Lab 1 Report

Group B33

2023/11/19

Group Members: Priyansh Gupta, Afshan Hashemi Hosseinabad, Ragini Gunda

Statement of contribution: Assignment 1 is mainly done by Ragini Gunda, Priyansh Gupta was mainly responsible for assignment 2 and, assignment 3 is mostly done by Afshan Hashemi Hosseinabad. Results from all assignments are discussed afterwards between all of us and the group report is formulated based on the discussion.

Assignment 1: Handwritten digit recognition with K-nearest neighbours.

1. Import the data into R and divide it into training, validation and test sets (50%/25%/25%).

```
data <- read.csv("optdigits.csv",header = FALSE)
data$V65 <- as.factor(data$V65)
num <- dim(data)[1]
set.seed(12345)
id = sample(1:num, floor(num*0.5))
trainData <- data[id,]
id_1 <- setdiff(1:num,id)
set.seed(12345)
id_2 <- sample(id_1,floor(num*0.25))
validationData <- data[id_2,]
id_3 <- setdiff(id_1,id_2)
testData <- data[id_3,]
```

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

a) Confusion Matrices:

```
cm_1 <- table(trainData$V65,confmat_train)
print("Confusion matrix for the training data")
cm_1

```

Confusion matrix for the training data:

```
confmat_train
    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
```

```
cm_2 <- table(testData$V65,confmat_test)
print("Confusion matrix for the test data")
cm_2
```

Confusion matrix for the testing data:

confmat_test

	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

Comment on the quality of predictions for different digits:

It can be seen that model succeeded best on predicting digits 0 to fit the input image to the correct output in the training set, the quality of the model is as follows:

```
1. diagonal_values <- diag(cm_1)
2. print(diagonal_values)
3. sorted_indices <- order(diagonal_values,decreasing = TRUE)
4. print(sorted_indices)
5. sorted_values <- diagonal_values[sorted_indices]
6. print(sorted_values)
```

```
> print(diagonal_values)
  0   1   2   3   4   5   6   7   8   9
202 179 190 185 159 171 190 178 188 183
> sorted_indices <- order(diagonal_values,decreasing = TRUE)
> print(sorted_indices)
 [1]  1  3  7  9  4 10  2  8  6  5
> sorted_values <- diagonal_values[sorted_indices]
>
> print(sorted_values)
  0   2   6   8   3   9   1   7   5   4
202 190 190 188 185 183 179 178 171 159
```

The model is succeeded on predicting digit 7 in the test set. The quality of predictions for different digits is:

```
1. diagonal_values_test <- diag(cm_2)
2. print(diagonal_values_test)
3. sorted_indices_test <- order(diagonal_values_test,decreasing = TRUE)
4. print(sorted_indices_test)
5. sorted_values_test <- diagonal_values_test[sorted_indices_test]
6. print(sorted_values_test)
```

```
> diagonal_values_test <- diag(cm_2)
> print(diagonal_values_test)
  0   1   2   3   4   5   6   7   8   9
 77  81  98 107  94  93  90 111  70  85
> sorted_indices_test <- order(diagonal_values_test,decreasing = TRUE)
> print(sorted_indices_test)
 [1]  8  4  3  5  6  7 10  2  1  9
> sorted_values_test <- diagonal_values_test[sorted_indices_test]
> print(sorted_values_test)
  7   3   2   4   5   6   9   1   0   8
111 107  98  94  93  90  85  81  77  70
```

b). Miss-classification errors for the training and test data:

```
1. miss_class = function(y1,y2){
2.   n=length(y1)
3.   return(1-sum(diag(table(y1,y2)))/n)
4. }
5. print("Miss-classification of training data")
6. miss_class(trainData$V65,confmat_train)
```

```
[1] "Miss-classification of training data"
> miss_class(trainData$V65,confmat_train)
[1] 0.04500262
```

```
1. print("Miss-classification of test data")
2. miss_class(testData$V65,confmat_test)
```

```
[1] "Miss-classification of test data"
> miss_class(testData$V65,confmat_test)
[1] 0.05329154
```

```
1. accuracy = function(y1,y2)
2. {
3.   n=length(y1)
4.   return(sum(diag(table(y1,y2)))/n)
5.
6. }
7. print("Accuracy of the training data")
8. accuracy(trainData$V65,confmat_train)
9.
```

```
[1] "Accuracy of the training data"
> accuracy(trainData$V65,confmat_train)
[1] 0.9549974
```

```
1. print("Accuracy of the test data")
2. accuracy(testData$V65,confmat_test)
```

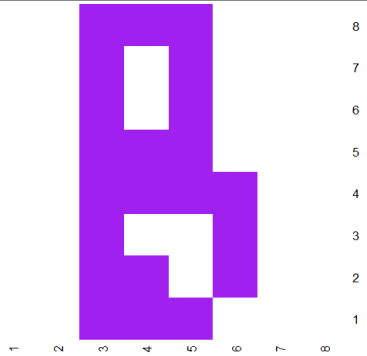
```
[1] "Accuracy of the test data"
> accuracy(testData$V65,confmat_test)
[1] 0.9467085
```

3. 3 Hardest and 2 easiest cases to classify:

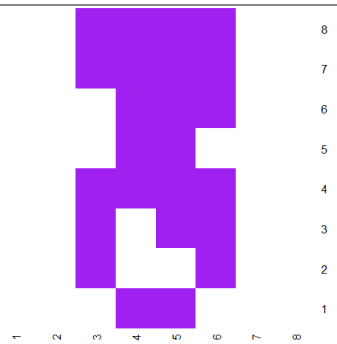
```
1. trainData_new <- trainData
2. trainData_new$prob <- train_kknn$prob[ ,"8"]
3. df <- data.frame(trainData_new)
4. hardest <- df[order(df$prob), ] %>% head(3) # 3 hardest cases to classify
5. easiest <- df[rev(order(df$prob)), ] %>% head(2) #2 easiest cases to classify
```

The easiest case:

```
1. heatmap(matrix(as.numeric(easiest[1,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv =
NA,col=c("white","purple"))
```

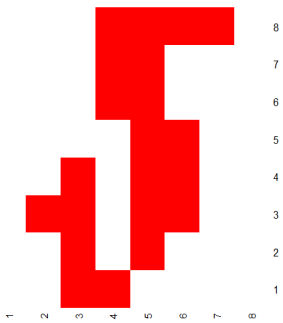


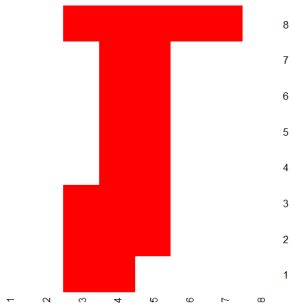
```
1.heatmap(matrix(as.numeric(easiest[2,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv =
NA,col=c("white","purple"))
2.
```

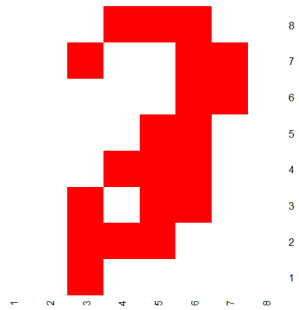


The Hardest case:

```
1. heatmap(matrix(as.numeric(hardest[1,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv =
NA,col=c("white","red"))
2.
3. heatmap(matrix(as.numeric(hardest[2,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv =
NA,col=c("white","red"))
4.
5. heatmap(matrix(as.numeric(hardest[3,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv =
NA,col=c("white","red"))
6.
```







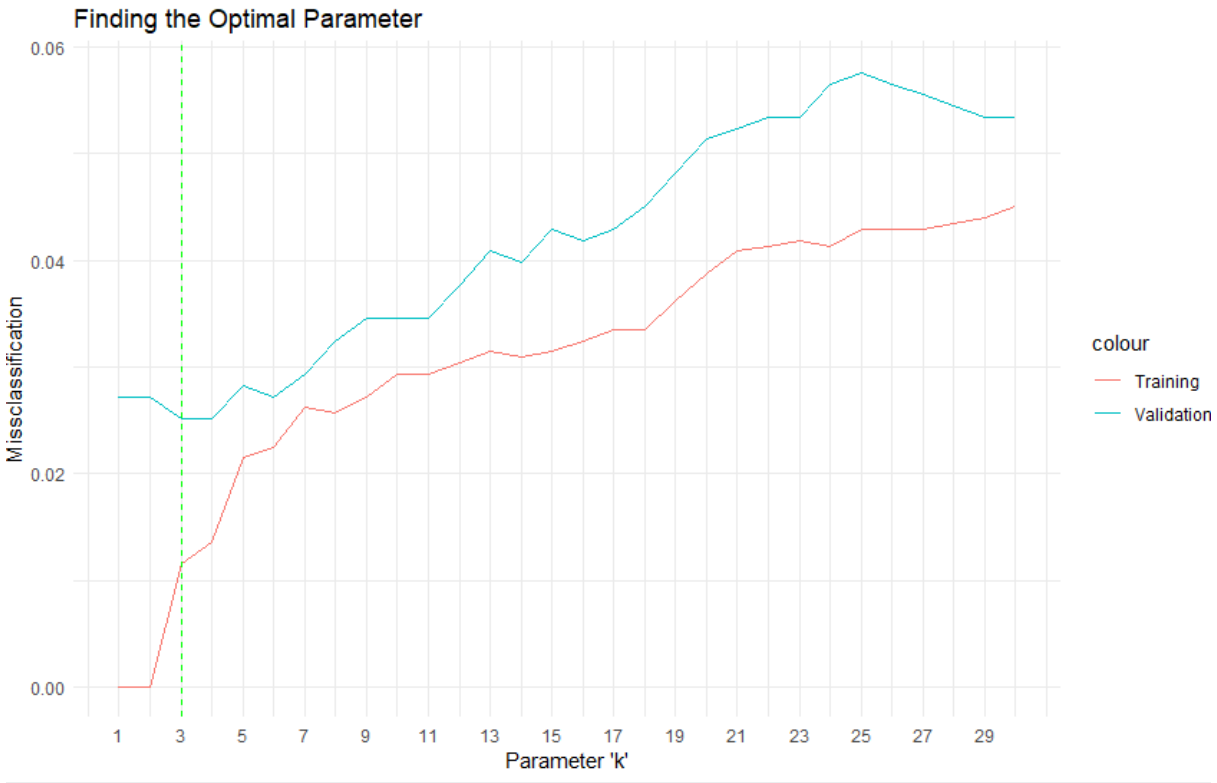
The heatmap for the easiest-to-classify cases shows clear and recognizable patterns of the digit '8.' The features are well-aligned with the typical structure of the digit, contributing to the model's accurate and confident predictions where as the heatmaps for the hardest-to-classify cases exhibit more ambiguous patterns. It appears challenging to visually discern the digit '8' in these instances, which aligns with the model's lower confidence in its predictions."

4. Fitting a K-nearest neighbor classifiers to the training data for different values of k=1,2,. . . ,30.

```
missclass_training <- c()
missclass_validation <- c()
for(k in 1:30){
  knn_training <- kknn(formula = V65~.,kernel = "rectangular",train =trainData,test = trainData,k=k
)
  knn_validation <- kknn(formula = V65~.,kernel = "rectangular",train =trainData,test =
validationData,k=k)
  missclass_training <- c(missclass_training,miss_class(trainData$V65,kknn_training$fitted.values))
  missclass_validation <-
c(missclass_validation,miss_class(validationData$V65,kknn_validation$fitted.values))
}
```

Plotting the dependence of the training and validation misclassification errors on the value of K:

```
k<-1:30
df1<-data.frame(missclass_training,k1 =k)
df2<-data.frame(missclass_validation,k2 =k)
df3 <- data.frame(df1,df2)
ggplot( ) +
  geom_line(aes(x=df3$k1,y=df3$missclass_training,
                colour="Training")) +
  geom_line(aes(x=df3$k2,y=df3$missclass_validation,
                colour="Validation")) +
  theme_minimal()+ggtitle("Finding the Optimal Parameter") +
  scale_x_continuous(breaks = seq(1,30,2)) +
  xlab("Parameter 'k'") + ylab("Missclassification") +
  geom_vline(xintercept = 3
             , linetype = "dashed",color = "green")
```



According to the plot, it can be seen that the optimal parameter for our model is *k* = 3 in which has the lowest Missclassification Error. In addition, when the model is tested with *k* = 3, we get the result as:

```
knn_test <- kknn(formula = V65~.,kernel = "rectangular",train =trainData,test = testData,k=3 )
cat("for k=3 :\n")
cat("MissClassification error on Training =",missclass_training[3],"\n")
```

```
4. cat("MissClassification error on Validation = ",missclass_validation[3],"\n")
5. cat("MissClassification error on Test = ",
6.     miss_class(testData$V65,kknn_test$fitted.values),"\n")
```

```
> cat("for k=3 :\n")
for k=3 :
> cat("MissClassification error on Training =",missclass_training[3],"\n")
MissClassification error on Training = 0.0115123
> cat("MissClassification error on Validation = ",missclass_validation[3],"\n")
MissClassification error on Validation =  0.02513089
> cat("MissClassification error on Test = ",
+     miss_class(testData$V65,kknn_test$fitted.values),"\n")
MissClassification error on Test =  0.02403344
```

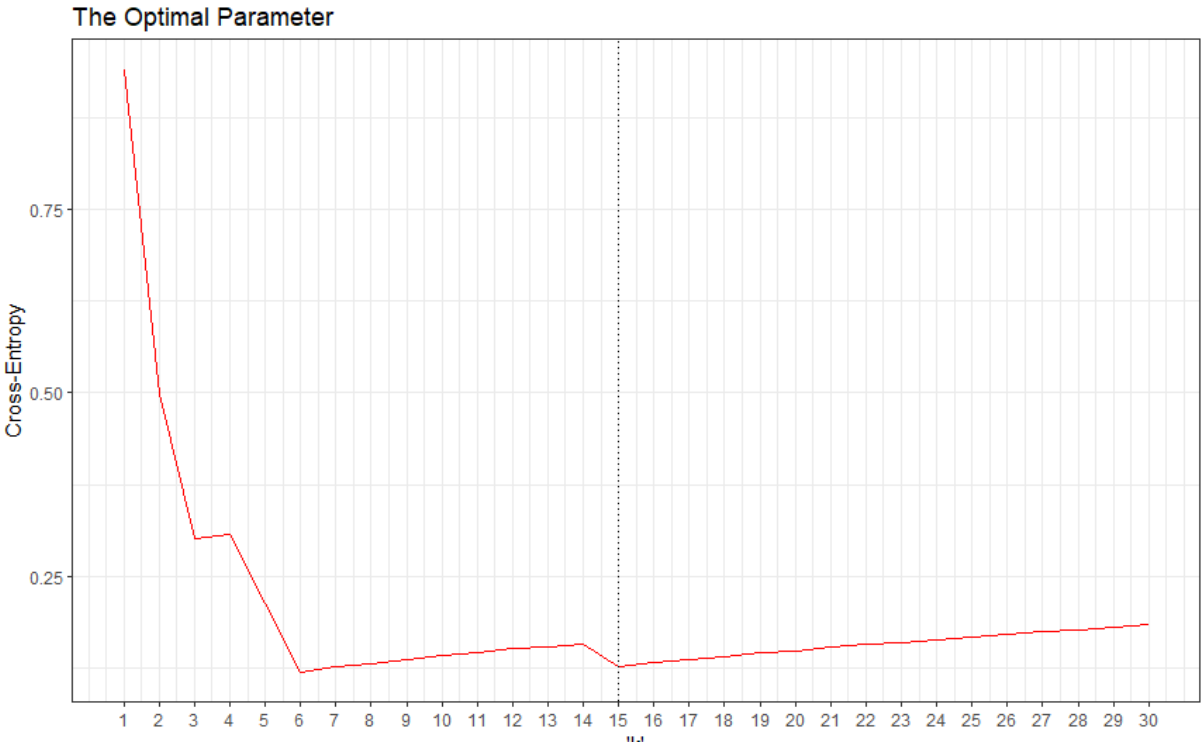
So, the miss-classification Error rate has the lowest value in the training set as expected. The miss-classification error is nearly equal on the Validation set, where the Test set shows that the model does not overfit. According to the plot, it can be perceived that the complexity of the model increases as the parameter k increases. Finally, we find that the optimal k value is 3.

In general, the complexity of the model increases as well as the variance in which means that when the complexity is high the observed variance is also high. When there is model complexity, we deal with a model with high bias. Our aim is to minimize these 2 variables so that there is always a trade-off between bias and variance. For k = 3; the bias is very small since the miss-classification error has the lowest and there is a small difference between the miss-classification Error on the training set and validation. That is why choosing k = 3 is the most suitable option.

5. **Fit K-nearest neighbor classifiers to training data for different values of k = 1, 2, . . . , 30 and compute the empirical risk for the validation data as cross-entropy.**

```
 1. library(data.table)
 2. library(kknn)
 3. crossEntropy_training <- c()
 4. crossEntropy_validation <- c()
 5. noOfClasses <- length(unique(trainData$V65))
 6. noOfRows_training <- nrow(trainData)
 7. noOfRows_validation <- nrow(validationData)
 8. training_onehot <- t(sapply(as.numeric(trainData$V65),function(x)
 9.   {c(rep(0,x-1), 1 ,rep(0 , noOfClasses-x ))}))
10.
11. validation_onehot <- t(sapply(as.numeric(validationData$V65),function(x)
12. {c(rep(0,x-1), 1 ,rep(0 , noOfClasses-x ))}))
13. for(i in 1:30)
14. {
15.   fitModel_training <- kknn(V65~.,kernel = "rectangular", train = trainData, test = trainData, k=i)
16.   fitModel_validation <- kknn(V65~.,kernel = "rectangular", train = trainData, test =
validationData, k=i)
17.
18.   #cross-entropy
19.   crossEntropy_training <- c(crossEntropy_training,sum(training_onehot * -log(fitModel_training$prob
+ 10^ - 15))/noOfRows_training)
20.
21.   crossEntropy_validation <- c(crossEntropy_validation,sum(validation_onehot * -
log(fitModel_validation$prob + 10^ - 15))/noOfRows_validation)
22.
23. }
24. crossEntropy_validation
```

```
25.
26. crossEntropy <- melt(data.table(k = 1:30, Validation = crossEntropy_validation), "k",variable.name =
" ")
27. ggplot(crossEntropy) + geom_line(aes(k, value),col="red") + theme_bw() +
28.   geom_vline(xintercept = 15, linetype = "dotted") + scale_x_continuous(breaks = seq(1,30)) +
29.   xlab("'k'") + ylab("Cross-Entropy") +
30.   ggtitle("The Optimal Parameter")
31.
```



The plot represents that $k = 6$ gives the lowest empirical risk error, so it can be defined as the optimal parameter for our model. From a probabilistic overview, miss-classification error is derived through Linear regression, while Cross-Entropy is from Logistic Regression. We want to classify the discrete return in which choosing Cross-Entropy is the suitable option. Moreover, unlikely to Misclassification Error, Cross Entropy penalizes the model more for wrong classification since it's derivation is higher.

Assignment 2: Linear regression and Ridge regression

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

```
parkinsons = read.csv("parkinsons.csv")
parkinson_data = as.data.frame(parkinsons[,c(5,7:22)])

n = dim(parkinson_data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.6))
traindata = parkinson_data[id,]
testdata = parkinson_data[-id,]
scaler <- preProcess(as.data.frame(traindata))
train_s <- predict(scaler,traindata)
test_s <- predict(scaler,testdata)
```

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.

```
lrm <- lm(motor_UPDRS ~ .-1, data = train_s)
train_predicted <- predict(lrm, newdata = train_s )
test_predicted <- predict(lrm, newdata = test_s)
MSETrain <- mean((train_predicted - train_s$motor_UPDRS)^2)
MSETest <- mean((test_predicted - test_s$motor_UPDRS)^2)
(c(MSE_of_train_data = MSETrain, MSE_of_test_data = MSETest ))
summary(lrm)
```

Output of the above code is:

```
(c(MSE_of_train_data = MSETrain, MSE_of_test_data = MSETest ))
MSE_of_train_data  MSE_of_test_data
        0.8785431         0.9354477
> summary(lrm)
Call:
lm(formula = motor_UPDRS ~ . - 1, data = train_s)
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
```

As we can see from summary variables Jitter.Abs, Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA, PPE contribute significantly to model with 0.001 significance level.

3. Implement 4 following basic functions by using basic R commands only (not using any external packages):

a. loglikelihood function

```
loglikelihood <- function(parameters) {
  theta <- parameters[1:16]
  sigma <- parameters[17]
  y <- as.matrix(train_s[,-1]) %*% theta
  n <- nrow(train_s)
  return (-n/2*log(2*pi) -n/2*log(sigma^2)-sum((train_s[,1]-y)^2)/(2*sigma^2))
}
```

b. Ridge function

```
ridgePenalty <- function(parameters, lambda){
  return(-loglikelihood(parameters)+ lambda*norm(as.matrix(parameters[1:16]),type="2")^2)
}
```

c. RidgeOPt function

```
ridgeOpt <- function(lambda){
  return (optim( par = c(rep(0,length=16),0.01),fn = ridgePenalty, lambda = lambda, method ="BFGS"))
}
```

d. DF function

```
df <- function(lambda){
  X <- as.matrix(train_s[,-1])
  P <- X %*% solve(t(X)%*%X + lambda*diag(16))%*%t(X)
  return(sum(diag(P)))
}
```

4. By using the RidgeOpt() method compute the optimal parameter $\boldsymbol{\theta}$ for $\lambda = 1, 10, 1000$.

```
MSE_Matrix<-matrix(c(MSE_Train_1,MSE_Test_1,MSE_Train_100,MSE_Test_100,
MSE_Train_1000,MSE_Test_1000), nrow =2, ncol = 3)
rownames(MSE_Matrix) <-c("Train_MSE",'Test_MSE')
colnames(MSE_Matrix) <-c("λ=1","λ=100","λ=1000")
print(MSE_Matrix)
```

Output is as follow:

```
print(MSE_Matrix)
                λ=1     λ=100    λ=1000
Train_MSE 0.8786272 0.8844091 0.9211176
Test_MSE  0.9349961 0.9323369 0.9539460
```

- **Which penalty parameter is most appropriate among the selected one?**
  As MSE value for testing data is lowest among for $\lambda = 100$ among all the selected values. It is the most appropriate value for penalty parameter.

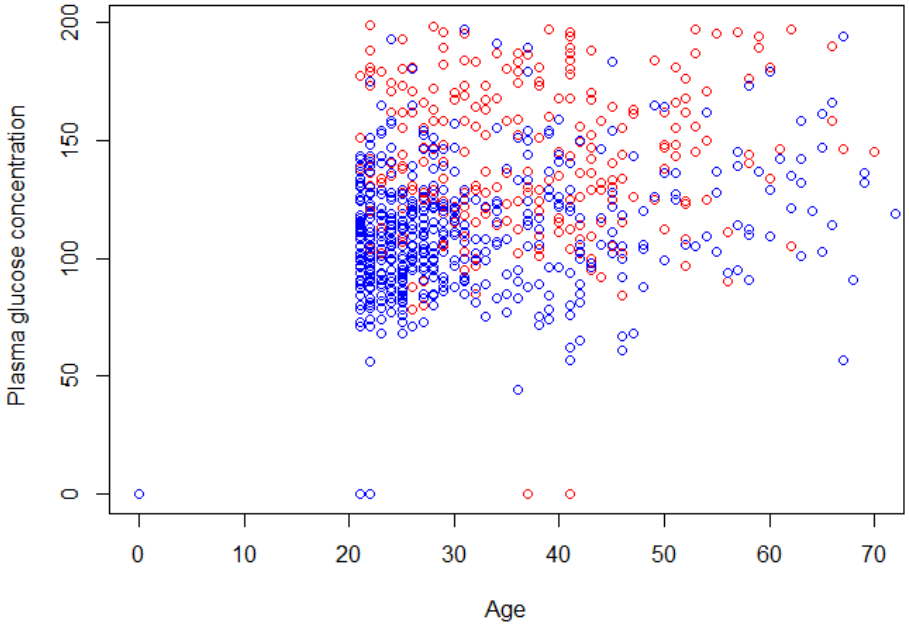
compute and compare the degree of freedom of these models and make appropriate conclusion.

```
c(DF_1 = df(1),DF_100 = df(100),DF_1000 = df(1000))
      DF_1    DF_100   DF_1000
13.860736  9.924887  5.643925
```

As it can be clearly seen from the result DF is inversely propositional to penalty parameter λ, i.e., df decreases as penalty parameter increases.

Assignment 3: Logistic regression and basic function

1. It is not easy to classify diabetes only using Plasma glucose concentration and Age since as we see in the scatterplot the points are not well separated and it is hard to draw a line or a curve between two groups that divide them nicely.



2. In this assignment we have divided the data 50/50 and used the code below to train our logistic models on the "train" data set and get predictions for the "test" data set

```
1)  logistic=glm(Diabetes ~ Age + Plasma,data = train,family = "binomial")
2)  LRpred=predict(logistic,newdata = test,type="response")
3)  cutoff=0.5
4)  LRpred[LRpred<cutoff]=0
5)  LRpred[LRpred>=cutoff]=1
6)  summary(logistic)
```

As we run the above code we will get the following results:

```
Call:
glm(formula = Diabetes ~ Age + Plasma, family = "binomial", data = train)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.047850   0.664918  -9.096  < 2e-16 ***
Age          0.047579   0.010939   4.349 1.37e-05 ***
Plasma       0.031033   0.004524   6.860 6.88e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 497.98  on 383  degrees of freedom
Residual deviance: 399.86  on 381  degrees of freedom
```

```
AIC: 405.86

Number of Fisher Scoring iterations: 4
```

The coefficients values for both age and plasma are fairly small so we have chosen good parameters. Based on the coefficient values we get the following equation:

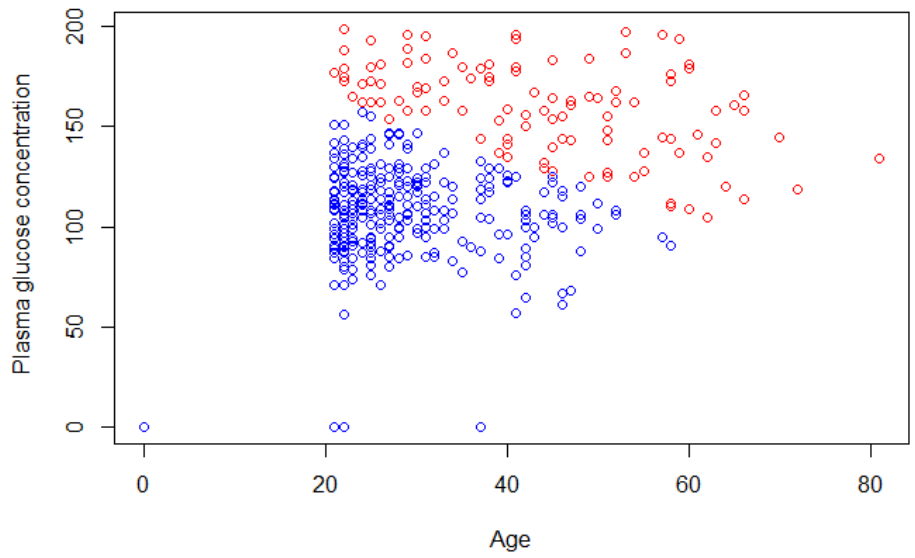
$$P(Diabetes) = sigm(-6.047 + 0.047 \cdot Age + 0.031 \cdot Plasma)$$

To calculate misclassification rate we should:

```
1)  confusionMX=table(LRpred,test$Diabetes)
2)  misclassificationRate=(1 - sum(diag(confusionMX))/sum(confusionMX))
```

Which outputs: 0.255

The scatterplot for the test data:



Unlike step one in this plot the blue and red pints can be divided easily so the quality of the classification is good.

3.

A) The equation of the decision boundary can be calculated using coefficients in oprder to that we need to calculate the slope and intercept of the line:

```
1.  coeff=logistic$coefficients
2.  slope=coeff[2]/(-1*coeff[3])
3.  intercept=coeff[1]/(-1*coeff[3])
```

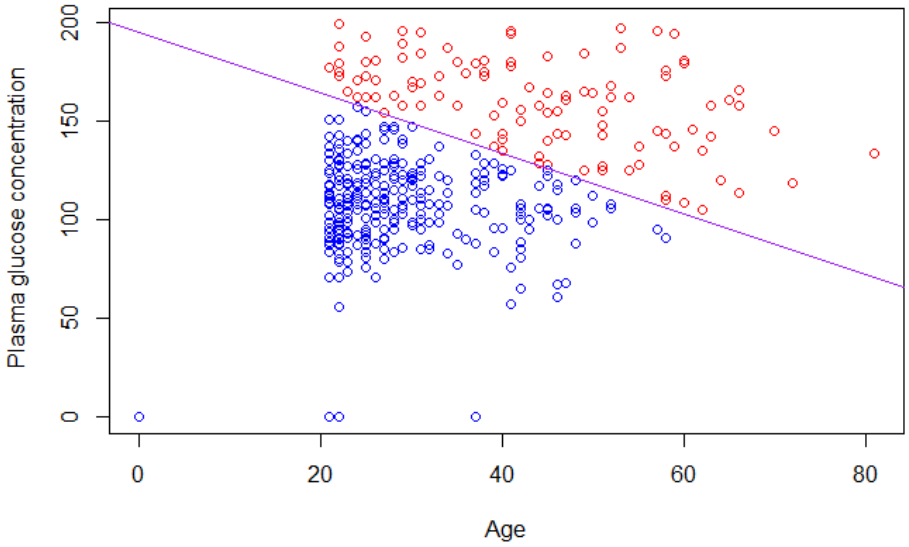
Using this calculation $slope = -1.53$ and $intercept = 195$ so we have:

$$y = 195 - x(1.53)$$

B) We can add a line in the data using the following code:

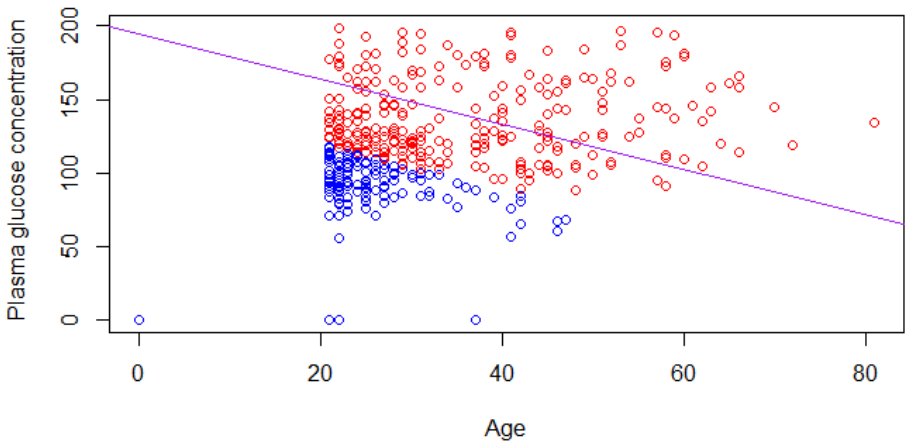
```
1)  abline(intercept,slope,col="purple")
```

We will get the following plot:

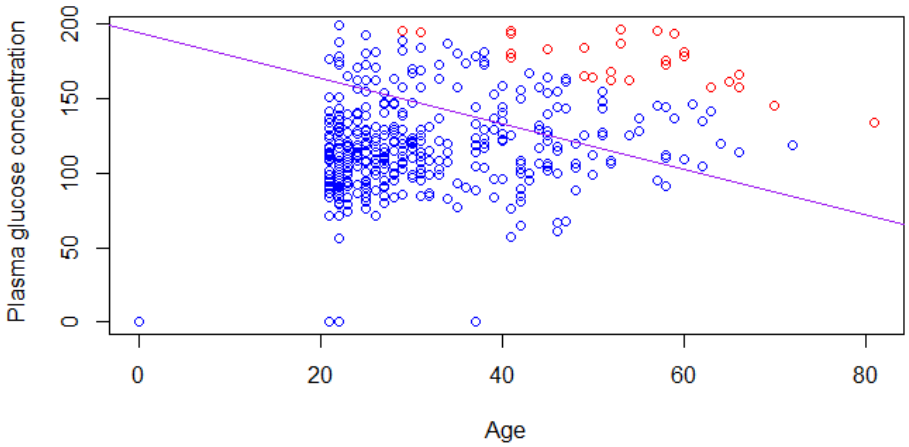


As we can see the two sides of the purple line has different colours, so the decision boundary has cached the data distribution well.

4. If we set $r = 0.2$ we will get the plot below and as we can see the model in not very accurate since there are some patients without diabetes that are classified as having it.



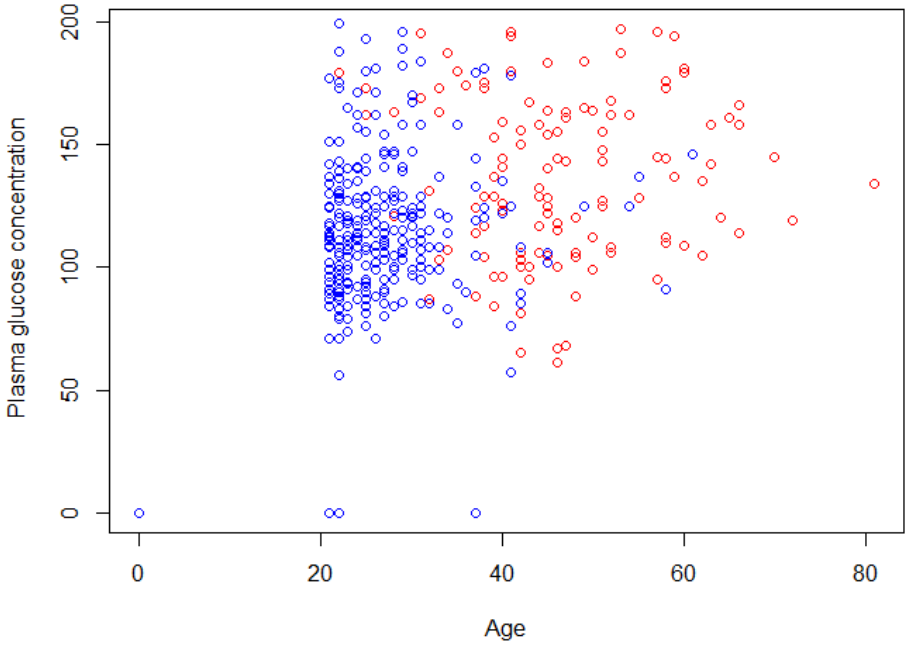
If we set $r = 0.8$, this is even worse than the previous case in medical terms since there are people with diabetes that are classified as non-diabetes, and this can endanger their life.



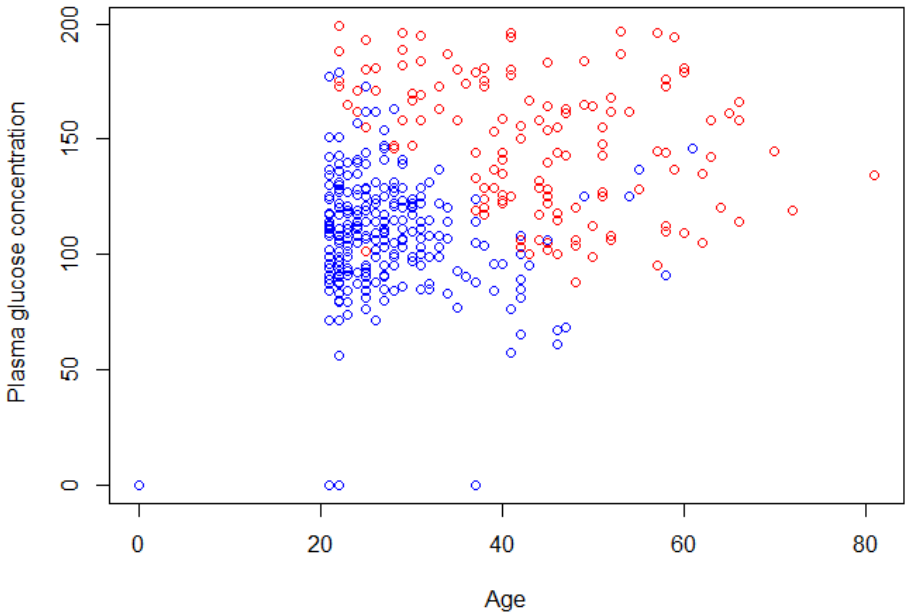
5. We can use the following code to add new binomial features to train our model with:

```
1)  train$phi1=phi1=train$Plasma^4
2)  train$phi2=phi2=train$Plasma^3*train$Age
3)  train$phi3=phi3=train$Plasma^2*train$Age^1
4)  train$phi4=phi4=train$Plasma^1*train$Age^3
5)  train$phi5=phi5=train$Age^4
```

As we train the model with the new features, we will get the following plot:



Misclassification rate in this model is 0.372 so, it does not work well and does not have a good accuracy compared to the model in part 2 but it has more of a curve compared to the previous decision boundary. But we can improve it by using more variables. For instance, we can use 7 variables, and we get the plot below with 0.299 misclassification rate.


Plasma glucose concentration
0
50
100
150
200
0
20
40
60
80
Age

Appendix:

Code for assignment 1:

```r
#install.packages("tidyverse")
library(tidyverse)
library(kknn)
library(dbplyr)

data <- read.csv("optdigits.csv",header = FALSE)
data$V65 <- as.factor(data$V65)
num <- dim(data)[1]
set.seed(12345)
id = sample(1:num, floor(num*0.5))
trainData <- data[id,]
id_1 <- setdiff(1:num,id)
set.seed(12345)
id_2 <- sample(id_1,floor(num*0.25))
validationData <- data[id_2,]
id_3 <- setdiff(id_1,id_2)
testData <- data[id_3,]

#install.packages("kknn")

train_kknn <- kknn(V65~., train = trainData, test = trainData, kernel = "rectangular" , k=30)
test_kknn <- kknn(V65~., train = trainData, test = testData, kernel="rectangular",k=30)
confmat_train <- train_kknn$fitted.values
confmat_test <- test_kknn$fitted.values
cm_1 <- table(trainData$V65,confmat_train)
print("Confusion matrix for the training data")
cm_1
cm_2 <- table(testData$V65,confmat_test)
print("Confusion matrix for the test data")
cm_2
diagonal_values <- diag(cm_1)
print(diagonal_values)
sorted_indices <- order(diagonal_values,decreasing = TRUE)
print(sorted_indices)
sorted_values <- diagonal_values[sorted_indices]

print(sorted_values)

diagonal_values_test <- diag(cm_2)
print(diagonal_values_test)
sorted_indices_test <- order(diagonal_values_test,decreasing = TRUE)
print(sorted_indices_test)
sorted_values_test <- diagonal_values_test[sorted_indices_test]

print(sorted_values_test)

miss_class = function(y1,y2){
  n=length(y1)
  return(1-sum(diag(table(y1,y2)))/n)
}
print("Miss-classification of training data")
miss_class(trainData$V65,confmat_train)

print("Miss-classification of test data")
miss_class(testData$V65,confmat_test)

accuracy = function(y1,y2)
{
  n=length(y1)
  return(sum(diag(table(y1,y2)))/n)

}
print("Accuracy of the training data")
accuracy(trainData$V65,confmat_train)
```

```r
print("Accuracy of the test data")
accuracy(testData$V65,confmat_test)
#install.packages("dbplyr")
trainData_new <- trainData
trainData_new$prob <- train_kknn$prob[ ,"8"]
df <- data.frame(trainData_new)
hardest <- df[order(df$prob), ] %>% head(3) # 3 hardest cases to classify
easiest <- df[rev(order(df$prob)), ] %>% head(2) #2 easiest cases to classify
heatmap(matrix(as.numeric(easiest[1,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv = NA,col=c("white","purple"))

heatmap(matrix(as.numeric(easiest[2,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv = NA,col=c("white","purple"))

heatmap(matrix(as.numeric(hardest[1,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv = NA,col=c("white","red"))

heatmap(matrix(as.numeric(hardest[2,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv = NA,col=c("white","red"))

heatmap(matrix(as.numeric(hardest[3,1:64]),byrow = TRUE,nrow = 8,ncol = 8),Colv = NA,Rowv = NA,col=c("white","red"))

missclass_training <- c()
missclass_validation <- c()
for(k in 1:30){
  kknn_training <- kknn(formula = V65~.,kernel = "rectangular",train =trainData,test = trainData,k=k )
  kknn_validation <- kknn(formula = V65~.,kernel = "rectangular",train =trainData,test = validationData,k=k)
  missclass_training <- c(missclass_training,miss_class(trainData$V65,kknn_training$fitted.values))
  missclass_validation <- c(missclass_validation,miss_class(validationData$V65,kknn_validation$fitted.values))
}

library(ggplot2)
k<-1:30
df1<-data.frame(missclass_training,k1 =k)
df2<-data.frame(missclass_validation,k2 =k)
df3 <- data.frame(df1,df2)
ggplot( ) +
  geom_line(aes(x=df3$k1,y=df3$missclass_training,
                colour="Training")) +
  geom_line(aes(x=df3$k2,y=df3$missclass_validation,
                colour="Validation")) +
  theme_minimal()+ggtitle("Finding the Optimal Parameter") +
  scale_x_continuous(breaks = seq(1,30,2)) +
  xlab("Parameter 'k'") + ylab("Missclassification") +
  geom_vline(xintercept = 3
             , linetype = "dashed",color = "green")

kknn_test <- kknn(formula = V65~.,kernel = "rectangular",train =trainData,test = testData,k=3 )
cat("for k=3 :\n")
cat("MissClassification error on Training =",missclass_training[3],"\n")
cat("MissClassification error on Validation = ",missclass_validation[3],"\n")
cat("MissClassification error on Test = ",
    miss_class(testData$V65,kknn_test$fitted.values),"\n")
library(data.table)
library(kknn)
crossEntropy_training <- c()
crossEntropy_validation <- c()
noOfClasses <- length(unique(trainData$V65))
noOfRows_training <- nrow(trainData)
noOfRows_validation <- nrow(validationData)
training_onehot <- t(sapply(as.numeric(trainData$V65),function(x)
  {c(rep(0,x-1), 1 ,rep(0 , noOfClasses-x ))}))

validation_onehot <- t(sapply(as.numeric(validationData$V65),function(x)
{c(rep(0,x-1), 1 ,rep(0 , noOfClasses-x ))}))
for(i in 1:30)
{
```

```
  fitModel_training <- kknn(V65~.,kernel = "rectangular", train = trainData, test = trainData, k=i)
  fitModel_validation <- kknn(V65~.,kernel = "rectangular", train = trainData, test =
validationData, k=i)

  #cross-entropy
  crossEntropy_training <- c(crossEntropy_training,sum(training_onehot * -
log(fitModel_training$prob + 10^ - 15))/noOfRows_training)

  crossEntropy_validation <- c(crossEntropy_validation,sum(validation_onehot * -
log(fitModel_validation$prob + 10^ - 15))/noOfRows_validation)

}
crossEntropy_validation

crossEntropy <- melt(data.table(k = 1:30, Validation = crossEntropy_validation), "k",variable.name
= " ")
ggplot(crossEntropy) + geom_line(aes(k, value),col="red") + theme_bw() +
  geom_vline(xintercept = 15, linetype = "dotted") + scale_x_continuous(breaks = seq(1,30)) +
  xlab("'k'") + ylab("Cross-Entropy") +
  ggtitle("The Optimal Parameter")

```

Code for assignment2:

```
require(ggplot2)
require(lattice)
library(caret)
#1. Divide it into training and test data (60/40) and scale it appropriately.
#In the coming steps, assume that motor_UPDRS is normally distributed and is a
#function of the voice characteristics, and since the data are scaled, no
#intercept is needed in the modelling.

parkinsons = read.csv("parkinsons.csv")
parkinson_data = as.data.frame(parkinsons[,c(5,7:22)])

n = dim(parkinson_data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.6))
traindata = parkinson_data[id,]
testdata = parkinson_data[-id,]
scaler <- preProcess(as.data.frame(traindata))
train_s <- predict(scaler,traindata)
test_s <- predict(scaler,testdata)

#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.

lrm <- lm(motor_UPDRS ~ .-1, data = train_s)
train_predicted <- predict(lrm, newdata = train_s )
test_predicted <- predict(lrm, newdata = test_s)
MSETrain <- mean((train_predicted - train_s$motor_UPDRS)^2)
MSETest <- mean((test_predicted - test_s$motor_UPDRS)^2)
(c(MSE_of_train_data = MSETrain, MSE_of_test_data = MSETest ))
summary(lrm)

#3.Implement 4 following basic functions by using basic R commands only ( not using any external
packages):
#a. loglikelihood function that for a given vector𝜽 and dispersion 𝜎 compute the log-likelihood
function
#logP(T|𝜽,𝜎) for the states model and traning data.

loglikelihood <- function(parameters) {
  theta <- parameters[1:16]
  sigma <- parameters[17]
  y <- as.matrix(train_s[,-1]) %*% theta
  n <- nrow(train_s)
  return (-n/2*log(2*pi) -n/2*log(sigma^2)-sum((train_s[,1]-y)^2)/(2*sigma^2))
}

```

```r
#b.Ridge function that for a given vectorθ, scalar σ and scalar λuses function from 3a
# and adds up a Ridge penalty λ‖θ‖^2 to the minus log likelihood.

ridgePenalty <- function(parameters, lambda){
  return(-loglikelihood(parameters)+ lambda*norm(as.matrix(parameters[1:16]),type="2")^2)
}

#c. RidgeOPt function that depends on scalar λ, uses function from 3b and function optim()
# with method="BFGS" to find the optimalθ and σ for given value of λ.

ridgeOpt <- function(lambda){
  return (optim( par = c(rep(0,length=16),0.01), fn = ridgePenalty, lambda = lambda, method =
"BFGS"))
}

#d. DF function that for a given scalar λ computes the degree of freedom of
#the Ridgemodel based on training data.
df <- function(lambda){
  X <- as.matrix(train_s[,-1])
  P <- X %*% solve(t(X)%*%X + lambda*diag(16))%*%t(X)
  return(sum(diag(P)))
}

#4. by using RidgeOpt(), compute optimal θ forλ =1, λ=100, λ=1000, use the estimated parameter to
#predict the motor_UPDRS value for training and test data and report the training and Test MSE
values.
# which penalty parameter is most appropriate among the selected one? compute and compare the degree
of freedom
# of these model and make appropriate conclusion.

ridgeOpt(lambda = 1)
train_predict <- as.matrix(train_s[,-1]) %*% ridgeOpt(lambda = 1)$par[1:16]
test_predict <- as.matrix(test_s[,-1]) %*% ridgeOpt(lambda = 1)$par[1:16]
MSE_Train_1 <- sum((train_predict - train_s$motor_UPDRS)^2) / nrow(train_s)
MSE_Test_1 <- sum((test_predict - test_s$motor_UPDRS)^2) / nrow(test_s)


ridgeOpt(lambda = 100)
train_predict <- as.matrix(train_s[,-1]) %*% ridgeOpt(lambda = 100)$par[1:16]
test_predict <- as.matrix(test_s[,-1]) %*% ridgeOpt(lambda = 100)$par[1:16]
MSE_Train_100 <- sum((train_predict - train_s$motor_UPDRS)^2) / nrow(train_s)
MSE_Test_100 <- sum((test_predict - test_s$motor_UPDRS)^2) / nrow(test_s)


ridgeOpt(lambda = 1000)
train_predict <- as.matrix(train_s[,-1]) %*% ridgeOpt(lambda = 1000)$par[1:16]
test_predict <- as.matrix(test_s[,-1]) %*% ridgeOpt(lambda = 1000)$par[1:16]
MSE_Train_1000 <- sum((train_predict - train_s$motor_UPDRS)^2) / nrow(train_s)
MSE_Test_1000 <- sum((test_predict - test_s$motor_UPDRS)^2) / nrow(test_s)

MSE_Matrix<-
matrix(c(MSE_Train_1,MSE_Test_1,MSE_Train_100,MSE_Test_100,MSE_Train_1000,MSE_Test_1000), nrow =2, ncol
= 3)
rownames(MSE_Matrix) <-c("Train_MSE", "Test_MSE")
colnames(MSE_Matrix) <-c("λ=1","λ=100","λ=1000")
print(MSE_Matrix)

c(DF_1 = df(1),DF_100 = df(100),DF_1000 = df(1000))

```

Code for assignment 3

Part1

```r
data=read.csv("pima-indians-diabetes.csv",header = FALSE)

colnames(data)=c("Number of times pregnant",
  "Plasma",
  " Diastolic blood pressure",
  "Triceps skinfold thickness",
```

```
  "2-Hour serum insulin",
  "Body mass index",
  "Diabetes pedigree function",
  "Age",
  "Diabetes")

haveDiabetes= (as.integer(unlist((data$Diabetes))))
dontHaveDiabetes= -1*(as.integer(unlist((data$Diabetes)))-1)
plasma= as.integer(unlist((data$`Plasma`)))
age= as.integer(unlist((data$Age)))
plot(haveDiabetes*age,haveDiabetes*plasma,ylab = " Plasma glucose concentration",
     xlab = "Age",col="red")
points(dontHaveDiabetes*age,dontHaveDiabetes*plasma,col="blue")

```

Part2,3

```
data=read.csv("pima-indians-diabetes.csv",header = FALSE)

colnames(data)=c("Number of times pregnant",
                 "Plasma",
                 " Diastolic blood pressure",
                 "Triceps skinfold thickness",
                 "2-Hour serum insulin",
                 "Body mass index",
                 "Diabetes pedigree function",
                 "Age",
                 "Diabetes")
# Devide date
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
# Train model

logistic=glm(Diabetes ~ Age + Plasma,data = train,family = "binomial")
LRpred=predict(logistic,newdata = test,type="response")
cutoff=0.5
LRpred[LRpred<cutoff]=0
LRpred[LRpred>=cutoff]=1
summary(logistic)

#Misclassification rate

confusionMX=table(LRpred,test$Diabetes)
misclassificationRate=(1 - sum(diag(confusionMX))/sum(confusionMX))

#Scatterplot
haveDiabetes=as.integer(unlist(LRpred))
dontHaveDiabetes= -1*(as.integer(unlist((LRpred)))-1)
plasma= as.integer(unlist((test$`Plasma`)))
age= as.integer(unlist((test$Age)))
plot(haveDiabetes*age,haveDiabetes*plasma,ylab = " Plasma glucose concentration",
     xlab = "Age",col="red")
points(dontHaveDiabetes*age,dontHaveDiabetes*plasma,col="blue")

###### Part3

#Calculate equation
coeff=logistic$coefficients
slope=coeff[2]/(-1*coeff[3])
intercept=coeff[1]/(-1*coeff[3])
#Plot decision boundary
abline(intercept,slope,col="purple")
```

Part4

```
###Seting r=0.2
data=read.csv("pima-indians-diabetes.csv",header = FALSE)
```

```r

colnames(data)=c("Number of times pregnant",
                 "Plasma",
                 " Diastolic blood pressure",
                 "Triceps skinfold thickness",
                 "2-Hour serum insulin",
                 "Body mass index",
                 "Diabetes pedigree function",
                 "Age",
                 "Diabetes")
# Devide date
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
# Train model

logistic=glm(Diabetes ~ Age + Plasma,data = train,family = "binomial")
LRpred=predict(logistic,newdata = test,type="response")
cutoff=0.2
LRpred[LRpred<cutoff]=0
LRpred[LRpred>=cutoff]=1

haveDiabetes=as.integer(unlist(LRpred))
dontHaveDiabetes= -1*(as.integer(unlist((LRpred)))-1)
plasma= as.integer(unlist((test$`Plasma`)))
age= as.integer(unlist((test$Age)))
plot(haveDiabetes*age,haveDiabetes*plasma,ylab = " Plasma glucose concentration",
     xlab = "Age",col="red")
points(dontHaveDiabetes*age,dontHaveDiabetes*plasma,col="blue")
coeff=logistic$coefficients
slope=coeff[2]/(-1*coeff[3])
intercept=coeff[1]/(-1*coeff[3])

#Plot decision boundary
abline(intercept,slope,col="purple")


```

Part5

```r
data=read.csv("pima-indians-diabetes.csv",header = FALSE)

colnames(data)=c("Number of times pregnant",
                 "Plasma",
                 " Diastolic blood pressure",
                 "Triceps skinfold thickness",
                 "2-Hour serum insulin",
                 "Body mass index",
                 "Diabetes pedigree function",
                 "Age",
                 "Diabetes")
# Devide date
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

#new features
train$phi1=phi1=train$Plasma^4
train$phi2=phi2=train$Plasma^3*train$Age
train$phi3=phi3=train$Plasma^2*train$Age^1
train$phi4=phi4=train$Plasma^1*train$Age^3
train$phi5=phi5=train$Age^4

logistic=glm(Diabetes ~ Plasma +Age+phi1+phi2+phi3+phi4+phi5
             ,data = train,family = "binomial")
LRpred=predict(logistic,newdata = test,type="response")
```

```
cutoff=0.5
LRpred[LRpred<cutoff]=0
LRpred[LRpred>=cutoff]=1
summary(logistic)

#Misclassification rate

confusionMX=table(LRpred,test$Diabetes)
misclassificationRate=(1 - sum(diag(confusionMX))/sum(confusionMX))

#Scatterplot
haveDiabetes=as.integer(unlist(LRpred))
dontHaveDiabetes= -1*(as.integer(unlist((LRpred)))-1)
plasma= as.integer(unlist((test$`Plasma`)))
age= as.integer(unlist((test$Age)))
plot(haveDiabetes*age,haveDiabetes*plasma,ylab = " Plasma glucose concentration",
     xlab = "Age",col="red")
points(dontHaveDiabetes*age,dontHaveDiabetes*plasma,col="blue")

```