Computer lab 1, group 29

Machine Learning, TDDE01

Oscar Hoffmann, oscho091
Martin Sörme, masor844
Hjalmar Öhman, hjaoh082

1. Statement of contribution

Hjalmar Öhman was responsible for assignment 1. In the second assignment, Oscar Hoffman was responsible and Martin Sörme was responsible for assignment 3. For each assignment this included the code writing and analysis. However, all members discussed the problems and took help from the other members in the tasks during the coding. After finishing the code writing and analysis, all three members sat down and discussed the problems and presented the solutions. This was done to ensure that all members understood the code, solutions and analyses for the tasks.

2. Assignments

Assignment 1 - Handwritten digit recognition with K-nearest neighbors.

Task 1.1

The optdigits.csv data was split into train, test and validation sets of 50%/25%/25% randomly. This was done by setting an “id” list of integers, selecting rows of the imported optdigits.csv data frame (*id=sample(1:n, floor(n*0.5)))*. Then these randomly generated rows were selected from the data and assigned to the training set (*train=data[id,]*). Similarly the rest of the data was split into set and validation. See appendix 1.1 for more detailed code.

Task 1.2

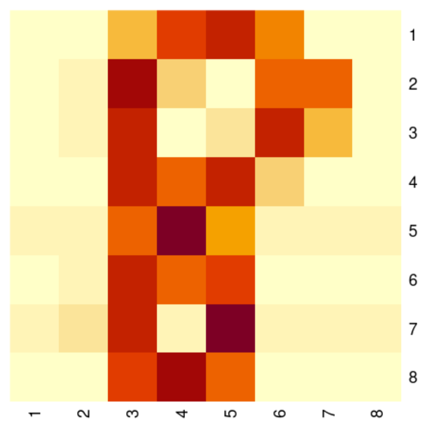
Then two models were created based on kknn. One with training as its test data set, and one with test as its test data set. *model_train* and *model_test* respectively.

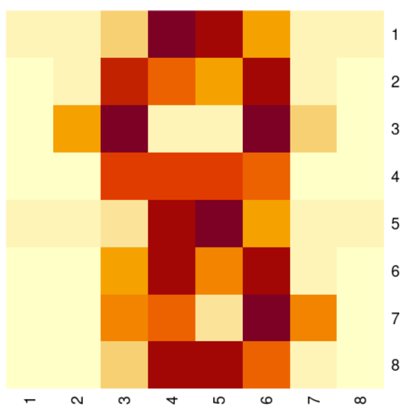
Predictions were made of the models on their respective test data and summarized into a confusion matrix. The overall prediction quality was quite similar for predictions on training data versus test data, around 95%. However, predictions made on training data were slightly better, about 1%. Simply put digits 1, 2, 7 and 9 were difficult to predict both on train and test data.

Task 1.3

The two cases where the model was most confident the digit was an eight and the three cases where it was the least difficult were extracted.

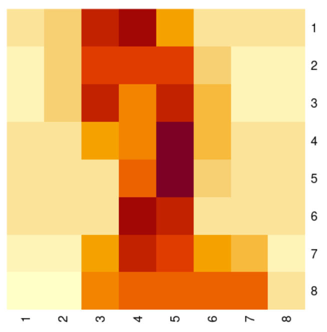
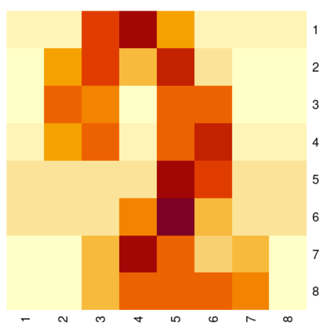
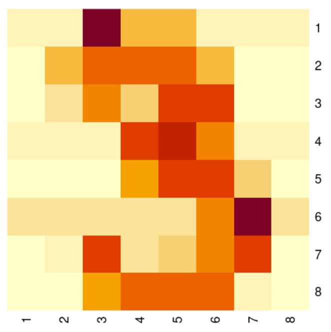
2 easiest cases:





They both are clearly eights, the second one more obvious than the first.

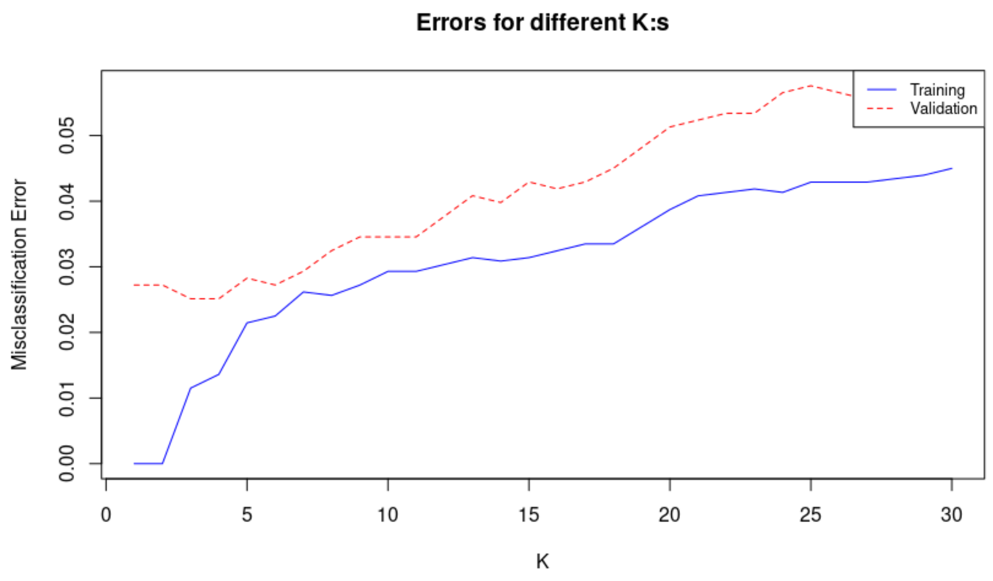
3 hardest cases:

All of these three are indeed difficult even for the human eye to see are eights. Looks more like 1,2,3.

Task 1.4

Misclassification error was plotted for different k values, shown by the graph below.



Complexity increases as K is increased, since there are more data points to take into regard when predicting the value. In this case it also leads to higher error rates, indicating that the model is getting overfitted for higher K:s.

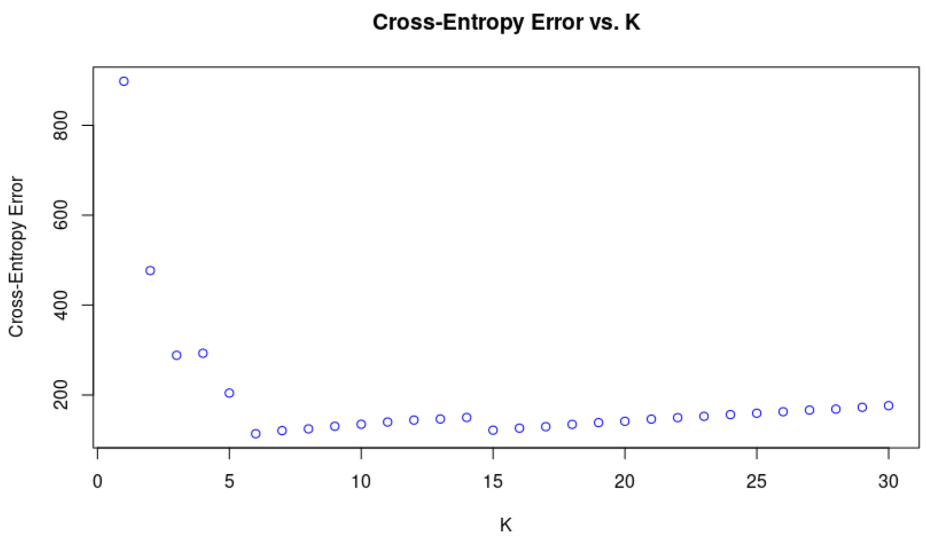
The optimal K for validation is 3. For K=3 the errors for the sets are:

Test Data:	0.02403344
Validation Data:	0.02513089
Training Data:	0.0115123

This shows that testing on the training data once again leads to a lower misclassification error, since that is what the model is trained on. However, tests and validation get a higher, about similar, error rate. This is reasonable as neither are separate for model training.

Task 1.5

The cross-entropy was plotted for different K:s, similarly as in Task 4.

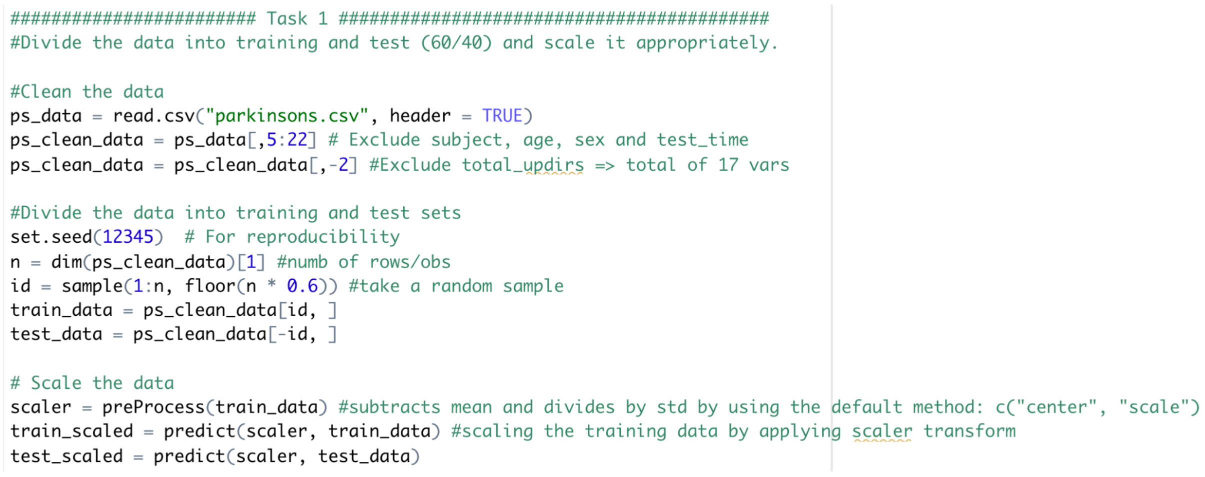


The optimal K here is 6. Cross-entropy might be a more suitable choice for this task of identifying digits, since it takes into account the model's confidence in its prediction, while misclassification rate regards a 99% incorrect classification the same as a 51% incorrect classification.

Assignment 2 - Linear regression and ridge regression

Task 2.1

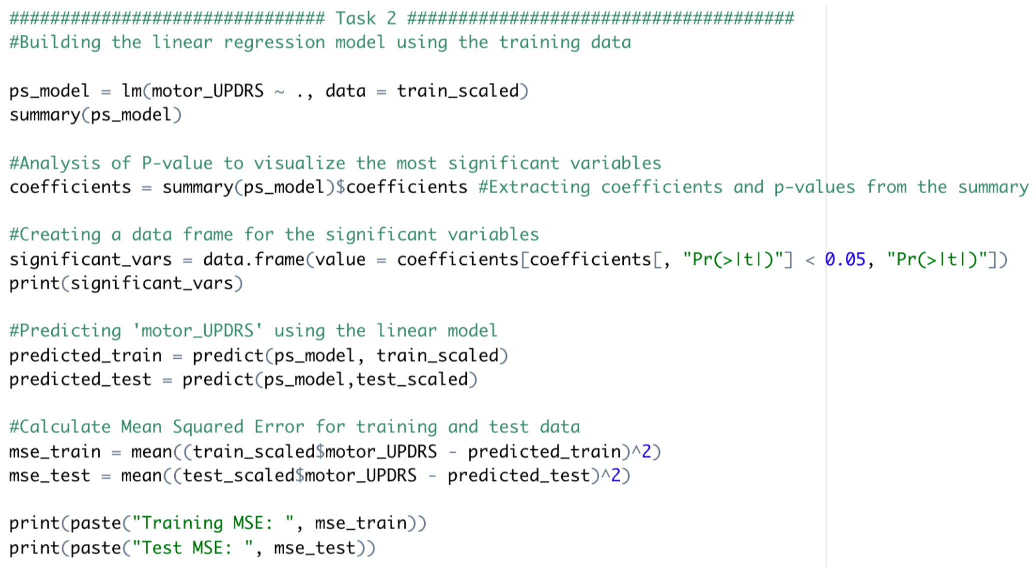
The first task was to import the “parkinsons.csv” file, divide it into training- and test data and scale it. The file was imported with the “read.csv()” function and scaled with the “preProcess()” function, which is a part of the external library “caret”. The preProcess() function uses the method = c("center", "scale")) by default, which subtracts the mean and divides by the standard deviation for each column variable in the csv file. This way the variables contribute equally to the analysis which is necessary for the model to perform well.



```r
###################### Task 1 ########################################
#Divide the data into training and test (60/40) and scale it appropriately.

#Clean the data
ps_data = read.csv("parkinsons.csv", header = TRUE)
ps_clean_data = ps_data[,5:22] # Exclude subject, age, sex and test_time
ps_clean_data = ps_clean_data[,-2] #Exclude total_updirs => total of 17 vars

#Divide the data into training and test sets
set.seed(12345)  # For reproducibility
n = dim(ps_clean_data)[1] #numb of rows/obs
id = sample(1:n, floor(n * 0.6)) #take a random sample
train_data = ps_clean_data[id, ]
test_data = ps_clean_data[-id, ]

# Scale the data
scaler = preProcess(train_data) #subtracts mean and divides by std by using the default method: c("center", "scale")
train_scaled = predict(scaler, train_data) #scaling the training data by applying scaler transform
test_scaled = predict(scaler, test_data)
```

Task 2.2

The second task was to compute a linear regression model and estimate MSE, as well as comment on the most significant variables.



```r
############################## Task 2 ####################################
#Building the linear regression model using the training data

ps_model = lm(motor_UPDRS ~ ., data = train_scaled)
summary(ps_model)

#Analysis of P-value to visualize the most significant variables
coefficients = summary(ps_model)$coefficients #Extracting coefficients and p-values from the summary

#Creating a data frame for the significant variables
significant_vars = data.frame(value = coefficients[coefficients[, "Pr(>|t|)"] < 0.05, "Pr(>|t|)"])
print(significant_vars)

#Predicting 'motor_UPDRS' using the linear model
predicted_train = predict(ps_model, train_scaled)
predicted_test = predict(ps_model,test_scaled)

#Calculate Mean Squared Error for training and test data
mse_train = mean((train_scaled$motor_UPDRS - predicted_train)^2)
mse_test = mean((test_scaled$motor_UPDRS - predicted_test)^2)

print(paste("Training MSE: ", mse_train))
print(paste("Test MSE: ", mse_test))
```

To achieve this a linear regression model was created with the “lm()” function. The model’s purpose was to predict the response variable “motor_UPDRS" based on the values of the predictors. The model summary provided coefficients and their P-vaules which were analyzed in order to comment on the significant variables. Small P-values (typically < 0.05) indicate strong evidence against the null hypothesis and in favor of significance. For these variables the null hypothesis can be rejected and it can be concluded that they contribute significantly to the model. To visualize this a separate table “significant_vars" was created where the variables with P-vaules < 0.05 were extracted. This can be seen below.

variable	**value**
Jitter.Abs.	3.316870e-05
Shimmer	4.054884e-03
Shimmer.APQ5	6.690868e-04
Shimmer.APQ11	6.365957e-07
NHR	4.849817e-05
HNR	6.450884e-11
DFA	6.566828e-43
PPE	6.750371e-12

To calculate the mean squared error we took the actual value of the response variable, which could be extracted from the “motor_UPDRS” column in the scaled training- and test data. We then subtracted the predicted value which was received from the model, squared it and took the mean. This measurement represents the average squared difference between the observed actual outcomes and the outcomes predicted by the model. In this case the MSE was ~0.879 for the training data and ~0.935 for the test data. The model has not seen the test data and since it is quite close to the training MSE it can be concluded that the model generalizes to new data quite well.

MSE not rounded:
Training MSE: 0.878543102826276
Test MSE: 0.935447712156708

Task 2.3

The third task was to implement four functions without any external packages. First, some global variables were defined that could be accessed by all the functions.

```r
#global variables for the functions
n = nrow(train_scaled) #Number of observations
y = as.matrix(train_scaled$motor_UPDRS) #The observed value in the motor_UPDRS column
x = as.matrix(train_scaled)[,2:17] #The predictor variables, columns Jitter... to PPE
```

The first function that was implemented was Log-likelihood.

```r
#Loglikelihood: how well does the model fit the data
loglikelihood <- function(theta, sigma) {
  return(-n / 2 * log(2 * pi * sigma^2) - (1 / (2 * sigma^2)) * sum((y - x %*% theta)^2))
}
```

After that the ridge function was implemented which call the loglikelihood function.

```r
#Ridge function
# Ridge penalty prevents overfitting by shrinking the coefficients
#=> less model complexity and improved generalization.
ridge_function = function(theta, lambda){
  sigma = theta[length(theta)]
  theta = theta[-17]
  # Calculate the Ridge penalty
  ridge_penalty = lambda * sum(theta^2)
  #call logLikelihood and add ridge_penalty to get ridge_value
  ridge_value = -loglikelihood(theta, sigma) + ridge_penalty
  return(ridge_value)
}
```

The third function optimized the theta and sigma variables. Optim() was fed a vector of 17 ones as an initial guess to start from.

```r
#Finds the parameter values that minimizes the loss function, starting from an initial guess.
RidgeOpt = function(lambda) {
  # Initial guesses for theta and sigma (17 ones)
  initial_guess = rep(1, 17)
  # Use optim to minimize the ridge regression loss function
  return(optim(initial_guess, fn = ridge_function, lambda = lambda, method = "BFGS"))
}
```

The last function that was implemented calculated the degrees of freedom.

```r
#Degree of freedom function
DF = function(lambda) {
  # Ensure the diagonal matrix I has the same dimensions as the number of predictors in x
  I = diag(ncol(x))
  # Compute the hat matrix H
  H = x %*% solve(t(x) %*% x + lambda * I) %*% t(x)
  # Calculate the degrees of freedom as the sum of the diagonal elements of H
  return(sum(diag(H)))
}
```

Task 2.4

In the last subtask the optimal values of theta were calculated using the ridgeOpt function for the given lambda values 1, 100 and 1000. The estimated parameters were then used to predict the response variable and calculate MSE for the training. and test data. This was implemented with a for loop that looped through a vector with the different lambda values.

```r
# Defining lambda values
lambda_values <- c(1, 100, 1000)

#converting into matrices
x_train <- as.matrix(train_scaled %>% select(Jitter...:PPE))
x_test <- as.matrix(test_scaled %>% select(Jitter...:PPE))

#Loop through the lambda vector
for (l in lambda_values) {
  #computing optimal theta coefficients
  theta_opt = RidgeOpt(lambda = l)
  #extracting optimal coefficients
  opt_extracted = as.matrix(theta_opt$par[-17])
  pred_train_opt = x_train %*% opt_extracted
  pred_test_opt  = x_test %*% opt_extracted
  #MSE for training and test
  mse_train_opt = mean((train_scaled$motor_UPDRS - pred_train_opt)^2)
  mse_test_opt  = mean((test_scaled$motor_UPDRS  - pred_test_opt)^2)
  #degrees of freedom for respective lambda value
  df = DF(lambda = l)
  #prints
  print(paste("lambda =",l,"MSE training:", mse_train_opt))
  print(paste("lambda =",l,"MSE training:", mse_test_opt))
  print(paste("lambda =",l,"Degrees of freedom:", df))
}
```

The optimal penalty parameter lambda is 100, as it yields the lowest test MSE, indicating the best predictive performance. With a degree of freedom around 9.925, this model balances complexity and generalization well. The chosen lambda minimizes overfitting, suggesting a model that's complex enough to capture underlying trends but not so much that it fits irrelevant noise.

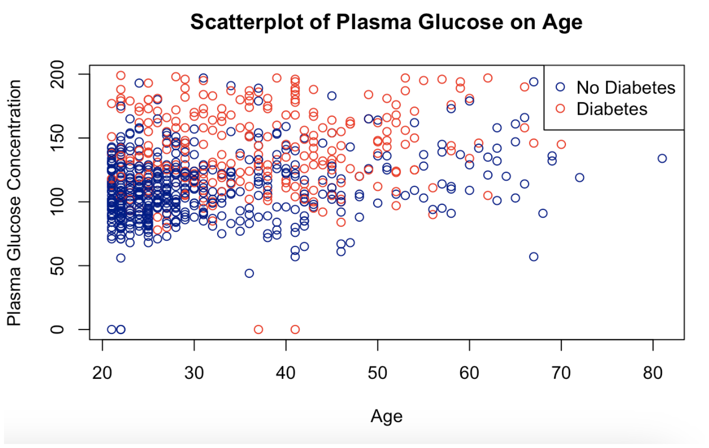
```
[1] "lambda = 1 MSE training: 0.878627189502917"
[1] "lambda = 1 MSE test: 0.934998015205606"
[1] "lambda = 1 Degrees of freedom: 13.8607362829965"
[1] "lambda = 100 MSE training: 0.884405556541252"
[1] "lambda = 100 MSE test: 0.932329601901113"
[1] "lambda = 100 Degrees of freedom: 9.92488712829542"
[1] "lambda = 1000 MSE training: 0.921117541108582"
[1] "lambda = 1000 MSE test: 0.953945694481502"
[1] "lambda = 1000 Degrees of freedom: 5.6439254878463"
```

Assignment 3 - Linear regression and ridge regression

Task 3.1

The “pima-indians-diabetes.csv” file consisted of data about medical details and the onset of diabetes in 5 years.

Firstly, the medical data “Age” and “Plasma glucose concentration” were shown in a scatter plot, as seen in the Figure to the right. In this figure, Red means diabetes and blue means no diabetes Based on this plot, classifying diabetes by a standard logistic regression model seems to be hard. There is no intuitive way to determine diabetes/no diabetes based on this, although a high plasma glucose concentration seems to be correlated to having diabetes.



Task 3.2

In the second task, a logistic regression model was trained using y = Diabetes as target, x1 = Plasma Glucose concentration, x2 = Age as features to make predictions for the observations using r = 0.5 as target. The probabilistic equation for this is the following as a result of the estimated model is the following:

$$P\,(Diabetes \;=\; 1) \;=\; \frac{1}{1+e^{-(\beta_0+\beta_{pgc}*pgc\,+\beta_{age}*age)}}$$

Where the values for β are derived from the following table, in the Estimate column:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.912449   0.462620  -12.78  < 2e-16 ***
pgc          0.035644   0.003290   10.83  < 2e-16 ***
age          0.024778   0.007374    3.36 0.000778 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

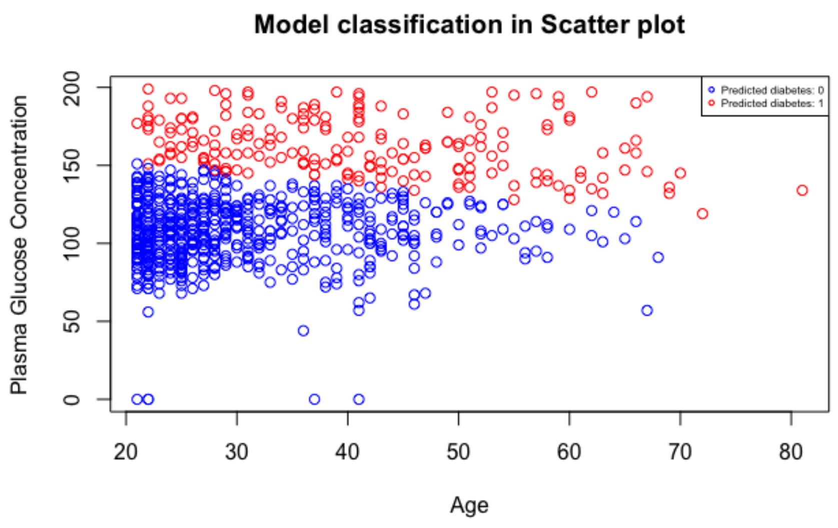
The table comes from the following code:

```
logistic_model <- glm(diabetes ~ pgc + age, data = dataframe, family = "binomial")

summary(logistic_model)
```

The training misclassification error can be derived from the following code:

```
prediction <- predict(logistic_model, dataframe, type = "response")
# r = 0.5
prediction1 <- ifelse(prediction > 0.5, 1, 0)

# Confusion matrix
confusion_matrix <- table(prediction1, diabetes)
print(confusion_matrix)

# Missclassification
misclass <- (1 - sum(diag(confusion_matrix)) / length(prediction1))
print(paste("Misclassification error:", misclass)) #0.2630208
```

The misclassification error from this model is roughly 0.26, which is quite high. It seems as if the model cannot accurately classify diabetes based on these two factors alone. Furthermore, we can plot the classifications that the model makes with the following plot.

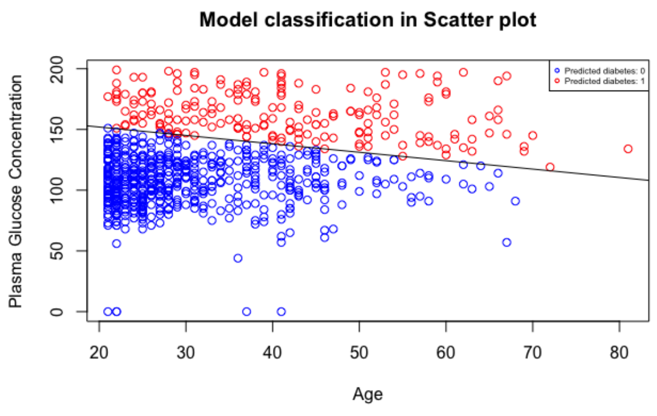


Task 3.3

Adding the curve showing this boundary can be done with the following code:

```
# Decision boundry line
abline(a = coef(logistic_model)[["(Intercept)"]] / (-coef(logistic_model)[["pgc"]]),
       b = coef(logistic_model)[["age"]] / (-coef(logistic_model)[["pgc"]]),
       col = "black")
```

Which results in this:



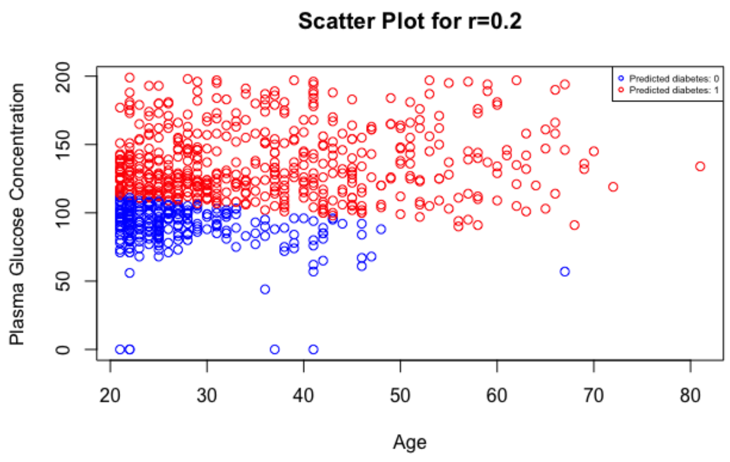
The decision boundary line can be observed. The equation for this is that coef(logistic_model)[["age"]] / (-coef(logistic_model)[["pgc"]]) is the slope and coef(logistic_model)[["(Intercept)"]] / (-coef(logistic_model)[["pgc"]] is the intercept.

Task 3.4

For the threshold 0.2:

```
# Prediction threshold r=0.2
prediction_2 <- ifelse(prediction > 0.2, 1, 0)

# r=0.2
plot(age, pgc, col = color_values[as.factor(prediction_2)],
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatter Plot for r=0.2")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes: 1"),
       col = c("blue", "red"), pch = 1, cex = 0.5)


# Missclassification for r=0.2
confusion_matrix2 <- table(prediction_2, diabetes)
print(confusion_matrix2)
## Comment: Frequent prediction of diabates when this is not the case

misclass2 <- (1 - sum(diag(confusion_matrix2)) / sum(confusion_matrix2))
print(paste("Misclassification error:", misclass2)) #0.37239583
```

The same is done but with a new threshold, which results in a worse misclassification rate of roughly 0.37. Plotting this we get the following:

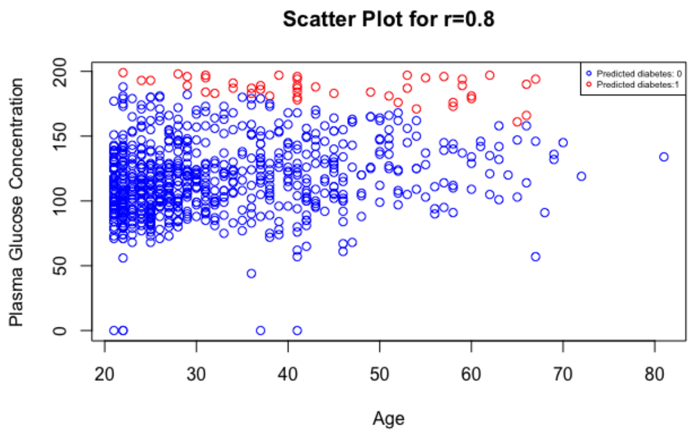


We can observe that the model now often predicts diabetes, compared to the model with the threshold of 0.5.

For the threshold 0.8:

```
# Prediction threshold r=0.8
prediction_3 <- ifelse(prediction > 0.8, 1, 0)

# r=0.8
plot(age, pgc, col = color_values[as.factor(prediction_3)],
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatter Plot for r=0.8")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
       col = c("blue", "red"), pch = 1, cex=0.5)



# Misclassification for r=0.8
confusion_matrix3 <- table(prediction_3, diabetes)
print(confusion_matrix3)
## Comment: Frequent missed prediction of diabetes
misclass3 <- (1 - sum(diag(confusion_matrix3)) / sum(confusion_matrix3))
print(paste("Misclassification error:", misclass3)) #0.31510416
```

This still results in a worse misclassification rate than the model with r=0.5, but a better one than r=0.2. The misclassification rate for this one is roughly 0.31. If we plot the predictions:



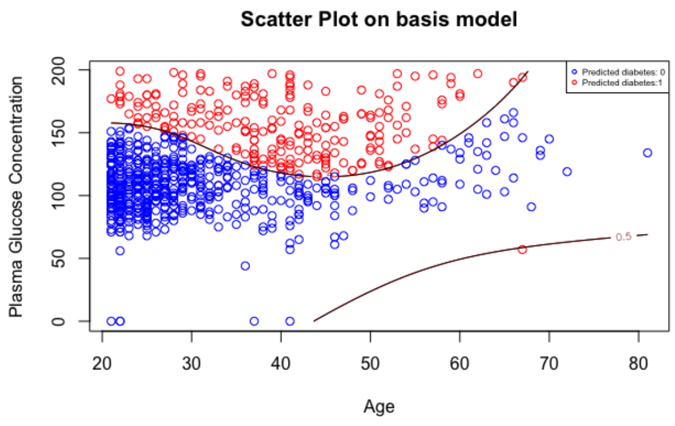
We can then observe that the model seldom predicts diabetes, which is due to the high threshold. When the r value is 0.2, the model will predict mostly diabetes=true, while it is 0.8 the model will predict mostly diabetes=false. Overall, both values lead to a high misclassification rate than using r=0.5.

Task 3.5

The basis function expansion trick is done by doing the following:

```
dataframe$z1 <- pgc^4
dataframe$z2 <- pgc^3 * age
dataframe$z3 <- pgc^2 * age^2
dataframe$z4 <- pgc * age^3
dataframe$z5 <- age^4
y <- diabetes

model <- glm(y ~ pgc + age + z1 + z2 + z3 + z4 + z5, data = dataframe, family = "binomial")
summary(model)

prediction_basis <- predict(model, dataframe, type = "response")
prediction_basis <- ifelse(prediction_basis > 0.5, 1, 0)

# Confusion Matrix and misclassification rate
cm_basis <- table(prediction_basis, y)
print(cm_basis)
misclass_basis <- (1 - sum(diag(cm_basis)) / sum(cm_basis))
print(paste("Misclassification error of new model:", misclass_basis))


plot(age, pgc, col = color_values[as.factor(prediction_basis)],
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatter Plot on basis model")

legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
       col = c("blue", "red"), pch = 1, cex=0.5)
```

The misclassification rate for this model is roughly 0.24, which is an improvement from the first model but still quite high. So the prediction accuracy is a bit higher than the first model. Plotting this with the new boundary line:



It can be observed that the shape of the boundary line has changed, but it still cannot accurately predict diabetes. A conclusion from this could be that predicting diabetes solely from age and plasma glucose concentration seems to be difficult.

Appendix

Assignment 1

```
#####TASK1#####
rm(list = ls())

data = read.csv("optdigits.csv", header = FALSE)
n = dim(data)[1]

set.seed(12345)
id = sample(1:n, floor(n * 0.5))

train = data[id, ]
rest = data[-id, ]

n2 = dim(rest)[1]
id2 = sample(1:n2, floor(n2 * 0.5))

test = rest[id2, ]
validation = rest[-id2, ]

rm(rest)




#####TASK2#####
library(kknn)

# Convert the outcome variable to a factor
train$V65 <- as.factor(train$V65)
test$V65 <- as.factor(test$V65)

# Fitting 30-nearest neighbor classifier on training data
model_train <- kknn(V65 ~ ., train = train, test = train, k = 30, kernel = "rectangular")

# Fitting 30-nearest neighbor classifier on test data
model_test <- kknn(V65 ~ ., train = train, test = test, k = 30, kernel = "rectangular")

# Making predictions on training and test data
prediction_train <- predict(model_train)
prediction_test <- predict(model_test)

# Confusion matrices
cm_train <- table(prediction_train, train$V65)
cm_test <- table(prediction_test, test$V65)
```

```
# Computing misclassification rates from confusion matrices
misclass_train <- 1 - sum(diag(cm_train)) / sum(cm_train)
misclass_test  <- 1 - sum(diag(cm_test)) / sum(cm_test)

# Displaying results
print("Confusion Matrix for Training Data:")
print(cm_train)

print("Confusion Matrix for Test Data:")
print(cm_test)

print(paste("Misclassification Rate for Training Data:", misclass_train))
print(paste("Misclassification Rate for Test Data:", misclass_test))

##Comment on the quality of predictions for different digits and on the overall
##prediction quality
#Ans: Overall good. ~95% accuracy. 0.045 vs. 0.048 miss class. Surprisingly similar
considering one had same data for test and train.
```

```
#####TASK3#####

# Extract probabilities guessing 8 for each image in the train data.
prob_8 <- model_train$prob[, 9]

# Easiest cases
easiest_cases <- order(prob_8, decreasing = TRUE)
easiest_cases <- easiest_cases[1:2]

# Hardest cases
hardest_cases <- order(prob_8)
hardest_cases <- hardest_cases[1:3]

# Extract the easiest and hardest features from the train data
easiest_cases_data <- train[easiest_cases, ]
hardest_cases_data <- train[hardest_cases, ]
```

```
# Remove last column and reshape into 8x8 numeric matrix
easiest_case_1_matrix <- matrix(as.numeric(easiest_cases_data[1, -65]), nrow = 8, ncol =
8)
easiest_case_2_matrix <- matrix(as.numeric(easiest_cases_data[2, -65]), nrow = 8, ncol =
8)

# Plot in heatmaps
heatmap(t(easiest_case_1_matrix), Colv = "Rowv", Rowv = NA)
heatmap(t(easiest_case_2_matrix), Colv = "Rowv", Rowv = NA)


# Remove last column and reshape into 8x8 numeric matrix
hardest_cases_1 <- matrix(as.numeric(hardest_cases_data[1, -65]), nrow = 8, ncol = 8)
hardest_cases_2 <- matrix(as.numeric(hardest_cases_data[2, -65]), nrow = 8, ncol = 8)
hardest_cases_3 <- matrix(as.numeric(hardest_cases_data[3, -65]), nrow = 8, ncol = 8)

# Plot in heatmaps
heatmap(t(hardest_cases_1), Colv = "Rowv", Rowv = NA)
heatmap(t(hardest_cases_2), Colv = "Rowv", Rowv = NA)
heatmap(t(hardest_cases_3), Colv = "Rowv", Rowv = NA)

##Comment on whether these cases seem to be hard or easy to recognize visually.
##Ans: First one pretty difficult. Second one easy. All the three "hardest cases" did indeed
seem difficult.




#####TASK4#####

# Create a sequence of K values from 1 to 30
key <- 1:30

# Initialize vectors to store misclassification errors for training and validation
missclass_errortr <- numeric(30)
missclass_errorv <- numeric(30)

# Loop over different values of K
for (i in key) {
  # Fit K-nearest neighbor classifier on training data
  nearest1 <- kknn(as.factor(V65) ~ ., train = train, test = train, k = i, kernel = "rectangular")

  # Calculate misclassification error for training data
  cm_nearest1 <- table(train$V65, predict(nearest1))
  missclass_error_train <- 1 - sum(diag(cm_nearest1) / sum(cm_nearest1))
  missclass_errortr[i] <- missclass_error_train
  # Fit K-nearest neighbor classifier on validation data
  nearest2 <- kknn(as.factor(V65) ~ ., train = train, test = validation, k = i, kernel =
"rectangular")

  # Calculate misclassification error for validation data
  cm_nearest2 <- table(validation$V65, predict(nearest2))
  missclass_error_valid <- 1 - sum(diag(cm_nearest2) / sum(cm_nearest2))
  missclass_errorv[i] <- missclass_error_valid
}

# Plot misclassification errors for training and validation
plot(key, missclass_errortr, ylab = "Misclassification Error", xlab = "K", col = "blue", type = "l",
lty = 1, ylim = c(0, max(missclass_errortr, missclass_errorv)), main = "Errors for different
K:s")
lines(key, missclass_errorv, col = "red", type = "l", lty = 2)
legend("topright", legend = c("Training", "Validation"), col = c("blue", "red"), lty = 1:2, cex =
0.8)

# Find the index where validation error is minimized
optimal_k_index <- which.min(missclass_errorv)
optimal_k <- key[optimal_k_index]

# Fit K-nearest neighbor classifier on training data with optimal K
optimal_nearest <- kknn(as.factor(V65) ~ ., train = train, test = test, k = optimal_k, kernel =
"rectangular")

# Calculate misclassification error for test data
cm_optimal <- table(test$V65, predict(optimal_nearest))
missclass_error_test <- 1 - sum(diag(cm_optimal) / sum(cm_optimal))


##How does the model complexity change when K increases and how does it affect the
training and validation errors?
##ANS:  Complexity increases when K increases, as the number of parameters increase.
##       In this case it also leads to higher error rates, indicating overfitting.


##Report the optimal K according to this plot.
cat("Optimal K:", optimal_k, "\n")
##ANS: 1

##Finally, estimate the test error for the model having the optimal K,
##compare it with the training and validation errors and make necessary conclusions about
the model quality.
cat("Misclassification Error for Training Data:", missclass_errortr[optimal_k], "\n")
cat("Misclassification Error for Validation Data:", missclass_errorv[optimal_k], "\n")
cat("Misclassification Error for Test Data:", missclass_error_test, "\n")
```

```
##ANS:  Around 97% accuracy for validation and test, which is good.
##       Testing on training data naturally gives 100% accuracy when k=1.




# TASK 5 #

key <- 1:30
entropy_error <- numeric(length = 30)

for (i in key) {
  # Fit K-nearest neighbor classifier on validation data
  nearest_valid <- kknn(as.factor(V65) ~ ., train = train, test = validation, k = i, kernel =
"rectangular")

  # Initialize variables for cross-entropy calculation
  cross_entropy_valid <- 0

  # Loop over digits (0 to 9)
  for (digit in 0:9) {
        # Extract true labels for the digit in the validation set
        true_valid <- validation$V65 == digit

        # Extract probabilities for the digit in the validation set ("digit+1" because vector is 1
indexed in $prob)
        prob_valid <- nearest_valid$prob[true_valid, digit + 1] + 1e-15

        # Calculate cross-entropy for validation data by += summary of all probabilities for
the digit.

        cross_entropy_valid <- cross_entropy_valid + sum(-log(prob_valid))
  }

  # Store the cross-entropy error for the given K
  entropy_error[i] <- cross_entropy_valid
}

# Plot the dependence of the validation error on the value of K
plot(key, entropy_error, ylab = "Cross-Entropy Error", xlab = "K", col = "blue", main =
"Cross-Entropy Error vs. K")

# Find the optimal K value
optimal_k_entropy <- which.min(entropy_error)
cat("Optimal K (based on cross-entropy):", optimal_k_entropy, "\n")
```

```
## Assuming that response has multinomial distribution,
## why might the cross-entropy be a more suitable choice
## of the error function than the misclassification error for this problem?
##ANS:  Misclassification rate doesn't take into account the probabilities of the model's
predictions.
```

Assignment 2

```
#libraries
library(caret)
library(dplyr)

# Clear global environment
rm(list = ls())

############################ Task 1
#############################################
#Divide the data into training and test (60/40) and scale it appropriately.

#Clean the data
ps_data = read.csv("parkinsons.csv", header = TRUE)
ps_clean_data = ps_data[,5:22] # Exclude subject, age, sex and test_time columns
ps_clean_data = ps_clean_data[,-2] #Exclude total_updirs => total of 17 vars

#Divide the data into training and test sets
set.seed(12345)  # For reproducibility
n = dim(ps_clean_data)[1] #numb of rows/obs
id = sample(1:n, floor(n * 0.6)) #take a random sample
train_data = ps_clean_data[id, ]
test_data = ps_clean_data[-id, ]

#Scale the data
scaler = preProcess(train_data) #subtracts mean and divides by std by using the default
method: c("center", "scale")
train_scaled = predict(scaler, train_data) #scaling the training data by applying scaler
transform
test_scaled = predict(scaler, test_data)

################################# Task 2
########################################
#Building the linear regression model using the training data
ps_model = lm(motor_UPDRS ~ . , data = train_scaled)
summary(ps_model)
```

```
#Analysis of P-value to visualize the most significant variables
coefficients = summary(ps_model)$coefficients #Extracting coefficients and p-values from
the summary

#Creating a data frame for the significant variables
significant_vars = data.frame(value = coefficients[coefficients[, "Pr(>|t|)"] < 0.05, "Pr(>|t|)"])
print(significant_vars)

#Predicting 'motor_UPDRS' using the linear model
predicted_train = predict(ps_model, train_scaled)
predicted_test = predict(ps_model,test_scaled)

#Calculate Mean Squared Error for training and test data
mse_train = mean((train_scaled$motor_UPDRS - predicted_train)^2)
mse_test = mean((test_scaled$motor_UPDRS - predicted_test)^2)

print(paste("Training MSE: ", mse_train))
print(paste("Test MSE: ", mse_test))

################################ Task 3
########################################
#global variables for the functions
n = nrow(train_scaled) #Number of observations
y = as.matrix(train_scaled$motor_UPDRS) #The observed value in the motor_UPDRS
column
x = as.matrix(train_scaled)[,2:17] #The predictor variables, columns Jitter... to PPE

#Loglikelihood: how well the model fits the data
loglikelihood = function(theta, sigma) {
  return(-n / 2 * log(2 * pi * sigma^2) - (1 / (2 * sigma^2)) * sum((y - x %*% theta)^2))
}

#Ridge function
# Ridge penalty prevents overfitting by shrinking the coefficients
#=> less model complexity and improved generalization.
ridge_function = function(theta, lambda){
  sigma = theta[length(theta)]
  theta = theta[-17]
  # Calculate the Ridge penalty
  ridge_penalty = lambda * sum(theta^2)
  #call logLikelihood and add ridge_penalty to get ridge_value
  ridge_value = -loglikelihood(theta, sigma) + ridge_penalty
  return(ridge_value)
}

#Finds the parameter values that minimizes the loss function, starting from an initial guess.
RidgeOpt = function(lambda) {
  # Initial guesses for theta and sigma (17 ones)
  initial_guess = rep(1, 17)
  # Use optim to minimize the ridge regression loss function
  return(optim(initial_guess, fn = ridge_function, lambda = lambda, method = "BFGS"))
}

#Degree of freedom function
DF = function(lambda) {
  # Ensure the diagonal matrix I has the same dimensions as the number of predictors in x
  I = diag(ncol(x))
  # Compute the hat matrix H
  H = x %*% solve(t(x) %*% x + lambda * I) %*% t(x)
  # Calculate the degrees of freedom as the sum of the diagonal elements of H
  return(sum(diag(H)))
}

########################################Task
4########################################
# Defining lambda values
lambda_values <- c(1, 100, 1000)

#converting into matrices
x_train <- as.matrix(train_scaled %>% select(Jitter...:PPE))
x_test <- as.matrix(test_scaled %>% select(Jitter...:PPE))

#Loop through the lambda vector
for (l in lambda_values) {
  #computing optimal theta coefficients
  theta_opt = RidgeOpt(lambda = l)
  #extracting optimal coefficients
  opt_extracted = as.matrix(theta_opt$par[-17])
  pred_train_opt = x_train %*% opt_extracted
  pred_test_opt  = x_test %*% opt_extracted
  #MSE for training and test
  mse_train_opt = mean((train_scaled$motor_UPDRS - pred_train_opt)^2)
  mse_test_opt  = mean((test_scaled$motor_UPDRS  - pred_test_opt)^2)
  #degrees of freedom for respective lambda value
  df = DF(lambda = l)
  #prints
  print(paste("lambda =",l,"MSE training:", mse_train_opt))
  print(paste("lambda =",l,"MSE training:", mse_test_opt))
  print(paste("lambda =",l,"Degrees of freedom:", df))
}
```

Assignment 3

```
dataframe <- read.csv('pima-indians-diabetes.csv', header = FALSE)
color_values <- c("blue", "red")
set.seed(12345)


#################################### TASK 1
####################################

# Independent variables
pgc <- dataframe$V2 # Plasma glucose concentration
age <- dataframe$V8

# Dependent variables
diabetes <- dataframe$V9


plot(age, pgc, col = ifelse(diabetes == 1, "red", "darkblue"),
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatterplot of Plasma Glucose on Age")

# Legend
legend("topright", legend = c("No Diabetes", "Diabetes"),
       col = c("darkblue", "red"), pch = 1)


#Do you think that Diabetes is easy to classify by a standard logistic regression model
#that uses these two variables as features? Motivate your answer.
#Motivation: No, not easy to classify. Hard to determine based on only these factors.
#if we look at the plot there is no intuitive way to determine diabetes/no diabetes


#################################### TASK 2
####################################

logistic_model <- glm(diabetes ~ pgc + age, data = dataframe, family = "binomial")

summary(logistic_model)

prediction <- predict(logistic_model, dataframe, type = "response")
# r = 0.5
prediction1 <- ifelse(prediction > 0.5, 1, 0)

# Confusion matrix
confusion_matrix <- table(prediction1, diabetes)
print(confusion_matrix)

# Missclassification
misclass <- (1 - sum(diag(confusion_matrix)) / length(prediction1))
```

```
print(paste("Misclassification error:", misclass)) #0.2630208


plot(age, pgc, col = color_values[as.factor(prediction1)],
    xlab = "Age", ylab = "Plasma Glucose Concentration",
    main = "Model classification in Scatter plot")

# Legend
legend("topright", c("Predicted diabetes: 0", "Predicted diabetes: 1"), col = c("blue", "red"),
pch = 1, cex = 0.5)



#################################### TASK 3
####################################

# Plot
plot(age, pgc, col = color_values[as.factor(prediction1)],
    xlab = "Age", ylab = "Plasma Glucose Concentration",
    main = "Scatter Plot")

# Decision boundry line
abline(a = coef(logistic_model)[["(Intercept)"]] / (-coef(logistic_model)[["pgc"]]),
      b = coef(logistic_model)[["age"]] / (-coef(logistic_model)[["pgc"]]),
      col = "black")


#################################### TASK 4
####################################

# Prediction threshold r=0.2
prediction_2 <- ifelse(prediction > 0.2, 1, 0)

# r=0.2
plot(age, pgc, col = color_values[as.factor(prediction_2)],
    xlab = "Age", ylab = "Plasma Glucose Concentration",
    main = "Scatter Plot for r=0.2")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes: 1"),
      col = c("blue", "red"), pch = 1, cex = 0.5)


# Missclassification for r=0.2
confusion_matrix2 <- table(prediction_2, diabetes)
print(confusion_matrix2)
## Comment: Frequent prediction of diabates when this is not the case

misclass2 <- (1 - sum(diag(confusion_matrix2)) / sum(confusion_matrix2))
print(paste("Misclassification error:", misclass2)) #0.37239583
```

```
# Prediction threshold r=0.8
prediction_3 <- ifelse(prediction > 0.8, 1, 0)

# r=0.8
plot(age, pgc, col = color_values[as.factor(prediction_3)],
    xlab = "Age", ylab = "Plasma Glucose Concentration",
    main = "Scatter Plot for r=0.8")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
      col = c("blue", "red"), pch = 1, cex=0.5)



# Misclassification for r=0.8
confusion_matrix3 <- table(prediction_3, diabetes)
print(confusion_matrix3)
## Comment: Frequent missed prediction of diabetes
misclass3 <- (1 - sum(diag(confusion_matrix3)) / sum(confusion_matrix3))
print(paste("Misclassification error:", misclass3)) #0.31510416


## When the r value is 0.2, the model will predict mostly diabetes=true,
## while it is 0.8 the model will predict mostly diabetes=false.
## Overall, both values lead to a high misclassification rate.


##################################### TASK 5
#####################################

dataframe$z1 <- pgc^4
dataframe$z2 <- pgc^3 * age
dataframe$z3 <- pgc^2 * age^2
dataframe$z4 <- pgc * age^3
dataframe$z5 <- age^4
y <- diabetes

model <- glm(y ~ pgc + age + z1 + z2 + z3 + z4 + z5, data = dataframe, family = "binomial")
summary(model)

prediction_basis <- predict(model, dataframe, type = "response")
prediction_basis <- ifelse(prediction_basis > 0.5, 1, 0)

# Confusion Matrix and misclassification rate
cm_basis <- table(prediction_basis, y)
print(cm_basis)
misclass_basis <- (1 - sum(diag(cm_basis)) / sum(cm_basis))
print(paste("Misclassification error of new model:", misclass_basis))
```

```
plot(age, pgc, col = color_values[as.factor(prediction_basis)],
    xlab = "Age", ylab = "Plasma Glucose Concentration",
    main = "Scatter Plot on basis model")

legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
      col = c("blue", "red"), pch = 1, cex=0.5)

# Add the decision boundary line
x_vals <- seq(min(age), max(age), length.out = 100)
y_vals <- seq(min(pgc), max(pgc), length.out = 100)

# Expanded grid with basis functions
new_data <- expand.grid(age = x_vals, pgc = y_vals)
new_data$z1 <- new_data$pgc^4
new_data$z2 <- new_data$pgc^3 * new_data$age
new_data$z3 <- new_data$pgc^2 * new_data$age^2
new_data$z4 <- new_data$pgc * new_data$age^3
new_data$z5 <- new_data$age^4

# Add the decision boundary
contour(x = x_vals, y = y_vals, z = matrix(predict(model, newdata = new_data, type =
"response"), ncol = length(y_vals)), levels = 0.5, add = TRUE, col = "black")
```