Machine Learning Computer Lab 1 (Group A7)

Qinyuan Qi(qinqi464) Satya Sai Naga Jaya Koushik Pilla (satpi345)
Daniele Bozzoli(danbo826)

2023-12-04

Assignment 1: Handwritten digit recognition with K- nearest neighbors (Solved by Qinyuan Qi - qinqi464)

Answer:

(1)

The following code will import the data and divide the data into training, validation and test sets.

```
#######################  Assignment 1.1 ##########################################
# read data
data <- read.csv("optdigits.csv")
row_num <- nrow(data)
cols_num <- ncol(data)

# set the last column name as "label_value" since we need to create a formula later
names(data)[cols_num] <- "label_value"

# set data split ratio to 0.5, 0.25 and 0.25
ratio <- c(train = .5, validate = .25, test = .25)

# data pre-processing
# columns 1-64 are number based and do not need to be normalized, last column
# is integer represent number from 0-9 which don't need to process again

# set random seed
set.seed(12345)

# split data to training, test and validation set
train_id <- sample(1:row_num, floor(row_num * ratio[1]))
train_set <- data[train_id, ]

set.seed(12345)
test_val_id <- setdiff(1:row_num, train_id)
valid_id <- sample(test_val_id, floor(row_num * ratio[2]))
valid_set <- data[valid_id, ]

test_id <- setdiff(test_val_id, valid_id)
test_set <- data[test_id, ]
```

(2)

The code contain a function call to kknn with k=30, kernel = "rectangular" with distance default to 2. We calculate the confusion matrices and the misclassification errors of training and test data set as follows, the code attached in the appendix. As the data showed below, the overall prediction quality is not good enough when k=30. And the misclassification rate of number 0,6 is the lowest. number 8 and 9 have the highest misclassification rate.

Table 1: Confusion matrix for training data set

	0	1	2	3	4	5	6	7	8	9
0	173	0	0	0	0	0	0	0	0	0
1	2	148	0	0	0	0	0	0	0	0
2	2	26	151	0	1	0	0	0	1	0
3	1	6	13	133	8	0	0	1	3	2
4	0	4	5	48	141	4	3	1	8	0
5	0	1	4	17	15	156	2	4	12	9
6	0	2	0	1	8	24	195	12	28	10
7	0	1	0	1	6	13	0	176	44	41
8	0	0	0	0	0	0	0	1	109	76
9	0	0	0	0	0	0	0	0	0	58

Table 2: Confusion matrix for validation data set

	0	1	2	3	4	5	6	7	8	9
0	98	0	0	0	0	0	0	0	0	0
1	0	70	0	0	0	0	0	0	0	0
2	1	26	87	0	1	0	2	0	1	0
3	0	4	11	52	3	0	0	0	1	2
4	0	1	8	25	74	3	0	1	2	0
5	0	1	4	12	18	66	1	0	4	2
6	0	0	1	2	6	19	80	5	10	6
7	0	2	1	0	5	4	0	90	18	13
8	0	0	0	0	1	0	0	1	48	42
9	0	0	0	0	0	0	0	0	0	20

Table 3: Misclassification rates

	rate
Training	0.2464678
Validation	0.2827225

Table 4: Misclassification rates of digits using learning data set

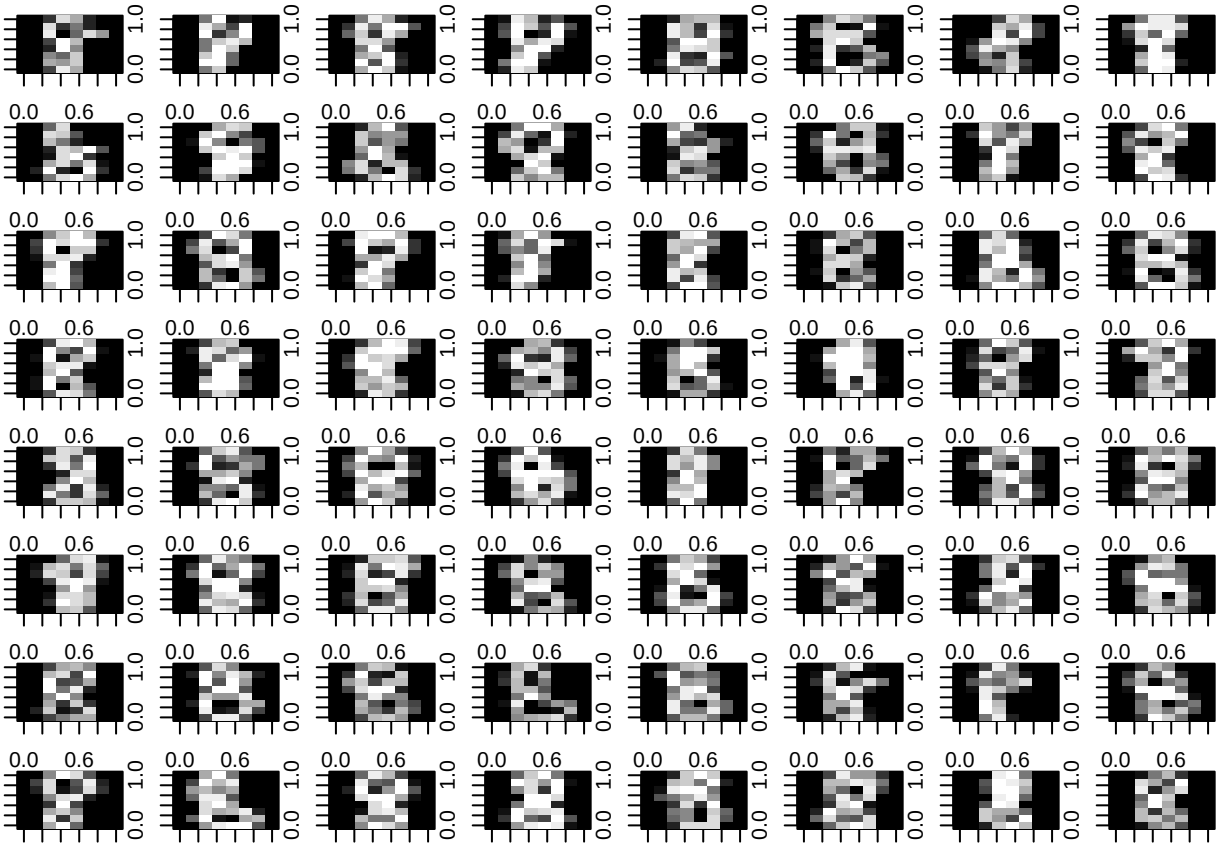
	rate
0	0.0026164
1	0.0209314
2	0.0115123
3	0.0350602

	rate
4	0.0198849
5	0.0214547
6	0.0026164
7	0.0099424
8	0.0502355
9	0.0722135

(3)

By using image function, we generate first 64 [8 image] from learning data set, and we get the following plot.(We use image function instead of heatmap so we can draw more images in one plot)

According to the plot, we can find that image at [3,2] and [5,2] are easy to identify. however, images such as [3,4] [4,2] [4,3] are very hard to figure out the number.



(4)

Using similar code as in 1.2, we fit the knn with k = 1 to 30.

When K increase, since all the points are related to more neighbors, we can say that the model become more complex. misclassification rates also increase. And validation data's misclassification rates always greater than train data's misclassification rates.

According to the plot, we know that optimized K = 1 which will generate the smallest misclassification rate.

When K = 1, we training 3 models, we got the following result in table 5. We can get the conclusion that

the test error rate is higher than the validation error rate. Both validation error rate and test error rate are greater than the value we get from the training data set.

Table 5: Misclassification rates of when K=1

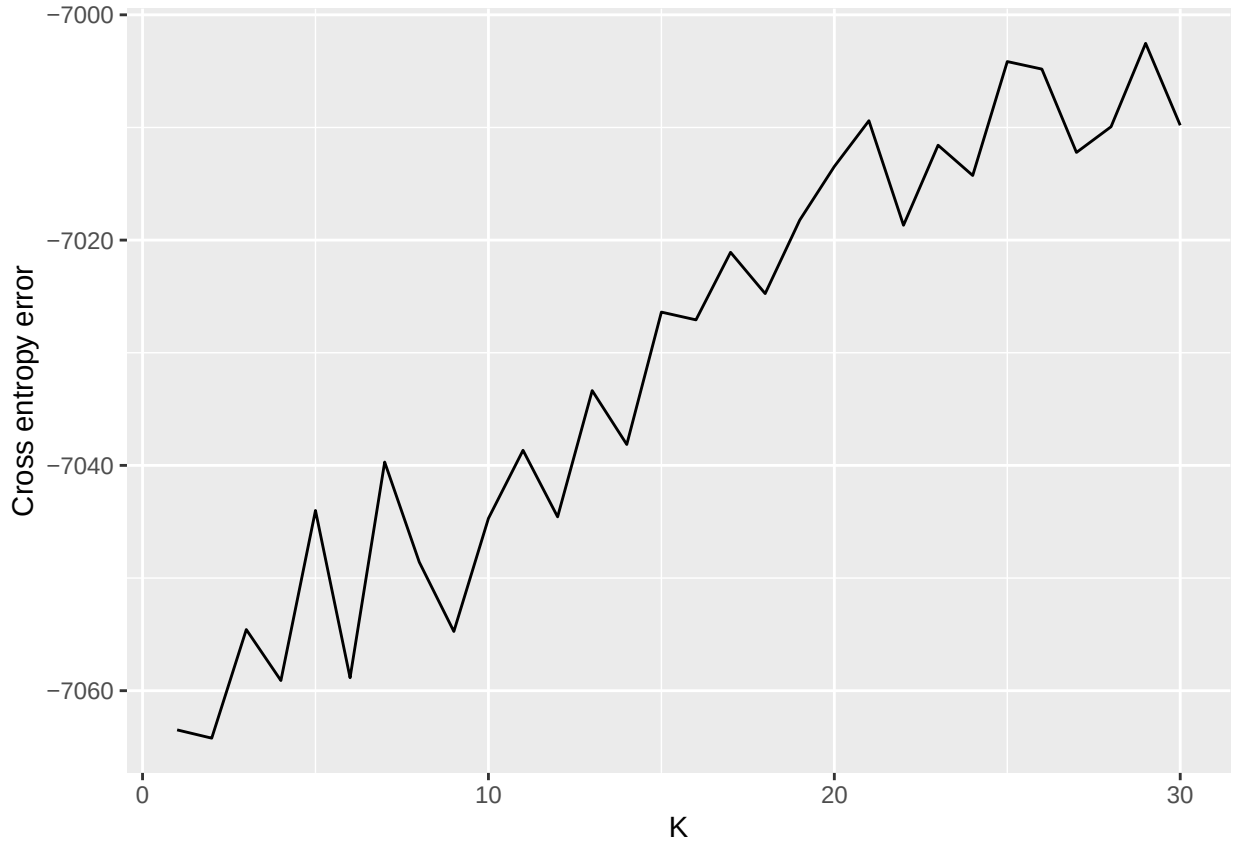
	Test Error
Training	0.0000000
Validation	0.0293194
Test	0.0334728

(5)

We plot the validation error using the code attached in the appendix, the plot of validation error and K as follows.

Since when K=2 get the minimal value, so the optimal K = 2.

The reason why we use cross-entropy is more suitable is: 1) It is sensitive to the predicted probabilities. 2) In multinomial distribution, the predicted value will more likely to be explained as a probability, and cross-entropy loss function is designed to compare the probability based prediction.



Assignment 2: Linear regression and ridge regressions (Solved by Satya Sai Naga Jaya Koushik Pilla)

Answer:

(1)

We read the data and divide data into training and test data (60/40) and normalize them.

```
############################  Assignment 2.1 ####################################
# Load the data
data <- read.csv("parkinsons.csv")

# remove useless data
data <- data[,-(1:4)]
data <- data[,-(2)]

row_num <- nrow(data)
cols_num <- ncol(data)

# Divide the data into training and test data (60/40)
# set data split ratio
ratio <- c(train = .6, test = .4)

# set random seed
set.seed(12345)
```

```r
# split data
train_id <- sample(1:row_num, floor(row_num * ratio[1]))
train_set <- data[train_id, ]
test_id <- setdiff(1:row_num, train_id)
test_set <- data[test_id, ]

# scale data.
preProcValues <- preProcess(train_set, method = c("scale"))
train_set <- predict(preProcValues,train_set)
test_set <- predict(preProcValues,test_set)
```

(2)

We create a linear model, we get the output as show below.

We have test MSE = 0.9354477.

According to the output of the model, we also know that Jitter.RAP,Jitter.DDP,Shimmer.APQ3,Shimmer.DDA contribute significantly to the model.

```
## test mse is  0.9354477
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ ., data = train_set)
##
## Coefficients:
##   (Intercept)      Jitter...    Jitter.Abs.     Jitter.RAP    Jitter.PPQ5
##      5.932650       0.186931      -0.169609      -5.269544      -0.074568
##    Jitter.DDP        Shimmer    Shimmer.dB.   Shimmer.APQ3   Shimmer.APQ5
##      5.249558       0.592436      -0.172655      32.070932      -0.387507
## Shimmer.APQ11    Shimmer.DDA            NHR            HNR           RPDE
##      0.305546     -32.387241      -0.185387      -0.238543       0.004068
##           DFA            PPE
##     -0.280318       0.226467
```

(3)

Functions implemented as below.

```r
############################  Assignment 2.3 ####################################
# Loglikelihood function
loglikelihood <- function(x,y, theta, sigma) {
  n <- length(x)
  log_likelihood_value <- -0.5 * (n * log(2 * pi * sigma^2)) -
                          sum((t(theta) * x - y)^2) / (2 * sigma^2)
  return(log_likelihood_value)
}

# ridge
ridge <- function(par,x,y,lambda) {

  param_length <- length(par)

  theta <- par[1:param_length - 1]
  sigma <- par[param_length]
```

```
  loglikelihood_result <- loglikelihood(x,y,theta, sigma)
  ridge_penalty <- lambda * sum(theta^2)
  return(loglikelihood_result - ridge_penalty)
}

# ridgeopt
ridgeopt <- function(initial_values, x,y,lambda ) {
  # Use optim function

  result <- optim(par = initial_values, ridge, method = "BFGS",
                  x = x, y = y,lambda = lambda)
  return(result)
}

# df
df <- function(x, y, theta, sigma) {
  hat_y <- t(theta) %*% x
  return(sum(cov(hat_y, y)) / sigma^2)
}
```

(4)

Functions implemented as below.

```
############################  Assignment 2.4 ######################################

x <- train_set[,-1]
y <- train_set$motor_UPDRS
colnum_num <- ncol(x)

#lambda <- 1
theta <- rep(1, ncol(x))
sigma <- 1
initial_values <- c(theta, sigma)
result1 <- ridgeopt(initial_values=initial_values, x = x, y=y,lambda = 1 )
result1
```

```
## $par
##  [1] 4.055120e+13 4.054138e+13 4.055246e+13 4.054097e+13 4.055281e+13
##  [6] 4.055120e+13 4.055263e+13 4.054136e+13 4.055099e+13 4.054134e+13
## [11] 4.055100e+13 4.055284e+13 4.055097e+13 4.055102e+13 4.055118e+13
## [16] 4.054138e+13 1.973971e+06
##
## $value
## [1] -2.630679e+28
##
## $counts
## function gradient
##       21       21
##
## $convergence
## [1] 0
##
## $message
```

```
## NULL
```

```
#train_pred1 <- t(train_set[,-1]) %*% result1$par[,1:colnum_num-1] + result1$par[,colnum_num]
#train_mse1 <- mean((train_pred1 - train_set$motor_UPDRS)^2)
#test_pred1 <- test_set[,-1] %*% result1$par[,1:colnum_num-1] + result1$par[,colnum_num]
#test_mse1 <- mean((test_pred1 - test_set$motor_UPDRS)^2)
#df1 <- df(x,y,result1$par[,1:colnum_num-1],result1$par[,colnum_num])



#lambda <- 100
theta <- rep(1, ncol(x))
sigma <- 1
initial_values <- c(theta, sigma)
result100 <- ridgeopt(initial_values=initial_values, x = x, y=y,lambda = 100 )
result100
```

```
## $par
##  [1]  3.021198e+13  3.021013e+13  3.021303e+13  3.020802e+13  3.021422e+13
##  [6]  3.021191e+13  3.021362e+13  3.021007e+13  3.021079e+13  3.020980e+13
## [11]  3.021086e+13  3.021461e+13  3.021066e+13  3.021119e+13  3.021171e+13
## [16]  3.021013e+13 -4.014640e+05
##
## $value
## [1] -1.460368e+30
##
## $counts
## function gradient
##        6        6
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#train_pred100 <- train_set[,-1] %*% result100$par[,1:colnum_num-1] + result100$par[,colnum_num]
#train_mse100 <- mean((train_pred100 - train_set$motor_UPDRS)^2)
#test_pred100 <- test_set[,-1] %*% result100$par[,1:colnum_num-1] + result100$par[,colnum_num]
#test_mse100 <- mean((test_pred100 - test_set$motor_UPDRS)^2)
#df100 <- df(x,y,result100$par[,1:colnum_num-1],result100$par[,colnum_num])

#lambda <- 1000
theta <- rep(1, ncol(x))
sigma <- 1
initial_values <- c(theta, sigma)
result1000 <- ridgeopt(initial_values=initial_values, x = x, y=y,lambda = 1000 )
result1000
```

```
## $par
##  [1]  1.546986e+14  1.546977e+14  1.546991e+14  1.546967e+14  1.546997e+14
##  [6]  1.546986e+14  1.546994e+14  1.546977e+14  1.546980e+14  1.546975e+14
## [11]  1.546981e+14  1.546999e+14  1.546980e+14  1.546982e+14  1.546985e+14
## [16]  1.546977e+14 -4.014640e+05
##
## $value
```

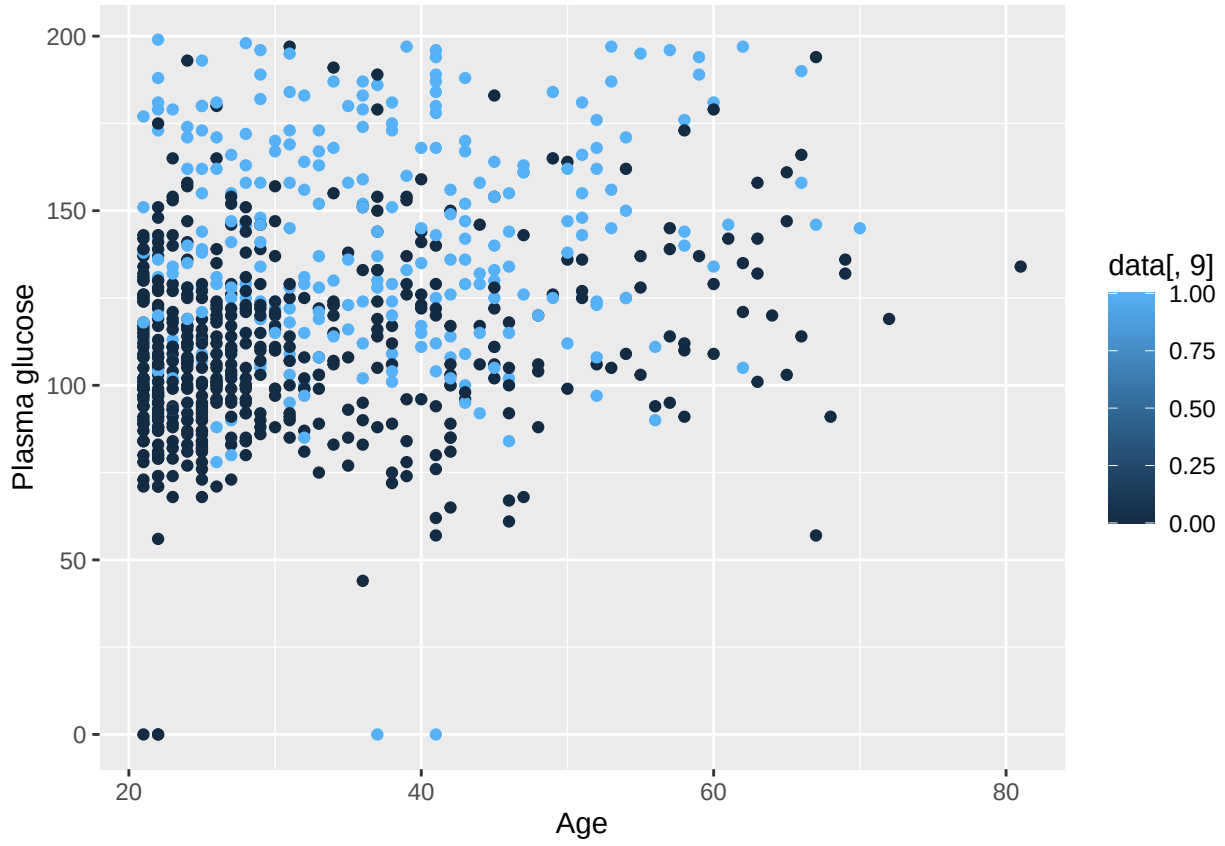
```
## [1] -3.829052e+32
##
## $counts
## function gradient
##        5        5
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#train_pred1000 <- train_set[,-1] %*% result1000$par[,1:colnum_num-1] + result1000$par[,colnum_num]
#train_mse1000 <- mean((train_pred1000 - train_set$motor_UPDRS)^2)
#test_pred1000 <- test_set[,-1] %*% result1000$par[,1:colnum_num-1] + result1000$par[,colnum_num]
#test_mse1000 <- mean((test_pred1000 - test_set$motor_UPDRS)^2)
#df1000 <- df(x,y,result1000$par[,1:colnum_num-1],result1000$par[,colnum_num])


#data2_4 <- data.frame(c("1","100","1000"),
#                      c(mse1,mse100,mse1000),
#                      c(pred1,pred100,pred1000),
#                      c(df1,df100,df1000))

#names(data2_4)[1] <- "lambda"
#names(data2_4)[2] <- "MSE"
#names(data2_4)[3] <- "Predicted Value"
#names(data2_4)[4] <- "df"

#knitr::kable(data2_4,
#             caption = "Values when lambda=1,100,1000")
```

Assignment 3. Logistic regression and basis function expansion (Solved by Daniele Bozzoli)

Answer:

(1)



We observe how a lot of people without diabete is highly concentrated in the lower part of the age axis, explaining how younger individuals tend to be non-diabetic, on the other hand, we notice how especially people with higher plasma glucose concentration tend to be diabetic, less concentrated in a specific age group, more spread across the x-age axis.

(2)(3)

Code as below. Looking at the graph, we notice how the classification is not very accurate. Though, it catches the fact that people with higher plasma glucose concentration are more commonly diabetic. We obtain a misclassification error = 0.266.

```
model <- glm(diabetes ~ plasma + age, fam = binomial(link="logit"), data = data)
summary(model)
```

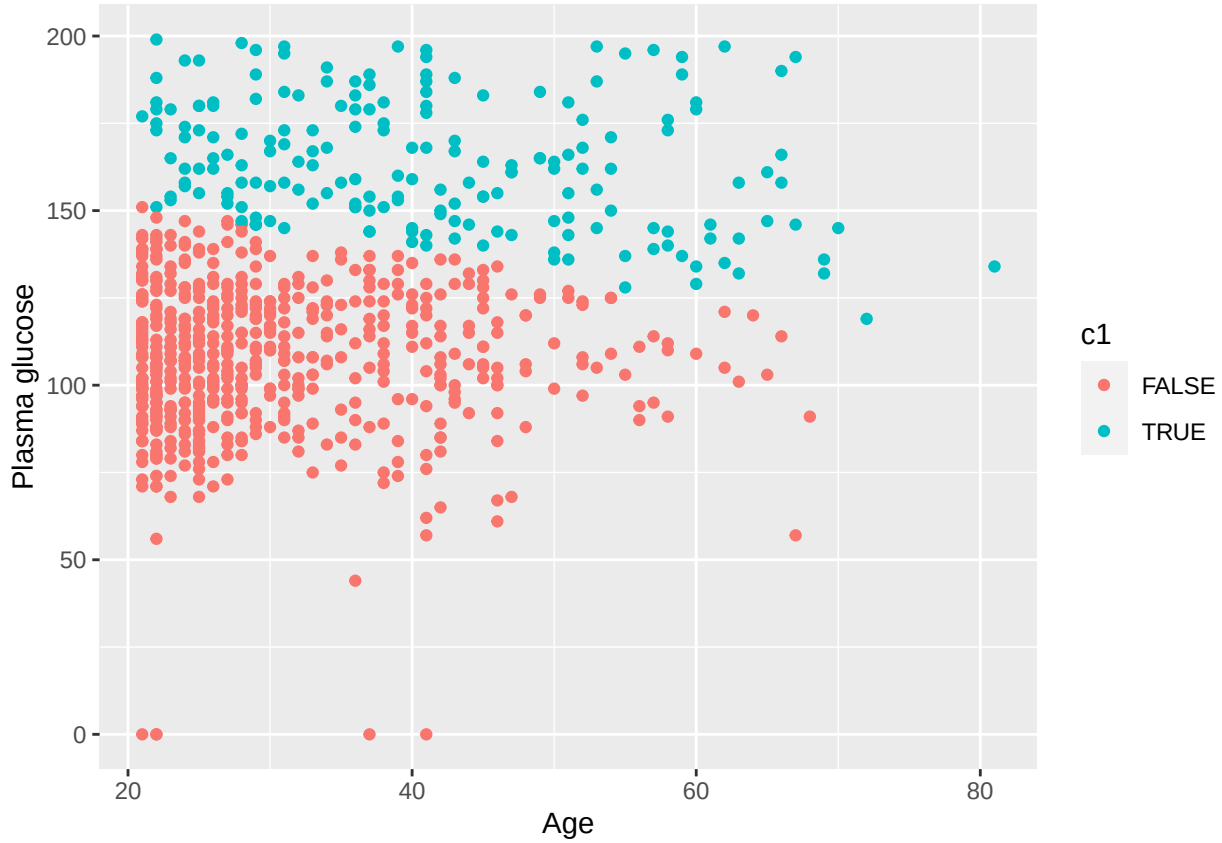
```
##
## Call:
## glm(formula = diabetes ~ plasma + age, family = binomial(link = "logit"),
##     data = data)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept) -5.897858   0.462450  -12.75  < 2e-16 ***
## plasma       0.035582   0.003288   10.82  < 2e-16 ***
## age          0.024502   0.007379    3.32 0.000899 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 991.38  on 766  degrees of freedom
## Residual deviance: 796.49  on 764  degrees of freedom
## AIC: 802.49
##
## Number of Fisher Scoring iterations: 4
```

```r
# r = 0.5

c1 <- fitted(model) >= .5

missC_error <- sum(data$diabetes != c1) / nrow(data)
cat("Misclassification error:", missC_error, "\n")
```

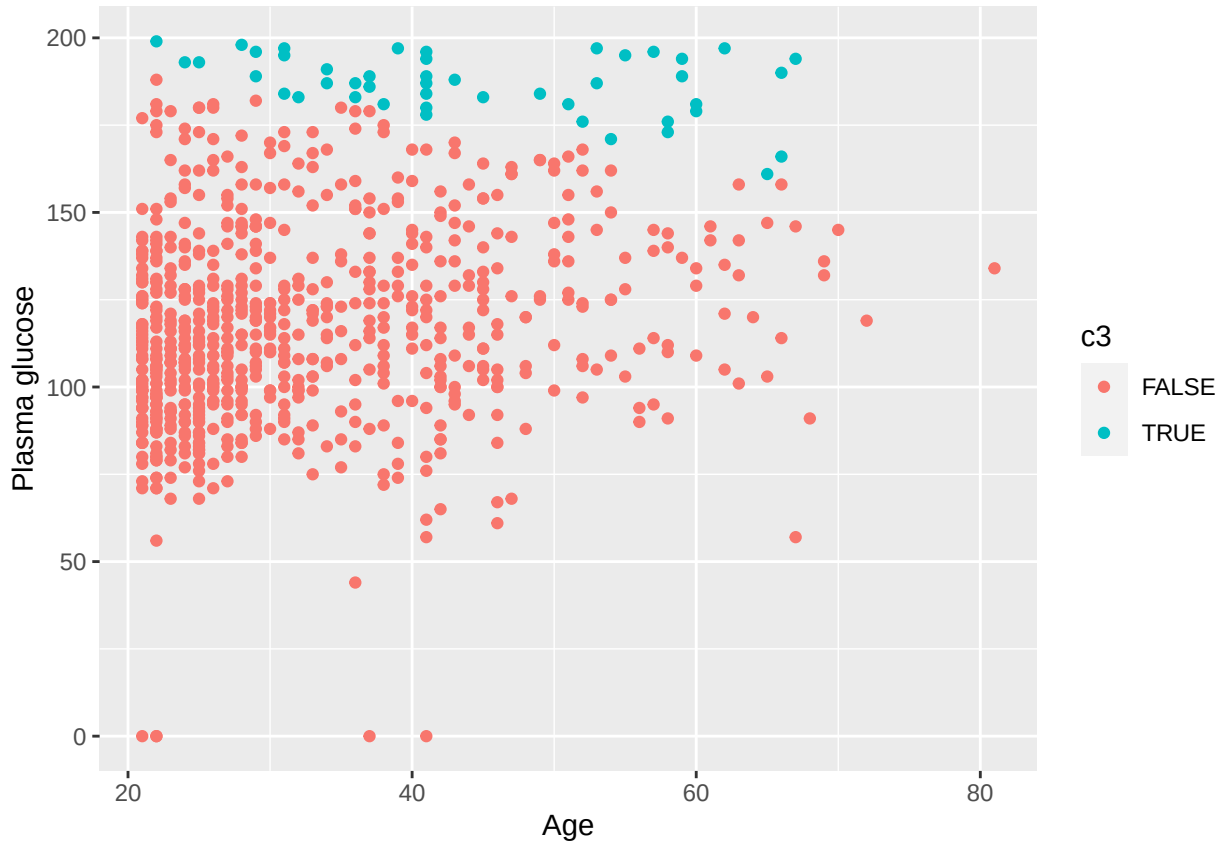
```
## Misclassification error: 0.2659713
```

```r
ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = c1))+
  ggplot2::labs(x = "Age", y="Plasma glucose")      # T = Diabetic, F = Non-diabetic
```



(4)

It looks like the line that splits the predicted observations into diabetic and non diabetic into the two cases (r= 0.2 and r=0.8) has the same slope, but different intercept. Both cases are pretty bad predictors as 0.2 and 0.8 are respectively too low and too high to get a good classification for our case.



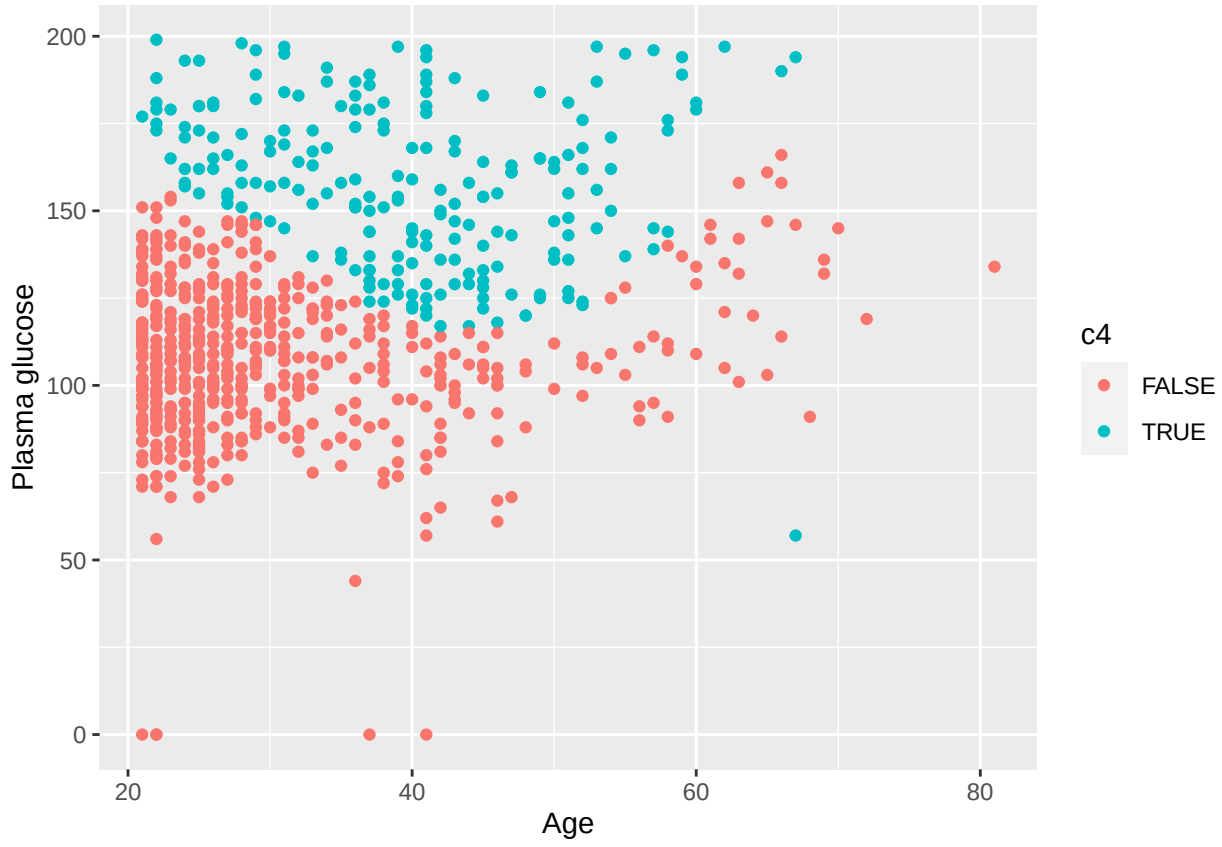
(5)

The outcome of this analysis is better than the first case, as we obtain a lower misclassification error than before, and also graphically we notice how this version captures what we were saying in the first part of the analysis, how people with higher plasma glucose concentration are more exposed to the risk of having diabete. We also notice how we strangely get a misclassificated predicted value in the bottom right corner of the graph.

```
##
## Call:
## glm(formula = diabetes ~ plasma + age + z1 + z2 + z3 + z4 + z5,
##     family = binomial(link = "logit"), data = newdata)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.279e+00  1.129e+00  -8.217  < 2e-16 ***
## plasma       3.772e-02  9.473e-03   3.981 6.85e-05 ***
## age          1.453e-01  2.072e-02   7.014 2.32e-12 ***
## z1           1.266e-08  5.610e-09   2.257  0.02402 *
## z2          -1.760e-07  7.638e-08  -2.304  0.02122 *
## z3           8.424e-07  3.439e-07   2.450  0.01430 *
## z4          -1.682e-06  6.317e-07  -2.662  0.00776 **
## z5           8.045e-07  4.056e-07   1.983  0.04732 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 991.38  on 766  degrees of freedom
## Residual deviance: 741.07  on 759  degrees of freedom
## AIC: 757.07
##
## Number of Fisher Scoring iterations: 5
```

```
## Misclassification error: 0.2464146
```



Appendix: All code for this report

```
############################  Init code For Assignment 1 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(kknn)
library(ggplot2)
library(Metrics)
library(caret)


#######################  Assignment 1.1 ############################################
# read data
data <- read.csv("optdigits.csv")
row_num <- nrow(data)
cols_num <- ncol(data)

# set the last column name as "label_value" since we need to create a formula later
names(data)[cols_num] <- "label_value"

# set data split ratio to 0.5, 0.25 and 0.25
ratio <- c(train = .5, validate = .25, test = .25)

# data pre-processing
# columns 1-64 are number based and do not need to be normalized, last column
# is integer represent number from 0-9 which don't need to process again

# set random seed
set.seed(12345)

# split data to training, test and validation set
train_id <- sample(1:row_num, floor(row_num * ratio[1]))
train_set <- data[train_id, ]

set.seed(12345)
test_val_id <- setdiff(1:row_num, train_id)
valid_id <- sample(test_val_id, floor(row_num * ratio[2]))
valid_set <- data[valid_id, ]

test_id <- setdiff(test_val_id, valid_id)
test_set <- data[test_id, ]
#######################  Assignment 1.2 ############################################

k_value <- 30
kknn_train <- kknn(label_value ~ .,
                         train_set,
                         train_set,
                         k = k_value,
                         kernel = "rectangular",scale = TRUE)

# generate confusion matrix for training data
confusion_matrices_train <- table(round(kknn_train$fit), train_set$label_value)

# print confusion matrix
```

```
#print(confusion_matrices_train)


# calculate error rate
error_rate_train <- mean(round(kknn_train$fit) != train_set$label_value)
#cat("Misclassification error for training data is: ", error_rate_train, "\n")

kknn_valid <- kknn(label_value ~ .,
                   train_set,
                   valid_set,
                   k = k_value,
                   kernel = "rectangular",scale = TRUE)

# generate confusion matrix for validate data
confusion_matrices_valid <- table(round(kknn_valid$fit), valid_set$label_value)


# print confusion matrix
#print(confusion_matrices_valid)

# calculate error rate
error_rate_valid <- mean(round(kknn_valid$fit) != valid_set$label_value)
#cat("Misclassification error for valid data is: ", error_rate_valid, "\n")

# misclassification rates
misclassification_rates_data_1_2 <- data.frame(name=c("Training","Validation"),
                                               rate=c(error_rate_train,
                                                      error_rate_valid))
names(misclassification_rates_data_1_2)[1] <- ""

# misclassification errors for each digits on training data
misclassification_rates_for_each_digits_data_1_2 <- data.frame(c(0,9),rate=rep(0,10))
names(misclassification_rates_for_each_digits_data_1_2)[1] <- ""

for(i in 1:10){
  misclassification_rates_for_each_digits_data_1_2[i,] <- c(i-1,mean(
    round(kknn_train$fit) != train_set$label_value &
    train_set$label_value==(i-1)))
}


# render result to tables

knitr::kable(confusion_matrices_train,
             caption = "Confusion matrix for training data set")
knitr::kable(confusion_matrices_valid,
             caption = "Confusion matrix for validation data set")
knitr::kable(misclassification_rates_data_1_2,
             caption = "Misclassification rates")
knitr::kable(misclassification_rates_for_each_digits_data_1_2,
             caption = "Misclassification rates of digits using learning data set")
######################  Assignment 1.3 ##########################################
```

```r
# Find all the 8 images
train_eight <- train_set[train_set[,65]==8, 1:64]

# draw 64 [8 images]
par(mfrow = c(8, 8), mar=c(1,1,1,1))

for(i in 1:64){
  row.id <- i
  m <- matrix(as.numeric(train_eight[row.id, 1:64]), nrow=8, ncol=8)
  image(m[,8:1], col=grey(seq(0, 1, length=16)))
}

######################  Assignment 1.4 ##########################################
# calculate error rate for different k values
error_rates_train <- c()
error_rates_valid <- c()

for(k_value in 1:30){

  kknn_train <- kknn(label_value ~ .,
                           train_set,
                           train_set,
                           k = k_value,
                           kernel = "rectangular",scale = TRUE)

  kknn_valid <- kknn(label_value ~ .,
                          train_set,
                          valid_set,
                          k = k_value,
                          kernel = "rectangular",scale = TRUE)

  # train error rate
  error_rates_train <- append(error_rates_train,
                       mean(round(predict(kknn_train)) != train_set$label_value))


  # valid error rate
  error_rates_valid <- append(error_rates_valid,
                       mean(round(predict(kknn_valid)) != valid_set$label_value))


}

# plot the error rate graph
k <- 1:30

error_rate_data <- data.frame(k, error_rates_train,error_rates_valid)

ggplot(data = error_rate_data) +
  geom_line(mapping = aes(x=k,y=error_rates_train,colour="train")) +
  geom_line(mapping = aes(x=k,y=error_rates_valid,colour="validation")) +
  labs(x = "K",y="Misclassification rates") +
  scale_color_manual(name = "Data set", values = c("train" = "blue", "validation" = "red"))
```

```r
k_value <- 1

kknn_train <- kknn(label_value ~ .,
                           train_set,
                           train_set,
                           k = k_value,
                           kernel = "rectangular",scale = TRUE)

kknn_valid <- kknn(label_value ~ .,
                           train_set,
                           valid_set,
                           k = k_value,
                           kernel = "rectangular",scale = TRUE)

kknn_test <- kknn(label_value ~ .,
                           train_set,
                           test_set,
                           k = k_value,
                           kernel = "rectangular",scale = TRUE)

# train error rate
error_rate_train <- mean(round(predict(kknn_train)) != train_set$label_value)


# valid error rate
error_rate_valid <- mean(round(predict(kknn_valid)) != valid_set$label_value)

# test error rate
error_rate_test <- mean(round(predict(kknn_test)) != test_set$label_value)

data1_4 <- data.frame(c("Training","Validation","Test"),
                      c(error_rate_train,error_rate_valid,error_rate_test))
names(data1_4)[1] <- ""
names(data1_4)[2] <- "Test Error"
knitr::kable(data1_4,
             caption = "Misclassification rates of when K=1")

######################  Assignment 1.5 ##########################################
# init value

k_values <- 1:30
epsilon <- 1e-15
validation_errors <- numeric(length(k_values))

for(k in k_values){
    kknn_train <- kknn(label_value ~ .,
                           train_set,
                           valid_set,
                           k = k,
                           kernel = "rectangular",scale = TRUE)

    pred <- round(predict(kknn_train))
```

```
    validation_errors[k] <- -sum(valid_set$label_value * log(pred + epsilon))
}

error_cross_entropy_data <- data.frame(k_values,validation_errors)

ggplot(data = error_cross_entropy_data) +
  geom_line(mapping = aes(x=k_values,y=validation_errors))  +
  labs(x = "K",y="Cross entropy error")

############################  Init code For Assignment 2 ########################
rm(list = ls())
library(caret)
knitr::opts_chunk$set(echo = TRUE)

############################  Assignment 2.1 ####################################
# Load the data
data <- read.csv("parkinsons.csv")

# remove useless data
data <- data[,-(1:4)]
data <- data[,-(2)]

row_num <- nrow(data)
cols_num <- ncol(data)

# Divide the data into training and test data (60/40)
# set data split ratio
ratio <- c(train = .6, test = .4)

# set random seed
set.seed(12345)

# split data
train_id <- sample(1:row_num, floor(row_num * ratio[1]))
train_set <- data[train_id, ]
test_id <- setdiff(1:row_num, train_id)
test_set <- data[test_id, ]

# scale data.
preProcValues <- preProcess(train_set, method = c("scale"))
train_set <- predict(preProcValues,train_set)
test_set <- predict(preProcValues,test_set)

############################  Assignment 2.2 ####################################
# Linear regression model

# apply linear regression(we already removed the data we don't need)
model <- lm(motor_UPDRS ~ ., data = train_set)

# predict the test value using the linear regression just created
test_pred <- predict(model, test_set)

# calculate test MSE
```

```
test_mse <- mean((test_pred - test_set$motor_UPDRS)^2)
cat("test mse is " , test_mse , "\n")
model
############################  Assignment 2.3 ####################################
# Loglikelihood function
loglikelihood <- function(x,y, theta, sigma) {
  n <- length(x)
  log_likelihood_value <- -0.5 * (n * log(2 * pi * sigma^2)) -
                          sum((t(theta) * x - y)^2) / (2 * sigma^2)
  return(log_likelihood_value)
}

# ridge
ridge <- function(par,x,y,lambda) {

  param_length <- length(par)

  theta <- par[1:param_length - 1]
  sigma <- par[param_length]

  loglikelihood_result <- loglikelihood(x,y,theta, sigma)
  ridge_penalty <- lambda * sum(theta^2)
  return(loglikelihood_result - ridge_penalty)
}

# ridgeopt
ridgeopt <- function(initial_values, x,y,lambda ) {
  # Use optim function

  result <- optim(par = initial_values, ridge, method = "BFGS",
                  x = x, y = y,lambda = lambda)
  return(result)
}

# df
df <- function(x, y, theta, sigma) {
  hat_y <- t(theta) %*% x
  return(sum(cov(hat_y, y)) / sigma^2)
}

############################  Assignment 2.4 ####################################

x <- train_set[,-1]
y <- train_set$motor_UPDRS
colnum_num <- ncol(x)

#lambda <- 1
theta <- rep(1, ncol(x))
sigma <- 1
initial_values <- c(theta, sigma)
result1 <- ridgeopt(initial_values=initial_values, x = x, y=y,lambda = 1 )
result1
#train_pred1 <- t(train_set[,-1]) %*% result1$par[,1:colnum_num-1] + result1$par[,colnum_num]
```

```r
#train_mse1 <- mean((train_pred1 - train_set$motor_UPDRS)^2)
#test_pred1 <- test_set[,-1] %*% result1$par[,1:colnum_num-1] + result1$par[,colnum_num]
#test_mse1 <- mean((test_pred1 - test_set$motor_UPDRS)^2)
#df1 <- df(x,y,result1$par[,1:colnum_num-1],result1$par[,colnum_num])



#lambda <- 100
theta <- rep(1, ncol(x))
sigma <- 1
initial_values <- c(theta, sigma)
result100 <- ridgeopt(initial_values=initial_values, x = x, y=y,lambda = 100 )
result100
#train_pred100 <- train_set[,-1] %*% result100$par[,1:colnum_num-1] + result100$par[,colnum_num]
#train_mse100 <- mean((train_pred100 - train_set$motor_UPDRS)^2)
#test_pred100 <- test_set[,-1] %*% result100$par[,1:colnum_num-1] + result100$par[,colnum_num]
#test_mse100 <- mean((test_pred100 - test_set$motor_UPDRS)^2)
#df100 <- df(x,y,result100$par[,1:colnum_num-1],result100$par[,colnum_num])

#lambda <- 1000
theta <- rep(1, ncol(x))
sigma <- 1
initial_values <- c(theta, sigma)
result1000 <- ridgeopt(initial_values=initial_values, x = x, y=y,lambda = 1000 )
result1000

#train_pred1000 <- train_set[,-1] %*% result1000$par[,1:colnum_num-1] + result1000$par[,colnum_num]
#train_mse1000 <- mean((train_pred1000 - train_set$motor_UPDRS)^2)
#test_pred1000 <- test_set[,-1] %*% result1000$par[,1:colnum_num-1] + result1000$par[,colnum_num]
#test_mse1000 <- mean((test_pred1000 - test_set$motor_UPDRS)^2)
#df1000 <- df(x,y,result1000$par[,1:colnum_num-1],result1000$par[,colnum_num])


#data2_4 <- data.frame(c("1","100","1000"),
#                      c(mse1,mse100,mse1000),
#                      c(pred1,pred100,pred1000),
#                      c(df1,df100,df1000))

#names(data2_4)[1] <- "lambda"
#names(data2_4)[2] <- "MSE"
#names(data2_4)[3] <- "Predicted Value"
#names(data2_4)[4] <- "df"

#knitr::kable(data2_4,
#             caption = "Values when lambda=1,100,1000")

############################  Init code For Assignment 3 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
data <- read.csv("pima-indians-diabetes.csv")

age <- data[, 8]
```

```r
plasma <- data[, 2]

names(data)[9] <- "diabetes"
names(data)[2] <- "plasma"
names(data)[8] <- "age"

x <- age
y <- plasma
df <- data.frame(x, y)

ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = data[, 9]))+
  ggplot2::labs(x = "Age", y="Plasma glucose")


model <- glm(diabetes ~ plasma + age, fam = binomial(link="logit"), data = data)
summary(model)

# r = 0.5

c1 <- fitted(model) >= .5

missC_error <- sum(data$diabetes != c1) / nrow(data)
cat("Misclassification error:", missC_error, "\n")

ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = c1))+
  ggplot2::labs(x = "Age", y="Plasma glucose")     # T = Diabetic, F = Non-diabetic


# r = 0.2

c2 <- fitted(model) >= .2

ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = c2))+
  ggplot2::labs(x = "Age", y="Plasma glucose")

# r = 0.8

c3 <- fitted(model) >= .8

ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = c3))+
  ggplot2::labs(x = "Age", y="Plasma glucose")


# Adding new variables, logit function with r= 0.5

z1 <- data$plasma^4
z2 <- data$plasma^3 * data$age
z3 <- data$plasma^2 * data$age^2
z4 <- data$plasma^1 * data$age^3
```

```r
z5 <- data$age^4

newdata <- cbind(data, z1, z2, z3, z4, z5)

newmodel <- glm(diabetes ~ plasma + age + z1 + z2 + z3 + z4 + z5 , family = binomial(link = "logit") , c

summary(newmodel)

c4 <- fitted(newmodel) >= .5

new_missC_error <- sum(data$diabetes != c4) / nrow(data)
cat("Misclassification error:", new_missC_error, "\n")

ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = c4))+
  ggplot2::labs(x = "Age", y="Plasma glucose")
```