Machine Learning Computer Lab 1 (Group A7)

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

2023-11-21

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

Answer:

(1)

The following code will input 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, test = .25, validate = .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(Check appendix) contain a function call to kknn with k=30 and kernel = ”rectangular”. We calculate the confusion matrices and the misclassification errors.

The confusion_matrices is as follows:

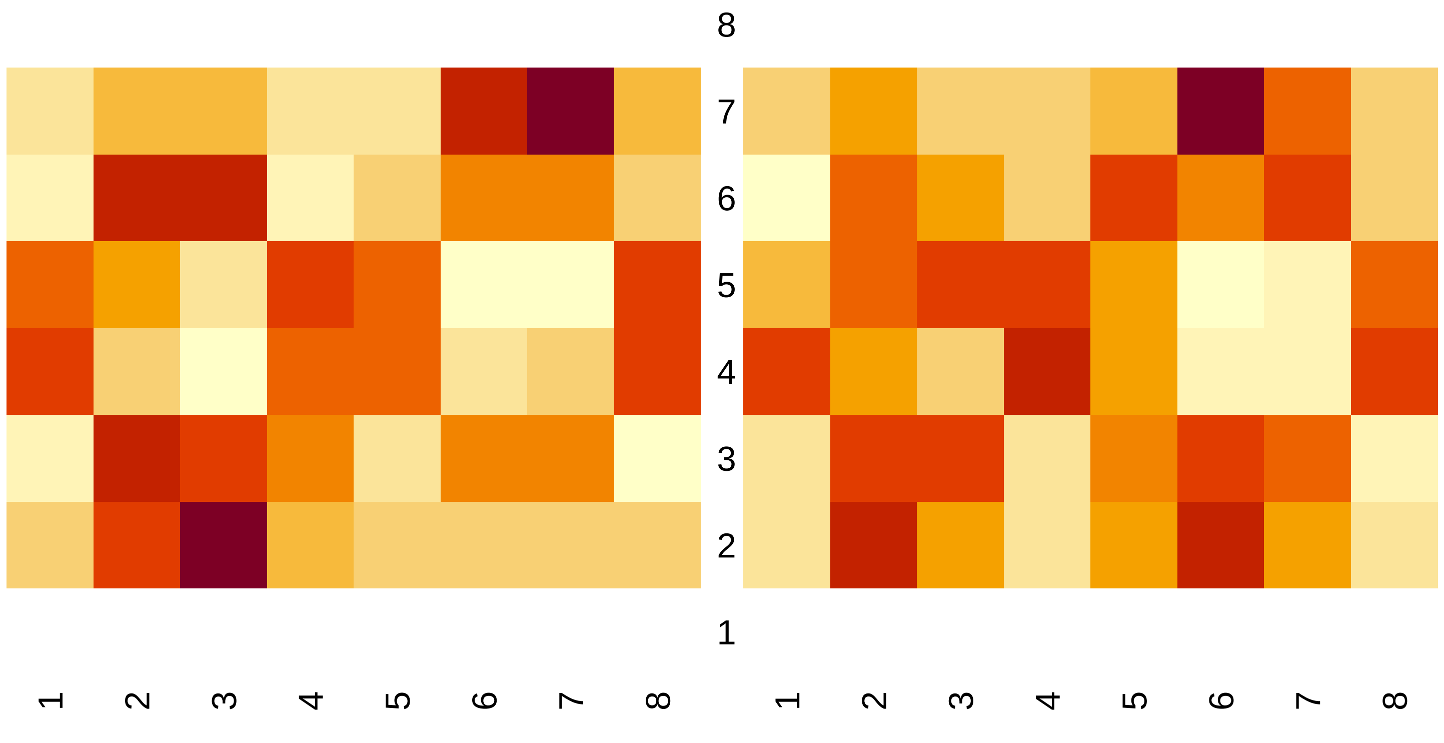
	0	1	2	3	4	5	6	7	8	9
0	98	0	0	0	0	0	0	0	0	0
1	0	70	0	0	0	0	1	0	0	0
2	1	26	87	0	1	0	2	0	1	0
3	0	4	11	52	3	0	0	0	1	2
4	0	1	8	25	74	3	0	1	2	0
5	0	1	4	12	18	66	1	0	4	2
6	0	0	1	2	6	19	80	5	10	6
7	0	2	1	0	5	4	0	90	18	13
8	0	1	0	0	1	1	0	1	48	42
9	0	0	0	1	1	1	0	0	0	20

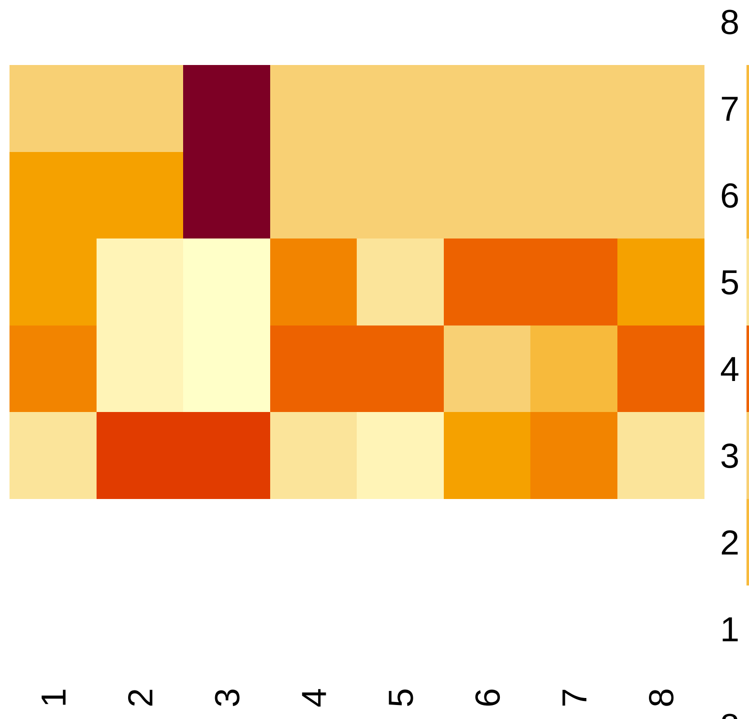
As what the data show in the confusion matrix, we can find that the accuracy of number 0,1,2,9 are super high. For most of the numbers, the accuracy are acceptable.And we got a total accuracy rate of 0.7172775, and total error rate is 0.2827225.

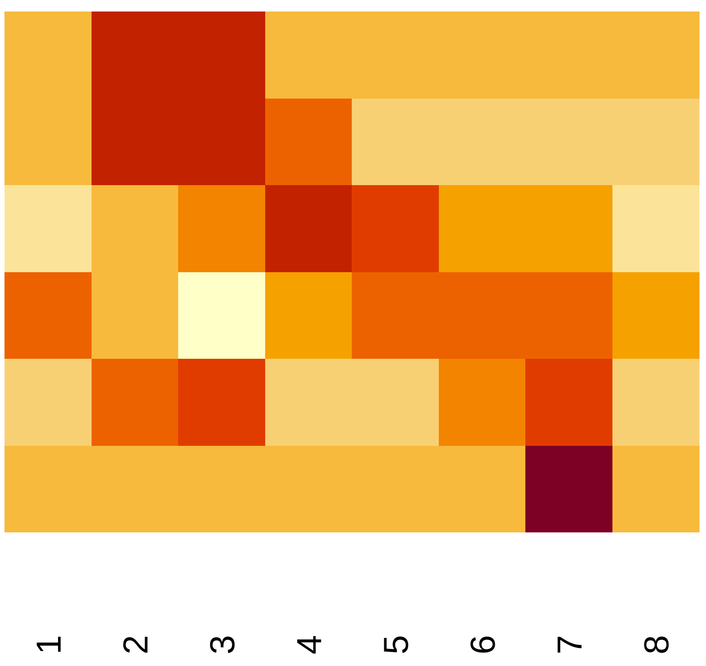
(3)

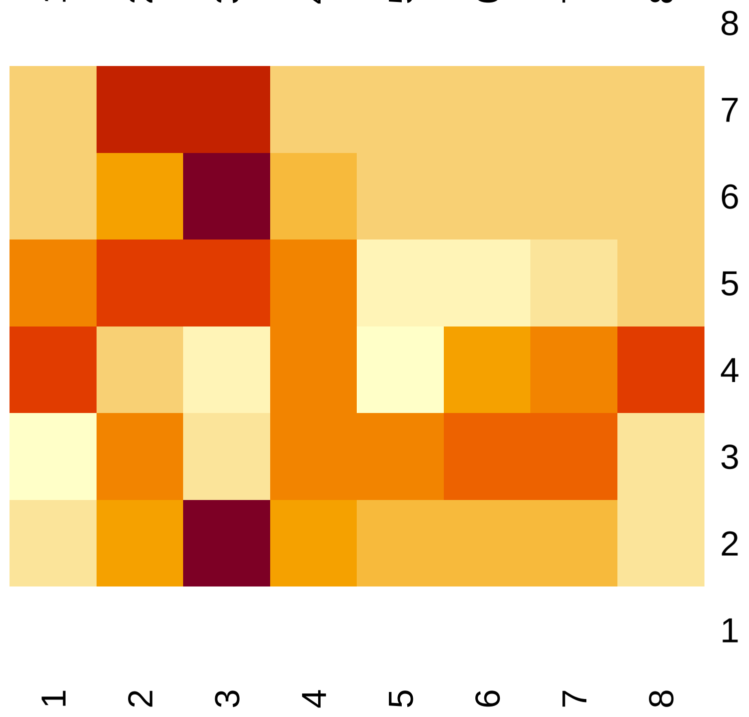
We use the following code to generate heatmaps. By show 205 heat images one by one, we found first 2 images(index 18,50) looks like 8, but next 3 images(index 1, 2,55) not like 8 at all.

The image is as follows.The code attached as appendix.





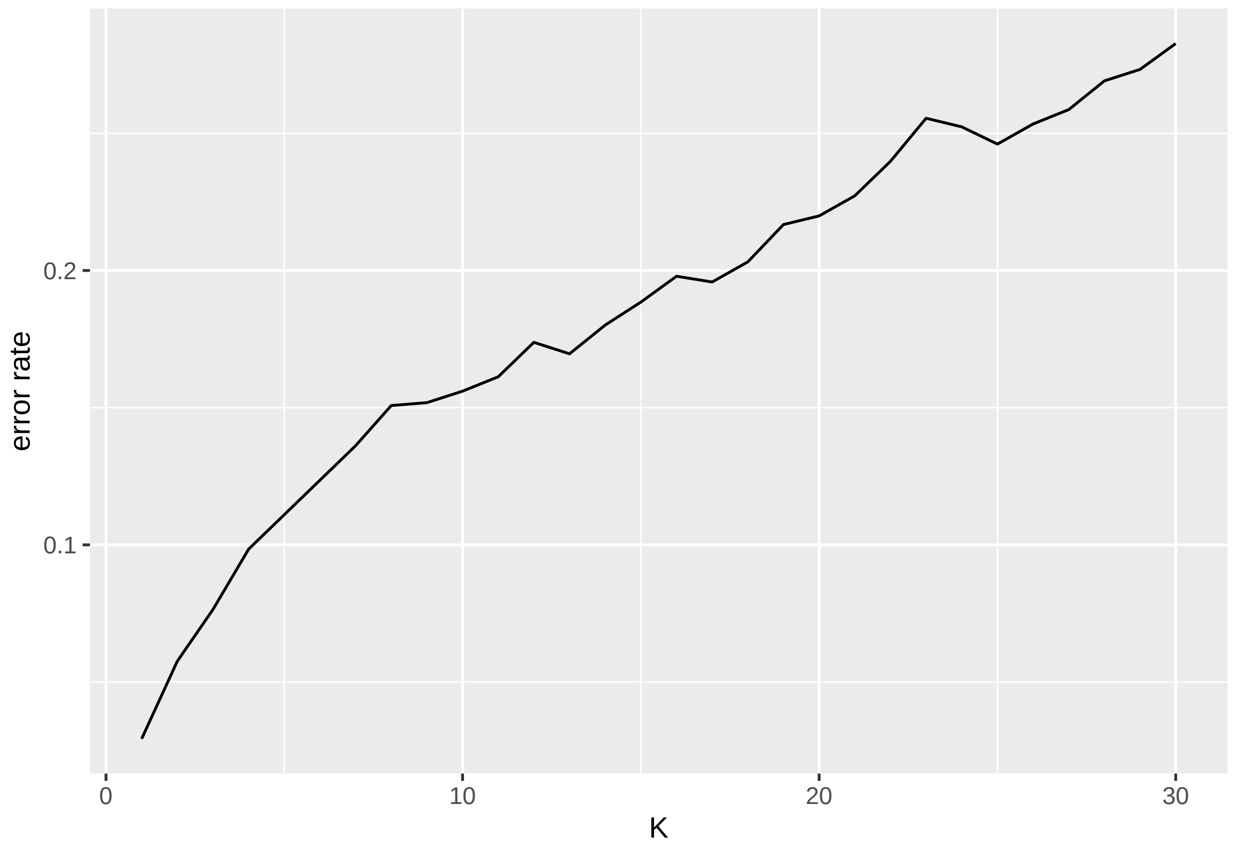




(4)

We fit the KNN with k = 1 to 30, we plot the error rates as follows, the code is attached in appendix.

When K increase, the model not become very complex,it seems it does not change a lot when k >= 2 And according to the graph, we can know that the optimal K is 1.



(5)

The cross-entropy is defined as $-\sum_{i=1}^{N}\sum_{m=1}^{M} I(Y_i = C_m) * log\hat{p}(y_i = C_m)$.

So we will calculate it using the following code(Not implemented).

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")
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, ]

# normalize data.
train_set <- as.data.frame(lapply(train_set, normalize))
test_set <- as.data.frame(lapply(test_set, normalize))
```

(2)

We create a linear model using following code, we get the parameters as show below.

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

# apply linear regression
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)
model
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ ., data = train_set)
##
## Coefficients:
##   (Intercept)       subject.            age            sex      test_time
##      0.041153      -0.020612      -0.045811       0.029830      -0.002456
##    total_UPDRS       Jitter...     Jitter.Abs.     Jitter.RAP    Jitter.PPQ5
##      1.012833       0.679831      -0.231111       3.062194      -0.149610
##     Jitter.DDP        Shimmer     Shimmer.dB.   Shimmer.APQ3   Shimmer.APQ5
##     -3.477175       0.404012      -0.059469      24.111950      -0.309738
## Shimmer.APQ11    Shimmer.DDA            NHR            HNR           RPDE
##      0.253228     -24.256718       0.004315      -0.008594      -0.054478
##            DFA            PPE
##     -0.007923       0.100989
```

And we get test MSE = 0.005370424 we know that Shimmer.APQ3 and Shimmer.DDA contribute significantly to the model

(3)

Functions implemented as below.

```
############################  Assignment 2.3 #####################################
# Loglikelihood function
loglikelihood <- function(theta, sigma) {
```

```r
  n <- length(Y)
  log_likelihood_values <- -0.5 * (log(2 * pi * sigma^2) + ((Y - theta %*% X) / sigma)^2)
  return(total_log_likelihood <- sum(log_likelihood_values))
}

# ridge
ridge <- function(param) {
  param_length <- length(param)
  theta <- param[1:param_length-2]
  sigma <- param[param_length-1]
  lambda <- param[param_length]
  loglikelihood_result <- loglikelihood(theta, sigma)
  ridge_penalty <- lambda * sum(theta^2)
  return(loglikelihood_result - ridge_penalty)
}

# ridgeopt
ridgeopt <- function(theta, sigma, lambda) {
  size_theta <- length(theta)
  param <- rep(0,size_theta+2)
  for(i in 1:size_theta){
    param[i] <- theta[i]
  }
  param[size_theta + 1] <- sigma
  param[size_theta + 2] <- lambda
  ridge_result <- optim(par = param, fn = ridge, method="BFGS")

}

# df
df <- function(X,lambda) {
  n <- nrow(X)
  # calculate hat_matrix
  hat_matrix <- X %*% solve(t(X) %*% X + lambda * diag(ncol(X)))
  # get degree of the hat_matrix
  df_ridge <- sum(diag(hat_matrix))
  return(df_ridge)
}
```

(4)

Functions implemented as below.

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

#sigma <- 0.01
#theta <- rep(1,21)
#X <- train_set[]
#lambda <- 1
#ridgeopt(theta,sigma,lambda)

#lambda <- 100
#ridgeopt(theta,sigma,lambda)
```

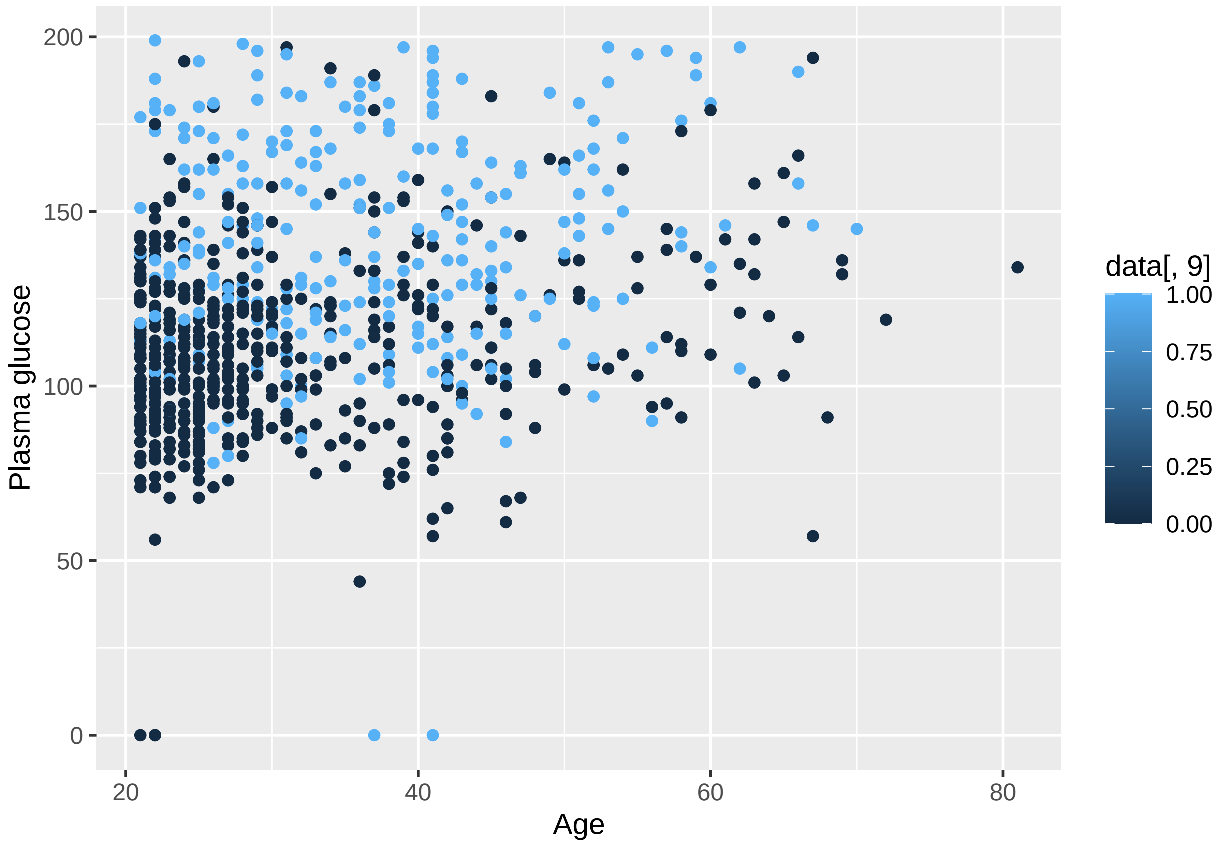
```
#lambda <- 1000
#ridgeopt(theta,sigma,lambda)
```

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

Answer:

(1)

```
data <- read.csv("pima-indians-diabetes.csv")

age <- data[, 8]
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")
```



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.

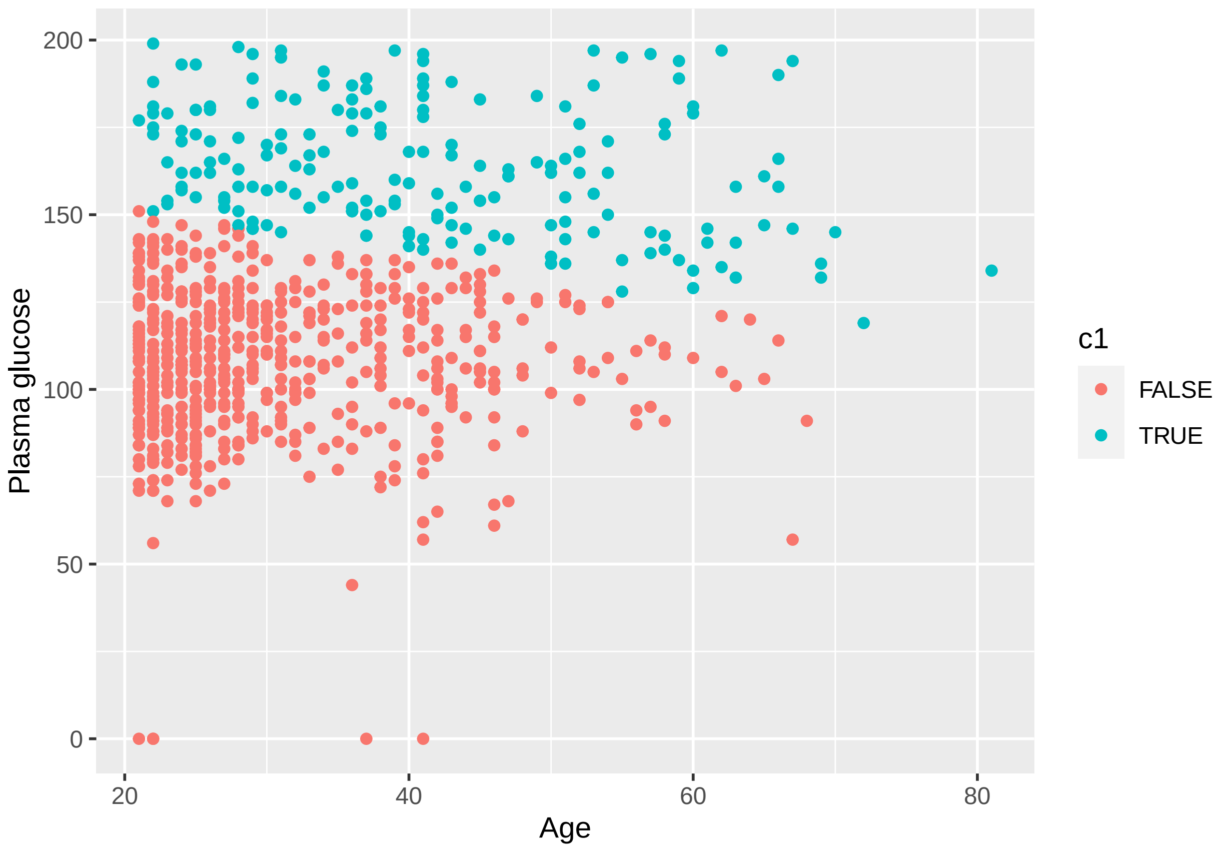
```r
model <- glm(diabetes ~ plasma + age, fam = binomial(link="logit"), data = data)
summary(model)
```

```
##
## 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
```

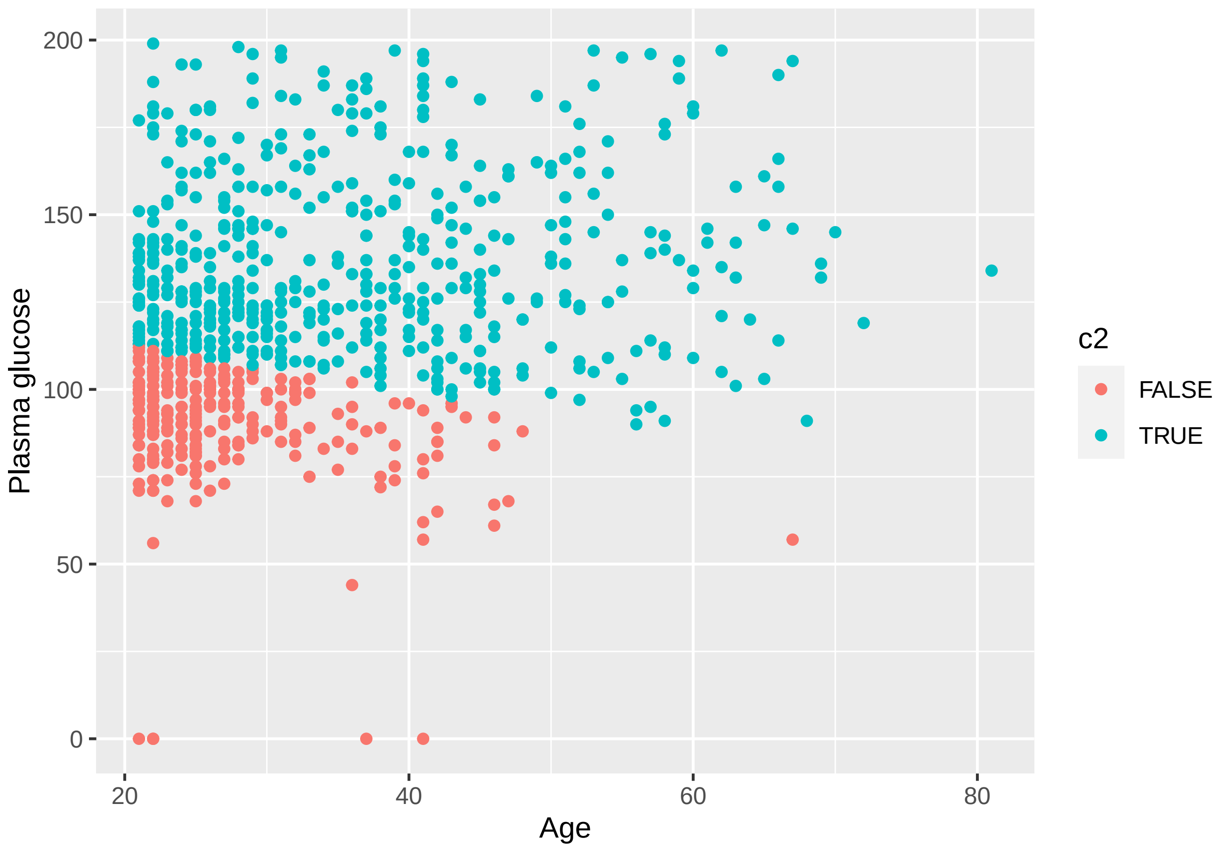
```r
ggplot2::ggplot(df, ggplot2::aes(age, plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = c1))+
  ggplot2::labs(x = "Age", y="Plasma glucose")     # T = Diabetic, F = Non-diabetic
```



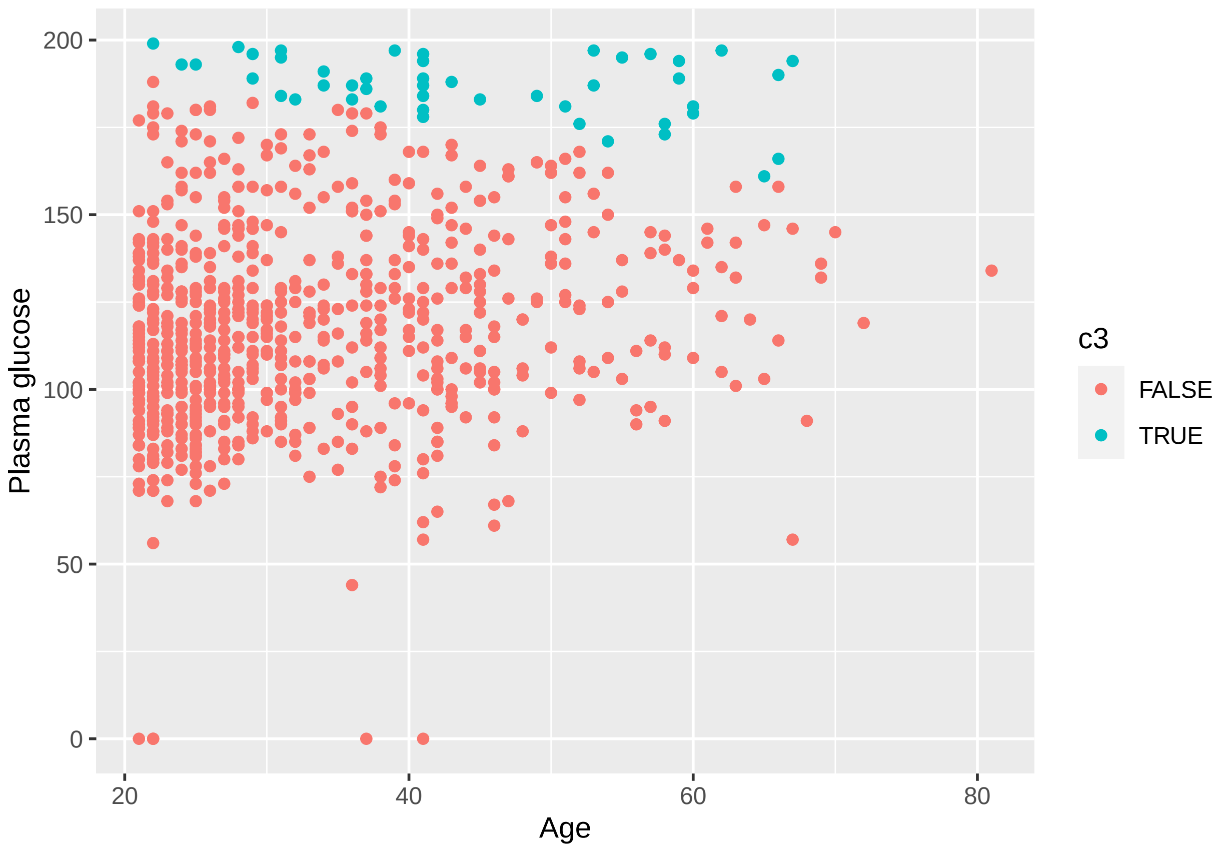
(4)

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.

```
# 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
# 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")
```



(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.

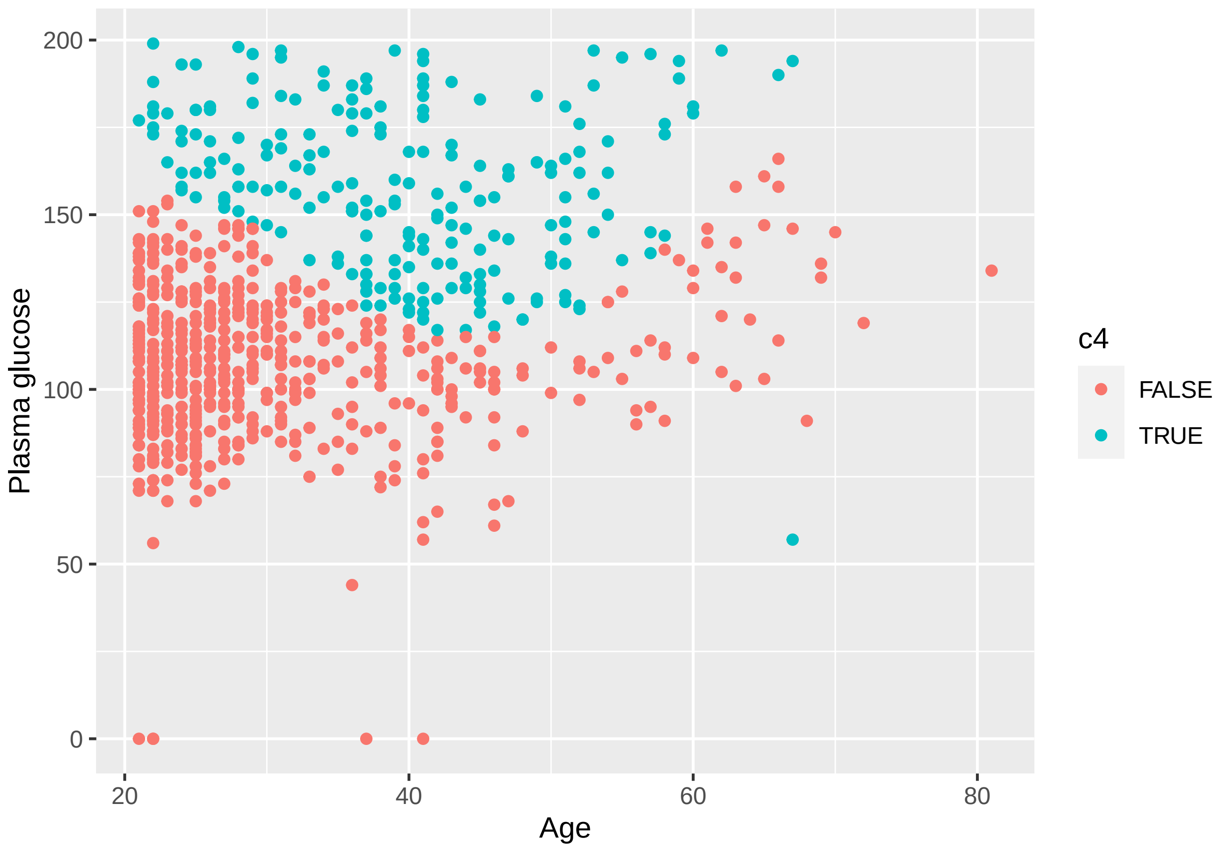
```r
# Adding new variables, logit function with r= 0.5

z1 <- data$plasma^4
z2 <- data$plasma^3 * data$age
z3 <- data$plasma^2 * data$age^2
z4 <- data$plasma^1 * data$age^3
z5 <- data$age^4

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

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

summary(newmodel)
```

```
##
## 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
```

```r
c4 <- fitted(newmodel) >= .5

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

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

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



Plasma glucose
200
150
100
50
0
20
40
60
80
Age
c4
FALSE
TRUE

Appendix: All code for this report

```r
###########################  Init code For Assignment 1 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(kknn)
######################  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, test = .25, validate = .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 ##########################################
# calculate error rate for k = 30
k_value <- 30
train_kknn <- train.kknn(label_value ~ .,
                         data = train_set,
                         kmax = k_value,
                         kernel = "rectangular")

predict_data <- predict(train_kknn,newdata=test_set)

# generate confusion matrix
confusion_matrices <- table(round(predict_data), test_set$label_value)

# calculate accuracy
accuracy <- sum(diag(confusion_matrices)) / sum(confusion_matrices)

# calculate error rate
error_rate <- 1 - accuracy
```

```r
######################  Assignment 1.3 ##########################################
# Find all the 8 images
eight_images <- train_set[train_set$label_value == 8, ]

# get the number of 8 images in training data, we got 205 images
eight_images_number <- nrow(eight_images)

image_number  <- 18
eight_images_matrix <- matrix(c(as.numeric(eight_images[image_number, 1:64])), ncol = 8)
heatmap(x = eight_images_matrix, Rowv = NA, Colv = NA)

image_number  <- 50
eight_images_matrix <- matrix(c(as.numeric(eight_images[image_number, 1:64])), ncol = 8)
heatmap(x = eight_images_matrix, Rowv = NA, Colv = NA)

image_number  <- 1
eight_images_matrix <- matrix(c(as.numeric(eight_images[image_number, 1:64])), ncol = 8)
heatmap(x = eight_images_matrix, Rowv = NA, Colv = NA)

image_number  <- 2
eight_images_matrix <- matrix(c(as.numeric(eight_images[image_number, 1:64])), ncol = 8)
heatmap(x = eight_images_matrix, Rowv = NA, Colv = NA)

image_number  <- 55
eight_images_matrix <- matrix(c(as.numeric(eight_images[image_number, 1:64])), ncol = 8)
heatmap(x = eight_images_matrix, Rowv = NA, Colv = NA)

######################  Assignment 1.4 ##########################################
# calculate error rate for different k values
error_rates_valid <- rep(0, 30)

for(k_value in 1:30){
  # apply kknn function
  valid_kknn <- kknn::kknn(label_value ~ .,
                          train_set,
                          valid_set,
                          k = k_value,
                          kernel = "rectangular")

 predict_data_valid <- predict(valid_kknn,newdata=valid_set)

  # generate confusion matrix
 confusion_matrices_valid <- table(round(predict_data_valid), valid_set$label_value)

 # calculate accuracy
 accuracy_valid <- sum(diag(confusion_matrices_valid)) / sum(confusion_matrices_valid)

  # calculate error rate
  error_rates_valid[k_value] <- 1 - accuracy_valid

}

# plot the error rate graph
```

```r
k <- 1:30

error_rate_data <- data.frame(k, error_rates_valid)
ggplot2::ggplot() +
  ggplot2::geom_line(error_rate_data,mapping = ggplot2::aes(x=k,y=error_rates_valid)) +
  ggplot2::labs(x = "K",y="error rate")
#error_rates_cross <- rep(30, 0)

#for(k_value in 1:30){
#  # apply kknn function
#  train_kknn <- train.kknn(label_value ~ .,
#                         data = train_set,
#                         kmax = k_value,
#                         kernel = "rectangular")

#  predict_data_test <- predict(train_kknn,newdata=test_set)

#  confusion_matrices_test <- table(round(predict_data_test), test_set$label_value)

  # calculate cross-entropy
  #ce <- 0
  #for(i in 1:10){
  #  total_count <- sum(confusion_matrices_test[i])
  #  for(j in 1:10){
  #    p <- confusion_matrices_test[i,j] / total_count
  #    print(p)
  #    ce <- ce + log(p) * confusion_matrices_test[i,j]
  #
  #  }
  #}
  #error_rates_cross[k_value] <- ce * (-1)
#}

#k <- 1:30

#error_rate_data <- data.frame(k,error_rates_cross)
#ggplot2::ggplot() +
#  ggplot2::geom_line(error_rate_data,mapping=ggplot2::aes(x=k,y=error_rates_cross),colour="blue") +
#  ggplot2::labs(x = "K",y="error rate")

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

###########################  Common Functions ##################################
# normalize data
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
###########################  Assignment 2.1 ####################################
# Load the data
data <- read.csv("parkinsons.csv")
row_num <- nrow(data)
```

```r
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, ]

# normalize data.
train_set <- as.data.frame(lapply(train_set, normalize))
test_set <- as.data.frame(lapply(test_set, normalize))
###########################  Assignment 2.2 ####################################
# Linear regression model

# apply linear regression
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)
model
###########################  Assignment 2.3 ####################################
# Loglikelihood function
loglikelihood <- function(theta, sigma) {
  n <- length(Y)
  log_likelihood_values <- -0.5 * (log(2 * pi * sigma^2) + ((Y - theta %*% X) / sigma)^2)
  return(total_log_likelihood <- sum(log_likelihood_values))
}

# ridge
ridge <- function(param) {
  param_length <- length(param)
  theta <- param[1:param_length-2]
  sigma <- param[param_length-1]
  lambda <- param[param_length]
  loglikelihood_result <- loglikelihood(theta, sigma)
  ridge_penalty <- lambda * sum(theta^2)
  return(loglikelihood_result - ridge_penalty)
}

# ridgeopt
ridgeopt <- function(theta, sigma, lambda) {
  size_theta <- length(theta)
  param <- rep(0,size_theta+2)
  for(i in 1:size_theta){
    param[i] <- theta[i]
  }
  param[size_theta + 1] <- sigma
  param[size_theta + 2] <- lambda
  ridge_result <- optim(par = param, fn = ridge, method="BFGS")

}

# df
df <- function(X,lambda) {
  n <- nrow(X)
  # calculate hat_matrix
  hat_matrix <- X %*% solve(t(X) %*% X + lambda * diag(ncol(X)))
  # get degree of the hat_matrix
  df_ridge <- sum(diag(hat_matrix))
  return(df_ridge)
}
###########################  Assignment 2.4 ####################################

#sigma <- 0.01
#theta <- rep(1,21)
#X <- train_set[]
#lambda <- 1
#ridgeopt(theta,sigma,lambda)

#lambda <- 100
#ridgeopt(theta,sigma,lambda)

#lambda <- 1000
#ridgeopt(theta,sigma,lambda)

###########################  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]
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")
```

```r
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$age^3
z5 <- data$age^4

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

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

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")
```