  3  0  2  0  0  0  2  1 86
```

Comment on the quality of predictions for different digits and on the overall prediction quality: The prediction from test data has a higher misclassifications error rate (0.037) comparing to the training data (0.026). It is a good model with such low MSE even for test data. It is also shown from both data set that the model usually predicts the number '8' as number '1', indicting the difficulty when predict for number '8'.

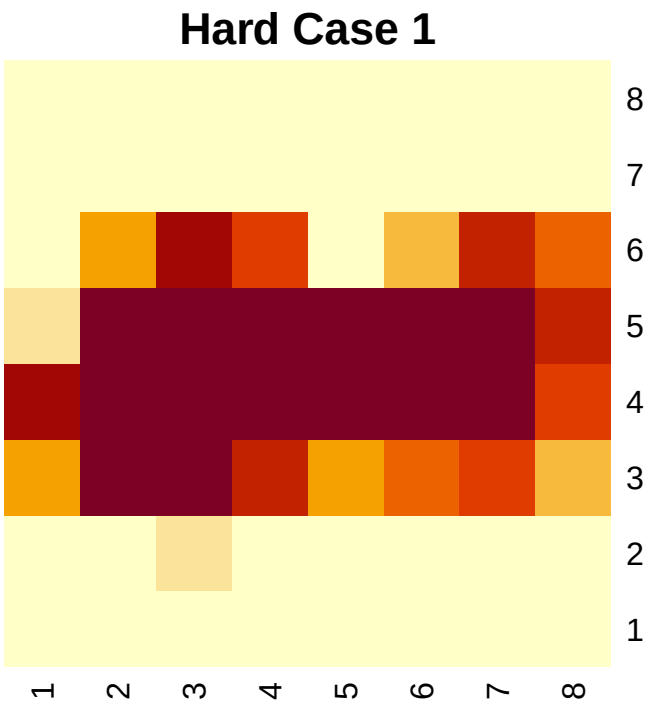
3.5 Filter 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify

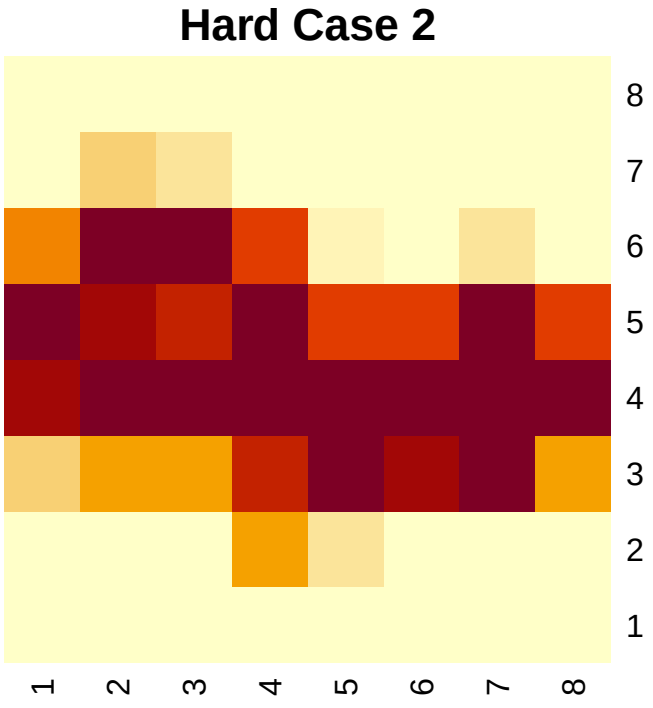
```
# filter the digital '8'
library(dplyr)

train_predict <- data.frame(train_digitals$V65, train_predictions,knn_train_model$prob)
train_predict$max_prob <- apply(train_predict[,3:12], 1, max)

train_predict_8 <- train_predict[train_predict$train_digitals.V65 == 8,]
# do not change the index while sorting
train_predict_8 <- train_predict_8[order(train_predict_8$X8), , drop = FALSE]

# get the 3 cases that were hardest to classify
hardest_cases_for_8 <- train_predict_8 %>% head(3)
easy_cases_for_8 <- train_predict_8 %>% tail(2)
```

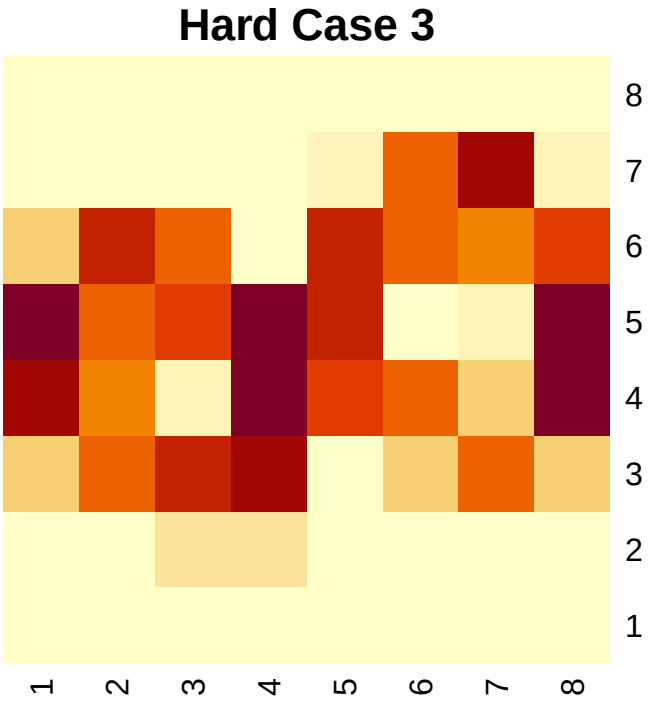
3.6 Analysis the difference of the hardest case and easiest cases

we can see on the heatmap that the hardest cases are more complex than the easiest cases. Dark-colored squares concentrated in the middle of the matrix while the easiest cases are more concentrated on the edges which looking more like the number 8.

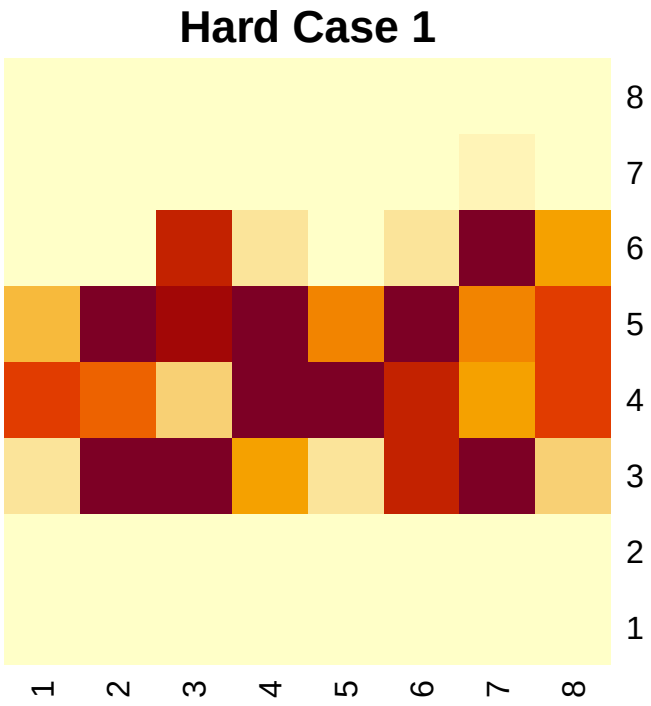
```
hardest_cases_index <- rownames(hardest_cases_for_8)
est_cases_index <- rownames(easy_cases_for_8)
# reindex the row index
row.names(train_digitals) <- NULL
full_hardest_cases <- train_digitals[hardest_cases_index,1:64]
full_est_cases <- train_digitals[est_cases_index,1:64]

hardest_matrixs <- lapply(1:nrow(full_hardest_cases),
  function(i) matrix(as.numeric(full_hardest_cases[i, ,
                      drop = FALSE]),nrow = 8,ncol = 8))
est_matrixs <- lapply(1:nrow(full_est_cases),
  function(i) matrix(as.numeric(full_est_cases[i, ,
                      drop = FALSE]),nrow = 8,ncol = 8))

for (i in 1:length(hardest_matrixs)) {
mat <- hardest_matrixs[[i]]
heatmap(mat, Colv = NA, Rowv = NA, scale = "none", main = paste("Hard Case", i))
}
```

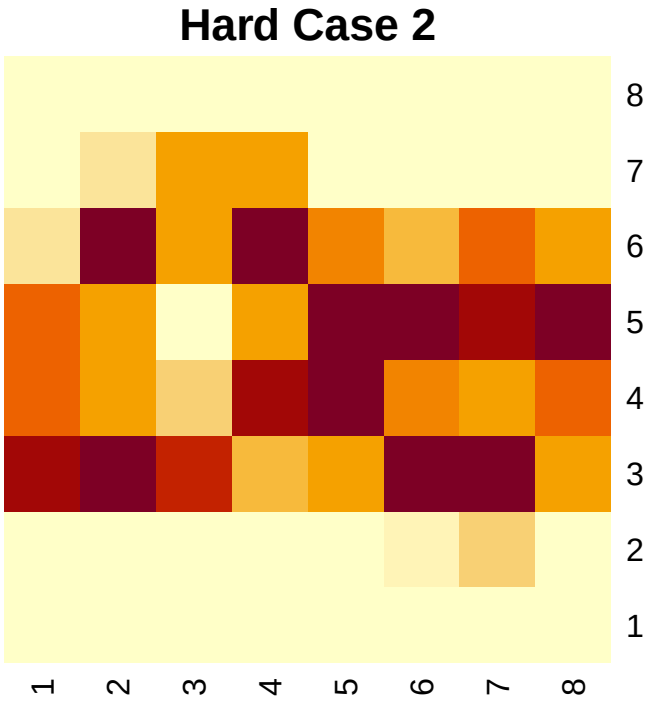

Hard Case 1
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8


Hard Case 2
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8




```
for (i in 1:length(est_matrixs)) {
mat <- est_matrixs[[i]]
heatmap(mat, Colv = NA, Rowv = NA, scale = "none", main = paste("Hard Case", i))

}
```

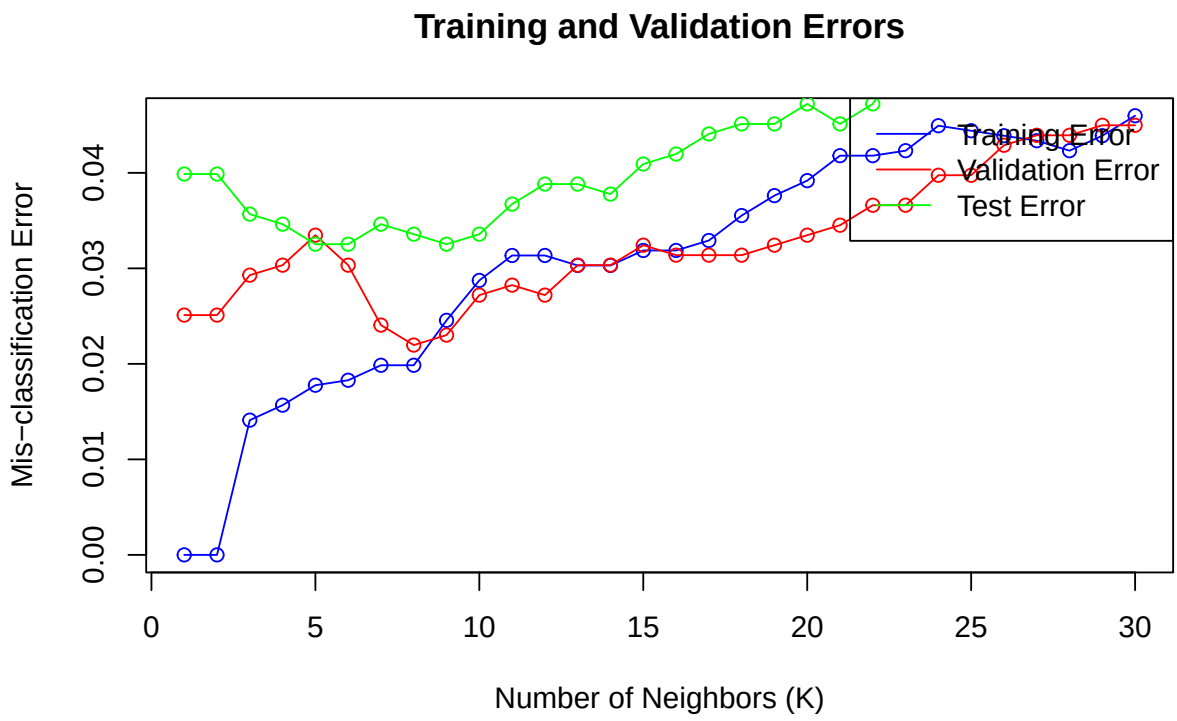

Hard Case 1
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8


Hard Case 2
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8

3.7 Training via different k in on the training and validation data

according the plot , k = 3 is best value on training data and validation data, though the performance of k = 1 is better than k = 3 on training data, it is not the best value on validation data due to the weak generalization ability, but when we apply it on test data, its performance is not as good as predicted

```
library(ggplot2)
train_error_rates <- list()
valid_error_rates <- list()
test_error_rates <- list()

for (ki in 1:30) {
# cat(paste("current k:",ki,"\n",sep=""))
train_ki_model <- kknn(formula, train = train_digitals, test = train_digitals,
                       k = ki, kernel = 'rectangular')
valid_ki_model <- kknn(formula, train = train_digitals, test = valid_digitals,
                       k = ki, kernel = 'rectangular')

test_ki_model <- kknn(formula, train = train_digitals, test = test_digitals,
                      k = ki, kernel = 'rectangular')

train_confusion <- table(train_digitals$V65, train_ki_model$fitted.values)
valid_confusion <- table(valid_digitals$V65, valid_ki_model$fitted.values)
test_confusion <- table(test_digitals$V65, test_ki_model$fitted.values)

train_error_rate <- sum(diag(train_confusion)) / sum(train_confusion)
valid_error_rate <- sum(diag(valid_confusion)) / sum(valid_confusion)

test_error_rate <- sum(diag(test_confusion)) / sum(test_confusion)

# print(train_error_rate)
# print(valid_error_rate)
train_error_rates[[ki]] <- 1 - train_error_rate
valid_error_rates[[ki]] <- 1 - valid_error_rate
test_error_rates[[ki]] <- 1 - test_error_rate

}
plot(1:30, train_error_rates, type = "o", col = "blue",
     ylim = range(c(train_error_rates, valid_error_rates)),
 xlab = "Number of Neighbors (K)", ylab = "Mis-classification Error",
 main = "Training and Validation Errors")
lines(1:30, valid_error_rates, type = "o", col = "red")
lines(1:30, test_error_rates, type = "o", col = "green")
legend("topright", legend = c("Training Error", "Validation Error","Test Error"),
       col = c("blue", "red","green"), lty = 1)
```



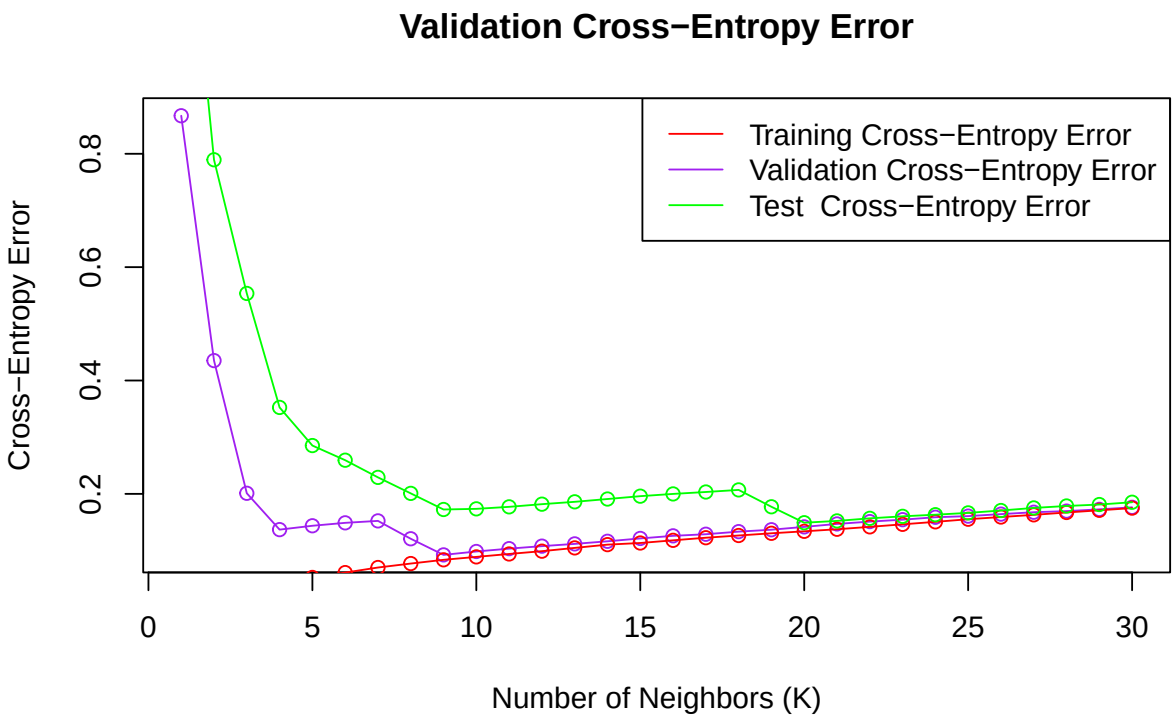

```
#
```

3.8 Change mis-classification error to cross-entropy

As is shown from the figure, the optimal K value is around 10 with the lowest validation Cross-entropy error value. In cross-entropy method, the predicted probabilities for different classes are taken into account, making it more reliable cmparing with mis-classification error method in this classification problem.

```
valid_cross_entropy_errors <- list()
train_cross_entropy_errors <- list()
test_cross_entropy_errors <- list()

for (ki in 1:30) {

  #print(ki)
  valid_ki_model <- kknn(formula, train = train_digitals, test = valid_digitals,
                         k = ki, kernel = 'rectangular')
  train_ki_model <- kknn(formula, train = train_digitals, test = train_digitals,
                         k = ki, kernel = 'rectangular')
  test_ki_model <- kknn(formula, train = train_digitals, test = test_digitals,
                        k = ki, kernel = 'rectangular')
  valid_probs <- valid_ki_model$prob
  train_probs <- train_ki_model$prob
  test_probs <- test_ki_model$prob

  valid_log_probs <- log(valid_probs + 1e-15)  # Add small constant to avoid log(0)
  train_log_probs <- log(train_probs + 1e-15)  # Add small constant to avoid log(0)
```

```r
  test_log_probs <- log(test_probs + 1e-15)  # Add small constant to avoid log(0)

# -1 means do not contain intercept
# One-hot encoding
#This type of matrix is typically used in machine learning and statistical modeling for feature
#engineering, particularly when converting categorical variables into dummy variables.
  valid_correct_class <- model.matrix(~V65 - 1, data = valid_digitals)  # One-hot encoding
  train_correct_class <- model.matrix(~V65 - 1, data = train_digitals)  # One-hot encoding
  test_correct_class <- model.matrix(~V65 - 1, data = test_digitals)  # One-hot encoding

  valid_cross_entropy_errors[[ki]] <- -sum(valid_correct_class
                                           * valid_log_probs) / nrow(valid_digitals)
  train_cross_entropy_errors[[ki]] <- -sum(train_correct_class
                                           * train_log_probs) / nrow(train_digitals)
  test_cross_entropy_errors[[ki]] <- -sum(test_correct_class
                                          * test_log_probs) / nrow(test_digitals)
  #print(-sum(valid_correct_class * valid_log_probs) )
  #print(-sum(train_correct_class * train_log_probs))
  #print(-sum(test_correct_class * test_log_probs) )
}


# plot(1:30, train_error_rates, type = "o", col = "blue",
# ylim = range(c(train_error_rates, valid_error_rates)),
#    xlab = "Number of Neighbors (K)", ylab = "Mis-classification Error",
# main = "Training and Validation Errors")
# lines(1:30, valid_error_rates, type = "o", col = "red")
# lines(1:30, test_error_rates, type = "o", col = "green")
# legend("topright", legend = c("Training Error", "Validation Error","Test Error"),
# col = c("blue", "red","green"), lty = 1)
plot(1:30, valid_cross_entropy_errors, type = "o", col = "purple",
   xlab = "Number of Neighbors (K)", ylab = "Cross-Entropy Error",
   main = "Validation Cross-Entropy Error")
lines(1:30, train_cross_entropy_errors, type = "o", col = "red")
lines(1:30, test_cross_entropy_errors, type = "o", col = "green")
legend("topright", legend = c("Training Cross-Entropy Error",
                "Validation Cross-Entropy Error","Test  Cross-Entropy Error"),
       col = c("red", "purple","green"), lty = 1)
```



4 Assignment 2: Linear regression and ridge regression

4.1 set up

We need to download some useful packages before the start.

```
install.packages("caret")
```

```
## Warning:   'caret'
```

```
library(caret)
```

4.2 Prepare the dataset

Task1. Firstly, we read the file and divided the data into training and test data (60/40).

```
data <- read.csv("parkinsons.csv") #
set.seed(12345)
ini_sample<- sample(1:nrow(data),0.6*nrow(data))
train_data<- data[ini_sample,]
test_data<- data[-ini_sample,]
```

And then we scaled the dataset appropriately.

```
scale_para<- preProcess(train_data)
train_data_scaled<- predict(scale_para,train_data)
test_data_scaled<- predict(scale_para,test_data)
```

4.3 Build models

Task2. Next, we computed a linear regression model , estimate training and test MSE. the p-values are summarized in the picture.

```
model<- lm(motor_UPDRS ~ .-subject. -age -sex -test_time -total_UPDRS - 1,train_data_scaled)
train_prediction<- predict(model,train_data_scaled)
train_mse<- mean((train_prediction - train_data_scaled$motor_UPDRS)^2)
test_prediction<- predict(model,test_data_scaled)
test_mse<- mean((test_prediction - test_data_scaled$motor_UPDRS)^2)
```

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
Jitter...       0.186931   0.149561   1.250 0.211431
Jitter.Abs.    -0.169609   0.040805  -4.157 3.31e-05 ***
Jitter.RAP     -5.269544  18.834160  -0.280 0.779658
Jitter.PPQ5    -0.074568   0.087766  -0.850 0.395592
Jitter.DDP      5.249558  18.837525   0.279 0.780510
Shimmer         0.592436   0.205981   2.876 0.004050 **
Shimmer.dB.    -0.172655   0.139316  -1.239 0.215315
Shimmer.APQ3   32.070932  77.159242   0.416 0.677694
Shimmer.APQ5   -0.387507   0.113789  -3.405 0.000668 ***
Shimmer.APQ11   0.305546   0.061236   4.990 6.34e-07 ***
Shimmer.DDA   -32.387241  77.158814  -0.420 0.674695
NHR            -0.185387   0.045567  -4.068 4.84e-05 ***
HNR            -0.238543   0.036395  -6.554 6.41e-11 ***
RPDE            0.004068   0.022664   0.179 0.857556
DFA            -0.280318   0.020136 -13.921  < 2e-16 ***
PPE             0.226467   0.032881   6.887 6.70e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 1: model

A smaller p-value means a larger contribution to the model. Specifically, Jitter.Abs., Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA and PPE contribute significantly to the model. We did predictions on both the test data and the training data using the linear regression model that we got. And then we calculated the MSE of training and test data.The results are as follows: test_MSE:0.9354477 training_MSE:0.8785431

Task3. Implement 4 following functions:

loglikelihiid function that for a given parameter vector theta and dispersion sigma.

```
logLikelihood <- function(theta, sigma, x, y) {
  n <- length(y)
  predictions <- x %*% theta
  residuals <- y - predictions
  log_likelihood <- -0.5 * n * log(2 * pi * sigma^2) - (t(residuals) %*% residuals)/ (2 * sigma^2)
```

```
  return(as.numeric(log_likelihood))
}
```

Ridge function that for given vector theta, scalar sigma and scalar lambda and adds up a Ridge penalty to the minus loglikelihood.

```
ridge <- function(theta, sigma, lambda, x, y) {
  log_likelihood <- logLikelihood(theta, sigma, x, y)
  ridge_penalty <- lambda * sum(theta^2)
  return(-log_likelihood + ridge_penalty)
}
```

Using function optim() with method="BFGS" to find the optimal theta and sigma for the given lambda.

```
ridgeopt <- function(lambda, x, y) {
  n <- ncol(x)
  init_params <- c(rep(0, n), 1)
  ridge_obj <- function(params) {
    theta <- params[1:n]
    sigma <- params[n + 1]
    return(ridge(theta, sigma, lambda, x, y))
  }
  opt <- optim(init_params, ridge_obj, method = "BFGS")
  theta_opt <- opt$par[1:n]
  sigma_opt <- opt$par[n + 1]
  return(list(theta = theta_opt, sigma = sigma_opt))
}
```

Computing the degrees of freedom of the Ridge model based on the training data.

```
freedom_degree <- function(lambda, x) {
  xT <- t(x) %*% x
  heat <- x %*% solve(xT + lambda * diag(ncol(x))) %*% t(x)
  df <- sum(diag(heat)) #trace
  return(df)
}
```

4.4 predict the values

Task4. Finally, we can compute optimal theta parameters for different lambda values by using function RidgeOpt.

```
train_data2<- as.matrix(train_data_scaled[,names(train_data)!="motor_UPDRS"])
test_data2<- as.matrix(test_data_scaled[,names(test_data)!="motor_UPDRS"])
train_value<- train_data_scaled$motor_UPDRS
test_value<- test_data_scaled$motor_UPDRS

lambda_values <- c(1, 100, 1000)

train_mse2<- c()
test_mse2<- c()
```

```r
df<- c()
theta_value<- list()
for (i in seq_along(lambda_values)){
  lambda<- lambda_values[i]
  ridgemodel<- ridgeopt(lambda,train_data2,train_value)
  thetavalue<- ridgemodel$theta

  theta_value[[i]]<- thetavalue

  train_predictions<- train_data2 %*% thetavalue
  train_mse2[i]<- mean((train_value - train_predictions)^2)

  test_predictions<- test_data2 %*% thetavalue
  test_mse2[i]<- mean((test_value - test_predictions)^2)

  df[i] <- freedom_degree(lambda,train_data2)

  result <- list(
    train_mse2 = train_mse2,
    test_mse2 = test_mse2,
    df = df,
    theta_value = theta_value
  )
}
print(result)
```

```
## $train_mse2
## [1] 0.09665726 0.09680434 0.10082977
##
## $test_mse2
## [1] 0.09250459 0.09274202 0.09725080
##
## $df
## [1] 18.857197 14.701717  9.321473
##
## $theta_value
## $theta_value[[1]]
##  [1] -0.026463038 -0.035283799  0.059207029 -0.002563679  0.959981195
##  [6]  0.168145629 -0.079347417 -0.043732685 -0.035093989 -0.054804671
## [11]  0.165626793 -0.026964285 -0.024931656 -0.142573230  0.077212413
## [16] -0.032327054  0.001465870 -0.004171716 -0.030467984 -0.006882262
## [21]  0.054681628
##
## $theta_value[[2]]
##  [1] -0.0226191438 -0.0325871283  0.0570391225 -0.0019559833  0.9529194191
##  [6]  0.1070246060 -0.0733873485 -0.0289585233 -0.0197662831 -0.0290450751
## [11]  0.0808540078  0.0120617713 -0.0239791846 -0.1021995385  0.0740933538
## [16] -0.0240649900  0.0009729876 -0.0024679539 -0.0284166393 -0.0083369428
## [21]  0.0572979530
##
## $theta_value[[3]]
##  [1] -0.0009068977 -0.0130847204  0.0428822485  0.0023423024  0.8921225040
##  [6]  0.0280785462 -0.0581572784 -0.0019913595 -0.0009807951 -0.0020047948
```

```
## [11]  0.0147796365  0.0157099419 -0.0208953140 -0.0291068011  0.0559883645
## [16] -0.0209099316 -0.0082453600 -0.0084473063 -0.0231997650 -0.0216914411
## [21]  0.0588202653
```

In general, a lower test MSE indicates that the model generalizes better. Higher degrees of freedom mean that models are more flexible and tend to fit details in the data, but can lead to overfitting; Lower degrees of freedom mean that the model is smoother, limiting the fit to the training data.

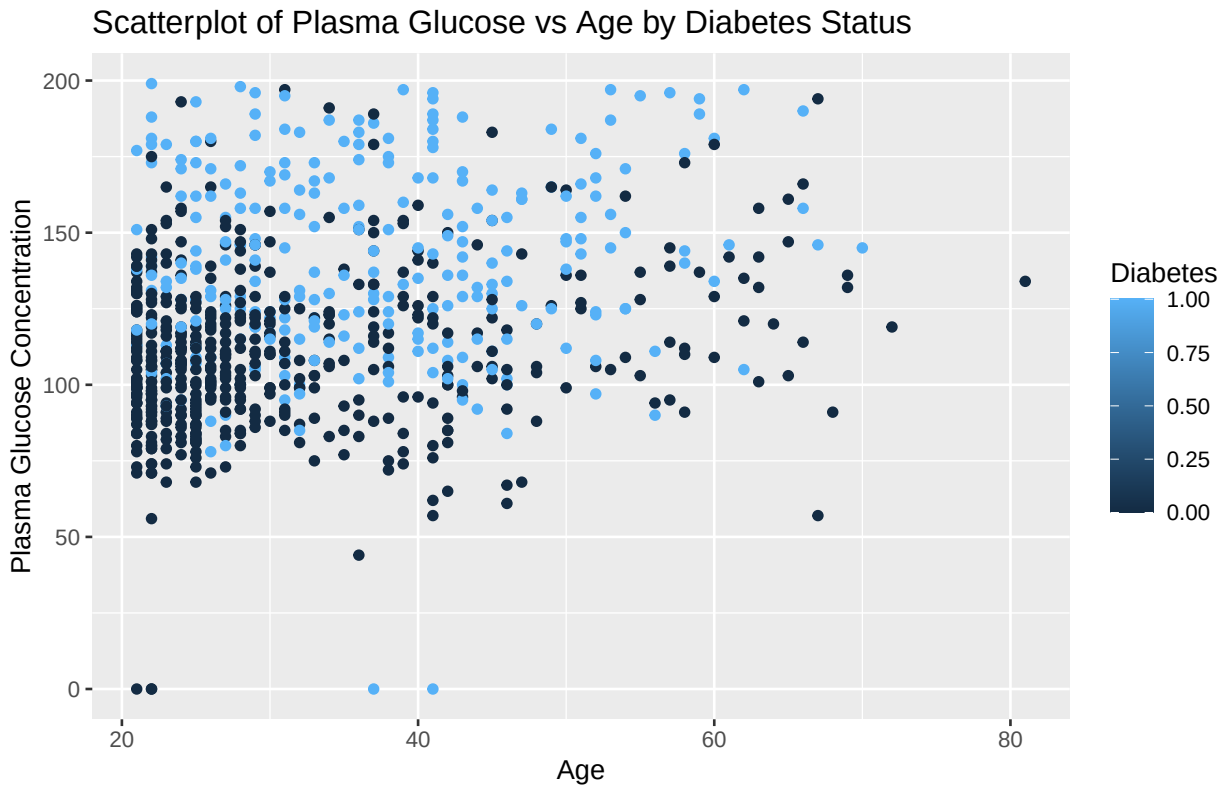
In this example, with the increase of parameter value, the MSEs of training and test data increase gradually and the degrees of freedom decrease gradually. However, when the parameter changes from 1 to 100, the MSEs change little (only 0.0002), and when the parameter changes from 100 to 1000, the MSEs increase more. Although parameter values equal to 1 and 100 are not much of a difference in MSEs, both are relatively small. However, when lambda equals 1, df is significant large and there is a risk of overfitting. So lambda equals 100 is the most appropriate choice.

5 Assignment 3. Logistic regression and basis function expansion

5.1 Read Data and show scatter plot

Read data and give a scatter plot showing a Plasma glucose concentration on Age where observations are colored by Diabetes levels.

```
diabetes <- read.csv('pima-indians-diabetes.csv',header = FALSE)
colnames(diabetes) <- c('Pregnancies','Plasma_glucose','blood_pressure','TricepsSkinFoldThickness','Ser
#
ggplot(diabetes,aes( x = diabetes$Age, y = diabetes$Plasma_glucose, color = diabetes$Diabetes)) +
  geom_point()+labs(x = "Age", y = "Plasma Glucose Concentration", color = "Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by Diabetes Status")
```



5.2 Train a logistic regression model when the threshold r = 0.5

```
formula <- Diabetes ~ Age + Plasma_glucose
diabetes$Diabetes <- as.factor(diabetes$Diabetes)
gml_model <- caret::train(formula, data = diabetes, method = "glm", family = "binomial")
#type = "prob" predict probability
#type = "raw" predict the raw value/ class
#diabetes_pred <- predict(gml_model, type = "prob")

classify_pred_res <- function(r,gml_model) {

  diabetes_pred <- predict(gml_model, type = "prob")
  diabetes_pred$predict <- lapply(1:nrow(diabetes_pred),
                      function(x) ifelse(diabetes_pred[x,2] > r, 1, 0))
  diabetes_pred$predict <- unlist(diabetes_pred$predict)
  diabetes_pred$raw <- diabetes$Diabetes
  diabetes_pred[, 3:4] <- lapply(diabetes_pred[, 3:4], as.factor)

  trainingData <-  gml_model$trainingData %>% select(-.outcome)
  diabetes_pred <- cbind(diabetes_pred, trainingData)

  diabetes_pred$Age <- gml_model$trainingData$Age
  diabetes_pred$Plasma_glucose <- gml_model$trainingData$Plasma_glucose
  return(diabetes_pred)
}
```

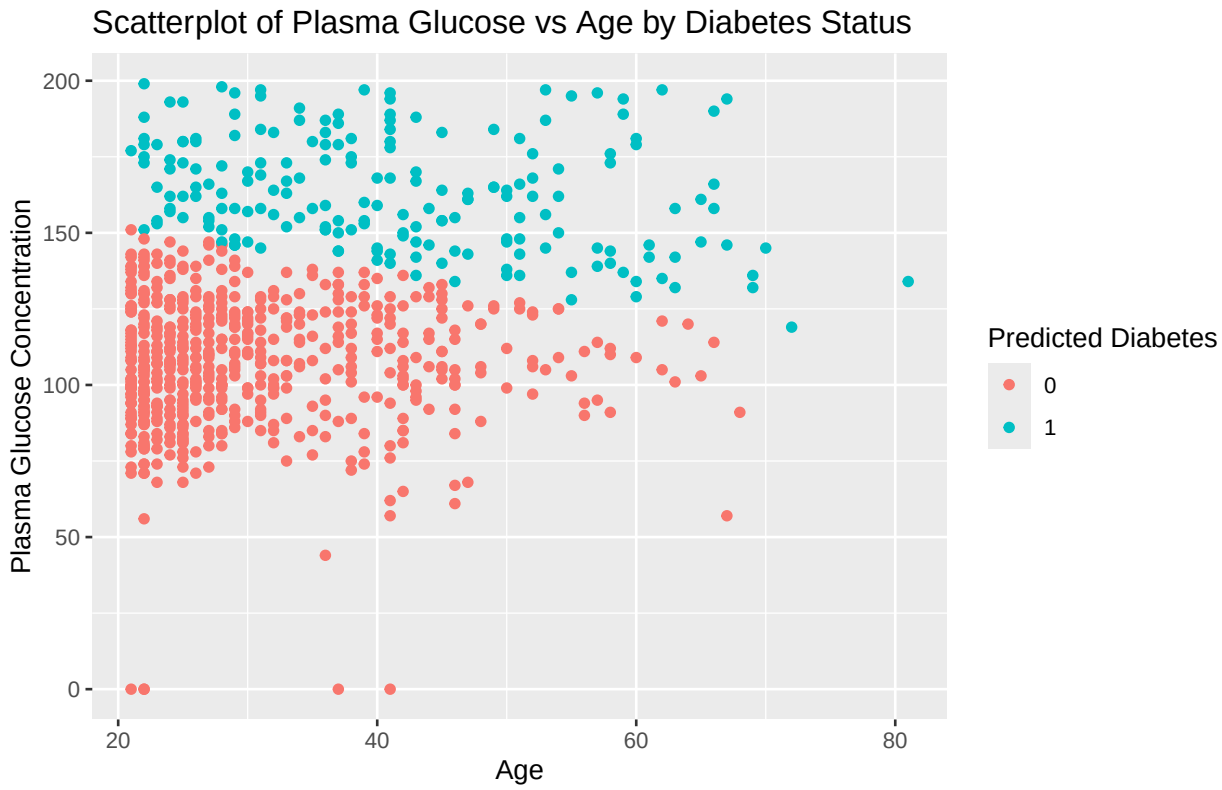
```
diabetes_pred <- classify_pred_res(0.5,gml_model)

diabetes_confusion <- table(diabetes_pred$raw, diabetes_pred$predict)
error_rate <- 1 - (sum(diag(diabetes_confusion)) / sum(diabetes_confusion))
cat(" training misclassification error:",error_rate)
```

```
##  training misclassification error: 0.2630208
```

5.3 Draw a scatter plot showing the predicted diabetes status

We can see that the logistic regression visually separates the two classes of diabetes status well, but the mis classification error is high due to the overlap of the two classes, maybe change can improve the performance.we will try later.

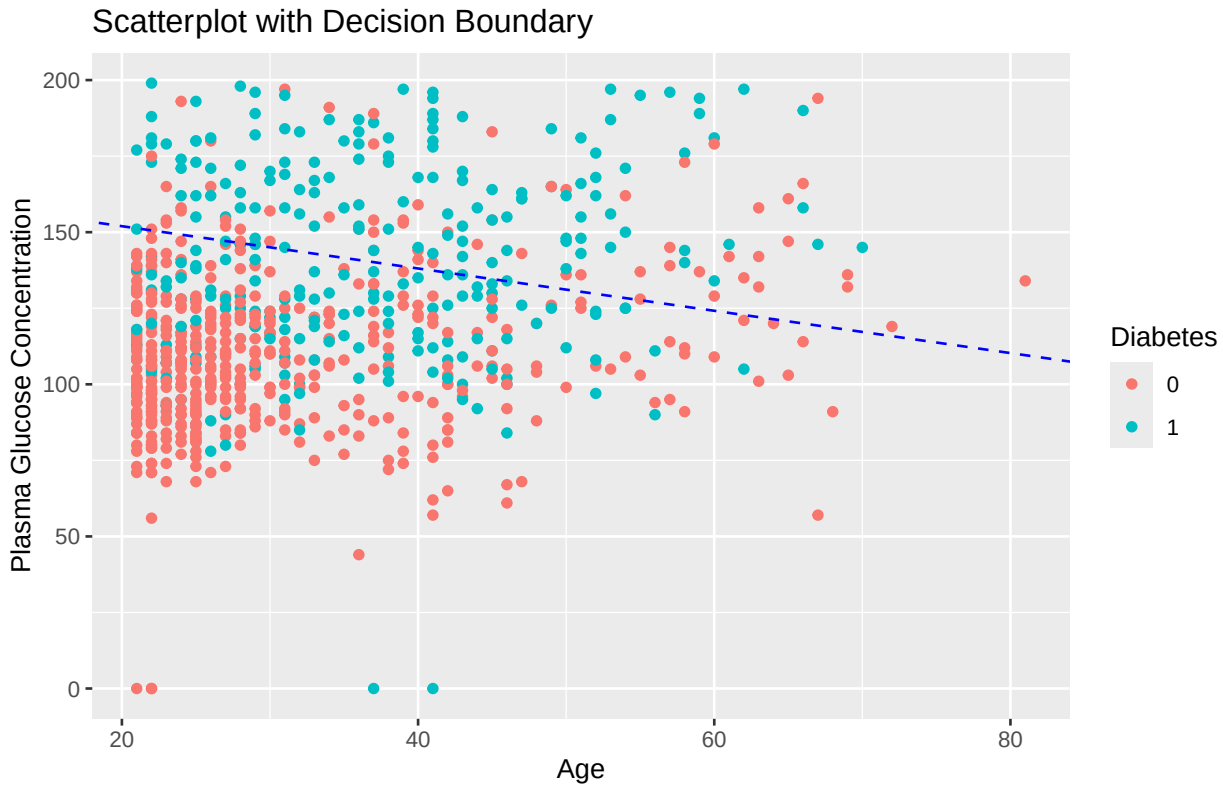
```
ggplot(diabetes,aes( x = diabetes_pred$Age, y = diabetes_pred$Plasma_glucose,
                     color = diabetes_pred$predict)) + geom_point()+
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by Diabetes Status")
```



5.4 Draw a decision boundary between the two predicted classes

We can see that the boundary line try to split the dots into two classes and put the most the red dots below the line and the most blue dots above the line, but when the age exceed the 50, the performance of the model is not good.it seems that the number of red dot below the boundary line is same as the number above the line, it results the high misclassification error.

```r
get_boundary_line <- function(gml_model,r, y_name) {
  coefficients <- gml_model$finalModel$coefficients
  boundary_parameter <- list()
  coef_names <- names(coefficients)
  y_value <- coefficients[[y_name]]
  boundary_parameter$Intercept <- -(coefficients[['(Intercept)']] / y_value) - (log((1/r) - 1)/y_value)
  # boundary_parameter$intercept <- intercept
  for (name in coef_names){
    if (name != '(Intercept)'){
      boundary_parameter[[name]] <- -coefficients[[name]] / y_value
    }
  }
  return(boundary_parameter)
}

boundary_parameter <- get_boundary_line(gml_model,0.5,'Plasma_glucose')
ggplot(diabetes,aes( x = diabetes$Age, y = diabetes$Plasma_glucose, color  = diabetes$Diabetes)) +
  geom_point()+
  geom_abline(slope = boundary_parameter$Age, intercept = boundary_parameter$Intercept,color = "blue", l
  labs(x = "Age", y = "Plasma Glucose Concentration", color = "Diabetes") +
  ggtitle("Scatterplot with Decision Boundary")
```



5.5 Change the thresholds r to 0.2 , 0.8 to see the what happened

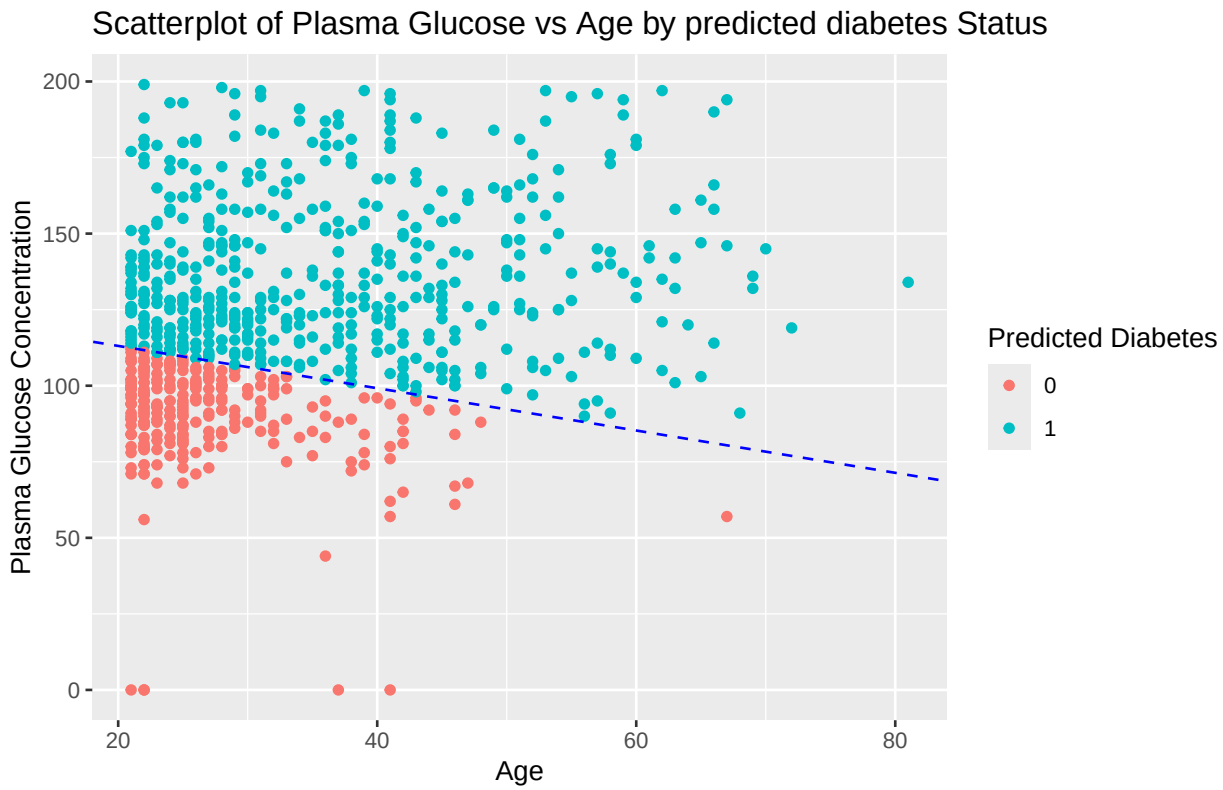
We can see that when r = 0.2, for the red dots , its TP is relatively high , but the Recall is low ,for the blue dots,its TP is lower than red dots, but the Recall is higher than red dots, it means that the model is

more likely to predict the blue dots as the positive class, but the blue dots are more likely to be the negative class, it results in the high misclassification error, when r = 0.8, the model is more likely to predict the red dots as the positive class, but the red dots are more likely to be the negative class, it results in the high misclassification error.

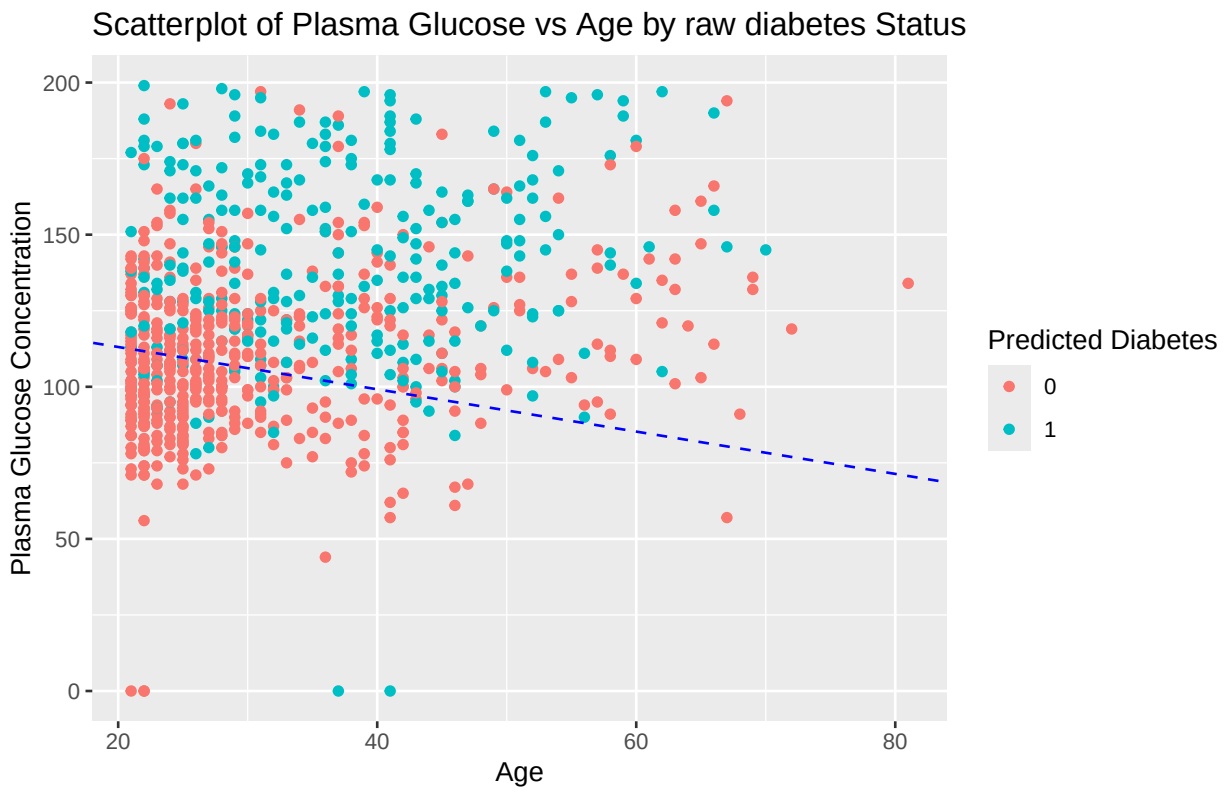
```
pred_res_0.2 <- classify_pred_res(0.2,gml_model)
pred_res_0.8 <- classify_pred_res(0.8,gml_model)
```

5.5.1 plot the scatter when r = 0.2

```
boundary_parameter_0.2 <- get_boundary_line(gml_model,0.2,'Plasma_glucose')

ggplot(diabetes,aes( x = pred_res_0.2$Age, y = pred_res_0.2$Plasma_glucose,
         color = pred_res_0.2$predict)) + geom_point()+
geom_abline(slope = boundary_parameter_0.2$Age,
intercept = boundary_parameter_0.2$Intercept, color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by predicted diabetes Status")
```

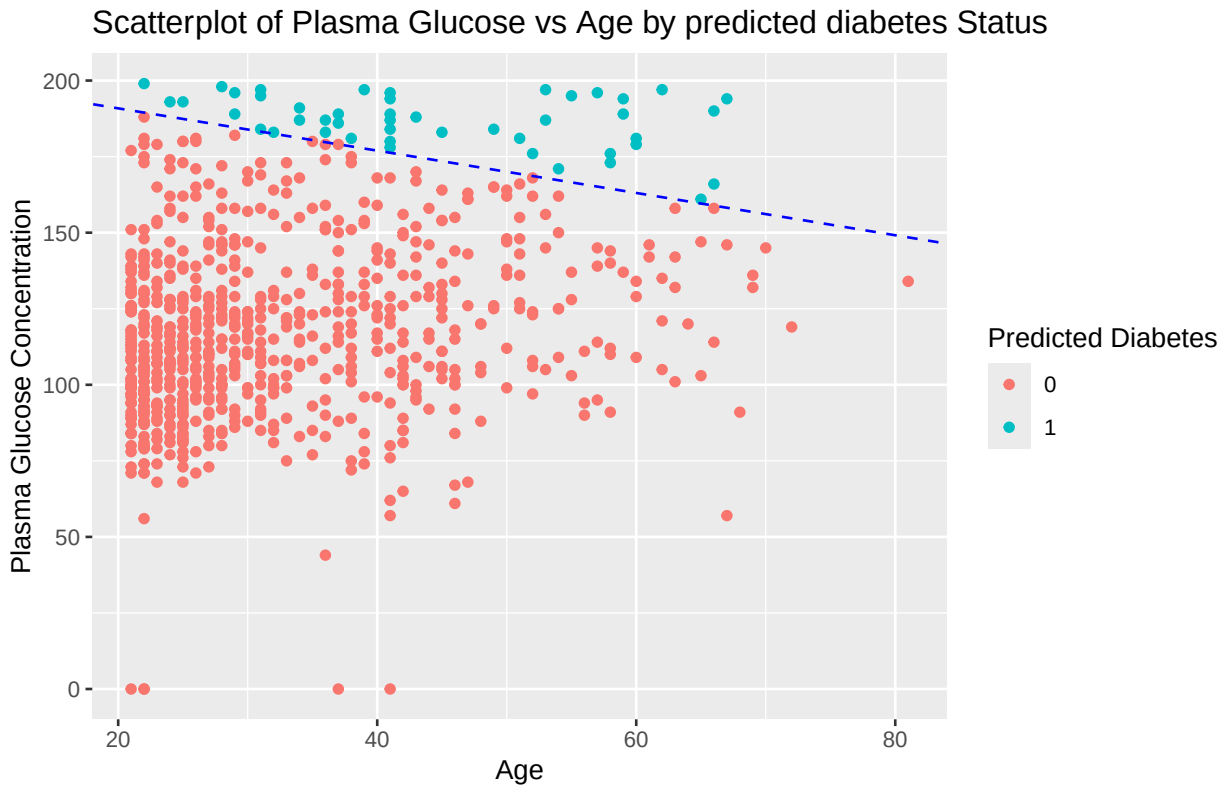



```
ggplot(diabetes,aes( x = pred_res_0.2$Age, y = pred_res_0.2$Plasma_glucose,
                     color = pred_res_0.2$raw)) + geom_point()+
geom_abline(slope = boundary_parameter_0.2$Age,
          intercept = boundary_parameter_0.2$Intercept,
          color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by raw diabetes Status")
```

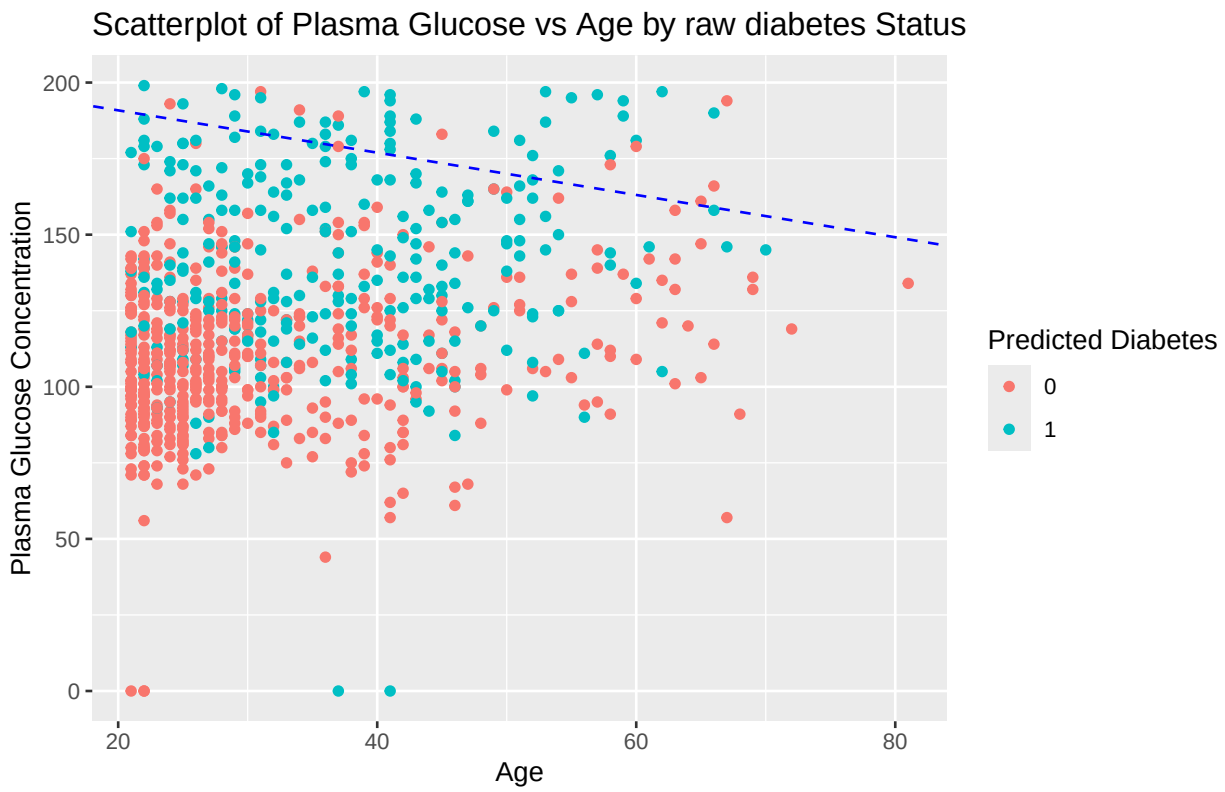


5.5.2 plot the scatter when r = 0.8

```
boundary_parameter_0.8 <- get_boundary_line(gml_model,0.8,'Plasma_glucose')

ggplot(diabetes,aes( x = pred_res_0.8$Age, y = pred_res_0.8$Plasma_glucose,
                     color = pred_res_0.8$predict)) + geom_point()+
geom_abline(slope = boundary_parameter_0.8$Age,
          intercept = boundary_parameter_0.8$Intercept,
          color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by predicted diabetes Status")
```




```
ggplot(diabetes,aes( x = pred_res_0.8$Age, y = pred_res_0.8$Plasma_glucose,
                     color = pred_res_0.8$raw)) + geom_point()+
geom_abline(slope = boundary_parameter_0.8$Age,
            intercept = boundary_parameter_0.8$Intercept,
            color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by raw diabetes Status")
```



5.5.3 Perform a basis function expansion trick

We can see that after add the basis function expansion, the misclassification error is lower than the previous model, it means that the basis function expansion can improve the performance of the model, look at the coefficients furtherly, the new added variables slightly affect the prediction, it means that the new added variables affect the prediction positively and the decision boundary become from a line to a multidimensional graphics.

```
diabetes$z1 <- diabetes$Plasma_glucose^4
diabetes$z2 <- diabetes$Plasma_glucose^3 * diabetes$Age
diabetes$z3 <- diabetes$Plasma_glucose^2 * diabetes$Age^2
diabetes$z4 <- diabetes$Plasma_glucose * diabetes$Age^3
diabetes$z5 <- diabetes$Age^4
formula <- Diabetes ~ Age + Plasma_glucose + z1 + z2 + z3 + z4 + z5
new_gml_model <- caret::train(formula, data = diabetes, method = "glm", family = "binomial")
new_pred_res <- classify_pred_res(0.5,new_gml_model)
new_diabetes_confusion <- table(new_pred_res$raw, new_pred_res$predict)
error_rate <- 1 - (sum(diag(new_diabetes_confusion)) / sum(new_diabetes_confusion))
cat(" training misclassification error:",error_rate)
```

```
##  training misclassification error: 0.2447917
```

```
new_boundary_parameter <- get_boundary_line(new_gml_model,0.5,'Plasma_glucose')
cat(new_gml_model$finalModel$coefficients)
```

```
## -9.309821 0.1456805 0.03793014 1.278015e-08 -1.7796e-07 8.51515e-07 -1.698011e-06 8.126623e-07
```

6 Assignment 4: Handwritten digit recognition with K-nearest neighbors

- Why can it be important to consider various probability thresholds in the classification problems, according to the book?

Page 50. In binary classification problems, we can express the most probable class with decision threshold r= 0.5. With decision threshold r= 0.5, the misclassification rate is minimized. However, it can happen when the the miscassification problem is asymmetric or imbalanced. For example, in medical diagnosis problems, it is more important not to falsely predict the negative class than to falsely to predict a positive class, where the minimized misclassification rate is not appropriate. Furthermore, the medical diagnosis problem can be imbalanced when the disorder is very rare. In this case, accurate predictions of the negative class is valued higher than the accurate prediction of positive class if only misclassification rate is considered.

- What ways of collecting correct values of the target variable for the supervised learning problems are mentioned in the book?

In supervised learning problems, the target variables can be manually labelled by a domain expert.Target variables can also be labelled from predictive models based. Or the output is labelled naturally during the collection of training data. (Page 6, paragraph 2)

- How can one express the cost function of the linear regression in the matrix form, according to the book? The cost function for the linear regression model can be written with matrix notations as (Page 41):

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}(x_i;\boldsymbol{\theta}) - y_i)^2 = \frac{1}{n}||\hat{y} - y||_2^2 = \frac{1}{n}||X\boldsymbol{\theta} - y||_2^2 = \frac{1}{n}|| \in ||_2^2$$

7 Appendix(Code)

```
# Load packages
library('ggplot2') # visualization
library('ggthemes') # visualization
library('scales') # visualization
library('dplyr') # data manipulation
library('randomForest') # classification algorithm
library('caret')

#assignment 1

digitals <- read.csv('../data/optdigits.csv',header = FALSE)
# change all the columns to factor
#digitals <- digitals %>% mutate_all(as.factor)
digitals$V65 <- as.factor(digitals$V65)
train_index <- createDataPartition(digitals$V65, p = 0.5, list = F)
train_digitals <- digitals[train_index,]
remainingData <- digitals[-train_index, ]
validationIndex <- createDataPartition(remainingData$V65, p = 0.5, list = FALSE)
valid_digitals <- remainingData[validationIndex, ]
test_digitals <- remainingData[-validationIndex, ]
```

```r
library(kknn)
formula <- V65~.
# if kenerl = 'rectangular' , so every point in the neighborhood is weighted equally
# both of the parameters of train and test use train_digital data
# if  your predict columns is continuous, kknn will recognized as a regression task
# under this situation, you can not get a probability of the prediction
knn_train_model <- kknn(formula, train_digitals, train_digitals,
                        kernel = 'rectangular',distance = 1,)
train_predictions <- fitted(knn_train_model)
knn_test_model <- kknn(formula, train = train_digitals, test = test_digitals,
                       k = 30, kernel = 'rectangular')
train_confusion <- table(train_digitals$V65, train_predictions)
test_confusion <- table(test_digitals$V65, knn_test_model$fitted.values)
test_error_rate <- 1 - sum(diag(test_confusion)) / sum(test_confusion)
train_error_rate <- 1-  sum(diag(train_confusion)) / sum(train_confusion)
# only observer the top 10 rows
table(train_digitals$V65, train_predictions)
table(test_digitals$V65, knn_test_model$fitted.values)
library(dplyr)

train_predict <- data.frame(train_digitals$V65, train_predictions,knn_train_model$prob)
train_predict$max_prob <- apply(train_predict[,3:12], 1, max)

train_predict_8 <- train_predict[train_predict$train_digitals.V65 == 8,]
# do not change the index while sorting
train_predict_8 <- train_predict_8[order(train_predict_8$X8), , drop = FALSE]

# get the 3 cases that were hardest to classify
hardest_cases_for_8 <- train_predict_8 %>% head(3)
easy_cases_for_8 <- train_predict_8 %>% tail(2)
hardest_cases_index <- rownames(hardest_cases_for_8)
est_cases_index <- rownames(easy_cases_for_8)
# reindex the row index
row.names(train_digitals) <- NULL
full_hardest_cases <- train_digitals[hardest_cases_index,1:64]
full_est_cases <- train_digitals[est_cases_index,1:64]

hardest_matrixs <- lapply(1:nrow(full_hardest_cases),
  function(i) matrix(as.numeric(full_hardest_cases[i, ,
                      drop = FALSE]),nrow = 8,ncol = 8))
est_matrixs <- lapply(1:nrow(full_est_cases),
  function(i) matrix(as.numeric(full_est_cases[i, ,
                      drop = FALSE]),nrow = 8,ncol = 8))

for (i in 1:length(hardest_matrixs)) {
mat <- hardest_matrixs[[i]]
heatmap(mat, Colv = NA, Rowv = NA, scale = "none", main = paste("Hard Case", i))
}

for (i in 1:length(est_matrixs)) {
mat <- est_matrixs[[i]]
heatmap(mat, Colv = NA, Rowv = NA, scale = "none", main = paste("Hard Case", i))
```

```
}
library(ggplot2)
train_error_rates <- list()
valid_error_rates <- list()
test_error_rates <- list()

for (ki in 1:30) {
# cat(paste("current k:",ki,"\n",sep=""))
train_ki_model <- kknn(formula, train = train_digitals, test = train_digitals,
                       k = ki, kernel = 'rectangular')
valid_ki_model <- kknn(formula, train = train_digitals, test = valid_digitals,
                       k = ki, kernel = 'rectangular')

test_ki_model <- kknn(formula, train = train_digitals, test = test_digitals,
                      k = ki, kernel = 'rectangular')

train_confusion <- table(train_digitals$V65, train_ki_model$fitted.values)
valid_confusion <- table(valid_digitals$V65, valid_ki_model$fitted.values)
test_confusion <- table(test_digitals$V65, test_ki_model$fitted.values)

train_error_rate <- sum(diag(train_confusion)) / sum(train_confusion)
valid_error_rate <- sum(diag(valid_confusion)) / sum(valid_confusion)

test_error_rate <- sum(diag(test_confusion)) / sum(test_confusion)

# print(train_error_rate)
# print(valid_error_rate)
train_error_rates[[ki]] <- 1 - train_error_rate
valid_error_rates[[ki]] <- 1 - valid_error_rate
test_error_rates[[ki]] <- 1 - test_error_rate

}
plot(1:30, train_error_rates, type = "o", col = "blue",
     ylim = range(c(train_error_rates, valid_error_rates)),
 xlab = "Number of Neighbors (K)", ylab = "Mis-classification Error",
 main = "Training and Validation Errors")
lines(1:30, valid_error_rates, type = "o", col = "red")
lines(1:30, test_error_rates, type = "o", col = "green")
legend("topright", legend = c("Training Error", "Validation Error","Test Error"),
       col = c("blue", "red","green"), lty = 1)
valid_cross_entropy_errors <- list()
train_cross_entropy_errors <- list()
test_cross_entropy_errors <- list()

for (ki in 1:30) {


  valid_ki_model <- kknn(formula, train = train_digitals, test = valid_digitals,
                         k = ki, kernel = 'rectangular')
  train_ki_model <- kknn(formula, train = train_digitals, test = train_digitals,
                         k = ki, kernel = 'rectangular')
  test_ki_model <- kknn(formula, train = train_digitals, test = test_digitals,
                        k = ki, kernel = 'rectangular')
```

```r
  valid_probs <- valid_ki_model$prob
  train_probs <- train_ki_model$prob
  test_probs <- test_ki_model$prob

  valid_log_probs <- log(valid_probs + 1e-15)  # Add small constant to avoid log(0)
  train_log_probs <- log(train_probs + 1e-15)  # Add small constant to avoid log(0)
  test_log_probs <- log(test_probs + 1e-15)  # Add small constant to avoid log(0)

# -1 means do not contain intercept
# One-hot encoding
#This type of matrix is typically used in machine learning and statistical modeling for feature
#engineering, particularly when converting categorical variables into dummy variables.
  valid_correct_class <- model.matrix(~V65 - 1, data = valid_digitals)  # One-hot encoding
  train_correct_class <- model.matrix(~V65 - 1, data = train_digitals)  # One-hot encoding
  test_correct_class <- model.matrix(~V65 - 1, data = test_digitals)  # One-hot encoding

  valid_cross_entropy_errors[[ki]] <- -sum(valid_correct_class
                                           * valid_log_probs) / nrow(valid_digitals)
  train_cross_entropy_errors[[ki]] <- -sum(train_correct_class
                                           * train_log_probs) / nrow(train_digitals)
  test_cross_entropy_errors[[ki]] <- -sum(test_correct_class
                                          * test_log_probs) / nrow(test_digitals)

}


# plot(1:30, train_error_rates, type = "o", col = "blue",
# ylim = range(c(train_error_rates, valid_error_rates)),
#    xlab = "Number of Neighbors (K)", ylab = "Mis-classification Error",
# main = "Training and Validation Errors")
# lines(1:30, valid_error_rates, type = "o", col = "red")
# lines(1:30, test_error_rates, type = "o", col = "green")
# legend("topright", legend = c("Training Error", "Validation Error","Test Error"),
# col = c("blue", "red","green"), lty = 1)
plot(1:30, valid_cross_entropy_errors, type = "o", col = "purple",
   xlab = "Number of Neighbors (K)", ylab = "Cross-Entropy Error",
   main = "Validation Cross-Entropy Error")
lines(1:30, train_cross_entropy_errors, type = "o", col = "red")
lines(1:30, test_cross_entropy_errors, type = "o", col = "green")
legend("topright", legend = c("Training Cross-Entropy Error",
                "Validation Cross-Entropy Error","Test  Cross-Entropy Error"),
       col = c("red", "purple","green"), lty = 1)

#assignment 2
install.packages("caret")
library(caret)
data <- read.csv("../data/parkinsons.csv") #
set.seed(42)
ini_sample<- sample(1:nrow(data),0.6*nrow(data))
train_data<- data[ini_sample,]
test_data<- data[-ini_sample,]
sacale_data<- train_data[,names(train_data)!="motor_UPDRS"]
scale_para<- preProcess(sacale_data)
```

```r
train_data_scaled<- predict(scale_para,train_data)
test_data_scaled<- predict(scale_para,test_data)
train_data_scaled$motor_UPDRS <- train_data$motor_UPDRS
test_data_scaled$motor_UPDRS <- test_data$motor_UPDRS
model<- lm(motor_UPDRS ~ .,train_data_scaled)
train_prediction<- predict(model,train_data_scaled)
train_mse<- mean((train_prediction - train_data_scaled$motor_UPDRS)^2)
test_prediction<- predict(model,test_data_scaled)
test_mse<- mean((test_prediction - test_data_scaled$motor_UPDRS)^2)
logLikelihood <- function(theta, sigma, x, y) {
  n <- length(y)
  predictions <- x %*% theta
  residuals <- y - predictions
  log_likelihood <- -0.5 * n * log(2 * pi * sigma^2) - (t(residuals) %*% residuals)/ (2 * sigma^2)
  return(as.numeric(log_likelihood))
}
ridge <- function(theta, sigma, lambda, x, y) {
  log_likelihood <- logLikelihood(theta, sigma, x, y)
  ridge_penalty <- lambda * sum(theta^2)
  return(-log_likelihood + ridge_penalty)
}
ridgeopt <- function(lambda, x, y) {
  n <- ncol(x)
  init_params <- c(rep(0, n), 1)
  ridge_obj <- function(params) {
    theta <- params[1:n]
    sigma <- params[n + 1]
    return(ridge(theta, sigma, lambda, x, y))
  }
  opt <- optim(init_params, ridge_obj, method = "BFGS")
  theta_opt <- opt$par[1:n]
  sigma_opt <- opt$par[n + 1]
  return(list(theta = theta_opt, sigma = sigma_opt))
}
freedom_degree <- function(lambda, x) {
  xT <- t(x) %*% x
  heat <- solve(xT + lambda * diag(ncol(x))) %*% t(x)
  df <- sum(diag(heat)) #trace
  return(df)
}
train_data2 <- as.matrix(train_data[,names(train_data)!="motor_UPDRS"])
test_data2<- as.matrix(test_data[,names(test_data)!="motor_UPDRS"])
train_value <- train_data$motor_UPDRS
test_value <- test_data$motor_UPDRS

lambda_values <- c(1, 100, 1000)

train_mse2<- c()
test_mse2<- c()
df<- c()
theta_value<- list()
for (i in seq_along(lambda_values)){
  lambda<- lambda_values[i]
```

```r
  ridgemodel<- ridgeopt(lambda,train_data2,train_value)
  thetavalue<- ridgemodel$theta

  theta_value[[i]]<- thetavalue

  train_predictions<- train_data2 %*% thetavalue
  train_mse2[i]<- mean((train_value - train_predictions)^2)

  test_predictions<- test_data2 %*% thetavalue
  test_mse2[i]<- mean((test_value - test_predictions)^2)

  df[i] <- freedom_degree(lambda,train_data2)

  result <- list(
    train_mse2 = train_mse2,
    test_mse2 = test_mse2,
    df = df,
    theta_value = theta_value
  )
}
print(result)

#assignment 3

diabetes <- read.csv('../data/pima-indians-diabetes.csv',header = FALSE)
colnames(diabetes) <- c('Pregnancies','Plasma_glucose','blood_pressure','TricepsSkinFoldThickness','Seru
#
ggplot(diabetes,aes( x = diabetes$Age, y = diabetes$Plasma_glucose, color = diabetes$Diabetes)) +
  geom_point()+labs(x = "Age", y = "Plasma Glucose Concentration", color = "Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by Diabetes Status")
formula <- Diabetes ~ Age + Plasma_glucose
diabetes$Diabetes <- as.factor(diabetes$Diabetes)
gml_model <- caret::train(formula, data = diabetes, method = "glm", family = "binomial")
#type = "prob" predict probability
#type = "raw" predict the raw value/ class
#diabetes_pred <- predict(gml_model, type = "prob")

classify_pred_res <- function(r,gml_model) {

  diabetes_pred <- predict(gml_model, type = "prob")
  diabetes_pred$predict <- lapply(1:nrow(diabetes_pred),
                       function(x) ifelse(diabetes_pred[x,2] > r, 1, 0))
  diabetes_pred$predict <- unlist(diabetes_pred$predict)
  diabetes_pred$raw <- diabetes$Diabetes
  diabetes_pred[, 3:4] <- lapply(diabetes_pred[, 3:4], as.factor)

  trainingData <-  gml_model$trainingData %>% select(-.outcome)
  diabetes_pred <- cbind(diabetes_pred, trainingData)

  diabetes_pred$Age <- gml_model$trainingData$Age
  diabetes_pred$Plasma_glucose <- gml_model$trainingData$Plasma_glucose
  return(diabetes_pred)
}
```

```r
diabetes_pred <- classify_pred_res(0.5,gml_model)

diabetes_confusion <- table(diabetes_pred$raw, diabetes_pred$predict)
error_rate <- 1 - (sum(diag(diabetes_confusion)) / sum(diabetes_confusion))
ggplot(diabetes,aes( x = diabetes_pred$Age, y = diabetes_pred$Plasma_glucose,
                     color = diabetes_pred$predict)) + geom_point()+
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by Diabetes Status")
get_boundary_line <- function(gml_model,r, y_name) {
  coefficients <- gml_model$finalModel$coefficients
  boundary_parameter <- list()
  coef_names <- names(coefficients)
  y_value <- coefficients[[y_name]]
  boundary_parameter$Intercept <- -(coefficients[['(Intercept)']] / y_value) - (log((1/r) - 1)/y_value)
  # boundary_parameter$intercept <- intercept
  for (name in coef_names){
    if (name != '(Intercept)'){
      boundary_parameter[[name]] <- -coefficients[[name]] / y_value
    }
  }
  return(boundary_parameter)
}

boundary_parameter <- get_boundary_line(gml_model,0.5,'Plasma_glucose')
ggplot(diabetes,aes( x = diabetes$Age, y = diabetes$Plasma_glucose, color  = diabetes$Diabetes)) +
  geom_point()+
  geom_abline(slope = boundary_parameter$Age, intercept = boundary_parameter$Intercept,color = "blue", l
  labs(x = "Age", y = "Plasma Glucose Concentration", color = "Diabetes") +
  ggtitle("Scatterplot with Decision Boundary")
pred_res_0.2 <- classify_pred_res(0.2,gml_model)
pred_res_0.8 <- classify_pred_res(0.8,gml_model)
boundary_parameter_0.2 <- get_boundary_line(gml_model,0.2,'Plasma_glucose')

ggplot(diabetes,aes( x = pred_res_0.2$Age, y = pred_res_0.2$Plasma_glucose,
         color = pred_res_0.2$predict)) + geom_point()+
geom_abline(slope = boundary_parameter_0.2$Age,
intercept = boundary_parameter_0.2$Intercept, color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by predicted diabetes Status")


ggplot(diabetes,aes( x = pred_res_0.2$Age, y = pred_res_0.2$Plasma_glucose,
                     color = pred_res_0.2$raw)) + geom_point()+
geom_abline(slope = boundary_parameter_0.2$Age,
          intercept = boundary_parameter_0.2$Intercept,
          color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by raw diabetes Status")
boundary_parameter_0.8 <- get_boundary_line(gml_model,0.8,'Plasma_glucose')

ggplot(diabetes,aes( x = pred_res_0.8$Age, y = pred_res_0.8$Plasma_glucose,
                     color = pred_res_0.8$predict)) + geom_point()+
geom_abline(slope = boundary_parameter_0.8$Age,
```

```
          intercept = boundary_parameter_0.8$Intercept,
          color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by predicted diabetes Status")


ggplot(diabetes,aes( x = pred_res_0.8$Age, y = pred_res_0.8$Plasma_glucose,
                     color = pred_res_0.8$raw)) + geom_point()+
geom_abline(slope = boundary_parameter_0.8$Age,
            intercept = boundary_parameter_0.8$Intercept,
            color = "blue", linetype = "dashed") +
labs(x = "Age", y = "Plasma Glucose Concentration", color = "Predicted Diabetes") +
ggtitle("Scatterplot of Plasma Glucose vs Age by raw diabetes Status")
diabetes$z1 <- diabetes$Plasma_glucose^4
diabetes$z2 <- diabetes$Plasma_glucose^3 * diabetes$Age
diabetes$z3 <- diabetes$Plasma_glucose^2 * diabetes$Age^2
diabetes$z4 <- diabetes$Plasma_glucose * diabetes$Age^3
diabetes$z5 <- diabetes$Age^4
formula <- Diabetes ~ Age + Plasma_glucose + z1 + z2 + z3 + z4 + z5
new_gml_model <- caret::train(formula, data = diabetes, method = "glm", family = "binomial")
new_pred_res <- classify_pred_res(0.5,new_gml_model)
new_diabetes_confusion <- table(new_pred_res$raw, new_pred_res$predict)
error_rate <- 1 - (sum(diag(new_diabetes_confusion)) / sum(new_diabetes_confusion))
cat(" training misclassification error:",error_rate)
new_boundary_parameter <- get_boundary_line(new_gml_model,0.5,'Plasma_glucose')
cat(new_gml_model$finalModel$coefficients)
```