Machine Learning Computer lab 1 block 1

Muhammad Umair, Usama Nadeem & Muhaiminul Islam

2022-11-19

Statement of contribution

The first assignment of lab 1 was done by group member Muhammad Umair. The second task was done by Usama Nadeem. The third task was done by Muhaiminul Islam. The whole report was compiled by Muhaiminul Islam.

Assignment 1. Handwritten digit recognition with K-nearest neighbors

1. Import the data into R and divide it into training, validation and test sets (50%/25%/25%) by using the partitioning principle specified in the lecture slides.

```
my_data <- read.csv("optdigits.csv", header = FALSE)
#converting target field into factor
my_data$V65 <- as.factor(my_data$V65)
n<-dim(my_data)[1]
set.seed(12345)
#spliting data
id<-sample(1:n, floor(n*0.5))
train<-my_data[id,]
rem_data<-my_data[-id,]

new_n<-dim(rem_data)[1]
set.seed(12345)
new_id<-sample(1:new_n, floor(new_n*0.5))
validation<-rem_data[new_id,]
test<-rem_data[-new_id,]
```

2. Use training data to fit 30-nearest neighbor classifier with function kknn() and kernel=”rectangular” from package kknn and estimate 1.Confusion matrices for the training and test data (use table()) 2. Misclassification errors for the training and test data Comment on the quality of predictions for different digits and on the overall prediction quality.

```
knn_train_classifier <- kknn(V65 ~. ,train = train,test = train,k = 30,kernel = "rectangular")
knn_test_classifier <- kknn(V65 ~. ,train = train,test = test,k = 30,kernel = "rectangular")
fitted_data <- fitted(knn_train_classifier)
predict <- fitted(knn_test_classifier)
confusion_matrix<-table(actual=train$V65, predicted=fitted_data)
```

```
confusion_matrix
```

```
##       predicted
## actual   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 is = 0.04500262
```

On training data, our model predicts digit 0 very well as can be seen in the confusion matrix that 100% of 0 digits are correctly predicted, and digit 4 is hardest to classify because the actual digit is 4 but it is predicted as 0,1,7,8,9 in 1,3,7,1,4 cases respectively. The accuracy of classifying digit 1 is also low because the actual number is 1 but it is classified as 2 in eleven cases.

```
##       predicted
## actual   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  92   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 is = 0.05334728
```

Misclassification error is = 0.05334728 Misclassification error rate of test data is lower than training data, as can be seen that model have problem in classifying 4 and 8. Actual digit is 8 and our model predicted it as 1 in seven cases.

3. Find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class). Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits (by using e.g. heatmap() function with parameters Colv=NA and Rowv=NA) and comment on whether these cases seem to be hard or easy to recognize visually.

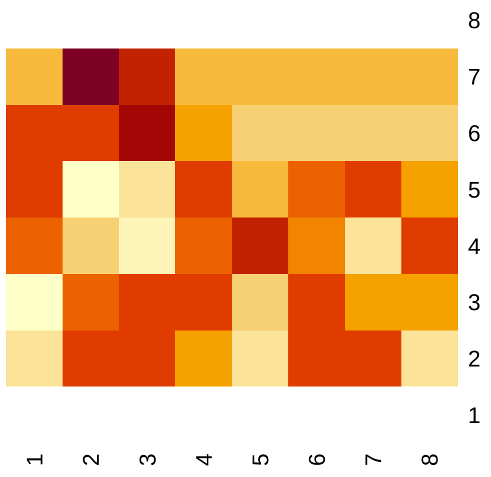
```
## The two indexes that are easy to identify as 8 are  10 14
```

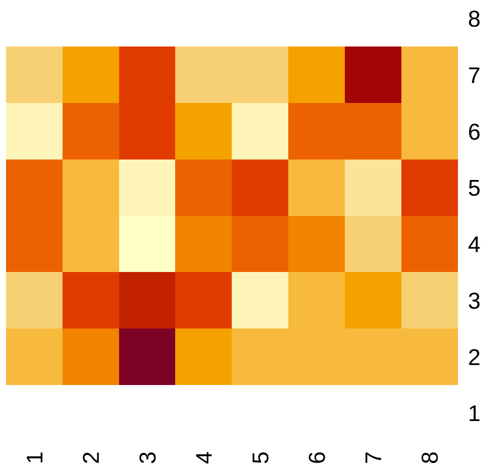
The two easiest classifying digit as 8 are at 10 and 14 indexes. The probability of these two indexes are high so these are easiest to classify.

```
## The three indexes that are hard to identify as 8 are  50 43 136
```

These three indexes are very hard to classify as digit "8" because the probability of classifying these indexes is the lowest.

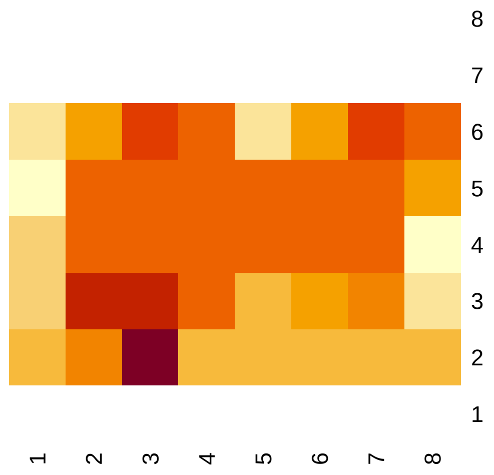
```r
for(grp in 1:length(easiest)){
dt<-matrix(data = as.numeric(train[indexes_of_eight[easiest[grp]],1:64])
,nrow = 8,ncol = 8)
heatmap(dt,Colv = NA, Rowv = NA)
}
```

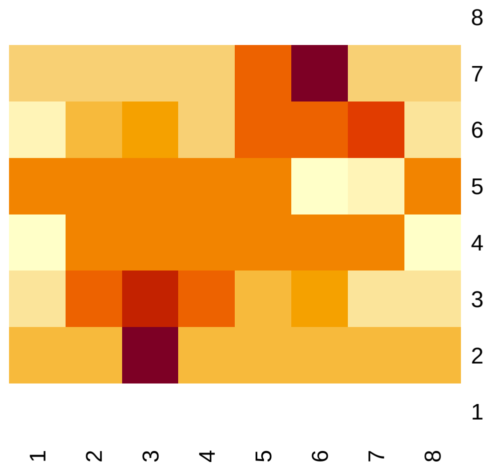


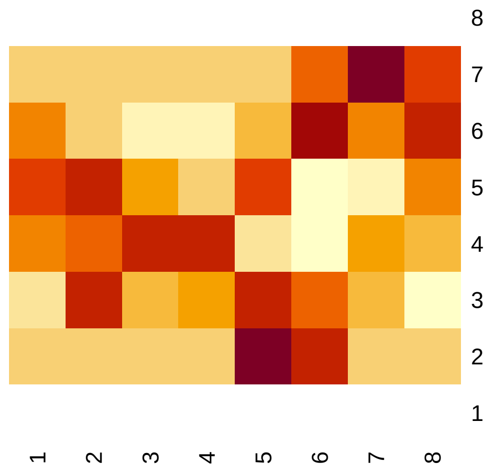


As can be seen that these digits are easy to recognize as 8.

```r
for(grp in 1:length(hardest)){
dt<-matrix(data = as.numeric(train[indexes_of_eight[hardest[grp]],1:64])
,nrow = 8,ncol = 8)
heatmap(dt,Colv = NA, Rowv = NA)}
```


1
2
3
4
5
6
7
8

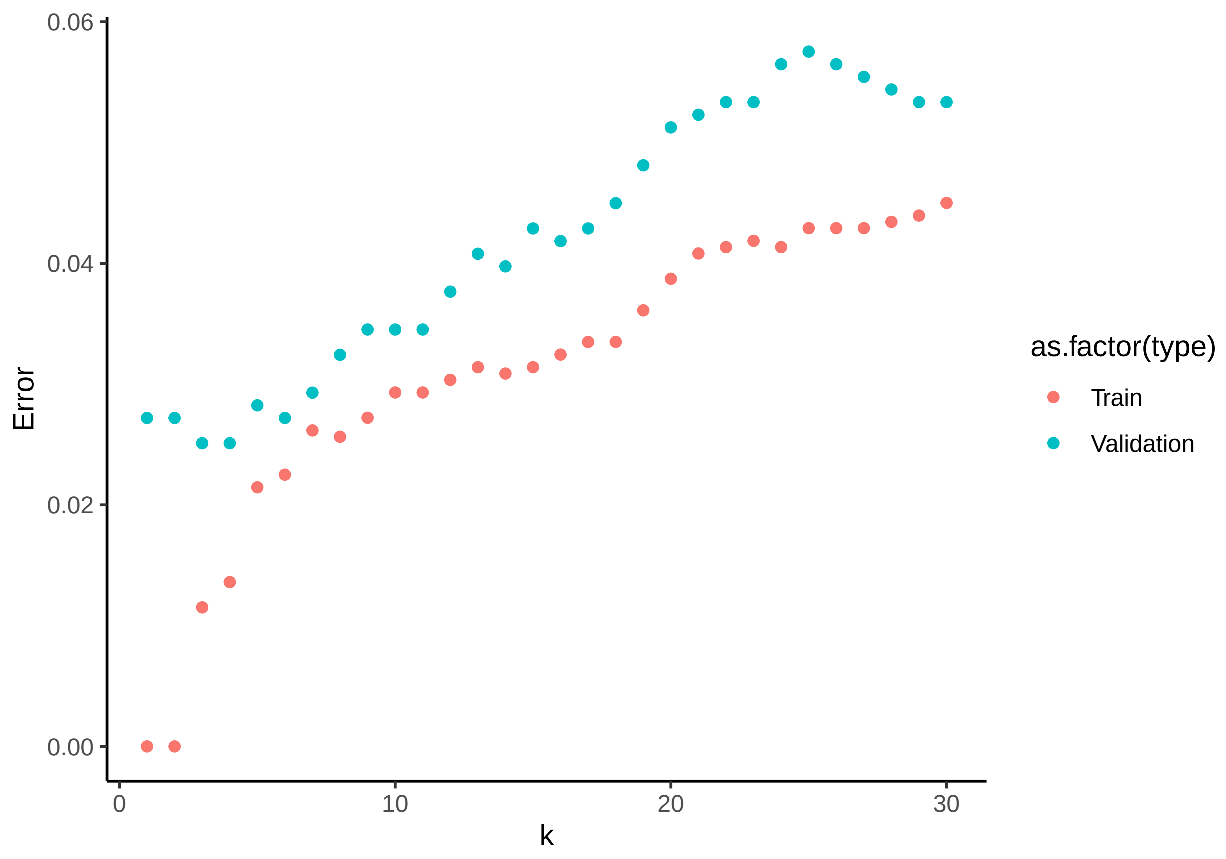

1
2
3
4
5
6
7
8


1
2
3
4
5
6
7
8

It is very hard to visually recognize these digits as 8.

4. Fit a K-nearest neighbor classifiers to the training data for different values of K = 1,2, … , 30 and plot the dependence of the training and validation misclassification errors on the value of K (in the same plot). How does the model complexity change when K increases and how does it affect the training and validation errors? Report the optimal k according to this plot. Finally, estimate the test error for the model having the optimal K, compare it with the training and validation errors and make necessary conclusions about the model quality.

```r
Misclassification_error_train <- c()
Misclassification_error_valid <- c()
for(k in 1:30)
{
  knn_train <- kknn(V65 ~. ,train = train,test = train,k = k,kernel = "rectangular")
  ftd_values<-fitted(knn_train)
  Misclassification_error_train[k] <- 1 - sum(diag(table(train$V65,ftd_values)))/length(ftd_values)

  knn_valid <- kknn(V65 ~. ,train = train,test = validation,k = k,kernel = "rectangular")
  ftd_values_valid<-fitted(knn_valid)
  Misclassification_error_valid[k] <- 1 - sum(diag(table(validation$V65,ftd_values_valid)))/length(ftd_v

}
valid_data <- data.frame(k = 1:30, Error =Misclassification_error_valid, type = "Validation")
training_data <- data.frame(k = 1:30, Error =Misclassification_error_train, type = "Train")
library(ggplot2)
merge_error_df <- rbind(valid_data, training_data)
ggplot(data = merge_error_df) + geom_point(aes(x = k, y = Error, color = as.factor(type))) + theme_clas
```



The misclassification error in training data is lower than in the validation data. When k value is 3 the misclassification error in validation data is lowest but at that value of k the misclassification error in training data is 3rd lowest.

```
knn_test_optimal <- kknn(V65 ~. ,train = train,test = test,k = 3,kernel = "rectangular")
ftd_values_opt<-fitted(knn_test_optimal)
Misclassification_error_testOpt <- 1 - sum(diag(table(test$V65,ftd_values_opt)))/length(ftd_values_opt)
cat("Misclassification error rate for test data at optimal value of k is =",Misclassification_error_tes
```

```
## Misclassification error rate for test data at optimal value of k is = 0.02405858
```

```
knn_valid_optimal <- kknn(V65 ~. ,train = train,test = validation,k = 3,kernel = "rectangular")
ftd_values_opt_v<-fitted(knn_valid_optimal)
Misclassification_error_validOpt <- 1 - sum(diag(table(validation$V65,ftd_values_opt_v)))/length(ftd_val
cat("Misclassification error rate for Validation data at optimal value of k is =",Misclassification_err
```

```
## Misclassification error rate for Validation data at optimal value of k is = 0.0251046
```

As the optimal value of k is 3 so at this value, the misclassification error of training data is 0.0115123 and the misclassification error of validation data is 0.0251046 and the misclassification error of test data is 0.02405858.

Misclassification of training data< Misclassification of test data < Misclassification of validation data

5. Fit K-nearest neighbor classifiers to the training data for different values of K =1,2, … , 30, compute the error for the validation data as cross-entropy (when computing log of probabilities add a small constant within log, e.g. 1e-15, to avoid numerical problems) and plot the dependence of the validation error on the value of K. What is the optimal K value here? Assuming that response has multinomial distribution, why might the cross-entropy be a more suitable choice of the error function than the misclassification error for this problem?

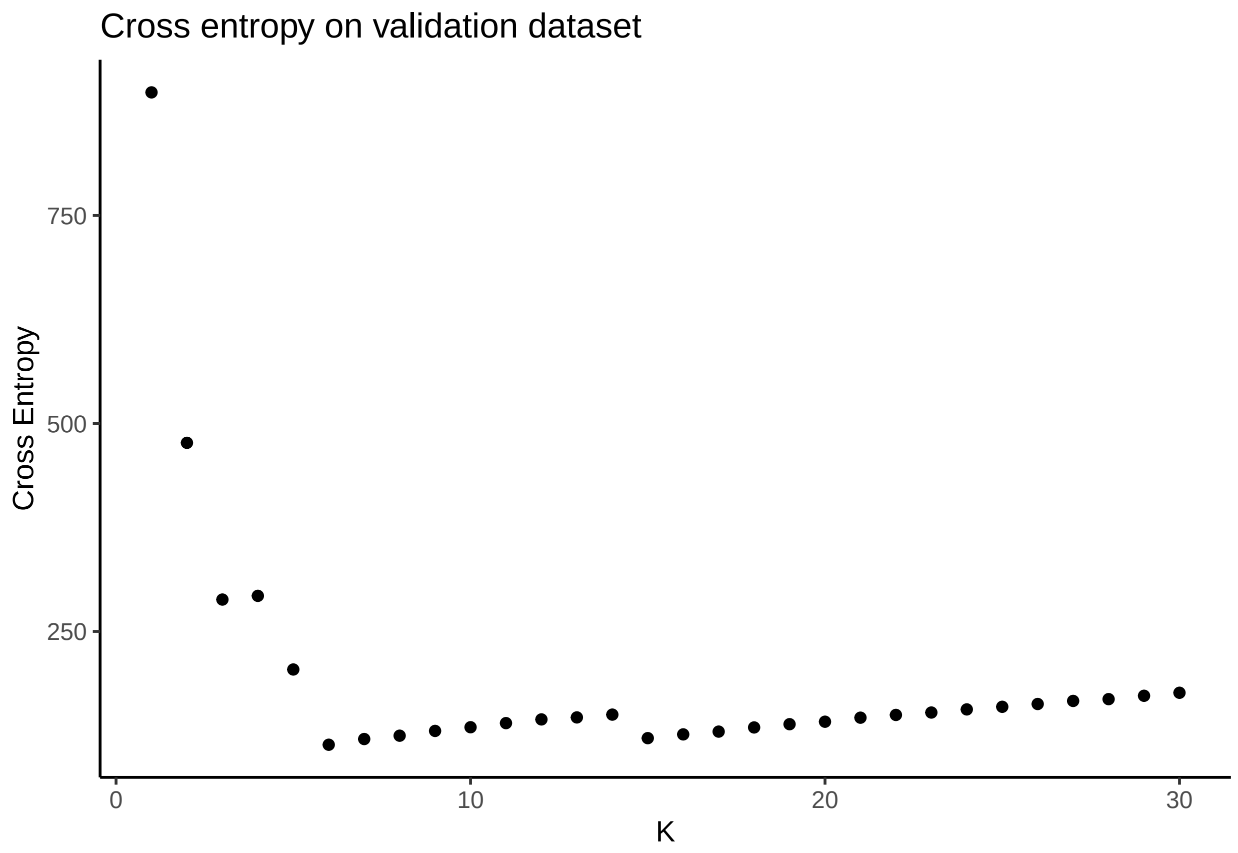
```r
I_matrix <- matrix(0, ncol = 10, nrow = nrow(validation))
True_labels <- as.numeric(as.character(validation$V65))
for(index in 1:length(True_labels)){
  I_matrix[index, True_labels[index]+1] <- 1 #insert 1 where Yi==Cm
}

cross_entropy<-c()
log_of_prob <- c()
for(kn in 1:30){
  knn_valid_entropy <- kkknn(V65 ~. ,
                     train = train,
                     test = validation,
                     k = kn,
                     kernel = "rectangular")
  probability_matrix <- knn_valid_entropy$prob
    for(i in 1:nrow(probability_matrix)){
    log_of_prob[i] <- sum(I_matrix[i,]*log(probability_matrix[i,] + 1e-15))

  }
  cross_entropy[kn] <- sum(-log_of_prob)
}

  cat("Minimum cross entropy loss is =",min(cross_entropy))
```

```
## Minimum cross entropy loss is = 113.768
```

```r
  cat("Optimal value of k is =",which(cross_entropy==min(cross_entropy)))
```

```
## Optimal value of k is = 6
```

```r
ggplot2::ggplot() + geom_point(aes(x = 1:30, y = cross_entropy)) + ggplot2::theme_classic() +
labs(title = "Cross entropy on validation dataset") + xlab("K") + ylab("Cross Entropy")
```



According to the cross entropy plot the optimal value of k is 6 because at this value of k the cross entropy loss is minimum and it is 113.768 .

Assignment 2

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.

```
mydata = read.csv("parkinsons.csv")
set.seed (12345)
n=dim(mydata)[1]
id=sample(1:n, floor(n*0.6))
train = mydata [id,]
test = mydata [-id,]
scaler=preProcess(train)
TrainScaled=predict(scaler,train)
TestScaled=predict(scaler,test)
```

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.

```
ModelTrained = lm (formula =
motor_UPDRS ~ Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 + Jitter.DDP +
Shimmer+ Shimmer.dB.+ Shimmer.APQ3 + Shimmer.APQ5 + Shimmer.APQ11 +
Shimmer.DDA +
NHR + HNR+
RPDE+
DFA+
PPE
, data = TrainScaled)
summary(ModelTrained)
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 +
##     Jitter.DDP + Shimmer + Shimmer.dB. + Shimmer.APQ3 + Shimmer.APQ5 +
##     Shimmer.APQ11 + Shimmer.DDA + NHR + HNR + RPDE + DFA + PPE,
##     data = TrainScaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9131 -0.7310 -0.1163  0.7355  2.1948
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.340e-15  1.583e-02   0.000 1.000000
## Jitter.Abs.  -1.537e-01  3.878e-02  -3.964 7.53e-05 ***
## Jitter.RAP   -5.395e+00  1.884e+01  -0.286 0.774600
## Jitter.PPQ5  -1.763e-02  7.503e-02  -0.235 0.814277
## Jitter.DDP    5.483e+00  1.884e+01   0.291 0.771064
## Shimmer       5.941e-01  2.060e-01   2.884 0.003955 **
## Shimmer.dB.  -1.517e-01  1.383e-01  -1.096 0.273011
## Shimmer.APQ3  3.083e+01  7.717e+01   0.400 0.689518
## Shimmer.APQ5 -3.936e-01  1.137e-01  -3.462 0.000543 ***
## Shimmer.APQ11 3.101e-01  6.114e-02   5.071 4.15e-07 ***
## Shimmer.DDA  -3.117e+01  7.717e+01  -0.404 0.686306
## NHR          -1.811e-01  4.545e-02  -3.985 6.88e-05 ***
## HNR          -2.339e-01  3.621e-02  -6.459 1.20e-10 ***
## RPDE          6.219e-03  2.260e-02   0.275 0.783236
## DFA          -2.804e-01  2.014e-02 -13.923  < 2e-16 ***
## PPE           2.367e-01  3.186e-02   7.428 1.38e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9396 on 3509 degrees of freedom
## Multiple R-squared:  0.1208, Adjusted R-squared:  0.1171
## F-statistic: 32.15 on 15 and 3509 DF,  p-value: < 2.2e-16
```

```
TrainMSE = mean ((ModelTrained$residuals)^2)
TestPrediction <- predict(ModelTrained, TestScaled)
TestResidual = TestPrediction - TestScaled$motor_UPDRS
TestMSE <- sum(TestResidual^2)/nrow(TestScaled)
```

```
TrainMSE
```

```
## [1] 0.8789342
```

```
TestMSE
```

```
## [1] 0.9367006
```

Test MSE is 0.9367006 and Train MSE is 0.8789342. Here 8 variables have major significance and are contributing to the model. These include:

1. Jitter Abs.
2. Shimmer.APQ5
3. Shimmer.APQ11
4. NHR
5. HNR
6. DFA
7. PPE
8. Shimmer

Least significant variable is RPDE with high P-value.

3. Implement 4 following functions by using basic R commands only (no external packages):

a. Likelihood function that for a given parameter vector θ and dispersion σ computes the log-likelihood function $logP(T|\theta, \sigma)$ for the stated model and the training data

```
loglikelihood <- function(theta, sigma)
{
X <- as.matrix(cbind(TrainScaled[,-c(1,2,3,4,5,6)])) # we don't need first 6 features
n <- nrow(X)
y <- TrainScaled[,6, drop = FALSE]
theta_1 <- theta

#theta_1 has 16 variables

first_term = -(n/2) * log(2 * pi * sigma^2)
second_term = sum((y - X%*%theta_1)^2) * (1/(2*sigma^2))
res <- first_term - second_term
return (res)
}
```

b. Ridge function that for given vector θ scalar σ and scalar λ uses function from 3a and adds up a Ridge penalty $\lambda||\theta||$ to the minus log-likelihood

```
Ridge <- function(theta, lambda){
sigma = theta[17]
theta1 = theta[1:16]
Neg_log_like <- -(loglikelihood(theta1, sigma))
res <- Neg_log_like + lambda * sum(theta^2)
}
```

c. RidgeOpt function that depends on scalar λ, uses function from 3b and function optim() with method="BFGS" to find the optimal θ and σ for the given λ.

```
RidgeOpt = function (lambda) #returns theta corresponding to lambda
{
# 22 - 6 = 16. So, 16 entries for theta and 1 for sigma
#pass both in the same vec so that both can be estimated
#lambda is the data provided by the user

temp = rep(1,17)
min_params = optim(temp ,fn = Ridge, method="BFGS", lambda = lambda)
return (min_params)
}
#par gives you the best guess for vec theta (first 16 indexes) and sigma (last index)
```

d. DF function that for a given scalar λ computes the degrees of freedom of the Ridge model based on the training data.

```
DF <- function(lambda){
matrix_X <- as.matrix(cbind(TrainScaled[,-c(1,2,3,4,5,6)]))
n <- nrow(matrix_X)
Hat_matrix <- matrix_X %*% solve((t(matrix_X)%*%matrix_X + lambda* diag(ncol(matrix_X))))%*% t(matrix_X
DegreeOfFreedom <- sum(diag(Hat_matrix))
return (DegreeOfFreedom)
}
```

4. By using function RidgeOpt, compute optimal θ parameters for $\lambda = 1$, $\lambda = 100$ and $\lambda = 1000$. Use the estimated parameters to predict the motor_UPDRS values for training and test data and report the training and test MSE values. Which penalty parameter is most appropriate among the selected ones? Compute and compare the degrees of freedom of these models and make appropriate conclusions.

```
train_x <- as.matrix(cbind(TrainScaled[,-c(1,2,3,4,5,6)]))

# y matrix is the value to be predicted which is at Col

test_x <- as.matrix(cbind(TestScaled[,-c(1,2,3,4,5,6)]))
test_y <- as.matrix(TestScaled[,6, drop = FALSE])
Lambda1 <- RidgeOpt(lambda = 1)
theta <- Lambda1$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
error_train_1 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
error_test_1 <- mean((fit_test - TestScaled$motor_UPDRS)^2)
Lambda1 <- RidgeOpt(lambda = 1)
theta <- Lambda1$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
TrainErrorForLambda1 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
TestErrorForLambda1 <- mean((fit_test - TestScaled$motor_UPDRS)^2)
```

```r
Lambda100 <- RidgeOpt(lambda = 100)
theta <- Lambda100$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
TrainErrorForLambda100 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
TestErrorForLambda100 <- mean((fit_test - TestScaled$motor_UPDRS)^2)
```

```r
Lambda1000 <- RidgeOpt(lambda = 1000)
theta <- Lambda1000$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
TrainErrorForLambda1000 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
TestErrorForLambda1000 <- mean((fit_test - TestScaled$motor_UPDRS)^2)
```

For Lambda = 1,

Train Error: 0.8836586 Test Error: 0.9412531 Degree of Freedom: 13.86074

For Lambda = 100

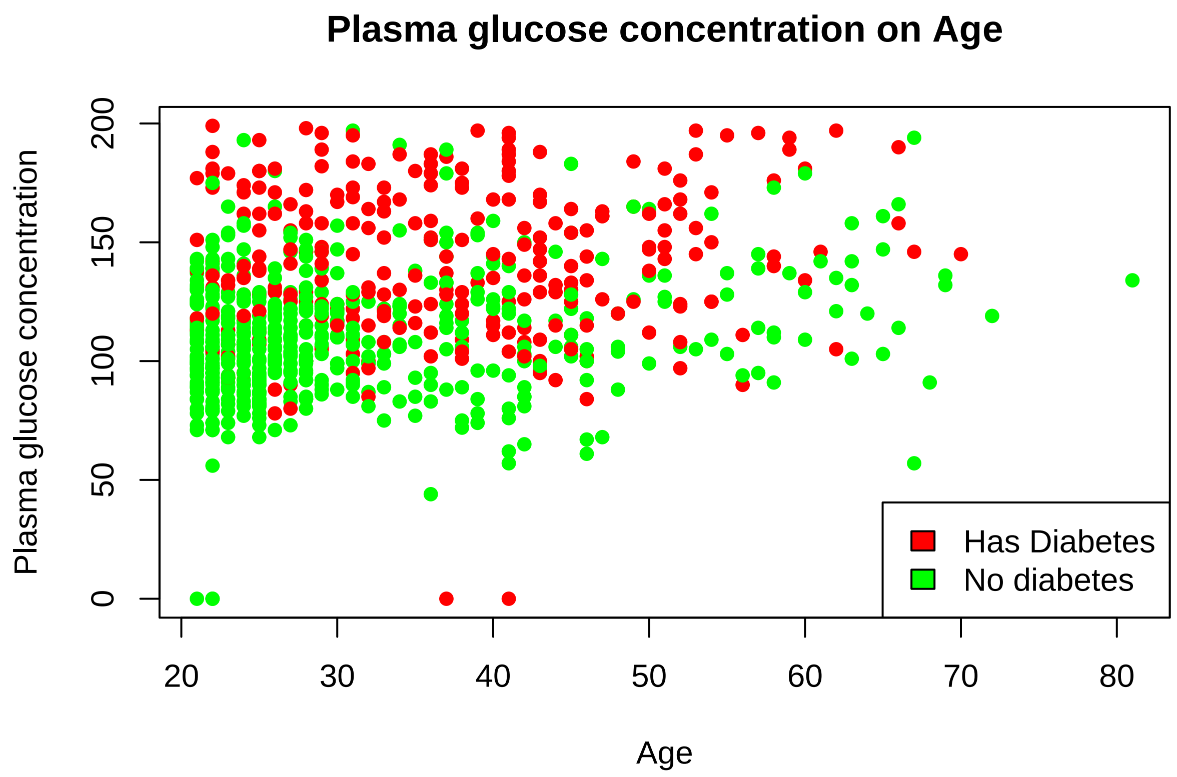
Train Error: 0.8896607 Test Error: 0.9389338 Degree of Freedom: 9.924887

For Lambda = 1000

Train Error: 0.9159136 Test Error: 0.951983 Degree of Freedom: 5.643925

Assignment 3

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? Motivate your answer.



=> If we observe the graph, we can see that some young people with less plasma glucose concentration has diabetes and also some old people with high plasma glucose concentration rate does not have diabetes. It will not be easy to classify by a standard logistic regression model that uses these two variables as features. If we add more features to classify diabetes, it may show more clear boundary to predict diabetes.

2. Train a logistic regression model with y=Diabetes as target x_1 =Plasma glucose concentration and x_2=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

```
logistic_model <- glm(V9 ~ V2+V8, data = xddata,  family = "binomial")
logistic_model
```

```
##
## Call:  glm(formula = V9 ~ V2 + V8, family = "binomial", data = xddata)
##
## Coefficients:
## (Intercept)           V2           V8
##    -5.91245      0.03564      0.02478
##
## Degrees of Freedom: 767 Total (i.e. Null);  765 Residual
## Null Deviance:       993.5
## Residual Deviance: 797.4     AIC: 803.4
```

```r
predict_reg <- predict(logistic_model, xddata, type = "response")

# Changing probabilities
predict_reg1 <- ifelse(predict_reg >0.5, 1, 0)
```

=> We know that,

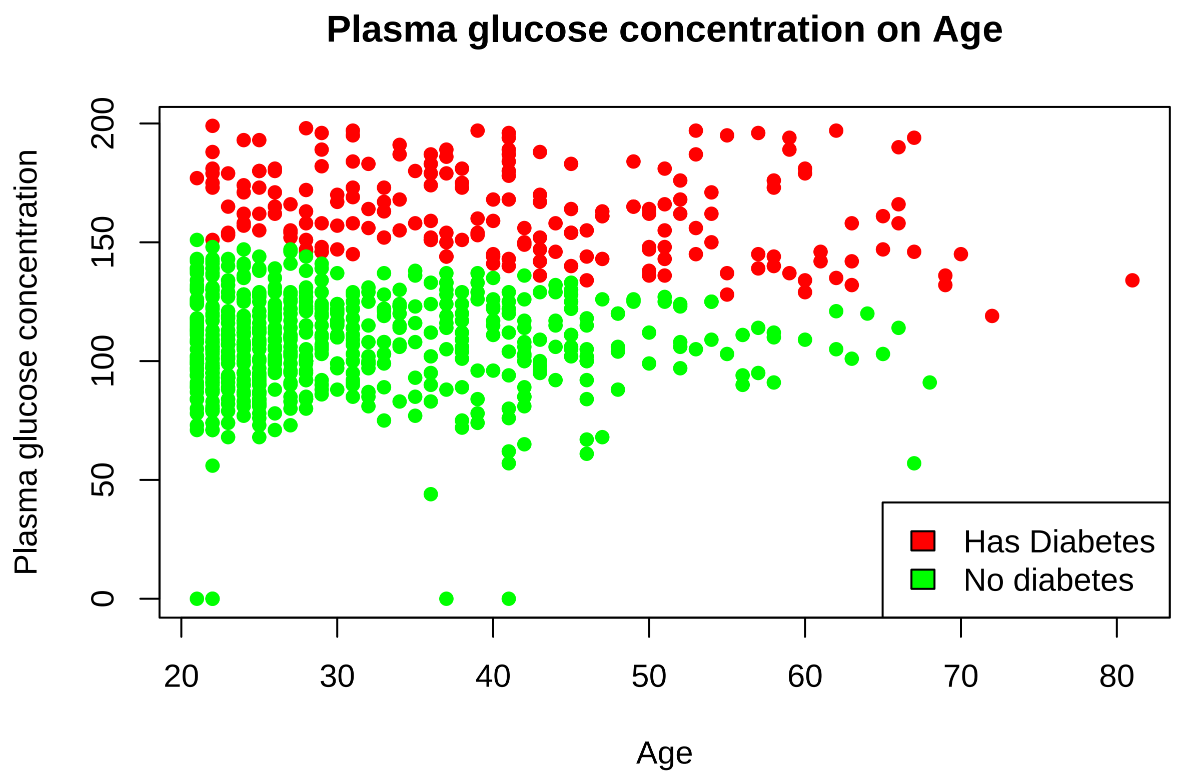
$$g(x_*, \hat{\theta}) = \frac{1}{1 + e^{-\hat{\theta}^T x_*}}$$

for our model the probability equation will be,

$$\frac{1}{1 + exp^{-(-5.91245 + 0.03564\ *\ plasma\ glucose\ concentration + 0.02478\ *\ age)}}$$

```r
# calculating miss classification error from confusion matrix


miss_error=function(x){
  n=sum(cm)
  y = 1- sum(diag(x))/n
  return(y)
}

cm= table(xddata$V9, predict_reg1) #confusion mat

miss_error(cm)
```

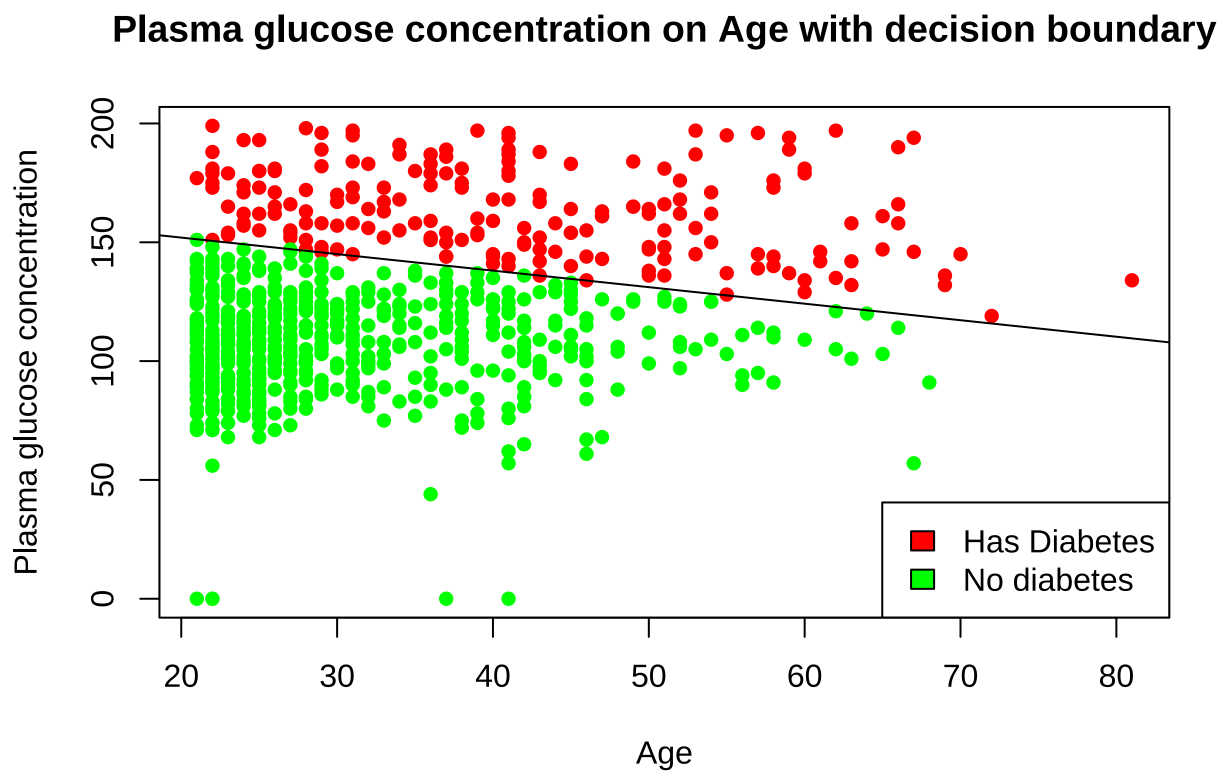
```
## [1] 0.2630208
```



Here, the training misclassification error is 0.2630208 which portrays that in about 73.69792% of cases the model correctly predicts if a person has diabetes or not.

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

```
slope = -coef(logistic_model)[3]/(coef(logistic_model)[2])

intercept <- -coef(logistic_model)[1]/(coef(logistic_model)[2])
```



=>

From confusion matrix, we can see that

```
##    predict_reg1
##       0   1
##   0 436  64
##   1 138 130
```

the model struggles to correctly predict who actually has diabetes, but it is more successful in predicting who does not have diabetes.

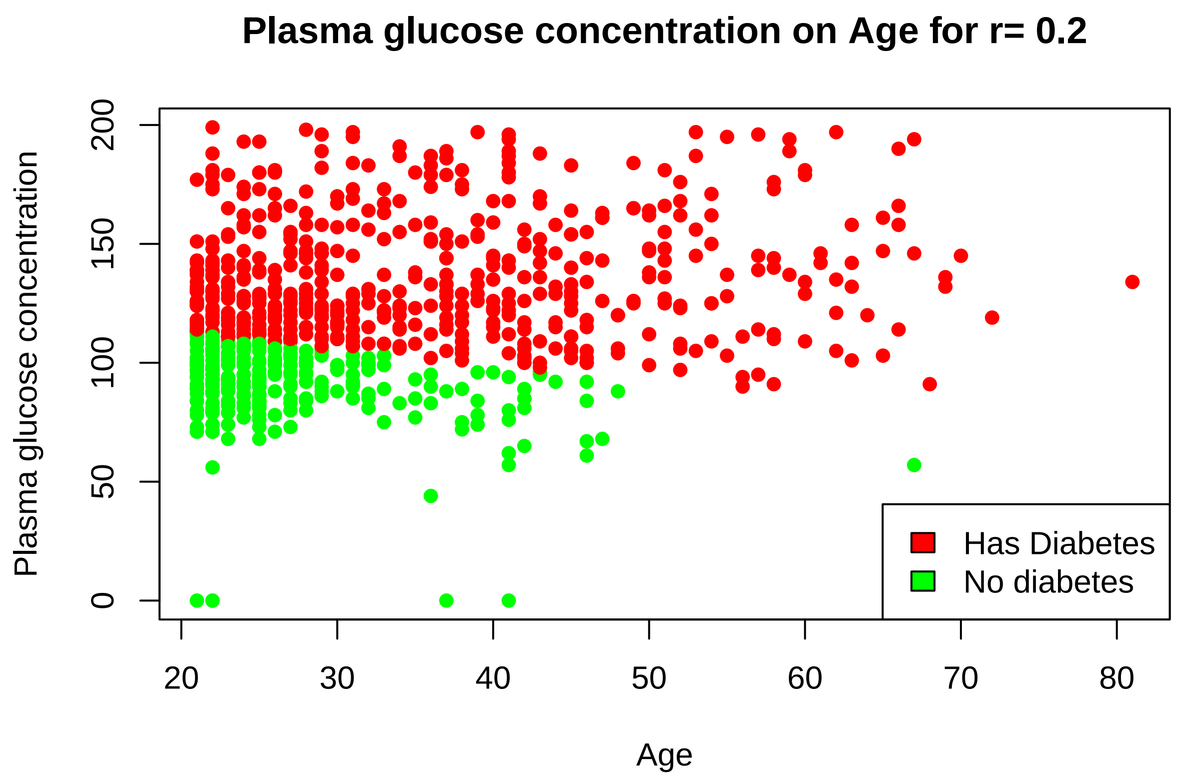
4. Make same kind of plots as in step 2 but use thresholds r= 0.2 and r = 0.8. By using these plots, comment on what happens with the prediction when r value changes

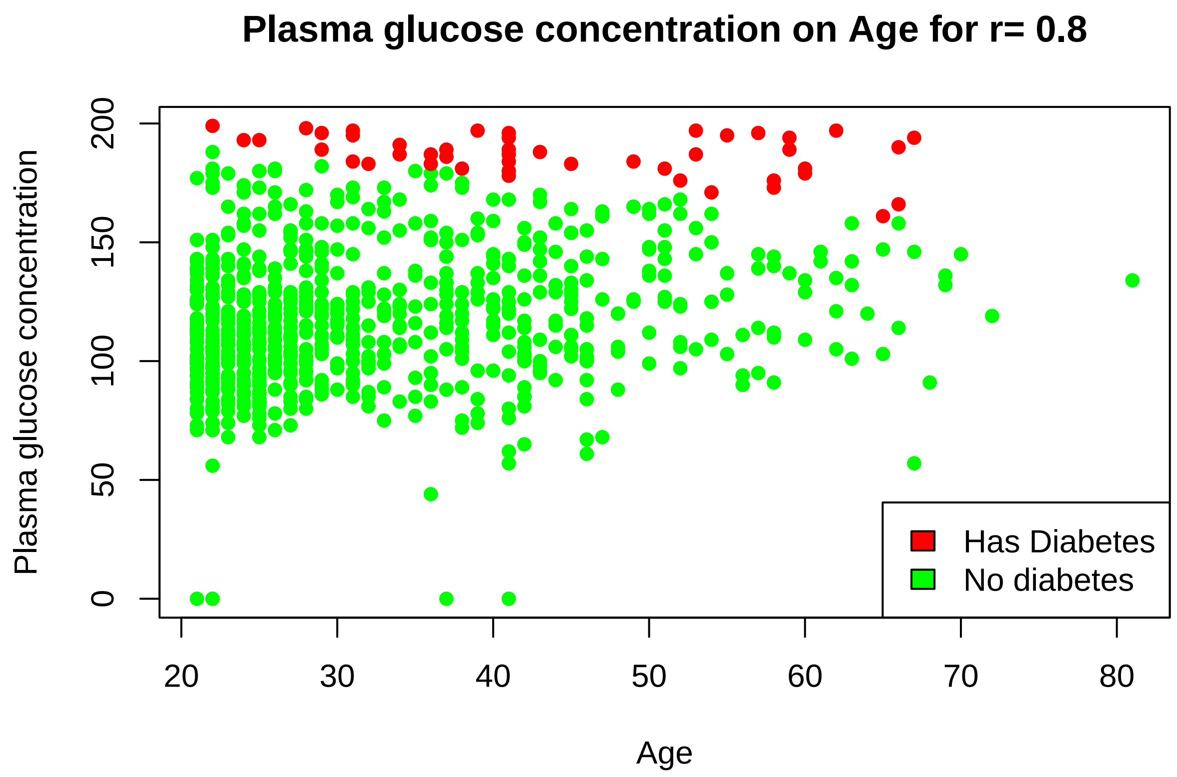
```
predict_reg2 <- ifelse(predict_reg >0.2, 1, 0)
cm2= table(xddata$V9, predict_reg2)

miss_error(cm2)
```

```
## [1] 0.3723958
```

```
predict_reg3 <- ifelse(predict_reg >0.8, 1, 0)
cm3= table(xddata$V9, predict_reg3)

miss_error(cm3)
```

```
## [1] 0.3151042
```


Plasma glucose concentration on Age for r= 0.2
Plasma glucose concentration
0
50
100
150
200
20
30
40
50
60
70
80
Age
🟥 Has Diabetes
🟩 No diabetes

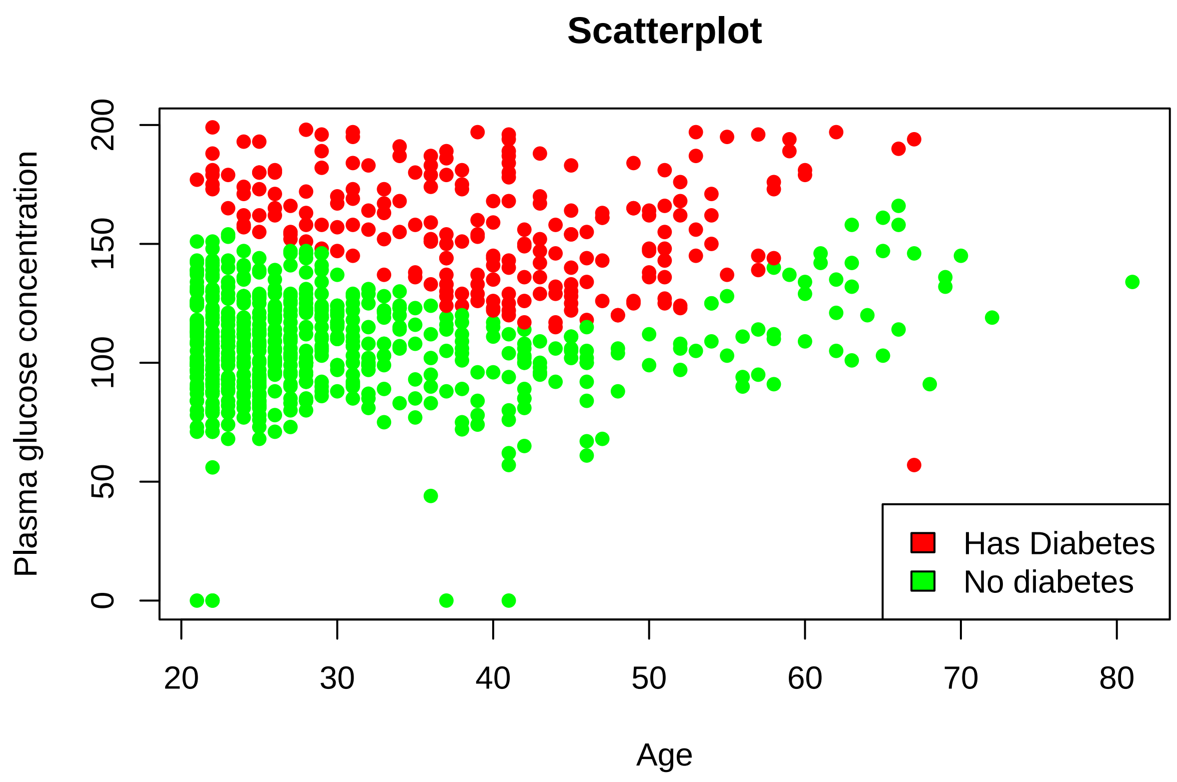


=> When, we set the threshold r= 0.2, we can observe that missclassifcation rate increases to 0.3723958, but it can predict who has diabetes more accurately. On the other hand, when we set the threshold r= 0.8, missclassification rate becomes 0.3151042. But, in this case it only predicts 36 cases of diabetes accurately while predicting 232 cases of having diabetes wrong.

5. Perform a basis function expansion trick by computing new features $z_1 = x_1^4$, $z_2 = x_1^3 x_2$, $z_3 = x_1^2 x_2^2$, $z_2 = x_1 x_2^3$, $z_2 = x_1 x_2^4$, adding them to the data set and then computing a logistic regression model with y as target and x1, x2, z1, z2, z3, z4, z5 as 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

```
xddata$z1 = xddata$V2^4
xddata$z2 = (xddata$V2^3)*(xddata$V8)
xddata$z3 = (xddata$V2^2)*(xddata$V8^2)
xddata$z4 = (xddata$V2)*(xddata$V8^3)
xddata$z5 = xddata$V8^4




logistic_model_bfe <- glm(V9 ~ V2+V8+z1+z2+z3+z4+z5, data = xddata,  family = "binomial")

predict_reg_bfe <- predict(logistic_model_bfe, xddata, type = "response")
predict_reg_bfe1 <- ifelse(predict_reg_bfe >0.5, 1, 0)
cm_bfe=table(xddata$V9, predict_reg_bfe1)
miss_error(cm_bfe)
```

```
## [1] 0.2447917
```



=> After adding basis function expansion trick into the logistic regression model we can see that the missclassfication rate has decreased to 0.2447917. This model is predicting both diabetes and non diabetes classes more accurately than the previous model. Here, the decision boundary in non-linear as we are adding features in the model after basis function expansion trick.

Appendix

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

my_data <- read.csv("optdigits.csv", header = FALSE)
#converting target field into factor
my_data$V65 <- as.factor(my_data$V65)
n<-dim(my_data)[1]
set.seed(12345)
#spliting data
id<-sample(1:n, floor(n*0.5))
train<-my_data[id,]
rem_data<-my_data[-id,]

new_n<-dim(rem_data)[1]
```

```r
set.seed(12345)
new_id<-sample(1:new_n, floor(new_n*0.5))
validation<-rem_data[new_id,]
test<-rem_data[-new_id,]
knn_train_classifier <- kknn(V65 ~. ,train = train,test = train,k = 30,kernel = "rectangular")
knn_test_classifier <- kknn(V65 ~. ,train = train,test = test,k = 30,kernel = "rectangular")
fitted_data <- fitted(knn_train_classifier)
predict <- fitted(knn_test_classifier)
confusion_matrix<-table(actual=train$V65, predicted=fitted_data)

confusion_matrix
total_correct<-sum(diag(confusion_matrix))
total<-length(fitted_data)
Misclassification_errors<-1-(total_correct/total)
cat("Misclassification error is =",Misclassification_errors)
confusion_matrix_test<-table(actual=test$V65, predicted=predict)
confusion_matrix_test
total_correct_test<-sum(diag(confusion_matrix_test))
total_test<-length(predict)
Misclassification_errors_test<-1-(total_correct_test/total_test)
cat("Misclassification error is =",Misclassification_errors_test)
indexes_of_eight<-which(train$V65==8)
probability_of_classifying<-knn_train_classifier$prob[indexes_of_eight,9]
easiest<-order(probability_of_classifying,decreasing = TRUE)[1:2]
cat("The two indexes that are easy to identify as 8 are ",easiest)
hardest<-order(probability_of_classifying,decreasing = FALSE)[1:3]
cat("The three indexes that are hard to identify as 8 are ",hardest)

for(grp in 1:length(easiest)){
dt<-matrix(data = as.numeric(train[indexes_of_eight[easiest[grp]],1:64])
,nrow = 8,ncol = 8)
heatmap(dt,Colv = NA, Rowv = NA)
}
for(grp in 1:length(hardest)){
dt<-matrix(data = as.numeric(train[indexes_of_eight[hardest[grp]],1:64])
,nrow = 8,ncol = 8)
heatmap(dt,Colv = NA, Rowv = NA)}
Misclassification_error_train <- c()
Misclassification_error_valid <- c()
for(k in 1:30)
{
  knn_train <- kknn(V65 ~. ,train = train,test = train,k = k,kernel = "rectangular")
  ftd_values<-fitted(knn_train)
  Misclassification_error_train[k] <- 1 - sum(diag(table(train$V65,ftd_values)))/length(ftd_values)

  knn_valid <- kknn(V65 ~. ,train = train,test = validation,k = k,kernel = "rectangular")
  ftd_values_valid<-fitted(knn_valid)
  Misclassification_error_valid[k] <- 1 - sum(diag(table(validation$V65,ftd_values_valid)))/length(ftd_

}
valid_data <- data.frame(k = 1:30, Error =Misclassification_error_valid, type = "Validation")
training_data <- data.frame(k = 1:30, Error =Misclassification_error_train, type = "Train")
library(ggplot2)
```

```r
merge_error_df <- rbind(valid_data, training_data)
ggplot(data = merge_error_df) + geom_point(aes(x = k, y = Error, color = as.factor(type))) + theme_clas

knn_test_optimal <- kknn(V65 ~. ,train = train,test = test,k = 3,kernel = "rectangular")
ftd_values_opt<-fitted(knn_test_optimal)
Misclassification_error_testOpt <- 1 - sum(diag(table(test$V65,ftd_values_opt)))/length(ftd_values_opt)
cat("Misclassification error rate for test data at optimal value of k is =",Misclassification_error_tes
knn_valid_optimal <- kknn(V65 ~. ,train = train,test = validation,k = 3,kernel = "rectangular")
ftd_values_opt_v<-fitted(knn_valid_optimal)
Misclassification_error_validOpt <- 1 - sum(diag(table(validation$V65,ftd_values_opt_v)))/length(ftd_va
cat("Misclassification error rate for Validation data at optimal value of k is =",Misclassification_err

I_matrix <- matrix(0, ncol = 10, nrow = nrow(validation))
True_labels <- as.numeric(as.character(validation$V65))
for(index in 1:length(True_labels)){
  I_matrix[index, True_labels[index]+1] <- 1 #insert 1 where Yi==Cm
}

cross_entropy<-c()
log_of_prob <- c()
for(kn in 1:30){
  knn_valid_entropy <- kknn(V65 ~. ,
                    train = train,
                    test = validation,
                    k = kn,
                    kernel = "rectangular")
  probability_matrix <- knn_valid_entropy$prob
    for(i in 1:nrow(probability_matrix)){
    log_of_prob[i] <- sum(I_matrix[i,]*log(probability_matrix[i,] + 1e-15))

  }
  cross_entropy[kn] <- sum(-log_of_prob)
}

  cat("Minimum cross entropy loss is =",min(cross_entropy))
  cat("Optimal value of k is =",which(cross_entropy==min(cross_entropy)))
ggplot2::ggplot() + geom_point(aes(x = 1:30, y = cross_entropy)) + ggplot2::theme_classic() +
labs(title = "Cross entropy on validation dataset") + xlab("K") + ylab("Cross Entropy")
library(caret)
library(lattice)

mydata = read.csv("parkinsons.csv")
set.seed (12345)
n=dim(mydata)[1]
id=sample(1:n, floor(n*0.6))
train = mydata [id,]
test = mydata [-id,]
scaler=preProcess(train)
TrainScaled=predict(scaler,train)
TestScaled=predict(scaler,test)


ModelTrained = lm (formula =
motor_UPDRS ~ Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 + Jitter.DDP +
Shimmer+ Shimmer.dB.+ Shimmer.APQ3 + Shimmer.APQ5 + Shimmer.APQ11 +
Shimmer.DDA +
NHR + HNR+
RPDE+
DFA+
PPE
, data = TrainScaled)
summary(ModelTrained)
TrainMSE = mean ((ModelTrained$residuals)^2)
TestPrediction <- predict(ModelTrained, TestScaled)
TestResidual = TestPrediction - TestScaled$motor_UPDRS
TestMSE <- sum(TestResidual^2)/nrow(TestScaled)



TrainMSE

TestMSE


loglikelihood <- function(theta, sigma)
{
X <- as.matrix(cbind(TrainScaled[,-c(1,2,3,4,5,6)])) # we don't need first 6 features
n <- nrow(X)
y <- TrainScaled[,6, drop = FALSE]
theta_1 <- theta

#theta_1 has 16 variables

first_term = -(n/2) * log(2 * pi * sigma^2)
second_term = sum((y - X%*%theta_1)^2) * (1/(2*sigma^2))
res <- first_term - second_term
return (res)
}


Ridge <- function(theta, lambda){
sigma = theta[17]
theta1 = theta[1:16]
Neg_log_like <- -(loglikelihood(theta1, sigma))
res <- Neg_log_like + lambda * sum(theta^2)
}

RidgeOpt = function (lambda) #returns theta corresponding to lambda
{
# 22 - 6 = 16. So, 16 entries for theta and 1 for sigma
#pass both in the same vec so that both can be estimated
#lambda is the data provided by the user

temp = rep(1,17)
min_params = optim(temp ,fn = Ridge, method="BFGS", lambda = lambda)
return (min_params)
```

```
}
#par gives you the best guess for vec theta (first 16 indexes) and sigma (last index)

DF <- function(lambda){
matrix_X <- as.matrix(cbind(TrainScaled[,-c(1,2,3,4,5,6)]))
n <- nrow(matrix_X)
Hat_matrix <- matrix_X %*% solve((t(matrix_X)%*%matrix_X + lambda* diag(ncol(matrix_X))))%*% t(matrix_X
DegreeOfFreedom <- sum(diag(Hat_matrix))
return (DegreeOfFreedom)
}


train_x <- as.matrix(cbind(TrainScaled[,-c(1,2,3,4,5,6)]))

# y matrix is the value to be predicted which is at Col

test_x <- as.matrix(cbind(TestScaled[,-c(1,2,3,4,5,6)]))
test_y <- as.matrix(TestScaled[,6, drop = FALSE])
Lambda1 <- RidgeOpt(lambda = 1)
theta <- Lambda1$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
error_train_1 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
error_test_1 <- mean((fit_test - TestScaled$motor_UPDRS)^2)
Lambda1 <- RidgeOpt(lambda = 1)
theta <- Lambda1$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
TrainErrorForLambda1 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
TestErrorForLambda1 <- mean((fit_test - TestScaled$motor_UPDRS)^2)
Lambda100 <- RidgeOpt(lambda = 100)
theta <- Lambda100$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
TrainErrorForLambda100 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
TestErrorForLambda100 <- mean((fit_test - TestScaled$motor_UPDRS)^2)

Lambda1000 <- RidgeOpt(lambda = 1000)
theta <- Lambda1000$par[1:16]
TrainFit <- train_x %*% theta
fit_test <- test_x %*% theta
TrainErrorForLambda1000 <- mean((TrainFit - TrainScaled$motor_UPDRS)^2)
TestErrorForLambda1000 <- mean((fit_test - TestScaled$motor_UPDRS)^2)

print (" +++Results for Lambda = 1000+++++")

TrainErrorForLambda1000
TestErrorForLambda1000
DF (1000)

print (" +++Results for Lambda = 100+++++")

TrainErrorForLambda100
```

```r
TestErrorForLambda100
DF(100)

print (" +++Results for Lambda = 1+++++")

TrainErrorForLambda1
TestErrorForLambda1
DF(1)

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

plot(xddata$V8,xddata$V2, main="Plasma glucose concentration on Age",
     xlab="Age", ylab="Plasma glucose concentration",pch=16,
     col=ifelse(xddata$V9==1,"red","green"))

legend("bottomright", legend=c("Has Diabetes", "No diabetes"),
       fill = c("red","green"))

logistic_model <- glm(V9 ~ V2+V8, data = xddata,  family = "binomial")
logistic_model

predict_reg <- predict(logistic_model, xddata, type = "response")

# Changing probabilities
predict_reg1 <- ifelse(predict_reg >0.5, 1, 0)


# calculating miss classification error from confusion matrix


miss_error=function(x){
  n=sum(cm)
  y = 1- sum(diag(x))/n
  return(y)
}

cm= table(xddata$V9, predict_reg1) #confusion mat

miss_error(cm)




plot(xddata$V8,xddata$V2, main="Plasma glucose concentration on Age",
     xlab="Age", ylab="Plasma glucose concentration",pch=16, col=ifelse(predict_reg1==1,"red","green"))


legend("bottomright", legend=c("Has Diabetes", "No diabetes"),
       fill = c("red","green"))
slope = -coef(logistic_model)[3]/(coef(logistic_model)[2])

intercept <- -coef(logistic_model)[1]/(coef(logistic_model)[2])
```

```r
plot(xddata$V8,xddata$V2, main="Plasma glucose concentration on Age with decision boundary",
     xlab="Age", ylab="Plasma glucose concentration",pch=16, col=ifelse(predict_reg1==1,"red","green"))

legend("bottomright", legend=c("Has Diabetes", "No diabetes"),
       fill = c("red","green"))
abline(a= intercept, b = slope)


cm

predict_reg2 <- ifelse(predict_reg >0.2, 1, 0)
cm2= table(xddata$V9, predict_reg2)

miss_error(cm2)

predict_reg3 <- ifelse(predict_reg >0.8, 1, 0)
cm3= table(xddata$V9, predict_reg3)

miss_error(cm3)

plot(xddata$V8,xddata$V2, main="Plasma glucose concentration on Age for r= 0.2",
     xlab="Age", ylab="Plasma glucose concentration",pch=16, col=ifelse(predict_reg2==1,"red","green"))


legend("bottomright", legend=c("Has Diabetes", "No diabetes"),
       fill = c("red","green"))



plot(xddata$V8,xddata$V2, main="Plasma glucose concentration on Age for r= 0.8",
     xlab="Age", ylab="Plasma glucose concentration",pch=16, col=ifelse(predict_reg3==1,"red","green"))


legend("bottomright", legend=c("Has Diabetes", "No diabetes"),
       fill = c("red","green"))


xddata$z1 = xddata$V2^4
xddata$z2 = (xddata$V2^3)*(xddata$V8)
xddata$z3 = (xddata$V2^2)*(xddata$V8^2)
xddata$z4 = (xddata$V2)*(xddata$V8^3)
xddata$z5 = xddata$V8^4



logistic_model_bfe <- glm(V9 ~ V2+V8+z1+z2+z3+z4+z5, data = xddata,  family = "binomial")

predict_reg_bfe <- predict(logistic_model_bfe, xddata, type = "response")
predict_reg_bfe1 <- ifelse(predict_reg_bfe >0.5, 1, 0)
cm_bfe=table(xddata$V9, predict_reg_bfe1)
miss_error(cm_bfe)
```

```
plot(xddata$V8,xddata$V2, main="Scatterplot",
     xlab="Age", ylab="Plasma glucose concentration",pch=16, col=ifelse(predict_reg_bfe1==1,"red","gree


legend("bottomright", legend=c("Has Diabetes", "No diabetes"),
       fill = c("red","green"))
```