Lab01 Assignment1

Group 20

Statement Of Contribution:

Assignment 1: NI Dongwei
Assignment 2: YAN Jin
Assignment 3: Collins Klinsmann Osondu, YAN Jin, NI Dongwei

Assignment 1. Handwritten digit recognition with Knearest neighbors.

1. Import the data into R and divide it into training, validation and test sets(50%/25%/25%) by using the partitioning principle specified in the lecture slides.

```
dim(train)
```

```
## [1] 1911   65
```

```
dim(valid)
```

```
## [1] 955  65
```

```
dim(test)
```

```
## [1] 957  65
```

2.Use training data to fit 30-nearest neighbor classifier with function kknn() and kernel=”rectangular” from package kknn and estimate

- Confusion matrices for the training and test data (use table())
- Misclassification errors for the training and test data

Comment on the quality of predictions for different digits and on the overall prediction quality.

- train:

```
optdigits_kknn_train_k30 <- kknn(as.factor(V65)~., train, train, k=30,
                                 kernel="rectangular")
fit <- fitted(optdigits_kknn_train_k30)
Misclass( fit, train$V65)
```

```
## Classification table:
##     obs
## pred   0   1   2   3   4   5   6   7   8   9
##    0 184   0   0   0   0   0   0   0   0   2
##    1   0 181   2   0   2   0   2   2  14   5
##    2   0   8 177   0   0   0   0   0   0   0
##    3   0   0   0 197   0   1   0   1   1   1
##    4   1   0   0   0 173   0   0   0   0   3
##    5   0   0   0   1   0 182   0   0   0   0
##    6   0   0   0   0   1   0 183   0   2   0
##    7   0   1   1   3   9   0   0 206   0   5
```

```
##    8   0   1   0   1   1   0   0   0 166   0
##    9   0   3   1   2   3   9   0   1   2 170
## Misclassification errors (%):
##    0    1    2    3    4    5    6    7    8    9
##  0.5  6.7  2.2  3.4  8.5  5.2  1.1  1.9 10.3  8.6
## Mean misclassification error: 4.8%
```

- test:

```
optdigits_kknn_test_k30 <- kknn(as.factor(V65)~., train, test, k=30,
                                kernel="rectangular")
fit <- fitted(optdigits_kknn_test_k30)
Misclass( fit, test$V65)
```

```
## Classification table:
##     obs
## pred   0   1   2   3   4   5   6   7   8   9
##    0  92   0   0   0   0   0   0   0   0   0
##    1   0  91   0   0   2   0   1   1   3   4
##    2   0   4 109   0   0   0   0   0   0   0
##    3   0   1   0  93   0   1   0   0   0   4
##    4   0   0   0   0  83   0   0   0   0   0
##    5   0   0   0   1   0  89   0   1   0   0
##    6   1   0   0   0   2   0  99   0   1   0
##    7   0   0   1   0   3   0   0  90   0   1
##    8   0   0   0   0   2   0   0   0  84   0
##    9   0   0   0   0   2   5   0   0   1  85
## Misclassification errors (%):
##    0    1    2    3    4    5    6    7    8    9
##  1.1  5.2  0.9  1.1 11.7  6.3  1.0  2.2  5.6  9.6
## Mean misclassification error: 4.5%
```

- for 0, 2, 6, 7 : best quality, at most 2.2% misclassification error for both train and test
- for 4, 9 : worst quality, at least 8.5% misclassification error for both train and test
- overall prediction quality: 4.8% mean misclassification error for train and 4.5% for test. Acceptable error rate(as a lab assignment) and differences.

3. Find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class). Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits (by using e.g. heatmap() function with parameters Colv=NA and Rowv=NA) and comment on whether these cases seem to be hard or easy to recognize visually.

```
originID <- which( train$V65 == 8)
# original id in train set

prob_train_k30 <- cbind(ID = originID,
                        optdigits_kknn_train_k30$prob[originID,]
                        ) %>%
  as.data.frame() %>%
  arrange(`8`)
#sort out the probabilities of observation 8 and arrange increasingly


head(prob_train_k30, 3L)
```

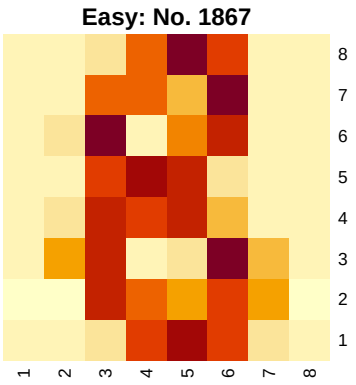
```
##     ID 0         1 2 3 4 5         6 7         8 9
## 1  427 0 0.8666667 0 0 0 0 0.0000000 0 0.1333333 0
## 2 1480 0 0.0000000 0 0 0 0 0.8666667 0 0.1333333 0
## 3 1033 0 0.8333333 0 0 0 0 0.0000000 0 0.1666667 0
```

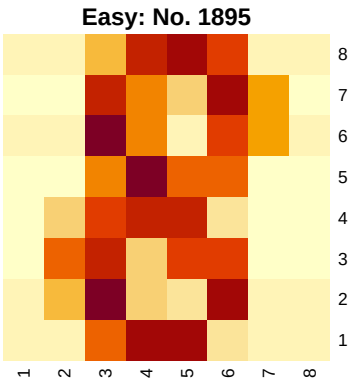
```
# hardest 427, 1480, 1033

tail(prob_train_k30, 2L)
```

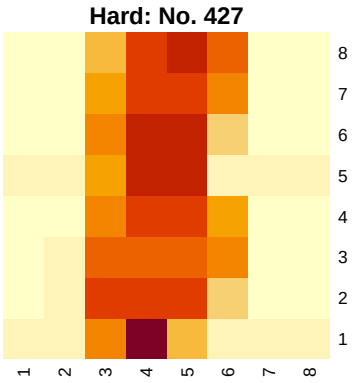
```
##       ID 0 1 2 3 4 5 6 7 8 9
## 184 1867 0 0 0 0 0 0 0 0 1 0
## 185 1895 0 0 0 0 0 0 0 0 1 0
```

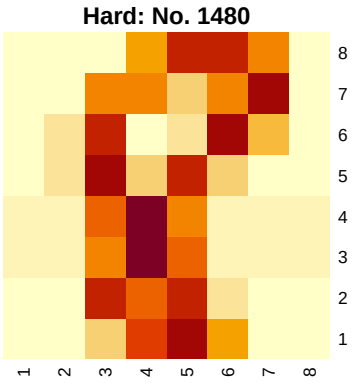
```
# easiest 1867, 1895
```

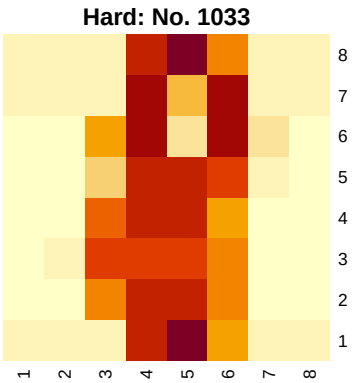
```
easy <- c(1867, 1895)
hard <- c(427, 1480, 1033)

for (i in c(easy, hard)) {
  if (any(i == easy) ) {
    title <- paste("Easy: No.", i)
  }else {
    title <- paste("Hard: No.", i)
  }

  heatmap(
    matrix(as.numeric(train[i,-65]), nrow = 8, ncol = 8, byrow = TRUE),
    Colv = NA, Rowv = NA, main = title
    )
}
```











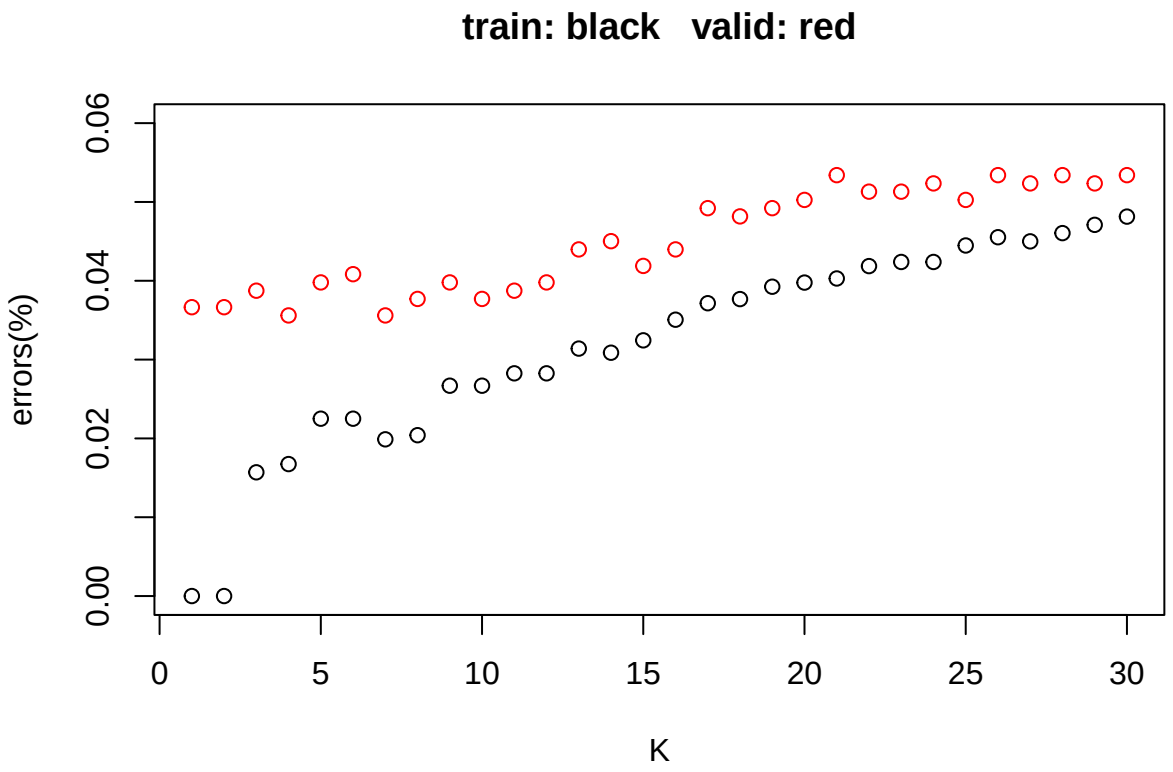
- visually the heatmap gives human the same difficulty to recognize as to our model , its clear and easy to recognize for human for those easiest-to-classify figures and similarly it's hard for human to recognize for the hardest ones.

4. Fit a K-nearest neighbor classifiers to the training data for different values of K = 1,2, . . . , 30 and plot the dependence of the training and validation misclassification errors on the value of K (in the same plot). How does the model complexity change when K increases and how does it affect the training and validation errors? Report the optimal K according to this plot. 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.

```
Err_train <- vector()
Err_valid <- vector()

for (i in 1:30) {
  optdig_kknn_train_1loop30 <- kknn(as.factor(V65)~., train, train, k = i ,
                                    kernel = "rectangular")
  fit <- fitted(optdig_kknn_train_1loop30)
  cfsMtx <- table(fit, train$V65)
  Err_train[i] <-( sum(cfsMtx)- sum(diag(cfsMtx))) / sum(cfsMtx)


  optdig_kknn_valid_1loop30 <- kknn(as.factor(V65)~., train, valid, k = i ,
                                    kernel = "rectangular")
  fit <- fitted(optdig_kknn_valid_1loop30)
  cfsMtx <- table(fit, valid$V65)
  Err_valid[i] <-( sum(cfsMtx)- sum(diag(cfsMtx))) / sum(cfsMtx)
```

```
}
```

```
plot(Err_train, main = "train: black   valid: red", ylim = c(0, 0.06),
     ylab = "errors(%)", xlab = "K")
points(Err_valid, col = "red")
```



- From this plot, we can learn that when K value is close to zero, the misclassification errors of training set should not be taken into consideration. e.g., when K == 1, the 1-nearest-neighbour is exactly each point itself, and of course there will be no misclassification errors.
  For this reason, K = 4 may possibly be the optimal value.
- Generally , the misclassification errors rises when K rises, and the errors of train itself is always lower than valid

```
optdigits_kknn_test_k4 <- kknn(as.factor(V65)~., train, test, k = 4,
                               kernel="rectangular")
fit <- fitted(optdigits_kknn_test_k4)
Misclass( fit, test$V65)
```

```
## Classification table:
##     obs
## pred   0   1   2   3   4   5   6   7   8   9
##    0  93   0   0   0   0   0   0   0   0   0
##    1   0  96   0   0   0   1   0   1   1   1
##    2   0   0 110   0   0   0   0   0   0   0
```

```
##    3   0   0   0  90   0   0   0   0   0   2
##    4   0   0   0   0  91   0   0   0   0   1
##    5   0   0   0   1   0  92   0   0   0   0
##    6   0   0   0   0   1   0 100   0   0   0
##    7   0   0   0   0   1   0   0  90   0   1
##    8   0   0   0   0   0   0   0   0  86   1
##    9   0   0   0   3   1   2   0   1   2  88
## Misclassification errors (%):
##   0   1   2   3   4   5   6   7   8   9
## 0.0 0.0 0.0 4.3 3.2 3.2 0.0 2.2 3.4 6.4
## Mean misclassification error: 2.3%
```

```
Err_train[4]
```

```
## [1] 0.01674516
```

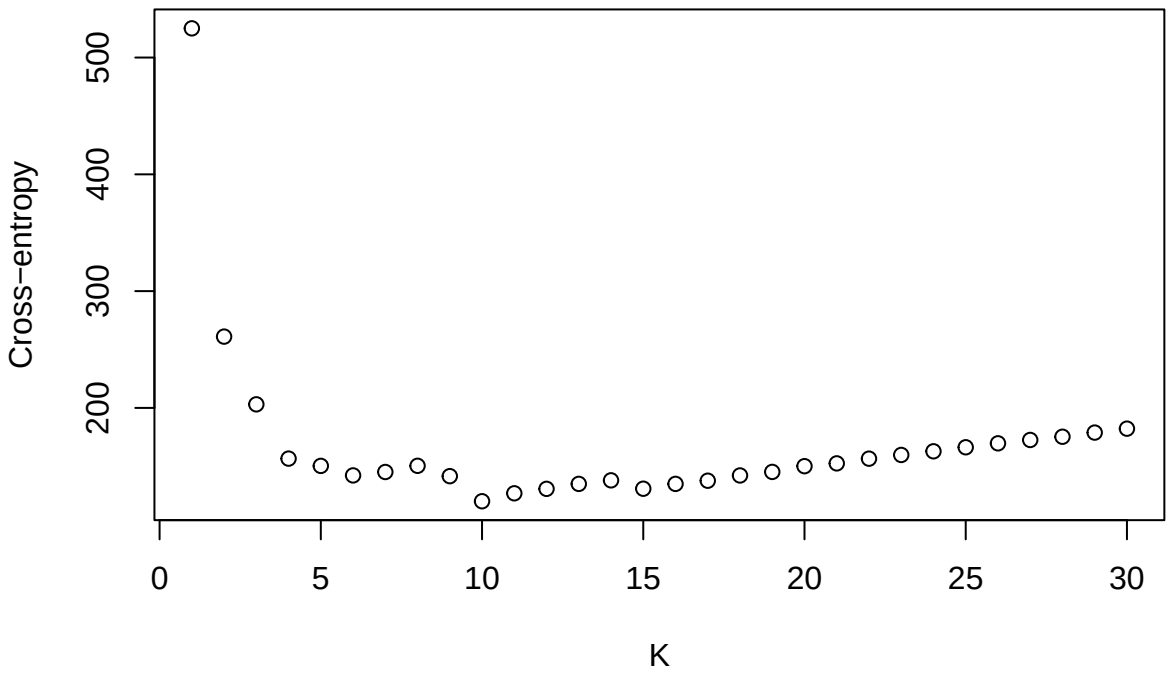
```
Err_valid[4]
```

```
## [1] 0.03560209
```

- When K = 3, the misclassification error rate of training set is 1.67%, and 3.56% for valid, 2.3% for test

5. Fit K-nearest neighbor classifiers to the training data for different values of K = 1,2, . . . , 30, compute the error for the validation data as cross-entropy (when computing log of probabilities add a small constant within log, e.g. 1e-15, to avoid numerical problems) and plot the dependence of the validation error on the value of K. What is the optimal K value here? 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?

```
H <- 0

for (i in 1:30) {
  optdigits_kknn_valid_q5 <- kknn(as.factor(V65)~., train,valid, k = i ,
                                  kernel = "rectangular")
  fit <- fitted(optdigits_kknn_valid_q5)
  prob_valid <- predict(optdigits_kknn_valid_q5, valid, type = 'prob')

  H[i] <- 0
  for (j in 1:dim(valid)[1]) {
    g <- prob_valid[j, 1 + valid[j, 65]]
    names(g) <- NULL

    H[i] <- H[i] + (-log(g + exp(-15)))
  }
}

plot(H, xlab = "K", ylab = "Cross-entropy")
```



- K == 10 is the optimal value due to the plot.
  Misclassification error we are calculating consider things purely under a binary context, i.e., the prediction is correct or is not correct. While cross-entropy also take the possibility into consideration. It not only affected by how wrongly you are, but also affected by how wrongly your wrongs are. An almost correct prediction will considered the same as a completely wrong prediction under misclassification error, but cross-entropy can show the differences.

Lab01 Assignment2

Question 1

Divide it into training and test data (60/40) and scale it appropriately. In the coming steps, assume that motor_UPDRS is normally distributed and is a function of the voice characteristics, and since the data are scaled, no intercept is needed in the modelling.

The code is given in Appendix.

Question 2

Compute a linear regression model from the training data, estimate training and test MSE and comment on which variables contribute significantly to the model.

Training MSE is 0.87854 Test MSE is 0.93945

When we use summary function we can find that some P values of parameters are marked with three stars. That means these parameters contribute most to the result. They are shown in the picture in Appendix.They are respectively Jitter.Abs Shimmer.APQ5 Shimmer.APQ11 NHR HNR DFA PPE

Question 3

Implementing 4 functions, which are "Loglikelihood function", " Ridge function", "RidgeOpt function", "DF function".

Four functions are given in Appendix.

We create the "Loglikelihood function" based on the following formula: $\ln p(\mathbf{y}|\mathbf{X};\theta) = -\frac{n}{2}\ln(2\pi\sigma_\epsilon^2) - \frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^{n}(\theta^{\mathrm{T}}\mathbf{x}_i - y_i)^2$

If we want to create "Ridge function", we need to add $\lambda||\theta||_2^2$ to the last formula.

Question 4

By using function RidgeOpt, compute optimal θ parameters for λ= 1, λ = 100 and λ = 1000. Use the estimated parameters to predict the motor_UPDRS values for training and test data and report the training and test MSE values. Which penalty parameter is most appropriate among the selected ones? Compute and compare the degrees of freedom of these models and make appropriate conclusions.

λ	$Training\_MSE$	$Test\_MSE$	DF
1	0.8786	0.9350	14.86
100	0.8844	0.9323	10.90
1000	0.9211	0.9539	6.42

By comparison we found that as the value of the λ increases, the value of DF decreases. At the same time, in general the predicted results would be less and less accurate. We also noticed that the optimum λ is 100, which means in order to get the relatively less errors, we should not use the extreme values of λ.

Appendix

```
my_data = read.csv("parkinsons.csv")
set.seed(12345)
n = nrow(my_data)
id = sample(1:n, floor(n*0.6))
train = my_data[id,]
test = my_data[-id,]


scaler = preProcess(train)
trainS = predict(scaler, train)
testS = predict(scaler, test)


# train MSE
trainS_1 = trainS %>% select(motor_UPDRS, 7:22)# what should we do if name of one column contains #
ml = lm(motor_UPDRS~.,trainS_1)
summary(ml)
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ ., data = trainS_1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0255 -0.7363 -0.1087  0.7333  2.1960
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.575e-15  1.583e-02   0.000 1.000000
## Jitter...      1.869e-01  1.496e-01   1.250 0.211496
## Jitter.Abs.   -1.696e-01  4.081e-02  -4.156 3.32e-05 ***
## Jitter.RAP    -5.270e+00  1.884e+01  -0.280 0.779688
## Jitter.PPQ5   -7.457e-02  8.778e-02  -0.850 0.395659
## Jitter.DDP     5.250e+00  1.884e+01   0.279 0.780541
## Shimmer        5.924e-01  2.060e-01   2.876 0.004055 **
## Shimmer.dB.   -1.727e-01  1.393e-01  -1.239 0.215380
## Shimmer.APQ3   3.207e+01  7.717e+01   0.416 0.677738
## Shimmer.APQ5  -3.875e-01  1.138e-01  -3.405 0.000669 ***
## Shimmer.APQ11  3.055e-01  6.124e-02   4.989 6.37e-07 ***
## Shimmer.DDA   -3.239e+01  7.717e+01  -0.420 0.674739
## NHR           -1.854e-01  4.557e-02  -4.068 4.85e-05 ***
## HNR           -2.385e-01  3.640e-02  -6.553 6.45e-11 ***
## RPDE           4.068e-03  2.267e-02   0.179 0.857576
## DFA           -2.803e-01  2.014e-02 -13.919  < 2e-16 ***
## PPE            2.265e-01  3.289e-02   6.886 6.75e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9396 on 3508 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.24 on 16 and 3508 DF,  p-value: < 2.2e-16
```

```r
Preds = predict(ml)
MSE_training = mean((trainS_1$motor_UPDRS - Preds)^2)

# test MSE
testS_1 = testS %>% select(motor_UPDRS, 7:22)
Preds_2 = predict(ml,testS_1)


MSE_test = mean((testS_1$motor_UPDRS - Preds_2)^2)



# The third question.

#a
n <- nrow(trainS)
trainS_1 <- tibble(trainS_1)
one <- tibble(one = rep(1,n))

trainS_1_new <- trainS_1%>%
  select(-motor_UPDRS)%>%
  mutate(one, .)


likelihood <- function(xita,sigma){
  n <- nrow(trainS)
  y_hat <- difference <- as.matrix(trainS_1_new) %*% as.matrix(xita)
  difference <- trainS_1$motor_UPDRS - y_hat
  square_difference <- difference^2
  sum <- sum(square_difference)
  return((-n/2) * log(2 * pi * sigma^2 ) - (1 / (2 * sigma^2)) * sum)

}

#b
Ridge <- function(lambda, xita, sigma){
  -likelihood(xita,sigma) + lambda * sum(xita^2)
}


#c
RidgeOpt <- function(lambda){

  to_optimize <- function(x){
    x1 = x[1:17]
    x2 = x[18]
    return(Ridge(lambda, x1, x2))
  }
  optim(rep(1, times = 18), fn = to_optimize, method = "BFGS")
}
```

```r
#d
DF <- function(lambda){
  X <- as.matrix(trainS_1_new)
  I <- diag(ncol(trainS_1))
  element_I <- diag(X %*% solve((t(X) %*% X + lambda * I)) %*% t(X))
  df <- sum(element_I)
  return(df)
}
df_1 <- DF(1)
df_100 <- DF(100)
df_1000 <- DF(1000)

#e

#when lambda equals 1, theta
theta_sigma_1 <- RidgeOpt(1)
theta_1 <- theta_sigma_1$par[1:17]

##MSE for training data
y_hat_tr_1 <- as.matrix(trainS_1_new) %*% as.matrix(theta_1)
MSE_train_1 <-mean((y_hat_tr_1 - trainS_1$motor_UPDRS)^2)

##MSE for test data
n_2 <- nrow(testS)
testS_1 <- tibble(testS_1)
one_2 <- tibble(one = rep(1,n_2))

testS_1_new <- testS_1%>%
  select(-motor_UPDRS)%>%
  mutate(one_2, .)

y_hat_test_1 <- as.matrix(testS_1_new) %*% as.matrix(theta_1)
MSE_test_1 <- mean((y_hat_test_1 - testS_1$motor_UPDRS)^2)




#when lambda equals 100, theta
theta_sigma_2 <- RidgeOpt(100)
theta_2 <- theta_sigma_2$par[1:17]


## MSE for training data

y_hat_tr_2 <- as.matrix(trainS_1_new) %*% as.matrix(theta_2)
MSE_train_2 <-mean((y_hat_tr_2 - trainS_1$motor_UPDRS)^2)

## MSE for test data
y_hat_test_2 <- as.matrix(testS_1_new) %*% as.matrix(theta_2)
MSE_test_2 <- mean((y_hat_test_2 - testS_1$motor_UPDRS)^2)
```

```r
#when lambda equals 1000, theta
theta_sigma_3 <- RidgeOpt(1000)
theta_3 <- theta_sigma_3$par[1:17]

## MSE for training data

y_hat_tr_3 <- as.matrix(trainS_1_new) %*% as.matrix(theta_3)
MSE_train_3 <-mean((y_hat_tr_3 - trainS_1$motor_UPDRS)^2)

## MSE for test data
y_hat_test_3 <- as.matrix(testS_1_new) %*% as.matrix(theta_3)
MSE_test_3 <- mean((y_hat_test_3 - testS_1$motor_UPDRS)^2)
```

Lab01 Assignment 3

Assignment 3. Logistic regression and basis function expansion

The data file **pima-indians-diabetes.csv** contains information about the onset of diabetes within 5 years in Pima Indians given medical details. The variables are (in the same order as in the dataset):

1. Number of times pregnant.
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skinfold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index (weight in kg/(height in m)^2).
7. Diabetes pedigree function.
8. Age (years).
9. Diabetes (0=no or 1=yes)

1.Make a scatterplot showing a Plasma glucose concentration on Age where observations are colored by Diabetes levels. 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.

```
data_PIdiabetes <- read.csv("pima-indians-diabetes.csv", col.names = c(1:9))

data_PIdiabetes$X9 <- factor(data_PIdiabetes$X9)

ggplot(data = data_PIdiabetes, aes(x = X8, y = X2, colour = X9)) +
  geom_point() +
  labs(x = "Age (years)", y = "Plasma glucose concentration",
       colour  = "Diabetes (0=no or 1=yes)" )
```

- Yes! Motivation: Looking at the plot, one can see that the plasma glucose concentration is more higher for the group with Diabetes level 1 compared to those with level 0. This is more evident when plasma glucose concentration is 150 or higher. At such one can conclude that it will be easy to classify Diabetes by a standard logistic regression

2.Train a logistic regression model with y = Diabetes as target x_1 = Plasma glucose concentration and x_2 = Age as features and make a prediction for all observations by using $r = 0.5$ as the classification threshold. Report the probabilistic equation of the estimated model (i.e., how the target depends on the features and the estimated model parameters probabilistically). Compute also the training misclassification error and make a scatter plot of the same kind as in step 1 but showing the predicted values of Diabetes as a color instead. Comment on the quality of the classification by using these results

- Logistic regression analysis belongs to the class of generalized linear models. In R generalized linear models are handled by the glm() function. The function is written as glm(response ~ predictor, family = binomial(link = "logit"), data). Please note that logit is the default for binomial; thus, we do not have to type it explicitly. The glm() function returns a model object, therefore we may apply extractor functions, such as summary(), fitted() or predict, among others, on it. However, please note that the output numbers are on the logit scale. To actually predict probabilities we need to provide the predict() function an additional argument type = "response"

```
# X2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
# X8. Age (years).
# X9. Diabetes (0=no or 1=yes).

glm_diabete <- glm(X9 ~ X2 + X8, data_PIdiabetes, family = binomial)

fit <- ifelse(fitted(glm_diabete) >= 0.5, 1, 0)
```

```
glm_diabete$coefficients
```

```
## (Intercept)          X2          X8
## -5.89785793  0.03558250  0.02450157
```

- we have

$$g(x) = \frac{1}{1 + e^{-\theta^T x}}$$

and use coefficients we got:

$$g(x) = \frac{1}{1 + e^{-(-5.89785793 + 0.03558250 x_1 + 0.02450157 x_2)}}$$

while x_1 = Plasma glucose concentration and x_2 = age.

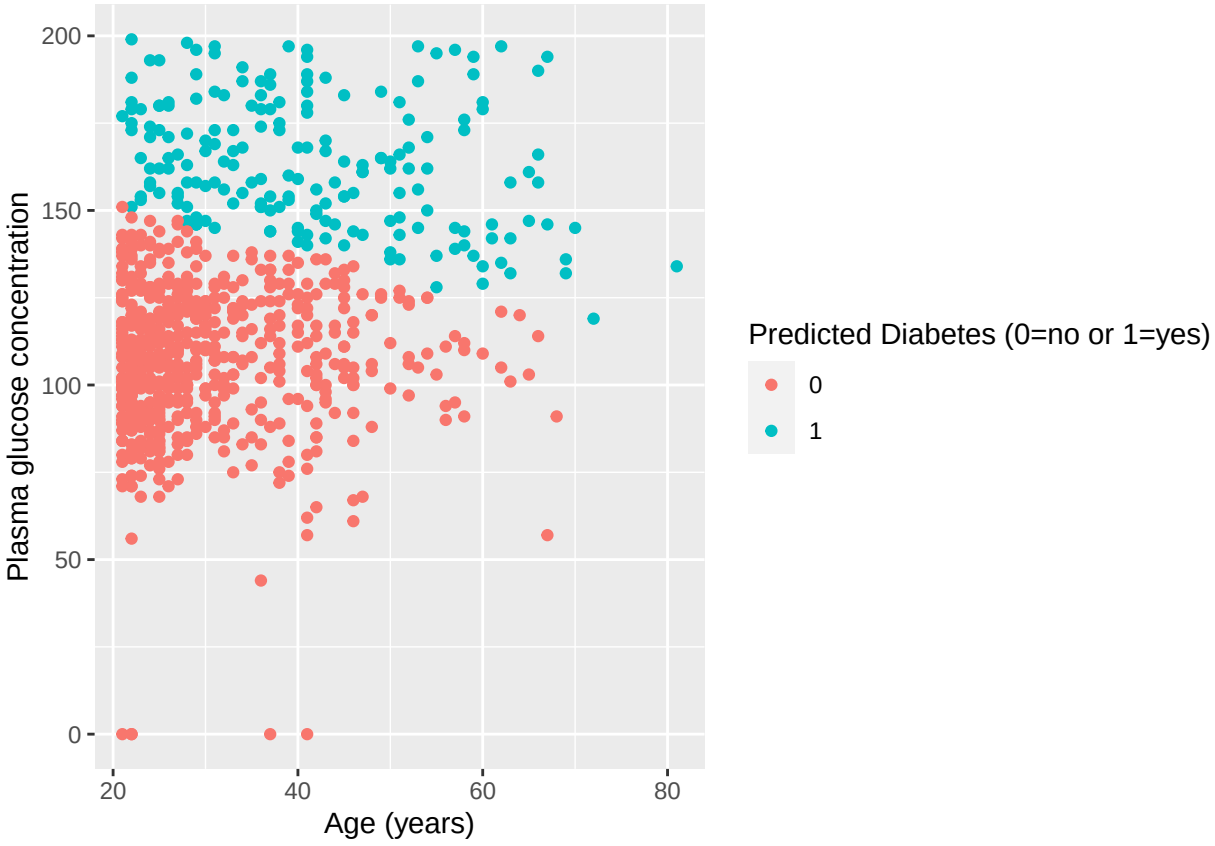
```
tq1 <- table(fit, data_PIdiabetes$X9)
tq1
```

```
##
## fit   0   1
##   0 436 140
##   1  64 127
```

```
# confusion matrix
1 - (sum(diag(tq1)) / sum(tq1))
```

```
## [1] 0.2659713
```

```
# misclassification error


data_PIdiabetes_q1 <- data_PIdiabetes
data_PIdiabetes_q1$X10 <- factor(fit)

ggplot(data = data_PIdiabetes_q1, aes(x = X8, y = X2, colour = X10)) +
  geom_point() +
  labs(x = "Age (years)", y = "Plasma glucose concentration",
       colour = "Predicted Diabetes (0=no or 1=yes)" )
```



- From the glm model, Both Plasma glucose and Age have a significant influence on Diabetes. From the results of this classification, we can see that our misclassification error is about 26.6%. This is not too bad I think. Just as seen with the original plot earlier, the plot with the predicted Diabetes also depits that level-1 Diabetes have more plasma glucose compare to level-0 group

3.Use the model estimated in step 2 to a) report the equation of the decision boundary between the two classes b) add a curve showing this boundary to the scatter plot in step 2. Comment whether the decision boundary seems to catch the data distribution well.

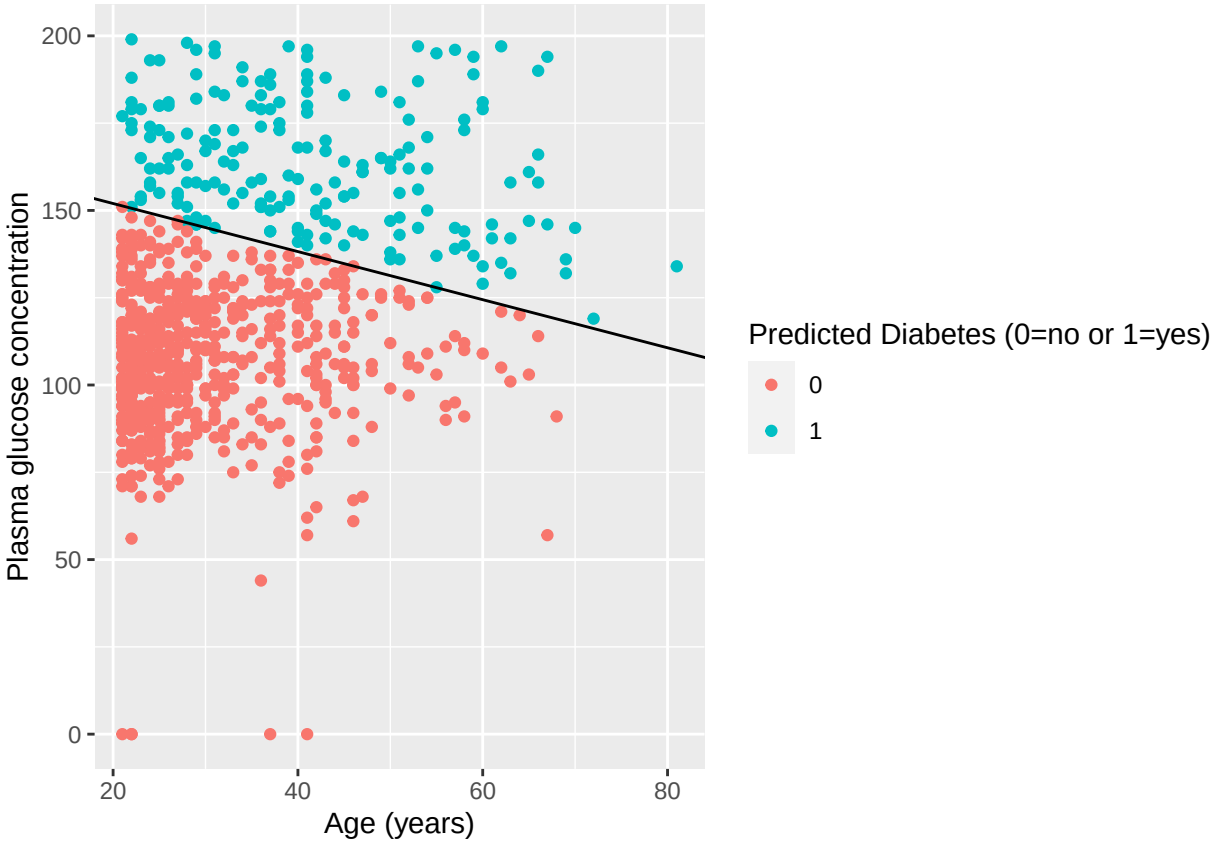
- when r = 0.5, we have

$$\frac{1}{1+e^{-\theta^T x}} = 0.5$$

then we have $\theta^T x = 0$, i.e. $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$ $(x_1 : concentration, x_2 : age)$

thus $slope = \frac{-\theta_2}{\theta_1}$, $intercept = \frac{-\theta_0}{\theta_1}$

```
theta <- coef(glm_diabete)

slope <- (-theta[3]) / theta[2]
intercept <- (-theta[1] ) / (theta[2])

ggplot(data = data_PIdiabetes_q1, aes(x = X8, y = X2, colour = X10)) +
  geom_point() +
  geom_abline(slope = slope, intercept = intercept) +

  labs(x = "Age (years)", y = "Plasma glucose concentration",
       colour  = "Predicted Diabetes (0=no or 1=yes)")
```



- Comments:I think this decision boundary does well for separating the data belonging two classes, even though not all data are put into the right sides of the boundary line.

4.Make same kind of plots as in step 2 but use thresholds r = 0.2 and r = 0.8. By using these plots, comment on what happens with the prediction when r value changes.

```
fit_r02 <- ifelse(fitted(glm_diabete) >= 0.2, 1, 0)

tr02 <- table(fit_r02, data_PIdiabetes$X9)
tr02
```

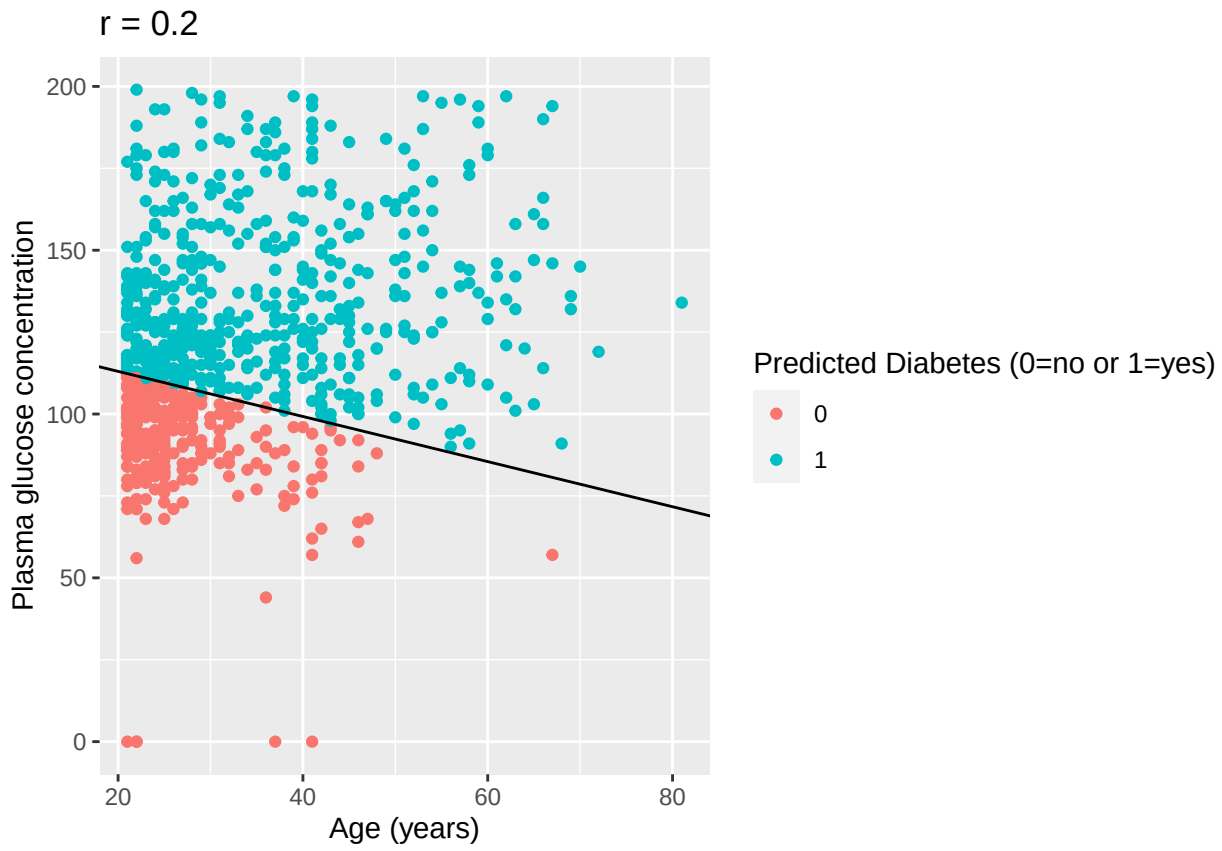
```
##
## fit_r02   0   1
##       0 238  25
##       1 262 242
```

```
#confusion matrix
1 - (sum(diag(tr02)) / sum(tr02))
```

```
## [1] 0.3741851
```

```
#misclassification error


data_PIdiabetes_r02 <- data_PIdiabetes
data_PIdiabetes_r02$X10 <- factor(fit_r02)

slope <- (-theta[3]) / theta[2]
intercept <- (-log(4) - theta[1] ) / (theta[2])
```

```
ggplot(data = data_PIdiabetes_r02, aes(x = X8, y = X2, colour = X10)) +
  geom_point() +
  geom_abline(slope = slope, intercept = intercept) +

  labs(x = "Age (years)", y = "Plasma glucose concentration",
       colour  = "Predicted Diabetes (0=no or 1=yes)", title = "r = 0.2" )
```




```
fit_r08 <- ifelse(fitted(glm_diabete) >= 0.8, 1, 0)


tr08 <- table(fit_r08, data_PIdiabetes$X9)
tr08
```

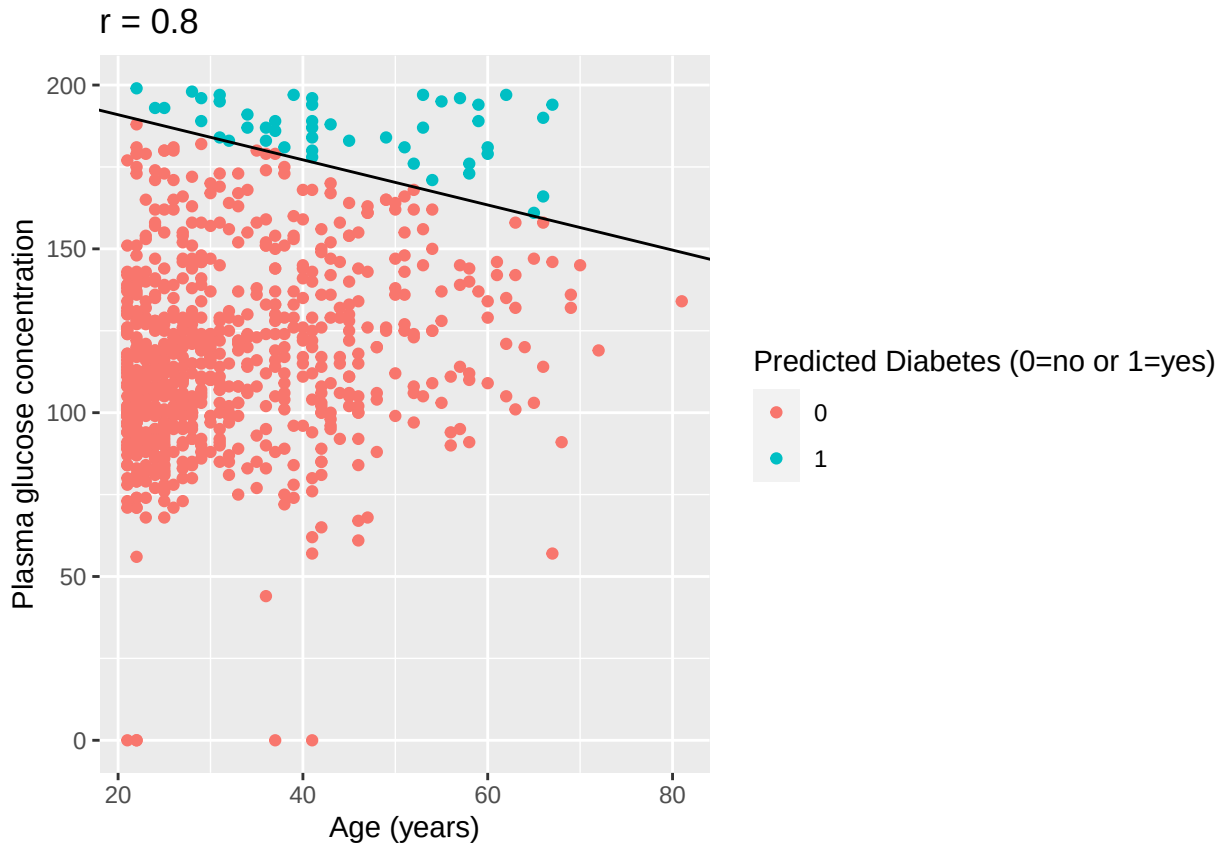
```
##
## fit_r08   0   1
##       0 490 231
##       1  10  36
```

```
#confusion matrix
1 - (sum(diag(tr08)) / sum(tr08))
```

```
## [1] 0.3142112
```

```
#misclassification error

data_PIdiabetes_r08 <- data_PIdiabetes
data_PIdiabetes_r08$X10 <- factor(fit_r08)
```

```
slope <- (-theta[3]) / theta[2]
intercept <- (log(4) - theta[1] ) / (theta[2])

ggplot(data = data_PIdiabetes_r08, aes(x = X8, y = X2, colour = X10)) +
  geom_point() +
  geom_abline(slope = slope, intercept = intercept) +

  labs(x = "Age (years)", y = "Plasma glucose concentration",
       colour  = "Predicted Diabetes (0=no or 1=yes)", title = "r = 0.8" )
```



- Comment: When r is 0.8, we can find that the method is more likely to regard the patients with diabetes as healthy. The probability is 87%. By contrast, when we use r = 0.2 as the threshold, we can find the patients with more probability, which is 91%.

```
242/(242+25)
```

```
## [1] 0.906367
```

```
231/(231+36)
```

```
## [1] 0.8651685
```

5.Perform a basis function expansion trick by computing new features $z_1 = x_1^4, z_2 = x_1^3 x_2^1, z_3 = x_1^2 x_2^2, z_4 = x_1^1 x_2^3, z_5 = x_2^4$, adding them to the data set and then computing a logistic regression model with y as target and $x_1, x_2, z_1, \ldots, z_5$ as features. Create a scatterplot of the same kind as in step 2 for this model and compute the training misclassification rate. What can you say about the quality of this model compared to the previous logistic regression model? How have the basis expansion trick affected the shape of the decision boundary and the prediction accuracy?

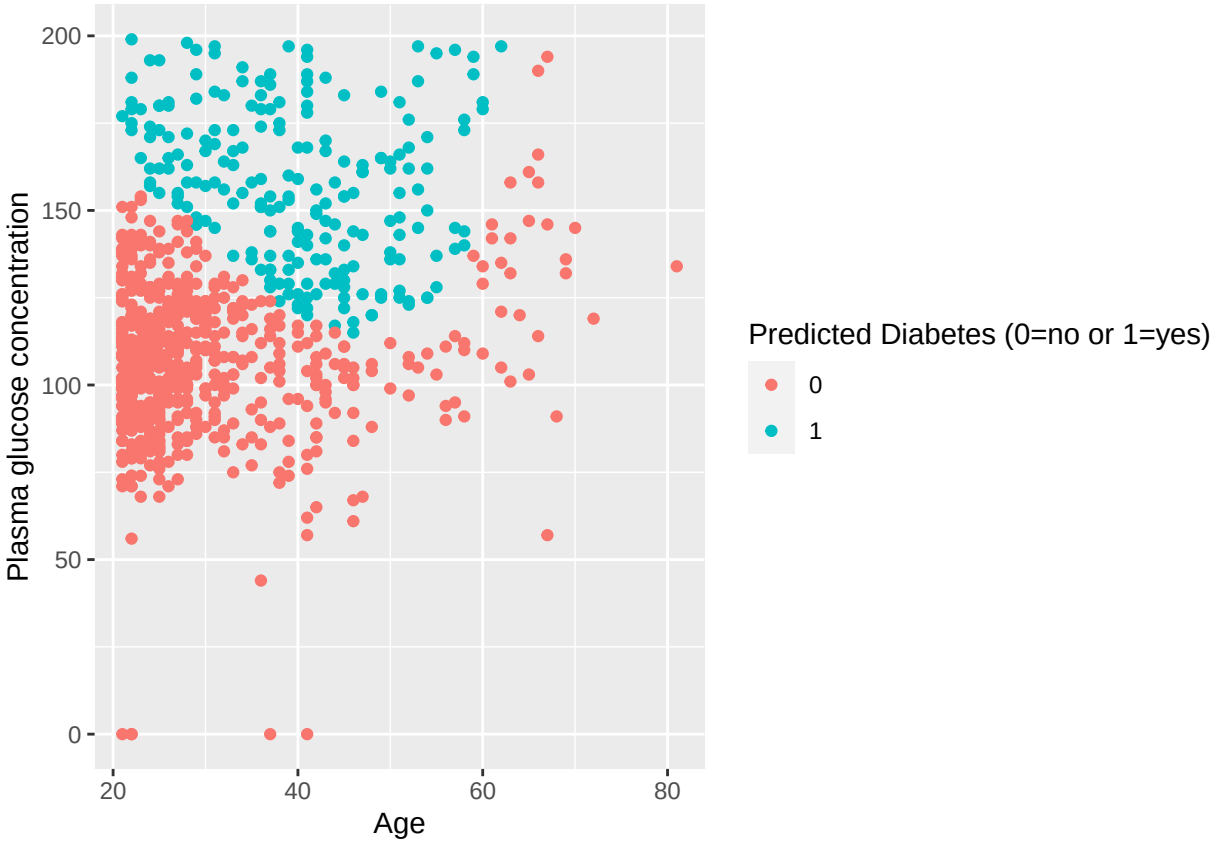
```r
data_PIdiabetes_q5 <-  data_PIdiabetes %>%  select(c(X2,X8,X9))
# X2,X8,X9
# 2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
# 8. Age (years).
# 9. Diabetes (0=no or 1=yes).

data_PIdiabetes_q5$Z1 <- data_PIdiabetes_q5$X2^4
data_PIdiabetes_q5$Z2 <- data_PIdiabetes_q5$X2^3 * data_PIdiabetes_q5$X8^1
data_PIdiabetes_q5$Z3 <- data_PIdiabetes_q5$X2^2 * data_PIdiabetes_q5$X8^2
data_PIdiabetes_q5$Z4 <- data_PIdiabetes_q5$X2^1 * data_PIdiabetes_q5$X8^3
data_PIdiabetes_q5$Z5 <- data_PIdiabetes_q5$X8^3

glm_diabete_q5 <- glm(X9 ~ X2 + X8 + Z1 + Z2 + Z3 + Z4 + Z5, data_PIdiabetes_q5, family = binomial)

fit <- ifelse(fitted(glm_diabete_q5) >= 0.5, 1, 0)

tq5 <- table(fit, data_PIdiabetes_q5$X9)
tq5
```

```
##
## fit   0   1
##   0 434 118
##   1  66 149
```

```r
# confusion matrix
1 - (sum(diag(tq5)) / sum(tq5))
```

```
## [1] 0.2398957
```

```r
# misclassification error

data_PIdiabetes_q5$X10 <- factor(fit)

ggplot(data = data_PIdiabetes_q5, aes(x = X8, y = X2, colour = X10)) +
  geom_point() +
  labs(x = "Age", y = "Plasma glucose concentration",
       colour  = "Predicted Diabetes (0=no or 1=yes)")
```



- Comment: When we increase the number of inputs and r equals 0.5, the eventual value of misclassification rate is similar to the one in the second question. The boundary line is no longer a straight line but a curve.