Lab1 Report

Group A7

2022-12-22

Group members:Hamza (hammu144), Dinesh (dinsu875) and Umamaheswarababu (umama339)

Statement of Contribution : Assignment 1 was mostly done by Hamza, Assignment 2 was mostly done by Dinesh and Assignment 3 was mostly done by Umamaheswarababu

Assignment 1. Handwritten digit recognition with K-nearest neighbors

Question.1

Loading data in R

```
csv_data <- read.csv("optdigits.csv", header = FALSE)
```

By using the partitioning principle. we will get training, validation and test sets.(0.5/0.25/0.25)

```
n=dim(csv_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=csv_data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=csv_data[id2,]
id3=setdiff(id1,id2)
test=csv_data[id3,]
```

Question.2

By using kknn function and using table function we will get confusion matrix.

```
library(kknn)
```

```
## Warning: package 'kknn' was built under R version 4.2.2
```

```
training_data.kknn <- kknn(as.factor(V65)~., train=train, test=train, k=30,kernel="rectangular")
train_fit <- fitted(training_data.kknn)
table(truth=train$V65, pred=train_fit)
```

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

```
misclassification_error <- 1- sum(diag(table(truth=train$V65, pred=train_fit))) / sum(table(trut
h=train$V65, pred=train_fit))
misclassification_error
```

```
## [1] 0.04500262
```

This formula Gives us misclassification error = 0.0450 by rounding.

Details of confusion matrix for train.

Table function will give us confusion matrix with the help of confusion matrix we can explain different mis-classifications. Here's some examples for mis-classifications as

Follow :

For integer (0) we don't get any miclassifcation.

For integer (7) we can see misclassification in second and (9th) column.

For integer (6) second column only showing with misclassification.

similarly we can check for other integers as well.

Correction : Asked for easiest and hardest one's.

The easiest digit to classify is '0' because we did not see any misclassifications for digit '0' in the confusion matrix. Similarly digits '2', '3' and '6' are also easy to classify as there are very few misclassifications for these digits.

The hardest digits to classify are '1', '4', '8' and '9' because we can see more missclassifications for these digits in the confusion matrix. For digit '1', more observations are misclassified as '2'. For digit '8', many observations are misclassified as '2'. For digits '4', the model misclassified some of the observations as '0','1','7','8' and '9'. Similarly for digit '9', the model misclassified some of the observations as '0','1','3','4','7' and '8'.

```
test_data.kknn <- kknn(as.factor(V65)~., train=train, test=test, k=30,kernel="rectangular")


test_fit <- fitted(test_data.kknn)

table(truth=test$V65, pred=test_fit)
```

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

```
misclassification_error <- 1- sum(diag(table(truth=test$V65, pred=test_fit))) / sum(table(truth=
test$V65, pred=test_fit))
misclassification_error
```

```
## [1] 0.05329154
```

This formula Gives us misclassification error = 0.0533 by rounding.

Details of confusion matrix for test.

Table function will give us confusion matrix with the help of confusion matrix we can explain different misclassificatons. Here’s some examples for misclassifications as

Follow :

For integer (0) we can see there’s misclassification in 5th column.

For integer (1) we can see misclassification in (3rd) and last column.

For integer (2) 9th column only showing with misclassification.

Similary with the help of confusion matrix we can explain misclassificatios for rest of integers. As far overall prediction we can say in lower triangle of matrix there’s less misclassifications as compare to upper triangle of matrix.

If we compare both matrix train confusion matrix didn’t give any misclassification for zero integer, but test gives misclassification for every integer. so train is somehow good.

Correction : Asked for easiest and hardest one’s.

We see from confusion matrix that no classifications are there for digit ‘6’ so it’s the easiest digit to classify. Similarly digits ‘0’, ‘2’, and ‘7’ are also easy to classify as there are very few misclassifications for these digits.

Digits ‘4’, ‘5’ and ‘8’ are hard to classify because we can see more misclassifications for these digits.

The conclusion on hardest/easiest digits to classify from the fitted model with train and test is that the easiest digits to classify are ‘0’, ‘2’ and ‘6’ and hardest digits to classify are ‘4’, ‘5’ and ‘8’.

Question.3

As in question it is ask for specific digit “8” we will make first subset of data-frame by using subset function with condition.

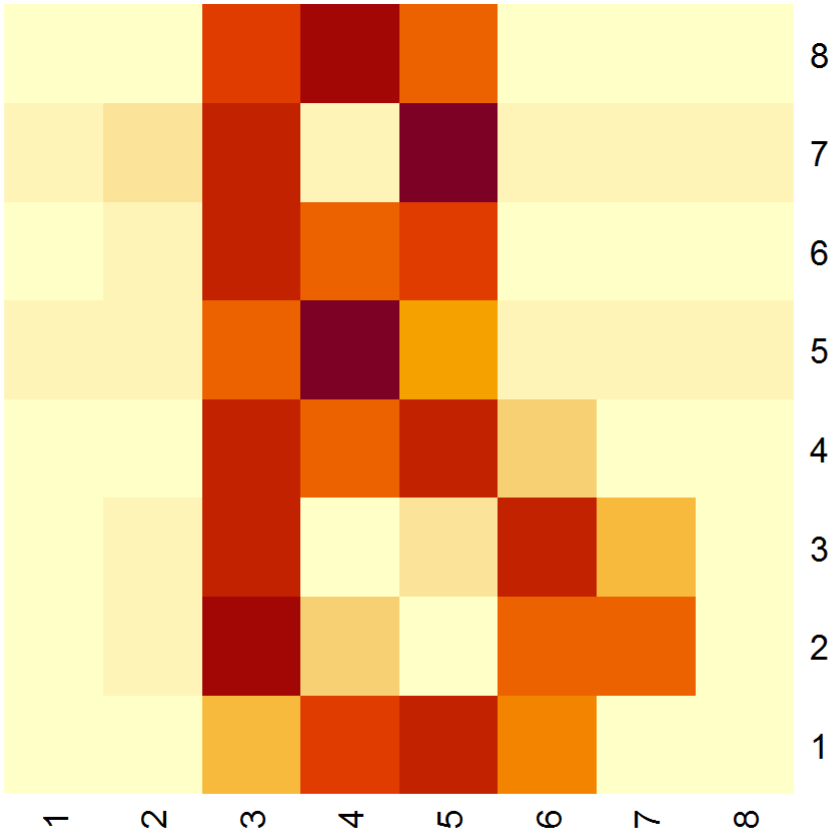
Then we will make sub-group of matrix (8*8) and then will get heatmap.

```
digit_col_train <- train$V65
case <-  digit_col_train == 8

probility_train <- training_data.kknn$prob

sorted_data<- order(probility_train[case,9], decreasing = TRUE)
sorted_data[1:2]
```

```
## [1] 10 14
```

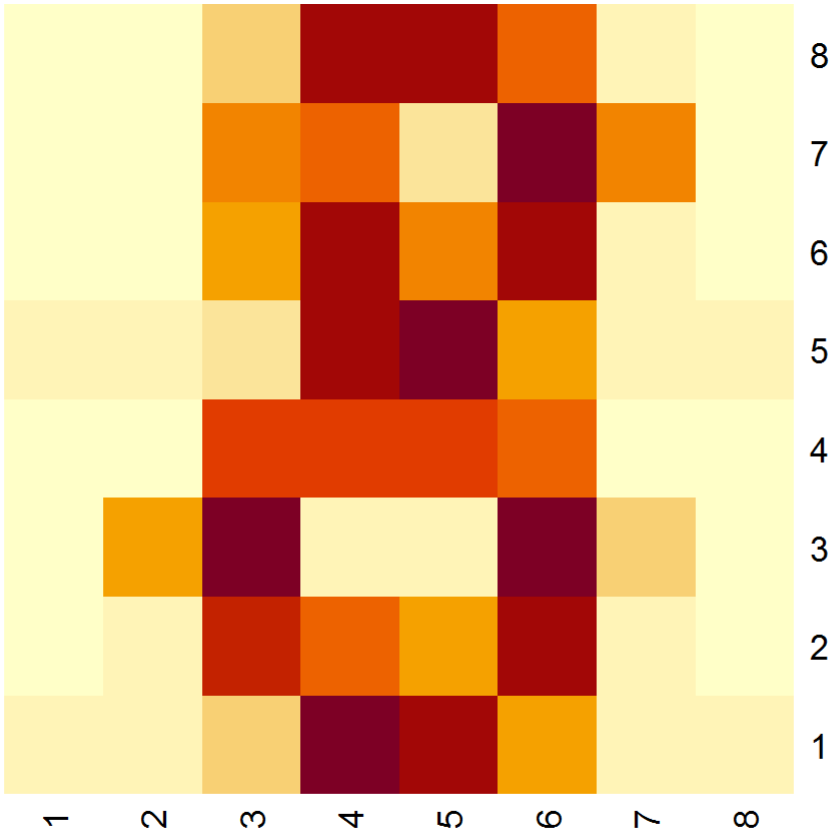
This will give us two case with higher probability and with the help of heatmap we can give them visual.

```
sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[10, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
```



We can see digit “8” clearly.

```
nw <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[14, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
```



second case for digit “8” easy to classify

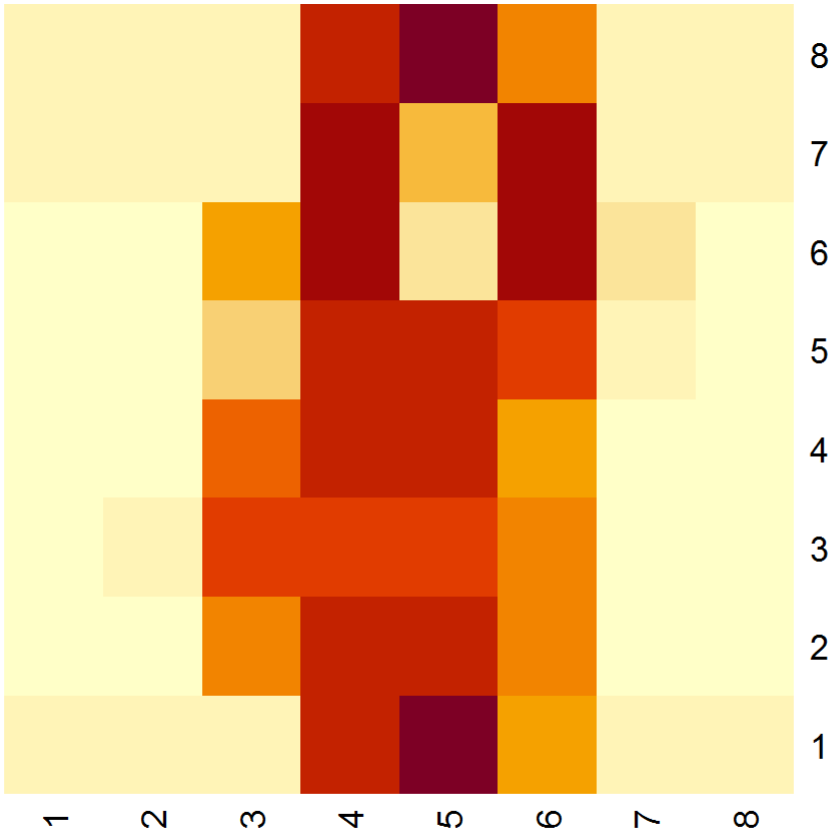
```
digit_col_train <- train$V65
case <-  digit_col_train == 8

probility_train <- training_data.kknn$prob

sorted_data<- order(probility_train[case,9], decreasing = FALSE)
sorted_data[1:3]
```

```
## [1]  50  43 136
```

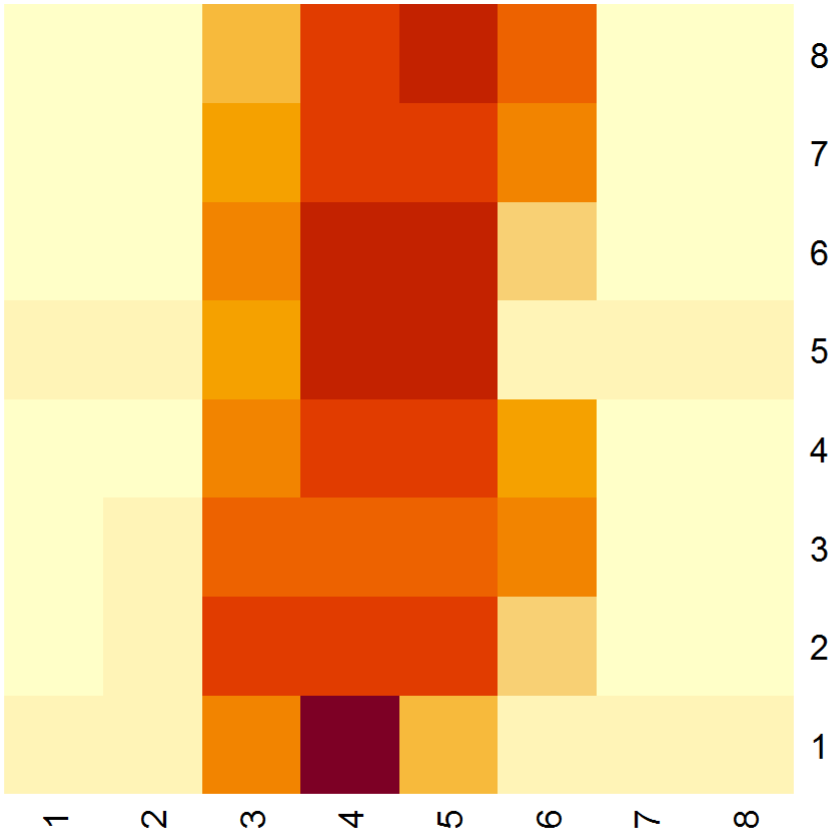
These are 3 case harder to classify.

We can have visual using heatmap function

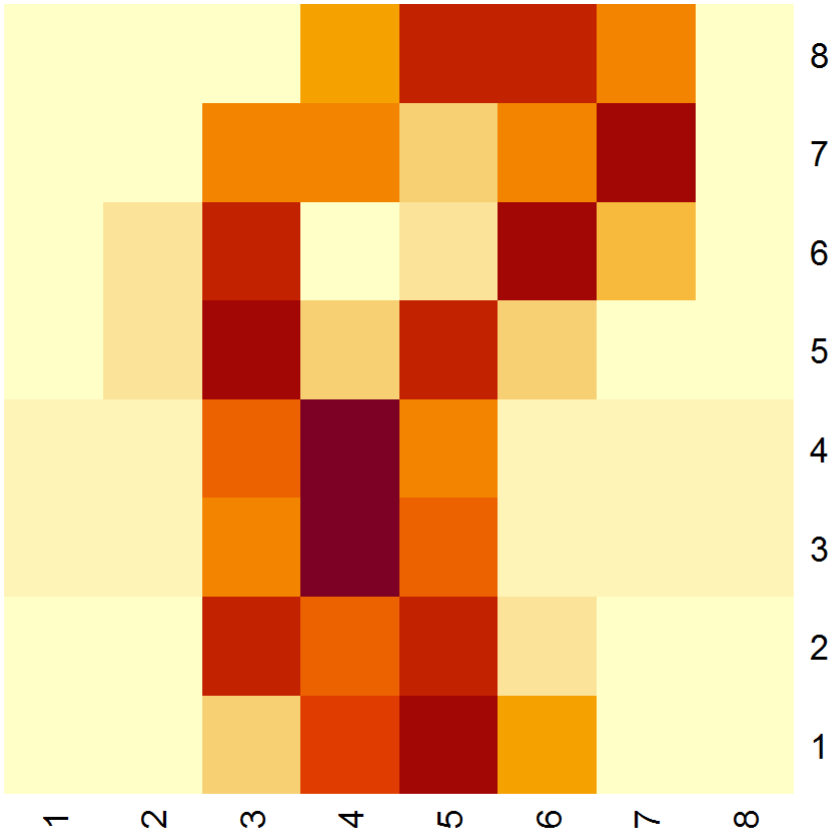
```
sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[43, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
```




```
sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[50, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
```




```
sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[136, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
```



These are three case harder to classify with visual with the help of heatmap.

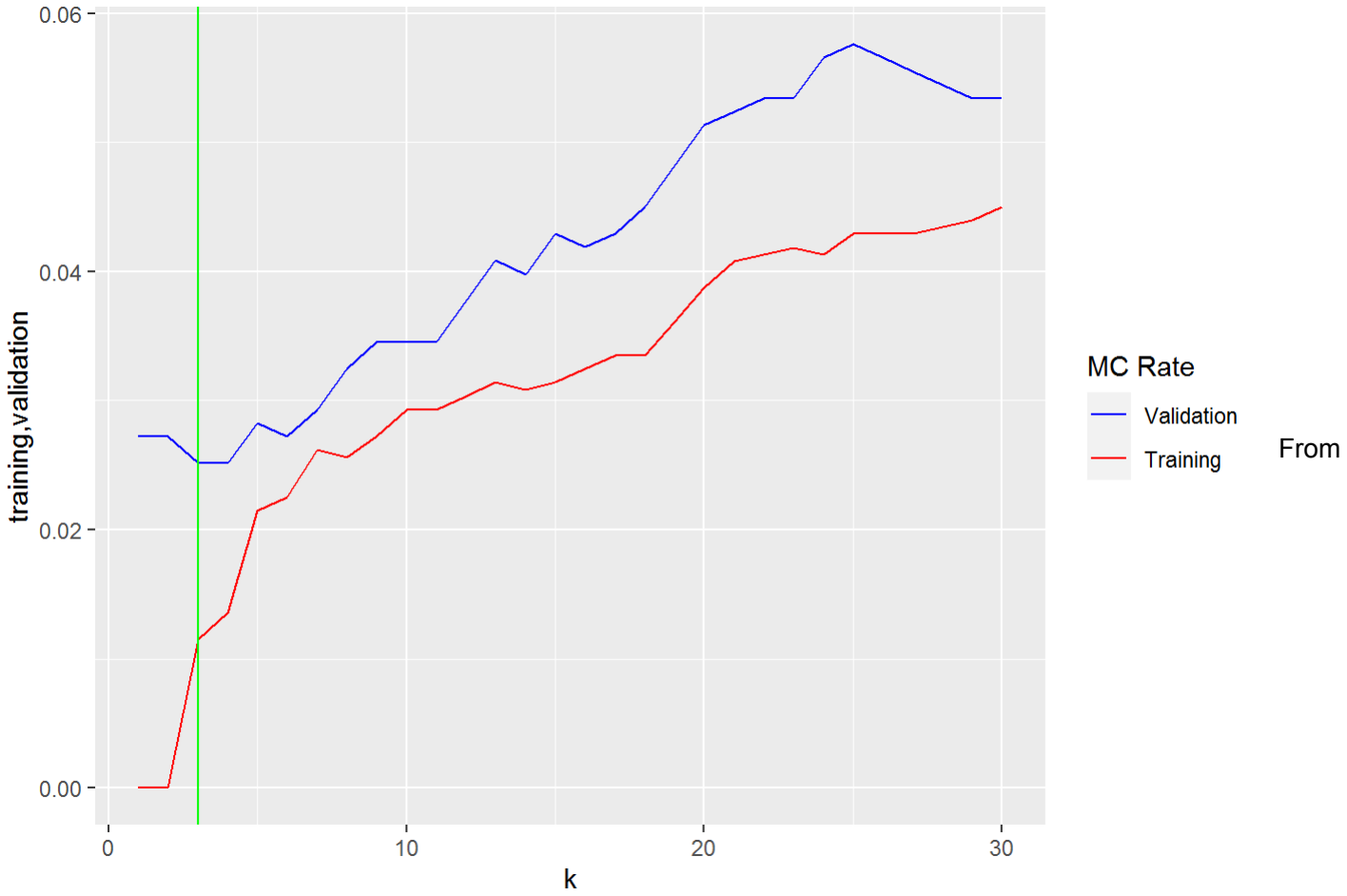
Question.4

For k =1,2…30 we need to check training data and valid data for this we use loop.

```
k=1
train_optim <- c()
valid_optim <- c()

for(i in 1:30){


  training_data.kknn <- kknn(as.factor(V65)~., train=train, test=train, k=i,kernel="rectangula
r")
  train_fit <- fitted(training_data.kknn)
  table(truth=train$V65, pred=train_fit)

  train_optim[i] <- 1- sum(diag(table(truth=train$V65, pred=train_fit))) / sum(table(truth=train
$V65, pred=train_fit))

  valid_data.kknn <- kknn(as.factor(V65)~., train=train, test=valid, k=i,kernel="rectangular")
  valid_fit <- fitted(valid_data.kknn)
  table(truth=valid$V65, pred=valid_fit)

  valid_optim[i] <- 1- sum(diag(table(truth=valid$V65, pred=valid_fit))) / sum(table(truth=valid
$V65, pred=valid_fit))

}


df <- data.frame(k = 1:30, training = c(train_optim), validation = c(valid_optim))

library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
ggplot(df) +
  geom_line(aes(y=training, x = k, color = "red")) +
  geom_line(aes(y=validation, x = k, color = "blue")) +
  ylab("training,validation")+
  scale_color_manual(name = "MC Rate", labels = c("Validation ", "Training "),
                     values =c("blue", "red"))+geom_vline(data=df, mapping=aes(xintercept=3), co
lor="green")
```



From plot misclassification error for validation is higher than as compare to training data.

Correction.

We fix prediction for validation data.

Complexity of model play an important role for each K value. For smaller values of k model seem complex, as the value of k is increased we observe less complexity. For smaller values of k we observe that error for training is almost zero but for validation it's differ. This shows that misclassifications for validation are higher than train so it's showing model over fit.

From the above plot we can see line graph is drawn by joining all the possible k values. We can see lowest misclassification error when k is approximately equal to 3. From the plot we can see that at k value around 3 model is almost fit doesn't seem to have any over or under fitted problems.

Estimating K-optimal value for test error

```
test_data.kknn <- kknn(as.factor(V65)~., train=train, test=test, k=3,kernel="rectangular")

test_fit <- fitted(test_data.kknn)

misclassification_error <- 1- sum(diag(table(truth=test$V65, pred=test_fit))) / sum(table(truth=
test$V65, pred=test_fit))
misclassification_error
```

```
## [1] 0.02403344
```

As we see values change from point k=3 from above graph so, this optimal k with error 0.0240.

Question.5

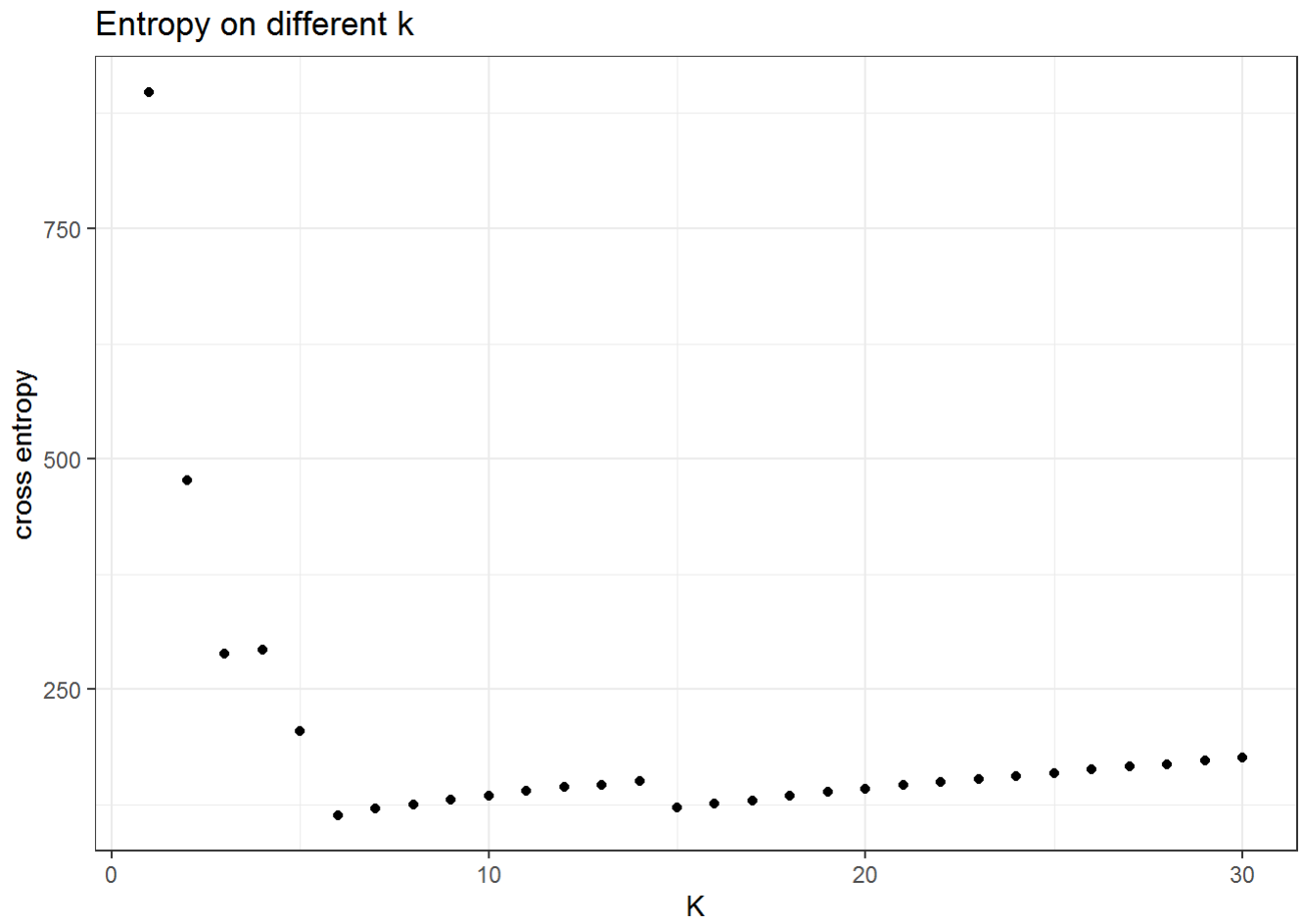
checking correctness for validation.

```
m <- matrix(0, ncol = 10, nrow = nrow(valid))
t_num <- as.numeric(as.character(valid$V65))
for(i in 1:length(t_num)){ m[i, t_num[i]+1] <- 1
}

colSums(m)
```

```
##  [1]  96 108  87  90 103  92  95  93  98  93
```

for entropy we chech differen values of K and check validation.

```
c_ent <- c()
for(i in 1:30){
  kkn_v <- kknn(as.factor(V65)~.,
                train = train,
                test = valid,
                k= i,
                kernel = "rectangular")



  prob_m <- kkn_v$prob
  l_probs <- c()
for(j in 1:nrow(prob_m)){
l_probs[j] <- sum(m[j,] * log(prob_m[j,] + 1e-15))
}

    c_ent[i] <- sum(-l_probs)
}


library(ggplot2)
ggplot() +
  geom_point(aes(x = 1:30, y = c_ent)) + theme_bw() +
  labs(title = "Entropy on different k") + xlab("K") + ylab("cross entropy")
```



so by this plot we can say k values is no more 3 its change. but rather when k = 6.

correction.

As we know in machine learning cross entropy play an important role. It helps to minimize the loss. If we have less loss in our model then it means that we have good model.

So if we observe cross entropy in above plot, value of cross entropy as we used value 3 before is higher that's why we are not saying k=3 is our optimal value on other hand if we observe value of entropy on value k = 6 it's seem more optimal less cross entropy value. so k =6 is our optimal value now.

Assignment 2. Linear Regression and Ridge Regression

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 afunction of the voice characteristics, and since the data are scaled, no intercept is needed in the modelling.

```
##2.1 Divide data

#Reading the data
data<-read.csv("parkinsons.csv")

# Removed some columns from the data
new_data <- subset(data, select = -c(subject., age, sex, test_time, total_UPDRS))

#Dividing the data(60/40)
n<-dim(new_data)[1]
set.seed(12345)
id<-sample(1:n,floor(n*0.6))
train<-new_data[id,]
test<-new_data[-id,]

#Scaling the data
library(caret)
scaler <- preProcess(train)
train_scale_data <- predict(scaler, train)
test_scale_data <- predict(scaler, test)
```

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.

```
model1 = lm(motor_UPDRS ~ 0 + ., data=train_scale_data)
summary(model1)
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ 0 + ., data = train_scale_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0255 -0.7363 -0.1087  0.7333  2.1960
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## Jitter...       0.186931   0.149561   1.250 0.211431
## Jitter.Abs.    -0.169609   0.040805  -4.157 3.31e-05 ***
## Jitter.RAP     -5.269544  18.834160  -0.280 0.779658
## Jitter.PPQ5    -0.074568   0.087766  -0.850 0.395592
## Jitter.DDP      5.249558  18.837525   0.279 0.780510
## Shimmer         0.592436   0.205981   2.876 0.004050 **
## Shimmer.dB.    -0.172655   0.139316  -1.239 0.215315
## Shimmer.APQ3   32.070932  77.159242   0.416 0.677694
## Shimmer.APQ5   -0.387507   0.113789  -3.405 0.000668 ***
## Shimmer.APQ11   0.305546   0.061236   4.990 6.34e-07 ***
## Shimmer.DDA   -32.387241  77.158814  -0.420 0.674695
## NHR            -0.185387   0.045567  -4.068 4.84e-05 ***
## HNR            -0.238543   0.036395  -6.554 6.41e-11 ***
## RPDE            0.004068   0.022664   0.179 0.857556
## DFA            -0.280318   0.020136 -13.921  < 2e-16 ***
## PPE             0.226467   0.032881   6.887 6.70e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9394 on 3509 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.25 on 16 and 3509 DF,  p-value: < 2.2e-16
```

Comment on which variables contribute significantly to the model

Jitter.Abs. , Shimmer.APQ5 , Shimmer.APQ11 , HNR , NHR, DFA ,PPE contributed significantly to motor UPDRS at the 0.001 significance level.

```
##MSE Train & Test
train_mse <- mean(model1$residuals^2)
# we can use this to estimate the training MSE train_mse <- sum(residuals(model1)^2)/nrow(train_
scale_data)

test_mse <- mean((test_scale_data$motor_UPDRS - predict(model1,test_scale_data))^2)
#we can use this to estimate the test MSE
#residuals_test <- predict(mod1, test_scale) - test_scale$motor_UPDRS
#test_mse <- sum(residuals_testˆ2)/nrow(test_scale)

list("Train MSEs"=train_mse,"Test MSEs"=test_mse)
```

```
## $`Train MSEs`
## [1] 0.8785431
##
## $`Test MSEs`
## [1] 0.9354477
```

When compared to the test MSE, the MSE on training data is lower.

Question 3: Implement 4 following functions by using basic R commands only

```
##2.3(a).Loglihood Function

loglikelihood <- function(theta){

  X <- as.matrix(train_scale_data[,-1])
  n <- nrow(X)
  y <- train_scale_data[,1]
  sigma <- theta[17]
  theta_new <- theta[1:16]
  return( -(n/2) * log(2 * pi * sigma^2) - (1/(2*sigma^2))  * sum((y - X%*%theta_new)^2)  )

}
```

```
##2.3(b). Ridge Function

Ridge <- function(theta, lambda){

  return( -(loglikelihood(theta)) + lambda * sum(theta^2) )

}
```

```
##2.3(c). RidgeOpt Function

RidgeOpt <- function(lambda){

  optim(rep(1,17),fn = Ridge, method="BFGS", lambda = lambda)
}
```

```
##2.3(d). DF function

DF <- function(lambda){
  X <- as.matrix(train_scale_data[,-1])
  n <- nrow(X)
  df <- X %*% solve((t(X)%*%X + lambda* diag(ncol(X)))) %*% t(X)
  return( sum(diag(df)) )
}
```

Question 4.Use RidgeOpt function , compute optimal θ parameters for λ=1,λ=100 and λ=1000.

```
Rid_opt_1 <- RidgeOpt(lambda = 1)
Rid_opt_100 <- RidgeOpt(lambda = 100)
Rid_opt_1000 <- RidgeOpt(lambda = 1000)
X_train <- as.matrix(train_scale_data[,-1])
X_test <- as.matrix(test_scale_data[,-1])

## Prediction with λ = 1
predicted_train <- X_train %*% Rid_opt_1$par[1:16]
predicted_test <- X_test %*% Rid_opt_1$par[1:16]

error_train_1 <- mean((predicted_train - train_scale_data$motor_UPDRS)^2)
error_test_1 <- mean((predicted_test - test_scale_data$motor_UPDRS)^2)

##Prediction with λ = 100
predicted_train <- X_train %*% Rid_opt_100$par[1:16]
predicted_test <- X_test %*% Rid_opt_100$par[1:16]

error_train_100 <- mean((predicted_train - train_scale_data$motor_UPDRS)^2)
error_test_100 <- mean((predicted_test - test_scale_data$motor_UPDRS)^2)

##Prediction with λ = 1000
predicted_train <- X_train %*% Rid_opt_1000$par[1:16]
predicted_test <- X_test %*% Rid_opt_1000$par[1:16]

error_train_1000 <- mean((predicted_train - train_scale_data$motor_UPDRS)^2)
error_test_1000 <- mean((predicted_test - test_scale_data$motor_UPDRS)^2)

list(error_train_1=error_train_1,error_train_100=error_train_100,error_train_1000=error_train_10
00)
```

```
## $error_train_1
## [1] 0.8786272
##
## $error_train_100
## [1] 0.8841695
##
## $error_train_1000
## [1] 0.9128817
```

```
list(error_test_1=error_test_1,error_test_100=error_test_100,error_test_1000=error_test_1000)
```

```
## $error_test_1
## [1] 0.9349982
##
## $error_test_100
## [1] 0.9322925
##
## $error_test_1000
## [1] 0.9481592
```

Which penalty parameter is most appropriate among the selected ones?

When λ = 1, the training MSE error is the lowest. But when λ = 1000,the training and prediction errors are both high. So, the penalty parameter λ = 100 is the most appropriate among the lambda values because it has the lowest test mse value.

```
df_1<-DF(1)
df_100<-DF(100)
df_1000<-DF(1000)
result <- data.frame(lambda = c(1, 100, 1000),
                     MSE_Train = c(error_train_1,error_train_100,error_train_1000),
                     MSE_Test = c(error_test_1,error_test_100,error_test_1000),
                     DOF = c(df_1, df_100, df_1000) )
result
```

```
##   lambda MSE_Train  MSE_Test       DOF
## 1      1 0.8786272 0.9349982 13.860736
## 2    100 0.8841695 0.9322925  9.924887
## 3   1000 0.9128817 0.9481592  5.643925
```

Compute and compare the degrees of freedom of these models and make appropriate conclusions.

By seeing the output of the result dataframe, we can say that if the lambda value gets increases, the degree of freedom gets decreased.

Reference

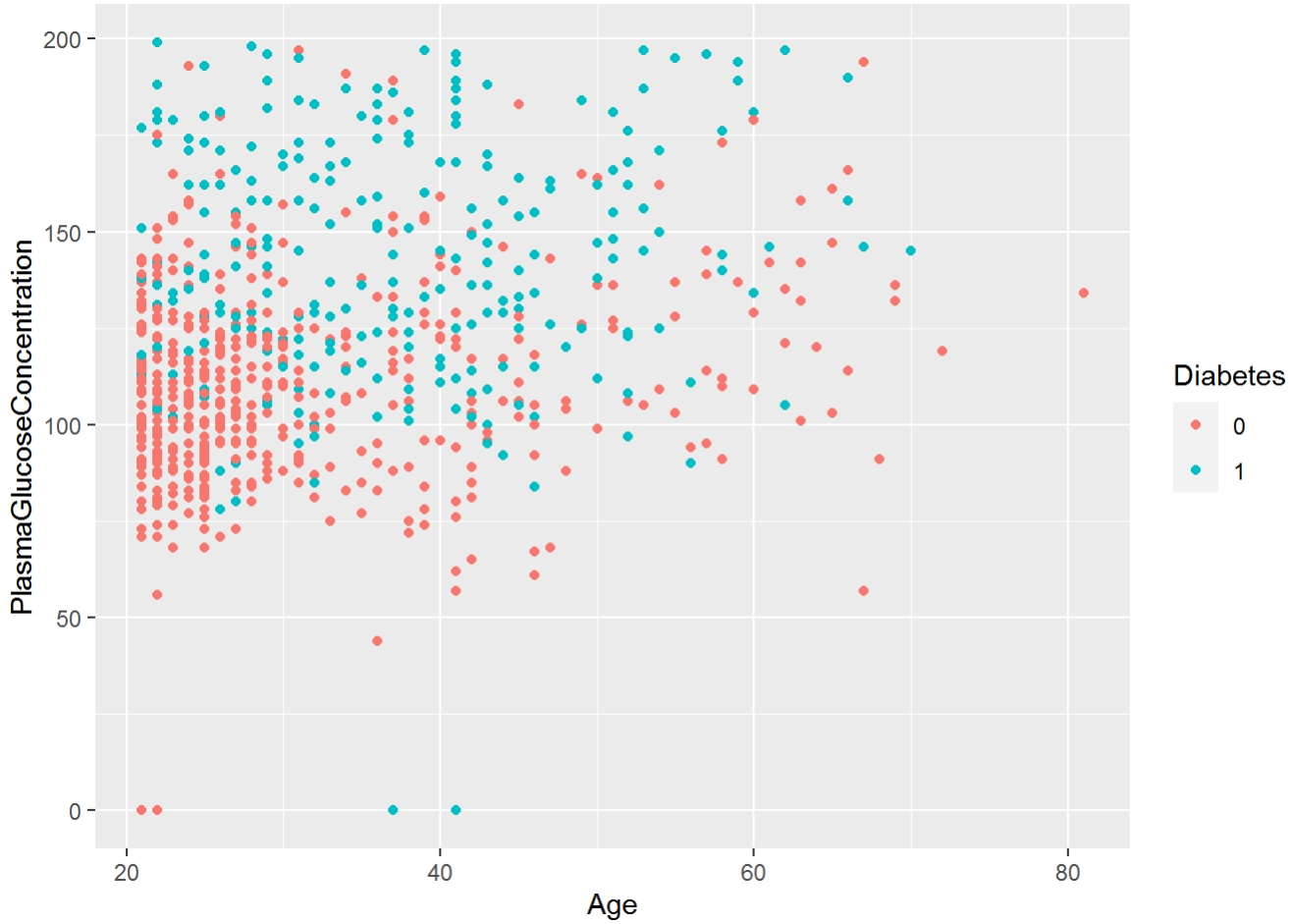
1. “More to know about the optimisation of linear regression” (via (https://www.joshua-entrop.com/post/optim_linear_reg/))
2. “More to know about Ridge Logistic” (via (https://freakonometrics.hypotheses.org/52773))

3.”Logistic Regression Regularized with Optimization” (via (https://www.r-bloggers.com/2017/02/logistic-regression-regularized-with-optimization/))

Assignment 3. Logistic regression and basis function expansion**

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?

```
library("ggplot2")

#Loading data as a dataframe
data <- read.csv("pima-indians-diabetes.csv")

#changing column names for easy interpretation
colnames(data)[c(2,8,9)]<-c('PlasmaGlucoseConcentration','Age','Diabetes')
data$Diabetes <- as.factor(data$Diabetes)
plot1<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=Diabetes))
print(plot1)
```



Logistic regression model can be used to classify the diabetes with these two variable because from the plot we can see that people with age less than 35 years and Plasma Glucose Concentration below 150 are less likely to have diabetes.

2. Train a logistic regression model with y=Diabetes as target x1= Plasma glucose concentration and x2 = 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.

```
#training a Logistic Regression model
train_model <- glm(Diabetes ~ PlasmaGlucoseConcentration + Age, data = data, family= "binomial")

#prediction of all observations
prediction<- predict(train_model, data, type = "response")
prediction_1<- ifelse(prediction>0.5,1,0)
```

The probabilistic equation of the estimated model

```
train_model$coefficients
```

```
##                (Intercept) PlasmaGlucoseConcentration
##                -5.89785793                 0.03558250
##                        Age
##                 0.02450157
```

The probabilistic equation of the estimated model is given by

$$p(Diabetes = 1) = \frac{1}{1 + e^{-5.89785793 + 0.03558250 * PlasmaGlucoseConcentration + 0.02450157 * Age}}$$

Misclassification error

```
#Computing model accuracy using confusion matrix
table(data$Diabetes,prediction_1)
```

```
##    prediction_1
##       0   1
##   0 436  64
##   1 140 127
```

```
#A function to calculate misclassification error
missclass<- function(pred,actual){
  n=length(pred)
  return(1-sum(diag(table(pred, actual)))/n)
}

#Computing misclassification error for r= 0.5
missclass(data$Diabetes,prediction_1)
```

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

```
#scatter plot with predicted values of diabetes
data$PredictedDiabetes <- as.factor(prediction_1)
plot2<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=PredictedD
iabetes))
print(plot2)
```

From the misclassification error we can say that around 26% of observations are misclassified. From the plot we can say that people with Plasma Glucose Concentration below 150 and age below 35 years are almost classified correctly. People with age above 35 years are mostly misclassified.

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.

The equation of the decision boundary between the two classes is given by $\theta^T \mathbf{x} = 0$

In this case the equation is given by

$$\mathbf{PlasmaGlucoseConcentration} = \frac{5.89785793}{0.03558250} - \frac{0.02450157}{0.03558250} * Age$$

```
#plotting decision boundary
x1<-15:85  #taking Age as x1
x2<-(5.89785793/0.03558250)-(0.02450157/0.03558250)*x1 # #taking Plasma Glucose as x2
boundary_df<-data.frame(x1,x2 )
plot3<-plot2 + geom_line(data = boundary_df, aes(x=x1,y=x2))
print(plot3)
```

Decision boundary is not catching the data distribution well. Many diabetic points present on both above and below the 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.

```
#prediction with r=0.2
prediction_2<- ifelse(prediction>0.2,1,0)
data$PredictedDiabetes_2 <- as.factor(prediction_2)
```

```
#Computing model accuracy using confusion matrix
table(data$Diabetes,prediction_2)
```

```
##    prediction_2
##       0   1
##   0 238 262
##   1  25 242
```

```
#Computing misclassification error
missclass(data$Diabetes,prediction_2)
```

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

```
##scatter plot of classification with r=0.2
plot4<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=PredictedD
iabetes_2))
plot4<-plot4 + geom_line(data = boundary_df, aes(x=x1,y=x2))
print(plot4)
```


```
#prediction with r=0.8
prediction_3<- ifelse(prediction>0.8,1,0)
data$PredictedDiabetes_3 <- as.factor(prediction_3)
```

```
#Computing model accuracy using confusion matrix
table(data$Diabetes,prediction_3)
```

```
##    prediction_3
##       0   1
##   0 490  10
##   1 231  36
```

```
#Computing misclassification error
missclass(data$Diabetes,prediction_3)
```

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

```
#scatter plot of classification with r=0.8
plot5<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=PredictedD
iabetes_3))
plot5<-plot5 + geom_line(data = boundary_df, aes(x=x1,y=x2))
print(plot5)
```

Models with r=0.2 and r=0.8 have misclassified the many observations compared to the model with r=0.5. The model with r=0.2 has classified many non-diabetic people as diabetic. The model with r=0.8 has classified many diabetic people as non-diabetic.

5.Perform a basis function expansion trick by computing new 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?

```
#computing and adding new features to the data set
data$z1<-data$PlasmaGlucoseConcentration^4
data$z2<-(data$PlasmaGlucoseConcentration^3)*(data$Age)
data$z3<-(data$PlasmaGlucoseConcentration^2)*(data$Age^2)
data$z4<-(data$PlasmaGlucoseConcentration)*(data$Age^3)
data$z5<-data$Age^4

new_train_model <- glm(Diabetes ~ PlasmaGlucoseConcentration+Age+z1+z2+z3+z4+z5, data = data, fa
mily= "binomial")
new_prediction<- predict(new_train_model, data, type = "response")
new_prediction<- ifelse(new_prediction>0.5,1,0)
```

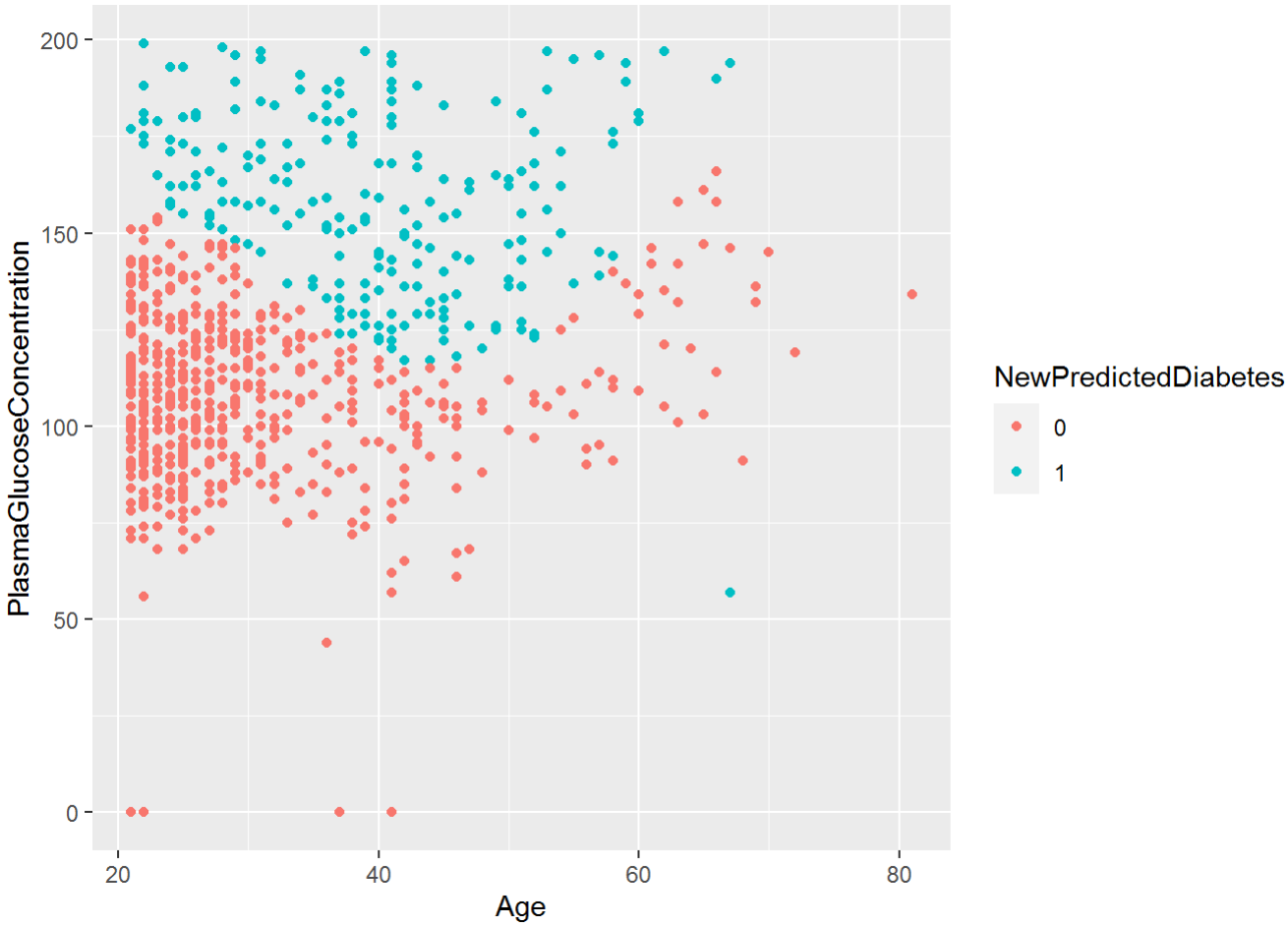
```
#Computing model accuracy using confusion matrix
table(data$Diabetes,new_prediction)
```

```
##    new_prediction
##       0   1
##   0 433  67
##   1 122 145
```

```
#Computing misclassification error
missclass(data$Diabetes,new_prediction)
```

```
## [1] 0.2464146
```

```
#scatter plot of classification after adding new features
data$NewPredictedDiabetes <- as.factor(new_prediction)
plot6<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=NewPredict
edDiabetes))
print(plot6)
```



Compared to the previous model this model's missclassification error is less and we can say that this model is better than the previous model. The decision boundary in this model have curvature (parabolic shape) in its shape. The decision boundary in the previous model is a straight line.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

csv_data <- read.csv("optdigits.csv", header = FALSE)

n=dim(csv_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=csv_data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=csv_data[id2,]
id3=setdiff(id1,id2)
test=csv_data[id3,]

library(kknn)


training_data.kknn <- kknn(as.factor(V65)~., train=train, test=train, k=30,kernel="rectangular")
train_fit <- fitted(training_data.kknn)
table(truth=train$V65, pred=train_fit)

misclassification_error <- 1- sum(diag(table(truth=train$V65, pred=train_fit))) / sum(table(trut
h=train$V65, pred=train_fit))
misclassification_error

test_data.kknn <- kknn(as.factor(V65)~., train=train, test=test, k=30,kernel="rectangular")


test_fit <- fitted(test_data.kknn)

table(truth=test$V65, pred=test_fit)

misclassification_error <- 1- sum(diag(table(truth=test$V65, pred=test_fit))) / sum(table(truth=
test$V65, pred=test_fit))
misclassification_error

digit_col_train <- train$V65
case <-  digit_col_train == 8

probility_train <- training_data.kknn$prob

sorted_data<- order(probility_train[case,9], decreasing = TRUE)
sorted_data[1:2]


sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[10, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
nw <- subset(train, subset = digit_col_train == 8 )
```

```
Reshape_matri <- matrix(as.numeric(sub_set[14, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)
digit_col_train <- train$V65
case <-  digit_col_train == 8

probility_train <- training_data.kknn$prob

sorted_data<- order(probility_train[case,9], decreasing = FALSE)
sorted_data[1:3]


sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[43, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)

sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[50, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)

sub_set <- subset(train, subset = digit_col_train == 8 )
Reshape_matri <- matrix(as.numeric(sub_set[136, 1:64]), nrow = 8, byrow = TRUE)


heatmap(Reshape_matri, Colv = NA, Rowv = NA)

k=1
train_optim <- c()
valid_optim <- c()

for(i in 1:30){


  training_data.kknn <- kknn(as.factor(V65)~., train=train, test=train, k=i,kernel="rectangula
r")
  train_fit <- fitted(training_data.kknn)
  table(truth=train$V65, pred=train_fit)

  train_optim[i] <- 1- sum(diag(table(truth=train$V65, pred=train_fit))) / sum(table(truth=train
$V65, pred=train_fit))

  valid_data.kknn <- kknn(as.factor(V65)~., train=train, test=valid, k=i,kernel="rectangular")
  valid_fit <- fitted(valid_data.kknn)
  table(truth=valid$V65, pred=valid_fit)

  valid_optim[i] <- 1- sum(diag(table(truth=valid$V65, pred=valid_fit))) / sum(table(truth=valid
$V65, pred=valid_fit))
```

```
}


df <- data.frame(k = 1:30, training = c(train_optim), validation = c(valid_optim))

library(ggplot2)

ggplot(df) +
  geom_line(aes(y=training, x = k, color = "red")) +
  geom_line(aes(y=validation, x = k, color = "blue")) +
  ylab("training,validation")+
  scale_color_manual(name = "MC Rate", labels = c("Validation ", "Training "),
                     values =c("blue", "red"))+geom_vline(data=df, mapping=aes(xintercept=3), co
lor="green")


test_data.kknn <- kknn(as.factor(V65)~., train=train, test=test, k=3,kernel="rectangular")

test_fit <- fitted(test_data.kknn)

misclassification_error <- 1- sum(diag(table(truth=test$V65, pred=test_fit))) / sum(table(truth=
test$V65, pred=test_fit))
misclassification_error

m <- matrix(0, ncol = 10, nrow = nrow(valid))
t_num <- as.numeric(as.character(valid$V65))
for(i in 1:length(t_num)){ m[i, t_num[i]+1] <- 1
}

colSums(m)



c_ent <- c()
for(i in 1:30){
  kkn_v <- kknn(as.factor(V65)~.,
                train = train,
                test = valid,
                k= i,
                kernel = "rectangular")



  prob_m <- kkn_v$prob
  l_probs <- c()
for(j in 1:nrow(prob_m)){
l_probs[j] <- sum(m[j,] * log(prob_m[j,] + 1e-15))
}

    c_ent[i] <- sum(-l_probs)
}
```

```
library(ggplot2)
ggplot() +
  geom_point(aes(x = 1:30, y = c_ent)) + theme_bw() +
  labs(title = "Entropy on different k") + xlab("K") + ylab("cross entropy")

##2.1 Divide data

#Reading the data
data<-read.csv("parkinsons.csv")

# Removed some columns from the data
new_data <- subset(data, select = -c(subject., age, sex, test_time, total_UPDRS))

#Dividing the data(60/40)
n<-dim(new_data)[1]
set.seed(12345)
id<-sample(1:n,floor(n*0.6))
train<-new_data[id,]
test<-new_data[-id,]

#Scaling the data
library(caret)
scaler <- preProcess(train)
train_scale_data <- predict(scaler, train)
test_scale_data <- predict(scaler, test)

model1 = lm(motor_UPDRS ~ 0 + ., data=train_scale_data)
summary(model1)


##MSE Train & Test
train_mse <- mean(model1$residuals^2)
# we can use this to estimate the training MSE train_mse <- sum(residuals(model1)^2)/nrow(train_
scale_data)

test_mse <- mean((test_scale_data$motor_UPDRS - predict(model1,test_scale_data))^2)
#we can use this to estimate the test MSE
#residuals_test <- predict(mod1, test_scale) - test_scale$motor_UPDRS
#test_mse <- sum(residuals_test^2)/nrow(test_scale)

list("Train MSEs"=train_mse,"Test MSEs"=test_mse)

##2.3(a).Loglihood Function

loglikelihood <- function(theta){

  X <- as.matrix(train_scale_data[,-1])
  n <- nrow(X)
  y <- train_scale_data[,1]
  sigma <- theta[17]
```

```
  theta_new <- theta[1:16]
  return( -(n/2) * log(2 * pi * sigma^2) - (1/(2*sigma^2))  * sum((y - X%*%theta_new)^2)  )

}

##2.3(b). Ridge Function

Ridge <- function(theta, lambda){

  return( -(loglikelihood(theta)) + lambda * sum(theta^2) )

}

##2.3(c). RidgeOpt Function

RidgeOpt <- function(lambda){

  optim(rep(1,17),fn = Ridge, method="BFGS", lambda = lambda)
}

##2.3(d). DF function

DF <- function(lambda){
  X <- as.matrix(train_scale_data[,-1])
  n <- nrow(X)
  df <- X %*% solve((t(X)%*%X + lambda* diag(ncol(X)))) %*% t(X)
  return( sum(diag(df)) )
}
Rid_opt_1 <- RidgeOpt(lambda = 1)
Rid_opt_100 <- RidgeOpt(lambda = 100)
Rid_opt_1000 <- RidgeOpt(lambda = 1000)
X_train <- as.matrix(train_scale_data[,-1])
X_test <- as.matrix(test_scale_data[,-1])

## Prediction with λ = 1
predicted_train <- X_train %*% Rid_opt_1$par[1:16]
predicted_test <- X_test %*% Rid_opt_1$par[1:16]

error_train_1 <- mean((predicted_train - train_scale_data$motor_UPDRS)^2)
error_test_1 <- mean((predicted_test - test_scale_data$motor_UPDRS)^2)

##Prediction with λ = 100
predicted_train <- X_train %*% Rid_opt_100$par[1:16]
predicted_test <- X_test %*% Rid_opt_100$par[1:16]

error_train_100 <- mean((predicted_train - train_scale_data$motor_UPDRS)^2)
error_test_100 <- mean((predicted_test - test_scale_data$motor_UPDRS)^2)

##Prediction with λ = 1000
predicted_train <- X_train %*% Rid_opt_1000$par[1:16]
predicted_test <- X_test %*% Rid_opt_1000$par[1:16]
```

```
error_train_1000 <- mean((predicted_train - train_scale_data$motor_UPDRS)^2)
error_test_1000 <- mean((predicted_test - test_scale_data$motor_UPDRS)^2)

list(error_train_1=error_train_1,error_train_100=error_train_100,error_train_1000=error_train_10
00)
list(error_test_1=error_test_1,error_test_100=error_test_100,error_test_1000=error_test_1000)

df_1<-DF(1)
df_100<-DF(100)
df_1000<-DF(1000)
result <- data.frame(lambda = c(1, 100, 1000),
                     MSE_Train = c(error_train_1,error_train_100,error_train_1000),
                     MSE_Test = c(error_test_1,error_test_100,error_test_1000),
                     DOF = c(df_1, df_100, df_1000) )
result

library("ggplot2")

#Loading data as a dataframe
data <- read.csv("pima-indians-diabetes.csv")

#changing column names for easy interpretation
colnames(data)[c(2,8,9)]<-c('PlasmaGlucoseConcentration','Age','Diabetes')
data$Diabetes <- as.factor(data$Diabetes)
plot1<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=Diabetes))
print(plot1)
#training a Logistic Regression model
train_model <- glm(Diabetes ~ PlasmaGlucoseConcentration + Age, data = data, family= "binomial")

#prediction of all observations
prediction<- predict(train_model, data, type = "response")
prediction_1<- ifelse(prediction>0.5,1,0)
train_model$coefficients
#Computing model accuracy using confusion matrix
table(data$Diabetes,prediction_1)
#A function to calculate misclassification error
missclass<- function(pred,actual){
  n=length(pred)
  return(1-sum(diag(table(pred, actual)))/n)
}

#Computing misclassification error for r= 0.5
missclass(data$Diabetes,prediction_1)
#scatter plot with predicted values of diabetes
data$PredictedDiabetes <- as.factor(prediction_1)
plot2<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=PredictedD
iabetes))
print(plot2)
#plotting decision boundary
x1<-15:85  #taking Age as x1
x2<-(5.89785793/0.03558250)-(0.02450157/0.03558250)*x1 # #taking Plasma Glucose as x2
boundary_df<-data.frame(x1,x2 )
```

```
plot3<-plot2 + geom_line(data = boundary_df, aes(x=x1,y=x2))
print(plot3)
#prediction with r=0.2
prediction_2<- ifelse(prediction>0.2,1,0)
data$PredictedDiabetes_2 <- as.factor(prediction_2)

#Computing model accuracy using confusion matrix
table(data$Diabetes,prediction_2)
#Computing misclassification error
missclass(data$Diabetes,prediction_2)
##scatter plot of classification with r=0.2
plot4<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=PredictedD
iabetes_2))
plot4<-plot4 + geom_line(data = boundary_df, aes(x=x1,y=x2))
print(plot4)
#prediction with r=0.8
prediction_3<- ifelse(prediction>0.8,1,0)
data$PredictedDiabetes_3 <- as.factor(prediction_3)
#Computing model accuracy using confusion matrix
table(data$Diabetes,prediction_3)
#Computing misclassification error
missclass(data$Diabetes,prediction_3)
#scatter plot of classification with r=0.8
plot5<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=PredictedD
iabetes_3))
plot5<-plot5 + geom_line(data = boundary_df, aes(x=x1,y=x2))
print(plot5)
#computing and adding new features to the data set
data$z1<-data$PlasmaGlucoseConcentration^4
data$z2<-(data$PlasmaGlucoseConcentration^3)*(data$Age)
data$z3<-(data$PlasmaGlucoseConcentration^2)*(data$Age^2)
data$z4<-(data$PlasmaGlucoseConcentration)*(data$Age^3)
data$z5<-data$Age^4

new_train_model <- glm(Diabetes ~ PlasmaGlucoseConcentration+Age+z1+z2+z3+z4+z5, data = data, fa
mily= "binomial")
new_prediction<- predict(new_train_model, data, type = "response")
new_prediction<- ifelse(new_prediction>0.5,1,0)
#Computing model accuracy using confusion matrix
table(data$Diabetes,new_prediction)
#Computing misclassification error
missclass(data$Diabetes,new_prediction)
#scatter plot of classification after adding new features
data$NewPredictedDiabetes <- as.factor(new_prediction)
plot6<- ggplot(data=data, aes(x=Age,y=PlasmaGlucoseConcentration))+geom_point(aes(col=NewPredict
edDiabetes))
print(plot6)
```