Machine Learning Computer Lab 1 (Group A7)

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

2024-03-31

Statement of Contribution

For Lab 1, we decided to split the three assignments equally, Qinyuan completed Assignment 1, Satya completed Assignment 2 and Daniele completed Assignment 3, after which, for verification's sake, we completed each other's assignments as well and validated our findings. The report was also compiled by three of us, with each handling their respective assignments.

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

Answer:

(1)

We divide the data into training, validation and test sets according to the ratios 0.5, 0.25 and 0.25. Please check the appendix for the code.

(2)

The code contains a function call to kknn with k=30, kernel = "rectangular". We calculate the confusion matrices and the misclassification errors of the training and test data set as follows, the code is attached in the appendix. As the data shown below, the diagonal of the confusion matrix is the number of correct predictions. The most correct prediction is digit 0, and the most incorrect prediction is digit 9. The misclassification rate of the training data set is 4.5%, and the misclassification rate of the test data set is 5.3%.

Table 1: Confusion matrix for training data set

	0	1	2	3	4	5	6	7	8	9
0	202	0	0	0	1	0	0	0	0	1
1	0	179	1	0	3	0	2	3	10	3
2	0	11	190	0	0	0	0	0	0	0
3	0	0	0	185	0	1	0	0	2	5
4	0	0	0	0	159	0	0	0	0	2
5	0	0	0	1	0	171	0	0	0	0
6	0	0	0	0	0	0	190	0	2	0
7	0	1	1	1	7	1	0	178	0	3
8	0	1	0	0	1	0	0	1	188	3
9	0	3	0	1	4	8	0	0	2	183

Table 2: Confusion matrix for test data set

	0	1	2	3	4	5	6	7	8	9
0	77	0	0	0	0	0	0	0	0	0
1	0	81	0	0	0	1	0	0	7	1
2	0	2	98	0	0	1	0	0	0	1
3	0	0	0	107	0	0	0	1	1	1
4	1	0	0	0	94	0	0	0	0	0
5	0	0	0	2	0	93	0	0	0	0
6	0	0	0	0	2	2	90	0	0	0
7	0	0	0	0	6	1	0	111	0	1
8	0	0	3	1	2	0	0	0	70	0
9	0	3	0	1	5	5	0	0	0	85

```
## Misclassification rate of training data set is  0.04500262
```

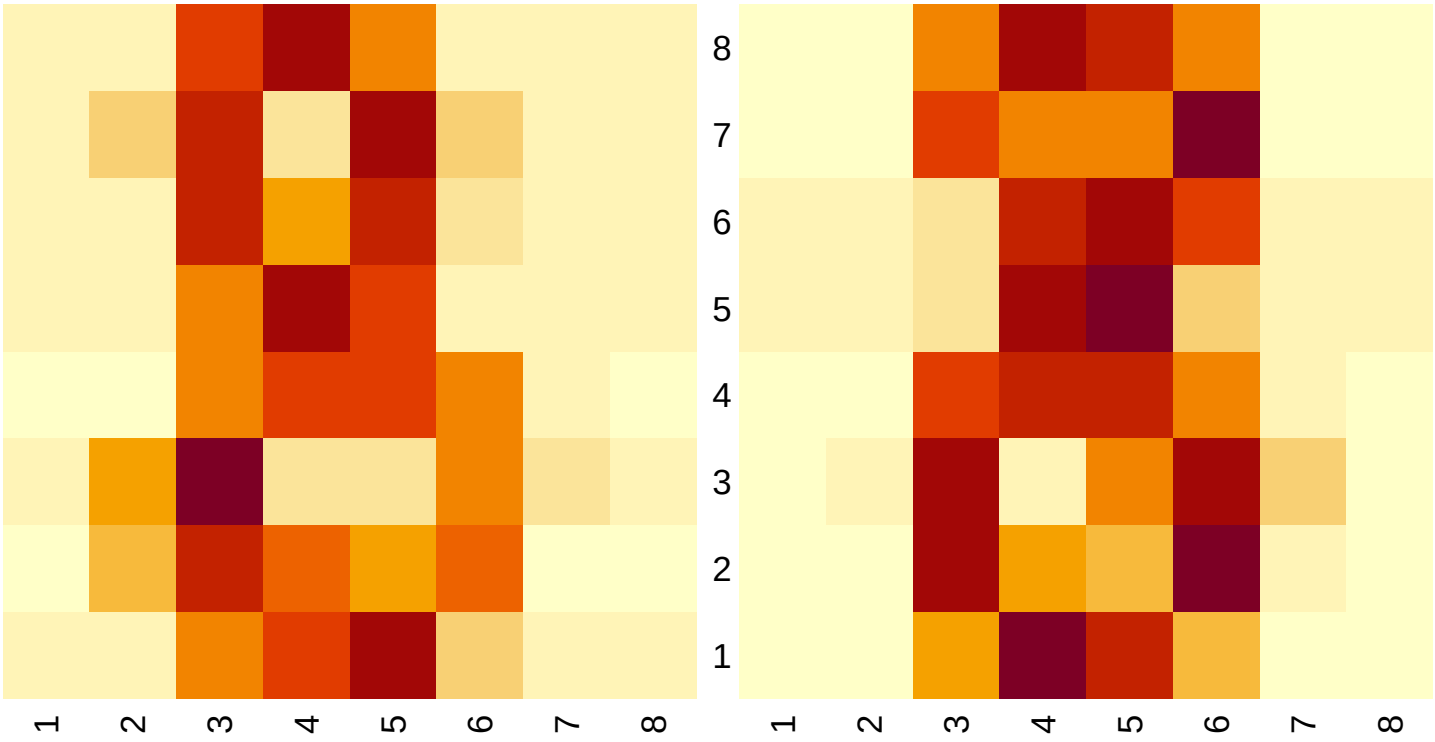
```
## Misclassification rate of test data set is  0.05329154
```

(3)

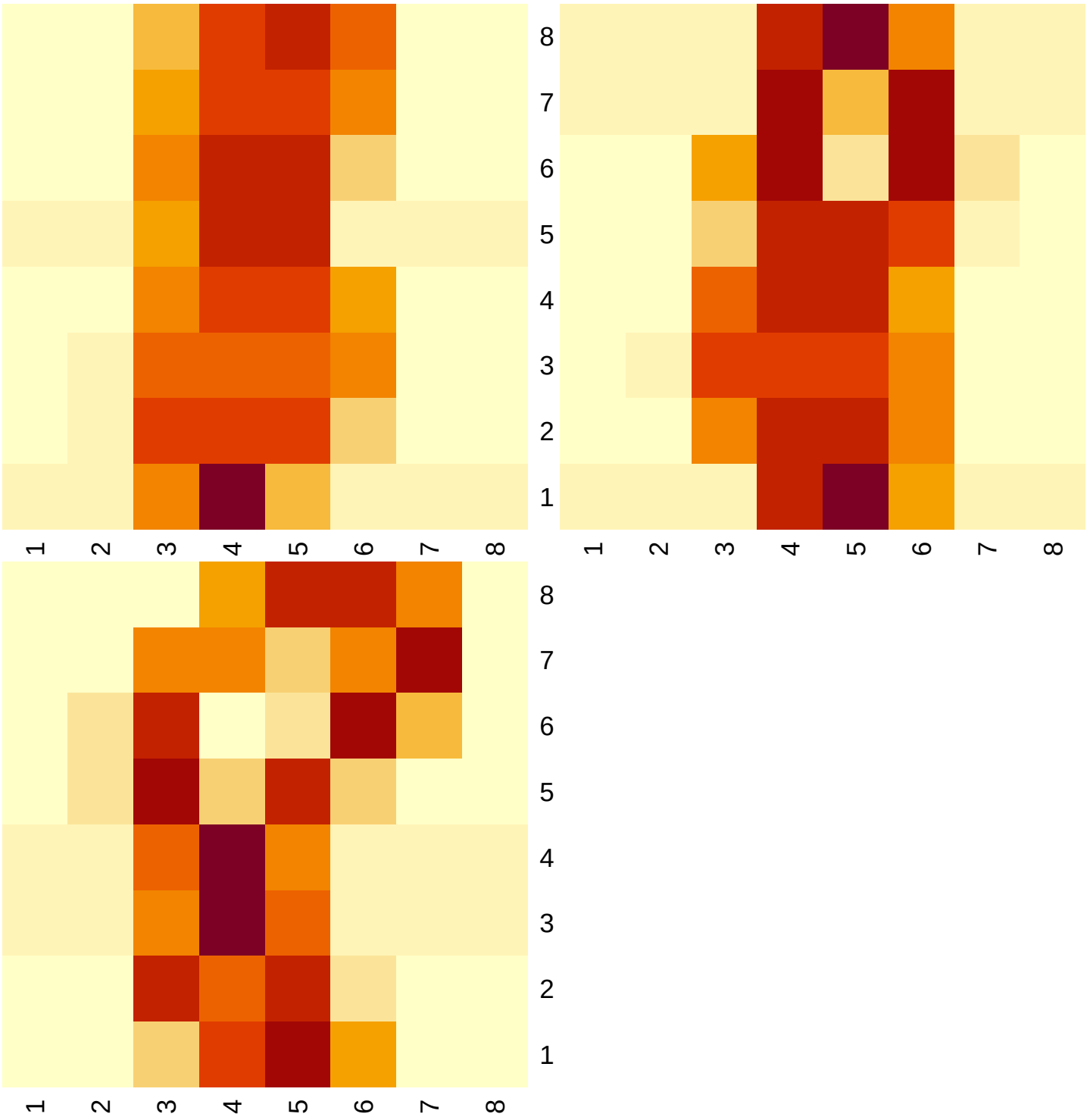
We use prob attribute to find out the easy/hard-to-identify images of label 8. The image samples are listed below.

As we can see from the heatmap, the easy-to-identify images are relatively clear and have a clear pattern, while the hard-to-identify images especially the 1st and 3rd ones are very hard to identify even for human beings.

```
## The following 2 images are easy to identify
```




```
## The following 3 images are hard to identify
```



(4)

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

When K is high, since points are related to more neighbors, we can say that the model becomes less complex. When K is small, the model becomes more complex.

From the plot below, we can find that the gap between the train and validation misclassification rate increases

after K=7.

When K = 3 or 4, it will generate the smallest misclassification rate on the validation data set, however, when k=3, the misclassification rate on the training data set is smaller, so K = 3 is our optimal K.

We set k=3, and training the model and calc models, we got misclassification = 2.4%, which is much better than k = 30.

We also calc the misclassification rate on test data using k=4, we got misclassification = 2.5%.

and when K=1 and 2, the misclassification error rate of training data is 0.

So the model here is pretty good than what we have in 1.2 which is 5.3% on test data.

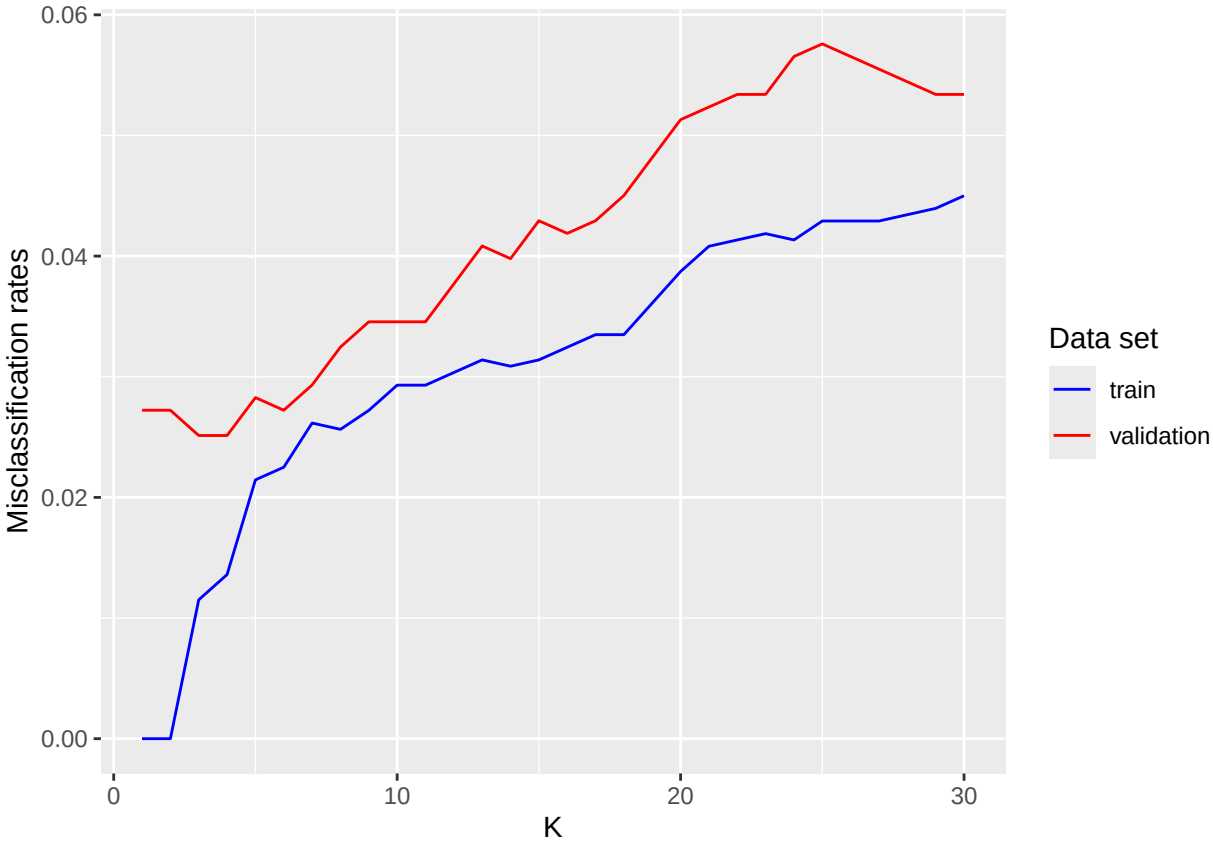


Table 3: Confusion matrix for test data set using optimal K=3

	0	1	2	3	4	5	6	7	8	9
0	77	0	0	0	0	0	0	0	0	0
1	0	81	0	0	0	1	0	0	7	1
2	0	2	98	0	0	1	0	0	0	1
3	0	0	0	107	0	0	0	1	1	1
4	1	0	0	0	94	0	0	0	0	0
5	0	0	0	2	0	93	0	0	0	0
6	0	0	0	0	2	2	90	0	0	0
7	0	0	0	0	6	1	0	111	0	1
8	0	0	3	1	2	0	0	0	70	0
9	0	3	0	1	5	5	0	0	0	85

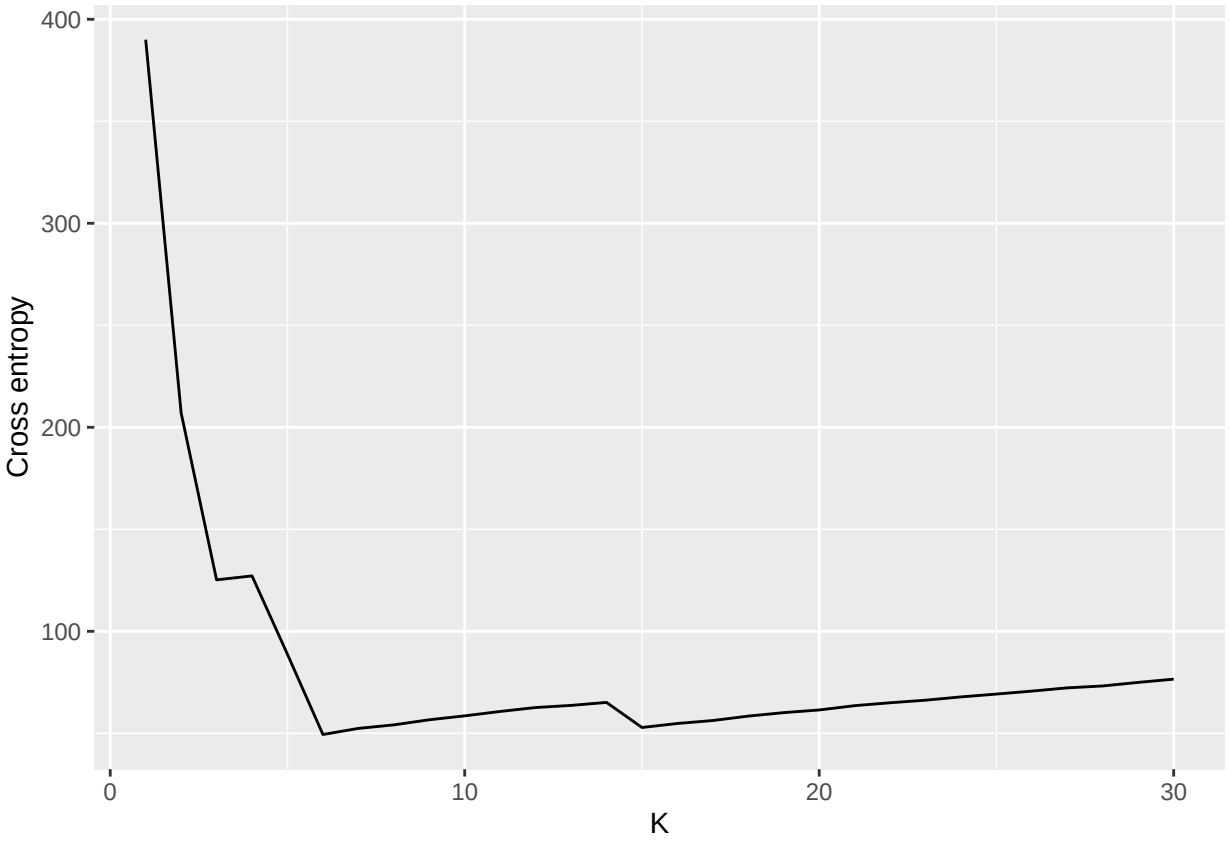
```
## Misclassification rate of test data set using optimal K=3 is  0.02403344
```

(5)

When K=6 we get the minimal cross entropy on test data, so the optimal K = 6.

If a model is good according to the cross entropy error, it means it predicted the correct class with high probability. When it guesses the wrong class, it will have a low probability.

The cross-entropy loss function is good as it utilizes the chance of the classification instead of the predicted value(class) which means low probability, say 50% for a class, will be treated differently with another one with a 90% probability for a class.



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

Answer:

(1)

We read the data, scale them, then divide the data into training and test data (60/40). Code in the appendix.

(2)

Training error and test error are as follows. We filter p_val < 0.05 as the significant factors, which are listed below.

```
## train mse is  0.873147
```

```
## test mse is  0.9297021
```

```
##                      p_val
## DFA           6.566828e-43
```

```
## PPE           6.750371e-12
## HNR           6.450884e-11
## Shimmer.APQ11 6.365957e-07
## Jitter.Abs.   3.316870e-05
## NHR           4.849817e-05
## Shimmer.APQ5  6.690868e-04
## Shimmer       4.054884e-03
```

(3)

The loglikelihood function has the following formula:

$$ln(p(y|X;\theta) = -\frac{n}{2}ln(2pi\sigma_{\varepsilon}^2) - \frac{1}{2\sigma_{\varepsilon}^2}\sum_{i=1}^{n}(\theta^T x_i - y_i)^2$$

According to the statement of the question, in the ridge function, we need to adds up a Ridge penalty to the minus of log-likelihood, which is as follows

$$ridge\_penalty - loglikelihood() = \lambda\sum(\theta^2) - loglikelihood()$$

RidgeOpt will use the optim function to find the optimal theta and sigma.

The df function has the following formula:

$$df(\hat{(}y)) = \frac{1}{\sigma^2}\sum_{i=1}^{n} cov(\hat{y_i}, y_i)$$

Functions implemented and listed in the appendix.

(4)

We use the function implemented in section 2.3 and calculate the optimal theta for $\lambda = 1, 100, 1000$.

θ are as follows:

```
## par value when lambda = 1 is
```

```
##  [1]  0.172925928 -0.168816137 -0.008761781 -0.067751148 -0.007506097
##  [6]  0.533145735 -0.144788327 -0.150099591 -0.373649472  0.311085590
## [11] -0.150581950 -0.184531493 -0.238334048  0.004998807 -0.276040090
## [16]  0.228528944
```

```
## par value when lambda = 100 is
```

```
##  [1]  0.04710354 -0.12994314  0.01583158 -0.02610922  0.01590333  0.03755943
##  [7]  0.02145389 -0.07634392 -0.10051651  0.22455993 -0.07638435 -0.14203382
## [13] -0.18294343  0.02774281 -0.24235354  0.21607987
```

```
## par value when lambda = 1000 is
```

```
##  [1]  0.006020365 -0.042161505 -0.003326839 -0.006841769 -0.003319359
##  [6]  0.005608257  0.012932840 -0.017988550 -0.010265955  0.073743613
## [11] -0.017991478 -0.038404615 -0.080388968  0.045516365 -0.128695109
## [16]  0.107032140
```

As we can see from the result above, when λ increases to a large number, say 1000, the value gets relatively smaller than when λ is small.

According to the formula of the ridge regression, when λ increases, ridge_penalty will increase, then this value will be used to calculate the optimal θ when calling optim function.

MSE and df information are as follows.

```
## train mse is  0.873147
```

```
## test mse is  0.9297021
```

	$\lambda = 1$	$\lambda = 100$	$\lambda = 1000$
mse_train	0.8732772	0.8790498	0.9145347
mse_test	0.9290341	0.9263156	0.9471367
df	0.1206838	0.1025754	0.0564014

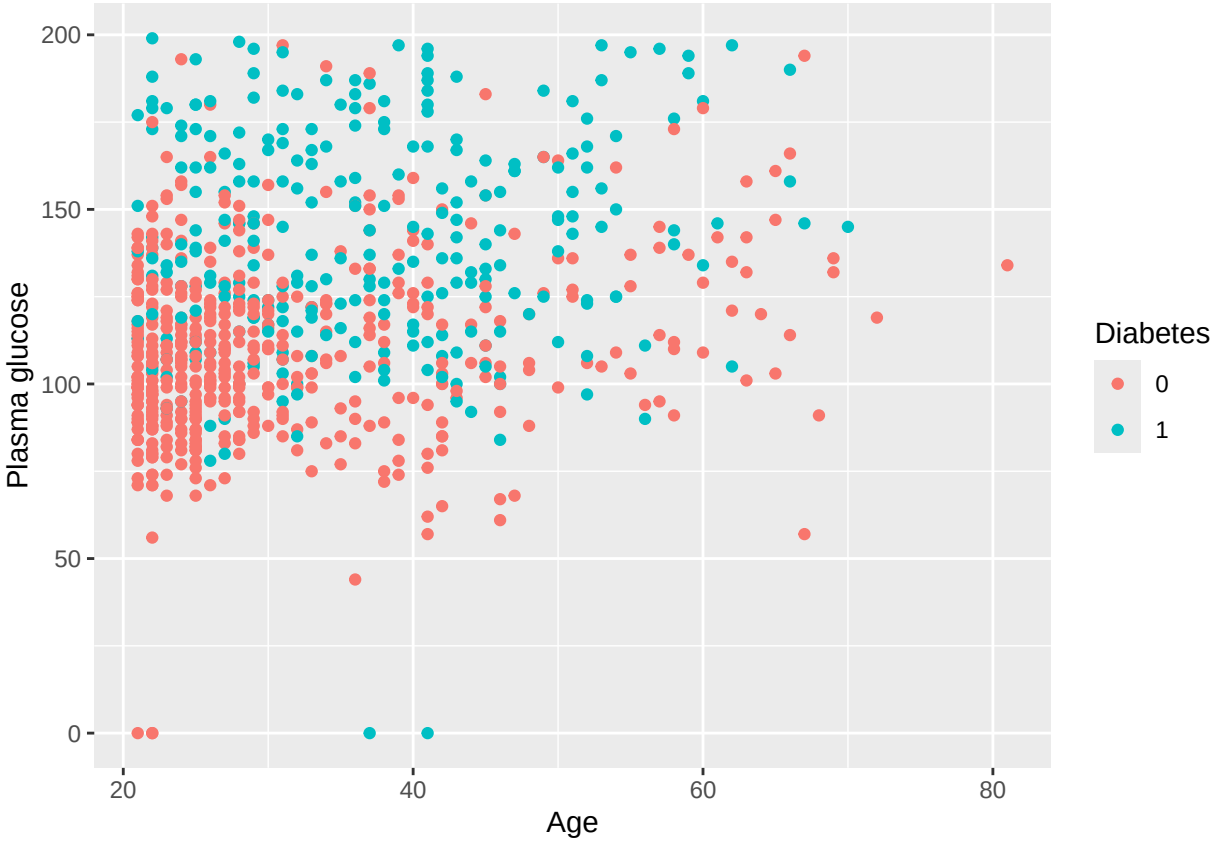
According to the table above, we find that when λ increases, training mse will increase and df will decrease at the same time.

Since our ridge model is trained using training data, a deviation value will be added to the model. when a large λ is used, it will oversimplify the model and some information will lost. we also find that mse_test get higher when λ is large, which means the model is overfitting. So in our case, $\lambda = 100$ is the best choice.

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

Answer:

(1)



The scatterplot above is for plasma glucose concentration on age colored by different diabetes values.

According to the plot, When find that when the age is relatively small, at the range of years 20-30, when the plasma glucose value is above 150, the person is more likely to have diabetes, and when the plasma glucose value is less than 125, the person is less likely to have diabetes.

While at high age, we can hardly find any pattern indicating the relationship between plasma glucose/age and diabetes.

Even if we have a logistic regression model above, since there are several points mixed at the high age, it is not easy to classify by a standard logistic regression model by only using 2 variables.

(2)-(3)

We build a logistic regression model with plasma and age as predictors. The model is as follows:

$$diabetes = -5.912449 + 0.035644 \times plasma + 0.024778 \times age$$

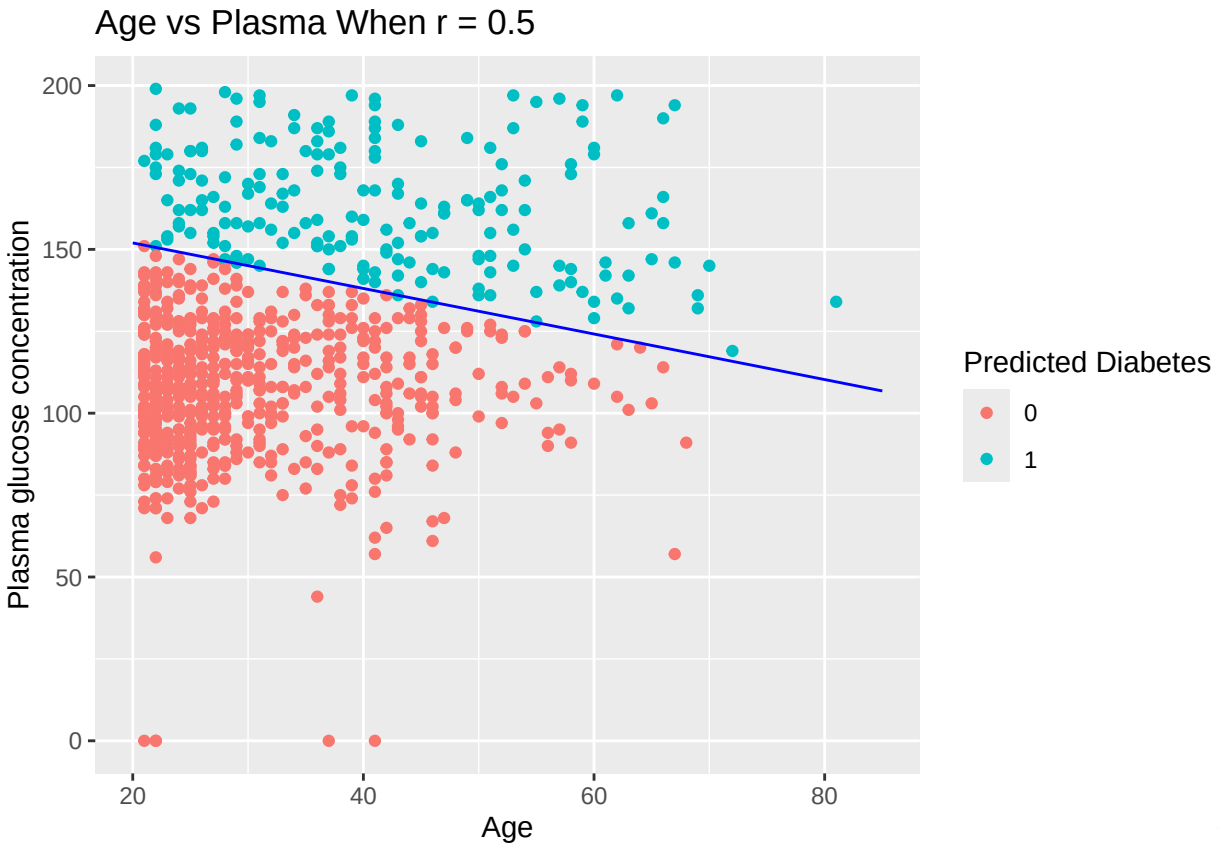
And the boundary line is as follows:

$$plasma = -(-5.912449 + 0.035644 \times age)/0.024778$$

Code in the appendix.

```
##
## Call:
## glm(formula = diabetes ~ plasma + age, family = binomial, data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.912449   0.462620  -12.78  < 2e-16 ***
## plasma       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
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 797.36  on 765  degrees of freedom
## AIC: 803.36
##
## Number of Fisher Scoring iterations: 4
```

```
## Misclassification error: 0.2630208
```



We check the graph plotted above, we find that the classification is not very accurate with a misclassification error = 0.263. Compared to the plot in 3.1, we can see that it catches the pattern we were talking about which is the low age side, but on the high age side, it is not very accurate. And the quality of the classification is not good enough.

(4)

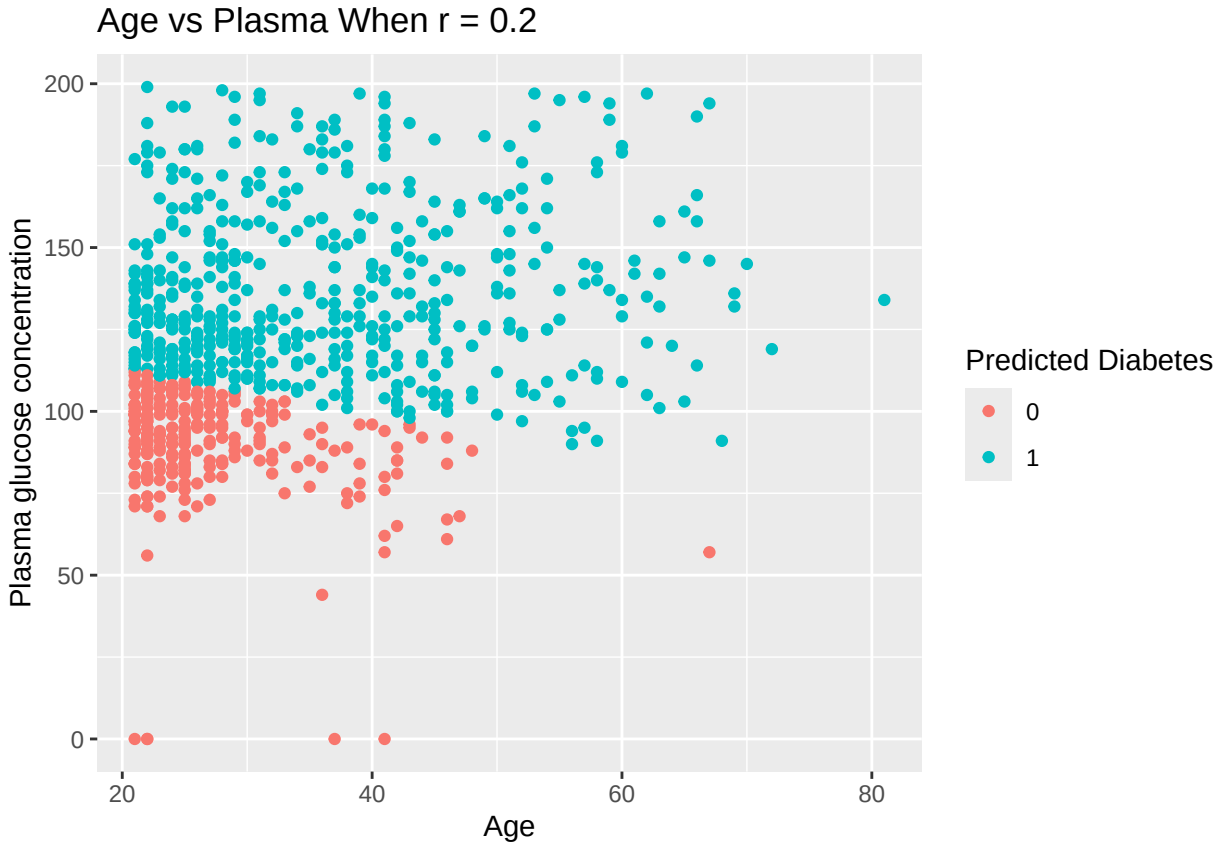
We set r= 0.2 and r=0.8 respectively, and plot the graphs as below.

According to the plots, we found that when r=0.2, the misclassification error is 0.37, and when r=0.8, the misclassification error is 0.31. The misclassification error is the much higher than r=0.5, which is 0.263.

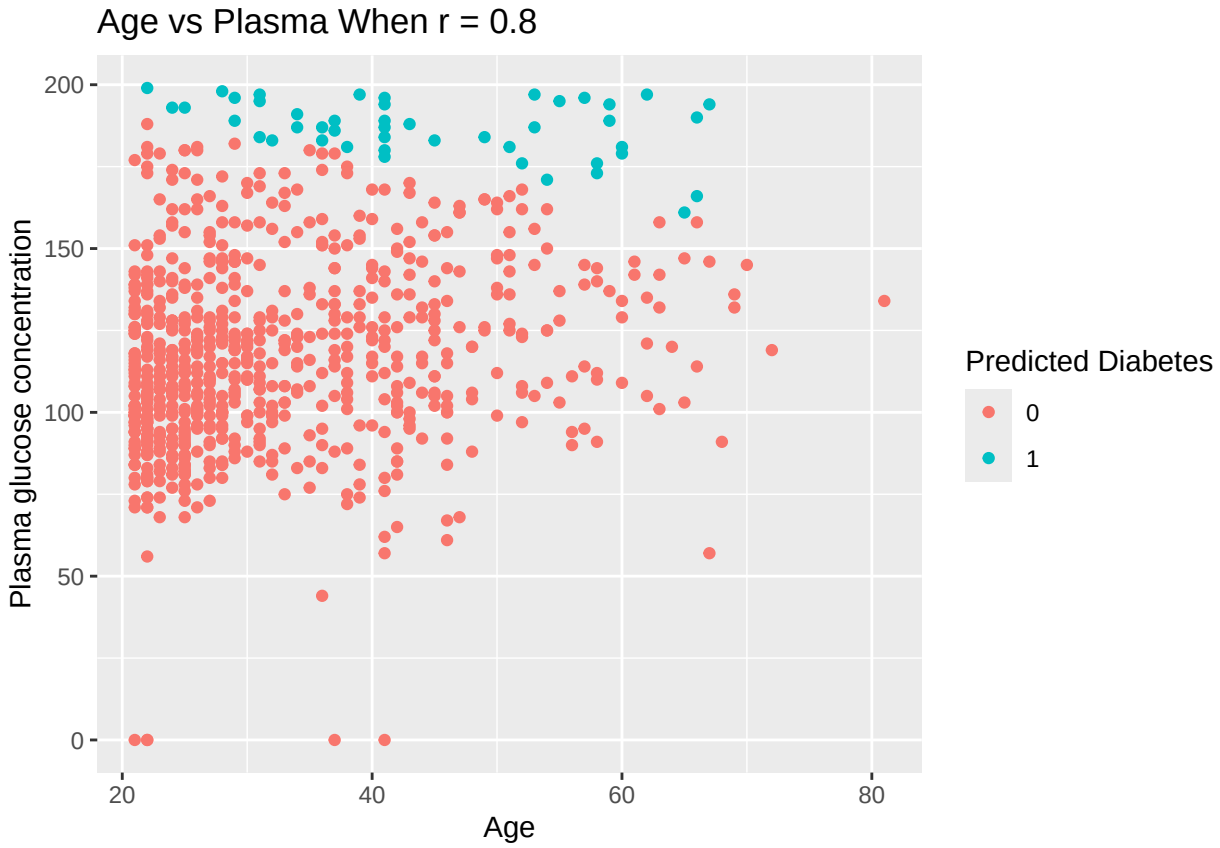
We also find that when we set r to a higher value, say 0.8, the model is more likely to predict fewer people having diabetes. while when we set r to a lower value, say 0.2, the model is more likely to predict more people having diabetes.

But in the real case, sometimes it's better to predict more people having diabetes, to make sure those high-risk people can get more attention and have body health checks.

```
## Misclassification error when r = 0.2: 0.3723958
```




```
## Misclassification error when r = 0.8: 0.3151042
```



(5)

We construct a new model by adding new variables $z_1 = plasma^4$,$z2 = plasma^3 \times age$,$z3 = plasma^2 \times age^2$,$z4 = plasma \times age^3$,$z5 = age^4$ and with default r = 0.5.

As can be seen from the result, we get a misclassification error of 0.24479, which is better than the model in 3.3 which is 0.263.

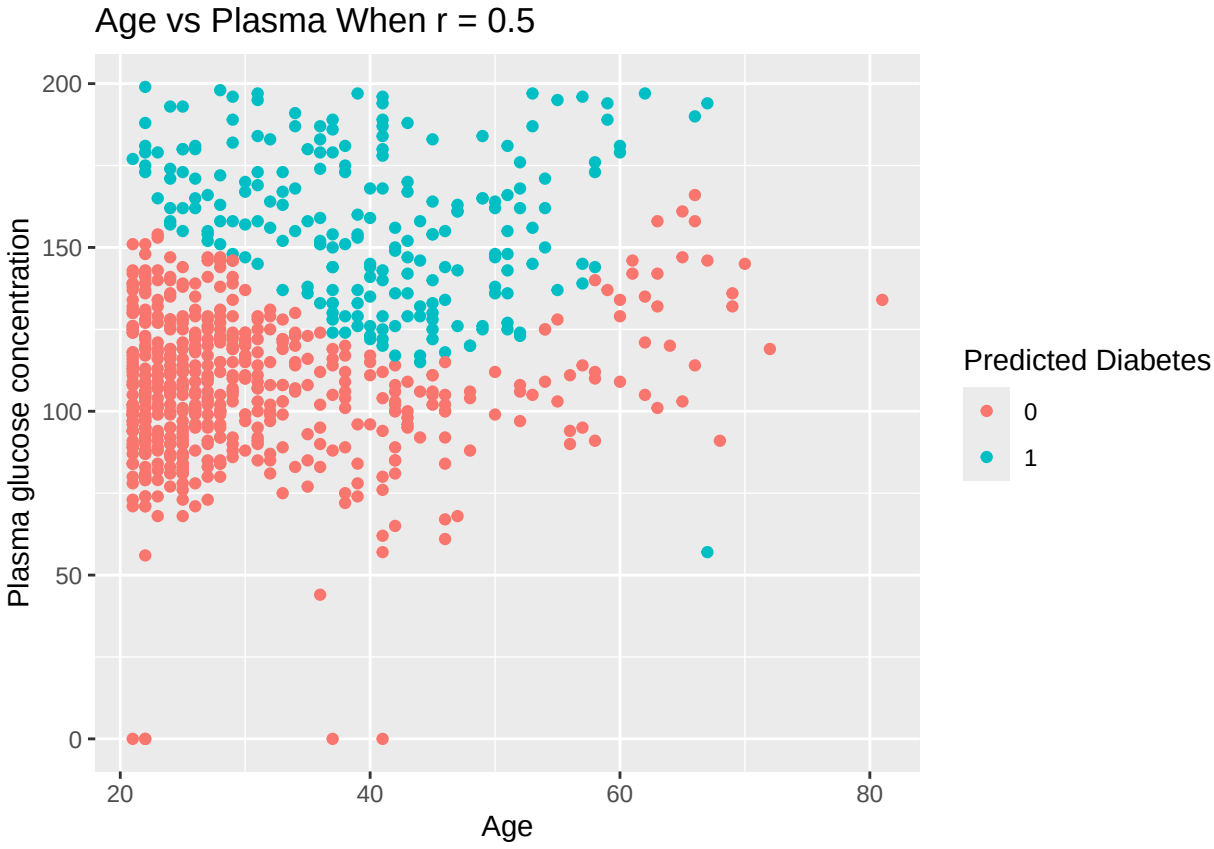
Using the basis expansion trick will make the decision boundary more complex than the previous model. And the new model is more likely to predict the correct class better. However, if the model is too complex, it may lead to overfitting.

According to the plot, we know that the boundary line in this case is a U-sharp curve.

```
##
## 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) -9.310e+00  1.129e+00  -8.243  < 2e-16 ***
## plasma       3.793e-02  9.473e-03   4.004 6.23e-05 ***
## age          1.457e-01  2.072e-02   7.031 2.05e-12 ***
## z1           1.278e-08  5.610e-09   2.278  0.02271 *
## z2          -1.780e-07  7.635e-08  -2.331  0.01976 *
## z3           8.515e-07  3.437e-07   2.478  0.01322 *
## z4          -1.698e-06  6.313e-07  -2.690  0.00715 **
## z5           8.127e-07  4.054e-07   2.004  0.04503 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 741.61  on 760  degrees of freedom
## AIC: 757.61
##
## Number of Fisher Scoring iterations: 5
```

```
## Misclassification error when r = 0.5: 0.2447917
```



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)
library(dplyr)

misclassification_func <- function(x1,x2){
  n <- length(x1)
  retuen_value <- 1 - sum(diag(table(x1,x2)))/n
}
######################  Assignment 1.1 ############################################
# read data
data <- read.csv("optdigits.csv", header=FALSE)
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, the last column is an integer represe

# 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, ]

test_val_id <- setdiff(1:row_num, train_id)

set.seed(12345)
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(as.factor(label_value) ~ .,
                         train_set,
                         train_set,
                         k = k_value,
                         kernel = "rectangular")

# generate a confusion matrix for training data
confusion_matrices_train <- table(kknn_train$fitted.values, train_set$label_value)
```

```r
kknn_test <- kknn(as.factor(label_value) ~ .,
                  train_set,
                  test_set,
                  k = k_value,
                  kernel = "rectangular")

# generate a confusion matrix for test data
confusion_matrices_test <- table(kknn_test$fitted.values, test_set$label_value)

# render results to tables

knitr::kable(confusion_matrices_train,
             caption = "Confusion matrix for training data set")
knitr::kable(confusion_matrices_test,
             caption = "Confusion matrix for test data set")

misclassification_train <- misclassification_func(kknn_train$fitted.values, train_set$label_value)
misclassification_test <- misclassification_func(kknn_test$fitted.values, test_set$label_value)

cat("Misclassification rate of training data set is ", misclassification_train, "\n")
cat("Misclassification rate of test data set is ", misclassification_test, "\n")
######################  Assignment 1.3 ##########################################

# get the probability of training data
train_prob <- kknn_train$prob

colnames(train_prob) <- c('lv_0','lv_1','lv_2','lv_3','lv_4','lv_5','lv_6','lv_7','lv_8','lv_9')

# copy training data, and we have 65 columns now
train_set_prob <- train_set

# we have 66 columns now
train_set_prob$predicted <- kknn_train$fitted.values

# add digit 8 probability to this dataset, and have 67 columns now
# we use 9 because it begin with digit 0
train_set_prob$prob <- train_prob[,9]

# order the dataset by the probability of digit 8
# begining with the smallest probability,so hard to identify

filtered_df <- dplyr::filter(train_set_prob,train_set_prob$label_value==8)
filtered_df <- filtered_df[order(filtered_df$prob),]

filtered_count <- nrow(filtered_df)

# easy image 1 (last several rows of the filtered_df)
easy_image1 <- filtered_df[filtered_count,]
easy_image1_matrix <- matrix(as.numeric(easy_image1[,c(1:64)]), ncol = 8, byrow = TRUE)

# easy image 2 (last several rows of the filtered_df)
easy_image2 <- filtered_df[filtered_count - 1,]
easy_image2_matrix <- matrix(as.numeric(easy_image2[,c(1:64)]), ncol = 8, byrow = TRUE)
```

```r
# easy image 3 (last several rows of the filtered_df)
easy_image3 <- filtered_df[filtered_count - 2,]
easy_image3_matrix <- matrix(as.numeric(easy_image3[,c(1:64)]), ncol = 8, byrow = TRUE)

# hard image 1 (first several rows of the filtered_df)
hard_image1 <- filtered_df[1,]
hard_image1_matrix <- matrix(as.numeric(hard_image1[,c(1:64)]), ncol = 8, byrow = TRUE)

# hard image 2 (first several rows of the filtered_df)
hard_image2 <- filtered_df[2,]
hard_image2_matrix <- matrix(as.numeric(hard_image2[,c(1:64)]), ncol = 8, byrow = TRUE)

# hard image 3 (first several rows of the filtered_df)
hard_image3 <- filtered_df[3,]
hard_image3_matrix <- matrix(as.numeric(hard_image3[,c(1:64)]), ncol = 8, byrow = TRUE)

cat("The following 2 images are easy to identify\n")
# draw heatmap
heatmap(x=easy_image1_matrix, Rowv=NA, Colv=NA)
heatmap(x=easy_image2_matrix, Rowv=NA, Colv=NA)

cat("The following 3 images are hard to identify\n")
heatmap(x=hard_image1_matrix, Rowv=NA, Colv=NA)
heatmap(x=hard_image2_matrix, Rowv=NA, Colv=NA)
heatmap(x=hard_image3_matrix, Rowv=NA, Colv=NA)
######################  Assignment 1.4 ##########################################
# calculate error rate for different k values
mis_rates_train <- c()
mis_rates_valid <- c()

for(k_value in 1:30){

  kknn_train <- kknn(as.factor(label_value) ~ .,
                           train_set,
                           train_set,
                           k = k_value,
                           kernel = "rectangular")
  pred_train <- kknn_train$fitted.values
  n_train <- length(train_set$label_value)
  mis_rate_train <- misclassification_func(train_set$label_value,pred_train)



  kknn_valid <- kknn(as.factor(label_value) ~ .,
                           train_set,
                           valid_set,
                           k = k_value,
                           kernel = "rectangular")
  pred_valid <- kknn_valid$fitted.values
  n_valid <- length(valid_set$label_value)
  mis_rate_valid <- misclassification_func(valid_set$label_value,pred_valid)

  # train error rate
```

```
  mis_rates_train <- c(mis_rates_train,mis_rate_train)
  # valid error rate
  mis_rates_valid <- c(mis_rates_valid,mis_rate_valid)
}

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

error_rate_data <- data.frame(k, mis_rates_train,mis_rates_valid)

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

k_value <- 3

kknn_test_optimal <- kknn(as.factor(label_value) ~ .,
                  train_set,
                  test_set,
                  k = k_value,
                  kernel = "rectangular")

# generate a confusion matrix for test data
confusion_matrices_test_optimal <- table(kknn_test_optimal$fitted.values, test_set$label_value)

knitr::kable(confusion_matrices_test,
             caption = "Confusion matrix for test data set using optimal K=3")

misclassification_test_optimal <- misclassification_func(kknn_test_optimal$fitted.values, test_set$label

cat("Misclassification rate of test data set using optimal K=3 is ", misclassification_test_optimal, "\n
###################### Assignment 1.5 ##########################################
# use cross-entropy in 1.5

# init value
k_values <- 1:30
epsilon <- 1e-15

entropies <- c()

# loop possible K values
for(k in k_values){
    kknn_valid <- kknn(as.factor(label_value) ~ .,
                           train_set,
                           valid_set,
                           k = k,
                           kernel = "rectangular")
    cross_entropy_sum = 0.0

    for(i in 0:9){
        cross_entropy_sum <- cross_entropy_sum -
```

```
            sum(
              log10(kknn_valid$prob[valid_set$label_value == i,i+1] + epsilon)
              )
    }

    entropies <- c(entropies,cross_entropy_sum)
}

error_cross_entropy_data <- data.frame(k_values,entropies)

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

############################  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",header=TRUE)

# filter data
data <- data %>%
dplyr::select(motor_UPDRS, starts_with("Jitter"),starts_with("Shimmer"),
            NHR,HNR,RPDE,DFA,PPE)

# scale data
data <- scale(data)
data = data.frame(data)

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

row_num <- nrow(data)

# 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, ]

############################  Assignment 2.2 ####################################
# Linear regression model
# apply linear regression(without intercept)
model <- lm(motor_UPDRS ~ ., data = train_set)

# predict the test value using the linear regression just created
train_pred <- predict(model, train_set)
# calculate train MSE
```

```r
train_mse <- mean((train_pred - train_set$motor_UPDRS)^2)
cat("train mse is " , train_mse , "\n")

# 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")

# we filter p_val < 0.05 as the significant factors using dplyr package
significant_df <- data.frame(p_val = summary(model)$coefficients[,4]) %>%
                  filter(p_val < 0.05) %>%
                  arrange(p_val)

print(significant_df)
##########################  Assignment 2.3 ####################################
# Loglikelihood function
loglikelihood <- function(theta, sigma) {
  n <- length(train_set[,1])
  x <- train_set[,-1]
  y <- train_set[,1]
  log_likelihood_value <- -n/2 * log(2 * pi * sigma^2,exp(1)) -
                          sum((theta %*% t(x)- y)^2) / (2 * sigma^2)
  return(log_likelihood_value)
}

# ridge
ridge <- function(theta,sigma,lambda) {
  loglikelihood_result <- loglikelihood(theta, sigma)
  ridge_penalty <- lambda * sum(theta^2)
  return(ridge_penalty - loglikelihood_result)
}

# ridgeopt
ridgeOpt <- function(lambda) {
  # Use the optim function
  result <- optim(par = numeric(length(train_set[1,-1])), fn = ridge, method = "BFGS",
                  sigma = sigma(model), lambda = lambda)
  return(result)
}

# df function
df <- function(lambda) {
  opt = ridgeOpt(lambda)
  sigma_sqr <- var(train_set[,1])
  y <- train_set[,1]
  n <- length(train_set[,1])
  y_hat <- numeric(n)
  for(i in 1:n){
    y_hat[i] <- opt$par %*% t(train_set[i,-1])
  }
  return (1 / sigma_sqr * cov(y_hat,y))
}
```

```
##########################  Assignment 2.4 ###################################
mse_new <- function(theta, subset){
  x <- subset[,-1]
  y <- subset[,1]
  n <- length(y)
  mse_value <- sum((theta %*% t(x) - y)^2) / n
  return(mse_value)
}


#lambda <- 1
result1 <- ridgeOpt(lambda = 1 )
df1 <- df(1)
mse_train_1 <- mse_new(result1$par, train_set)
mse_test_1 <- mse_new(result1$par, test_set)

#lambda <- 100
result100 <- ridgeOpt(lambda = 100 )
df100 <- df(100)
mse_train_100 <- mse_new(result100$par, train_set)
mse_test_100 <- mse_new(result100$par, test_set)

#lambda <- 1000
result1000 <- ridgeOpt(lambda = 1000 )
df1000 <- df(1000)
mse_train_1000 <- mse_new(result1000$par, train_set)
mse_test_1000 <- mse_new(result1000$par, test_set)

cat("par value when lambda = 1 is \n")
print(result1$par)
cat("par value when lambda = 100 is \n")
print(result100$par)
cat("par value when lambda = 1000 is \n")
print(result1000$par)

cat("train mse is " , train_mse , "\n")
cat("test mse is " , test_mse , "\n")
##########################  Init code For Assignment 3 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)

misclassification_func <- function(x1,x2){
  n <- length(x1)
  retuen_value <- 1 - sum(diag(table(x1,x2)))/n
}

##########################  3.1.1##############################################
data <- read.csv("pima-indians-diabetes.csv",header=FALSE)

# get age and plasma values
age <- data[, 8]
plasma <- data[, 2]
```

```r
# set the column names
names(data)[9] <- "diabetes"
names(data)[2] <- "plasma"
names(data)[8] <- "age"

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

# plot the data
ggplot2::ggplot(df, ggplot2::aes(age,plasma)) +
  ggplot2::geom_point(ggplot2::aes(colour = as.factor(data[, 9])))+
  ggplot2::labs(x = "Age", y="Plasma glucose") +
  guides(color = guide_legend(title = 'Diabetes'))
##########################  3.2,3,3 ########################################
model <- glm(diabetes ~ plasma + age,
             family = binomial,
             data = data)
summary(model)
diabete_prob <- predict(model, type="response")

r <- 0.5

diabet_pred <- ifelse(diabete_prob > r,1,0)

missC_error <- misclassification_func(data$diabetes,diabet_pred)

cat("Misclassification error:", missC_error, "\n")

# calc the boundary line
x <- seq(20, 85)
y <- -(model$coefficients["(Intercept)"] + model$coefficients["age"] * x) / model$coefficients["plasma"]

# make data frame
df_boundary <- data.frame(x, y)

ggplot(df, ggplot2::aes(age,plasma)) +
  geom_point(ggplot2::aes(colour = as.factor(diabet_pred)))+
  labs(x = "Age", y="Plasma glucose concentration") +
  guides(color = guide_legend(title = 'Predicted Diabetes')) +
  geom_line(data = df_boundary, aes(x = x, y = y), color = "blue") +
  ggtitle("Age vs Plasma When r = 0.5")

r <- 0.2

diabet_pred <- ifelse(diabete_prob > r,1,0)

missC_error <- misclassification_func(data$diabetes,diabet_pred)

cat("Misclassification error when r = 0.2:", missC_error, "\n")

ggplot(df, ggplot2::aes(age,plasma)) +
```

```r
  geom_point(ggplot2::aes(colour = as.factor(diabet_pred)))+
  labs(x = "Age", y="Plasma glucose concentration") +
  guides(color = guide_legend(title = 'Predicted Diabetes')) +
  ggtitle("Age vs Plasma When r = 0.2")

r <- 0.8

diabet_pred <- ifelse(diabete_prob > r,1,0)

missC_error <- misclassification_func(data$diabetes,diabet_pred)

cat("Misclassification error when r = 0.8:", missC_error, "\n")

ggplot(df, ggplot2::aes(age,plasma)) +
  geom_point(ggplot2::aes(colour = as.factor(diabet_pred)))+
  labs(x = "Age", y="Plasma glucose concentration") +
  guides(color = guide_legend(title = 'Predicted Diabetes')) +
  ggtitle("Age vs Plasma When r = 0.8")


# 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") , c

summary(newmodel)

r <- 0.5

diabete_prob <- predict(newmodel, type="response")

diabet_pred <- ifelse(diabete_prob > r,1,0)

missC_error <- misclassification_func(data$diabetes,diabet_pred)

cat("Misclassification error when r = 0.5:", missC_error, "\n")

ggplot(df, ggplot2::aes(age,plasma)) +
  geom_point(ggplot2::aes(colour = as.factor(diabet_pred)))+
  labs(x = "Age", y="Plasma glucose concentration") +
  guides(color = guide_legend(title = 'Predicted Diabetes')) +
  ggtitle("Age vs Plasma When r = 0.5")
```