TDDE01. Lab1. Group B24 report.

Statement of contribution

Firstly, general analysis was performed teamwise. Approaches and strategies of solving the tasks were elaborated teamwise as well.

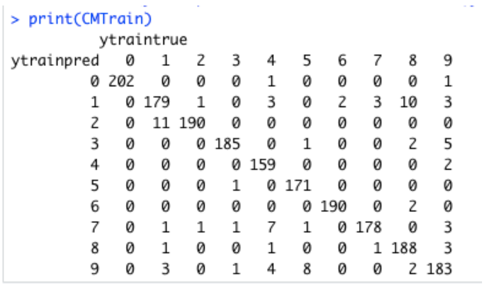
Student Elham was responsible for code writing and problem solving for the Assignment 1. Student Anton was responsible for code writing and problem solving for the Assignment 2. Student Elena was responsible for code writing and problem solving for the Assignment 3.

After completion of coding stage group analyzed the results together and peer reviewed the results of each other's work. Finally, each student corrected their solutions according to the received reviews from groupmates.

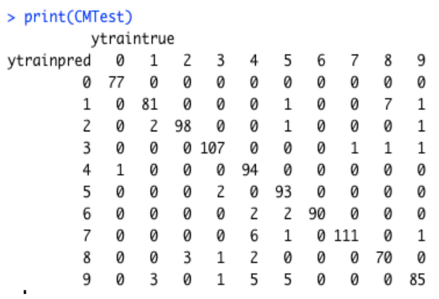
Assignment 1. Handwritten digit recognition with Knearest neighbors.

Task 1: Our raw data in this assignment was "optdigits.csv" which we uploaded it without header then factorized it to 10 levels bases on V65 to show the rows of the confusion matric just from 0 to 9 and then divided it randomly to train, test and data segments.

Task 2: We modeled our train and our test data by KKNN function with rectangular kernel, while number of neighbors was 30, and the column 65 was our true data. As you can see by having these 2 models, we could calculate confusion matrices for both models.



```
> print(CMTrain)
          ytraintrue
ytrainpred   0   1   2   3   4   5   6   7   8   9
         0 202   0   0   0   1   0   0   0   0   1
         1   0 179   1   0   3   0   2   3  10   3
         2   0  11 190   0   0   0   0   0   0   0
         3   0   0   0 185   0   1   0   0   2   5
         4   0   0   0   0 159   0   0   0   0   2
         5   0   0   0   1   0 171   0   0   0   0
         6   0   0   0   0   0   0 190   0   2   0
         7   0   1   1   1   7   1   0 178   0   3
         8   0   1   0   0   1   0   0   1 188   3
         9   0   3   0   1   4   8   0   0   2 183
```



```
> print(CMTest)
          ytraintrue
ytrainpred   0   1   2   3   4   5   6   7   8   9
         0  77   0   0   0   0   0   0   0   0   0
         1   0  81   0   0   0   1   0   0   7   1
         2   0   2  98   0   0   1   0   0   0   1
         3   0   0   0 107   0   0   0   1   1   1
         4   1   0   0   0  94   0   0   0   0   0
         5   0   0   0   2   0  93   0   0   0   0
         6   0   0   0   0   2   2  90   0   0   0
         7   0   0   0   0   6   1   0 111   0   1
         8   0   0   3   1   2   0   0   0  70   0
         9   0   3   0   1   5   5   0   0   0  85
```

We also calculated misclassification errors for both models and as you can see for train data the misclassification error is lessen than misclassification error for test data.

MCE for test data = 0.05329154 and for train data = 0.04500262

For train data the quality of prediction for 0 and 6 was higher than others while the worst prediction was for 1 and 9.

Number of wrong predictions for each digit in CMTrain:

"0"=1 , "1"=38 , "2"=12 , "3"=11 , "4"=14 , "5"=10 , "7"=18 , "8"=30 , "9"=35

For test data the quality of prediction for 0 and 6 was higher than others while the worst prediction was for 9, 1 and 4.

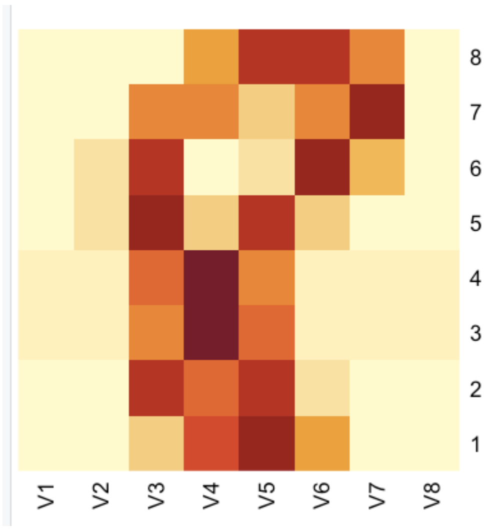
Number of wrong predictions for each digit in CMTest:

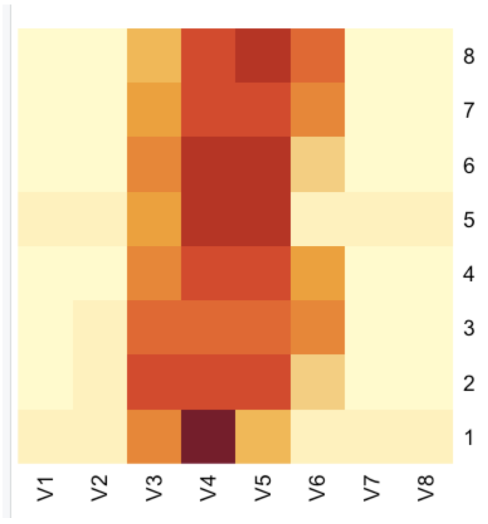
"0"= 1 , "1"= 14 , "2"=7 , "3"=7 , "4"=16 , "5"=12 , "6"=4 , "7"= 9 , "8"=14 , "9"=18

Task 3:

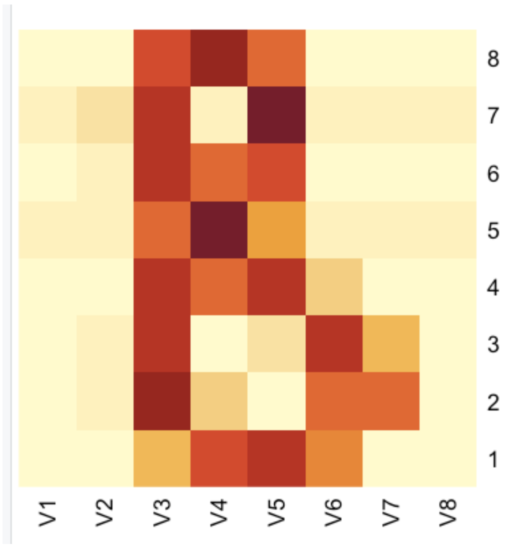
We needed the probability for digit “8” in our train model and the real value of that in our train data set. So, we made a data frame by these 2 data and named it “df”. As an example, we picked up rows 520, 431, 1294 as the hardest to predict and rows 129, 195(with the probability of 1 as the easiest to predict. In our next step we transferred these vectors to 8*8 matrices and plot them by heat maps. As we predicted and you can see in the pictures, by comparing the heat maps the cases with lower probabilities are difficult to recognize visually and those with high probabilities are pretty easy to recognize.

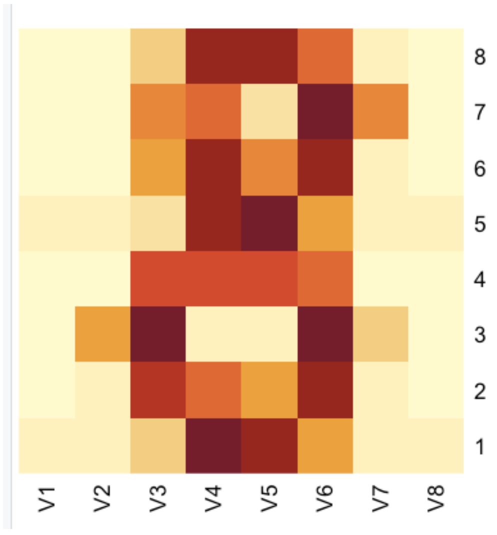
Difficult to recognize:





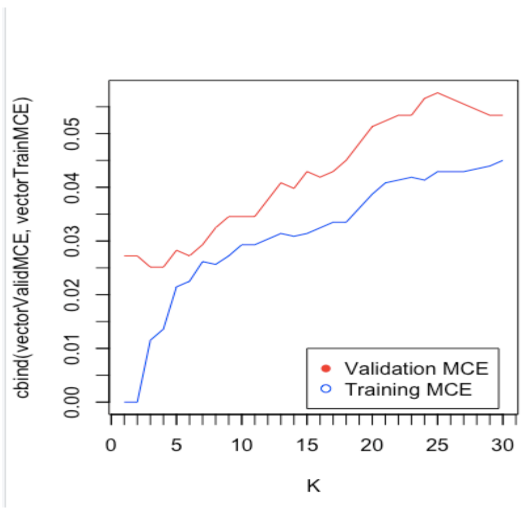
Easy to recognize:

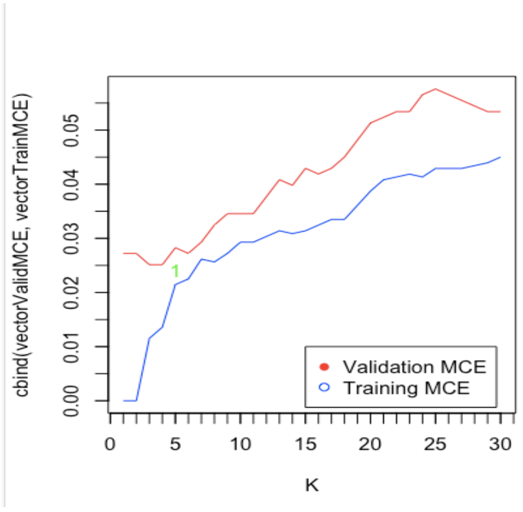




Task 4:

In this part for all K from 1 to 30 we modeled our train and valid data then calculated misclassification errors for both models and saved them to separate vectors. By having these vectors, we could plot MCEs and as we expected the MCE errors for validation data for different Ks were higher than those for training data. Almost for both data sets by increasing K, the errors increased because the accuracy of model decreased by increasing the number of neighbors(K).

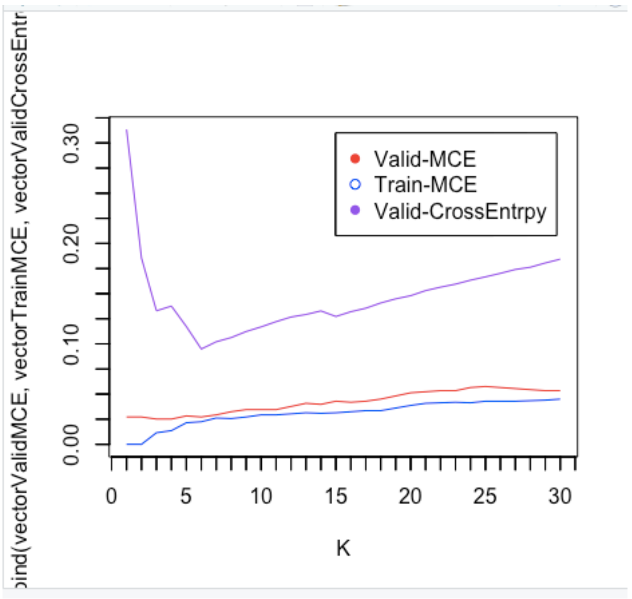




We modeled the test dataset with optimal k which is 3 and compare its' MCE error with validation and train errors on the same plot (it is shown by number 1 on the plot). We can see that the MCE error for test data is lessen than this error for validation data and a bit higher that MCE error for train data. Finally, we think that the optimal K for this model have a good quality.

Task 5:

At first, we modeled the valid data set for all 30 Ks and for each K compute the cross-entropy error. In order to compute it we added 2 more loops, one for sigma of m classes and another to sigma n which is the length of our valid data set. Then we earned 30 cross-entropy for each K and as it is figured below, we plotted train MCE, valid MCE and valid cross entropy at one plot. By considering cross-entropy the optimal k is around 6. We believe that since the cross entropy deal with the probability for each class ,instead of just calculating squared errors ,for multinominal distributions it can be a better choice.



Assignment 2. Linear regression and ridge regression

Task 1.

This assignment is about predicting Parkinson disease based on 16 variables giving a resulting Parkinson disease score, motor_UPDRS. The data came from parkinsons.csv. To start with unused parameters were removed from the matrix, age, test_time, those that were not mentioned in the instructions. Then divided into a training set (60%) and validation set (40%). Followed by being scaled, after which the intercept was removed. The linear regression model was based on this scaled data.

Task 2.

The model was computed and named Fit.

Mean squared error

The mean squared error for the training data is calculated by squaring the residuals of the model and taking the mean of those squares. It is the predicted value minus the actual value squared summed up and then divided with the size of the population.

	Estimated Mean Squared Error (MSE)
Train	0.8785431
Test	3.207502

MSE informs us about the distance between our models regression line and the actual values in the plot. Residuals were gotten from the summary(fit) and the predicted values for the scaled data set were retrieved with the function predict() and finally the actual values were taken from the scaled test data set.

Variable contribution

	Pr(>|t|)	
Jitter...	0.211431	
Jitter.Abs.	3.31e-05	***
Jitter.RAP	0.779658	
Jitter.PPQ5	0.395592	
Jitter.DDP	0.780510	
Shimmer	0.004050	**
Shimmer.dB.	0.215315	
Shimmer.APQ3	0.677694	
Shimmer.APQ5	0.000668	***
Shimmer.APQ11	6.34e-07	***
Shimmer.DDA	0.674695	
NHR	4.84e-05	***
HNR	6.41e-11	***
RPDE	0.857556	
DFA	< 2e-16	***
PPE	6.70e-12	***

The table above is the result from running the sum(fit) function on the model of the training data. Same with the following function: Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1.

In the table you can see stars next to certain variables, for example it is possible to see that Jitter.ABS, Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA and PPE have three stars, which shows an interval for the p-value for that variable. Three stars means that the p-value for these variables is somewhere in the interval 0 to 0.001. Meaning that we can say with statistical significance that these variables contribute to the model.

The p-value shows the relationship between the variable and the null hypothesis, and if it can be rejected with a certain level of confidence. The smaller the p-value, the more likely you are to reject the null hypothesis with a high degree of confidence. To reject the null hypothesis means that the result we want is statistically significant, or we can say with a high degree of certainty that the result we want is the correct one.

Task 3.

Log-likelihood

Based on formula (3.20) in Machine learning - a first course for engineers and scientists (MLFC) by Lindholm et al. visible on page 44.
The logarithm is applied to the likelihood function to ease with calculations since the curve will behave in the same way and reach max or min for the same value on x.
Theta is taken as the vector of coefficients from the model. So is sigma.

Ridge function

Based on lecture 1d slide 21. Formula was negative loglikelihood function with given theta and sigma plus the ridge penalty that was the given lambda multiplied with the sum of theta squared.

Ridge optimization function

Takes a given lambda and uses the optim() function in r. Optim() takes a given function and maximizes or minimizes it. We gave it an empty vector of 17 zeros. The sixteen first positions were for our coefficients in theta and the final seventeenth position was sigma. These were the values the function were supposed to optimize.

DF function based on given lambda

Trace() was used on the hat matrix that was calculated with formula you see here.

$$y = \boldsymbol{X\theta} = \boldsymbol{X}(\boldsymbol{X}^{T}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{T}\boldsymbol{y} = \boldsymbol{Py}$$

This gave the degrees of freedom for a given lambda.
X=scaled train data set.

Task 4.

RidgeOpt() was used to estimate optimal parameters. Seen in table below.

Optimal theta and sigma for given lambda

	Lamba=1	Lamba=100	Lamba=1000
Jitter...	0.29204469	0.0043588759	4.378481e-04
Jitter.Abs.	0.48971808	0.0073092249	7.342099e-04
Jitter.RAP	0.39705097	0.0059261336	5.952790e-04
Jitter.PPQ5	0.43842829	0.0065437057	6.573137e-04
Jitter.DDP	-0.40709224	-0.0060760037	-6.103334e-04
Shimmer	-0.05024367	-0.0007499056	-7.532802e-05
Shimmer.dB.	0.39705097	0.0059261336	5.952790e-04
Shimmer.APQ3	-0.53045919	-0.0079173014	-7.952910e-04
Shimmer.APQ5	-0.04094817	-0.0006111667	-6.139165e-05
Shimmer.APQ11	0.28184432	0.0042066316	4.225552e-04
Shimmer.DDA	-0.07010972	-0.0010464137	-1.051121e-04
NHR	0.10054090	0.0015006102	1.507360e-04
HNR	0.26884707	0.0040126430	4.030691e-04
RPDE	-0.21381983	-0.0031913408	-3.205695e-04
DFA	0.10350229	0.0015448102	1.551758e-04
PPE	0.53249384	0.0079476693	7.983416e-04
sigma	-0.05098402	-0.0007609555	-7.643781e-05

Then motor_UPDRS was estimated based on the optimal parameters which resulted in a matrix to big to show. After this mean squared error was calculated once more for train and test.

MSE for test and train

	λ=1	λ=1	λ=1000
MSE train data	2.8735	0.9969213	0.9993934
MSE test data	2.760689	1.012087	1.015156

Given that lambda gives the smallest mean squared error at 100 the appropriate penalty is lambda equals 100.

Degrees of freedom for the model, given lambda.

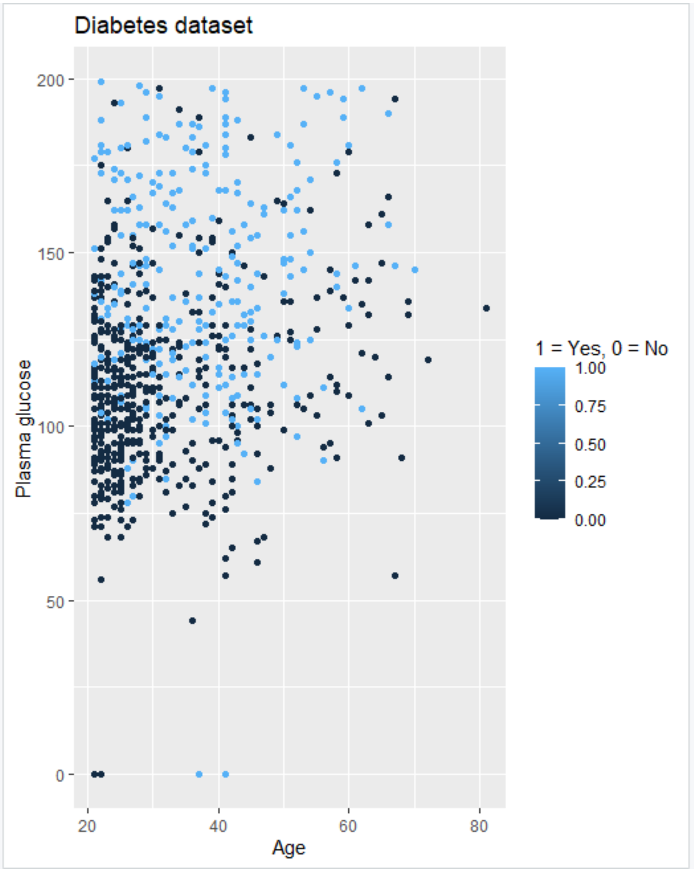
	λ=1	λ=100	λ=1000
Degrees of freedom	14.86012	10.88439	6.390824

Higher lambda leads to lower degrees of freedom. Degrees of freedom give the maximum amount variables that are free to vary in the data sample, meaning at lambda equals 100, 10 variables are free to vary. With more variables free to vary a better approximation of the data is given. So, the model becomes more complex with low lambda and high degrees of freedom.

Assignment 3. Logistic regression and basis function expansion

Task 1.

During this assignment raw data from "pima-indians-diabetes.csv" file was loaded and this full dataset was used in further analysis. The initial dataset was put on the plot where x axis is patients age and y axis is glucose levels in plasma. Data was colored by actual diabetes levels.



We assume that diabetes is relatively easy to classify, as there are only two possible outputs (whether a person has diabetes or not) and there is a certain interconnection between actually having diabetes and a pair of parameters (Age+Blood sugar levels).

Task 2.

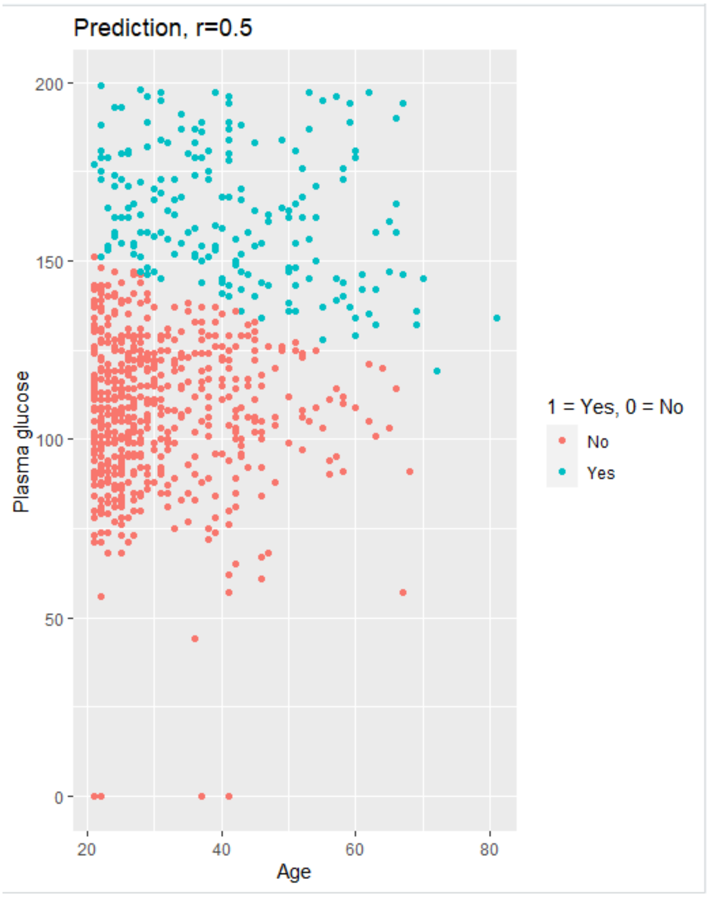
A logistic regression model was trained with the respect to $x1$ =Plasma glucose concentration and $x2$ =Age as features and y =Diabetes as target. Prediction was made for the whole dataset by using r=0.5 as the classification threshold.

The probabilistic equation of the estimated model looks like

p(y|X) = 1 / (1+exp(-(-5.91+0.035x1+0.024x2))).

The training misclassification error computed MSE = 58.29971.

Acquired prediction was put on the plot:



Furthermore, the confusion matrix was calculated:

Prediction/True data	0	1
No	TN = 436	FN = 138
Yes	FT = 64	TP = 130

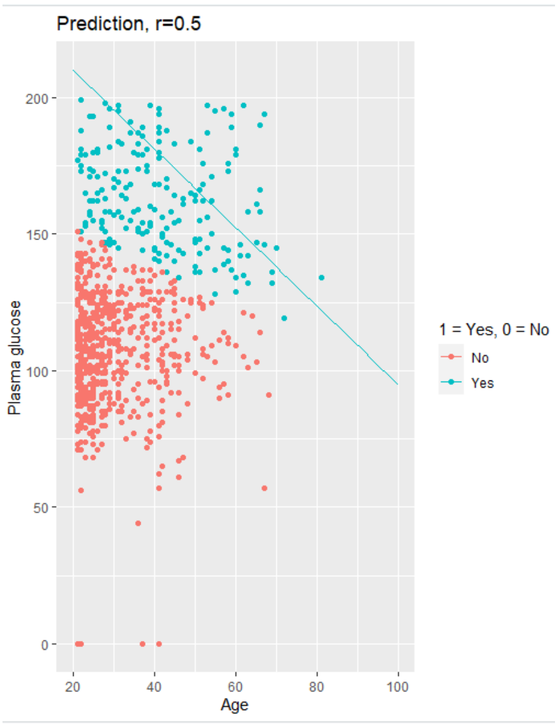
As can be seen from the data and the graph above, the quality is pretty mediocre because as it is seen on the plot, there is no concrete boundary between two domains of decisions so there is a high amount of errors (more than 25% errors, i.e., out of 768 objects 202 were false positive or false negative).

Task 3.

The equation of the decision boundary between the two classes looks like:

-5.91+0.035*x1+0.024*x2 = 0

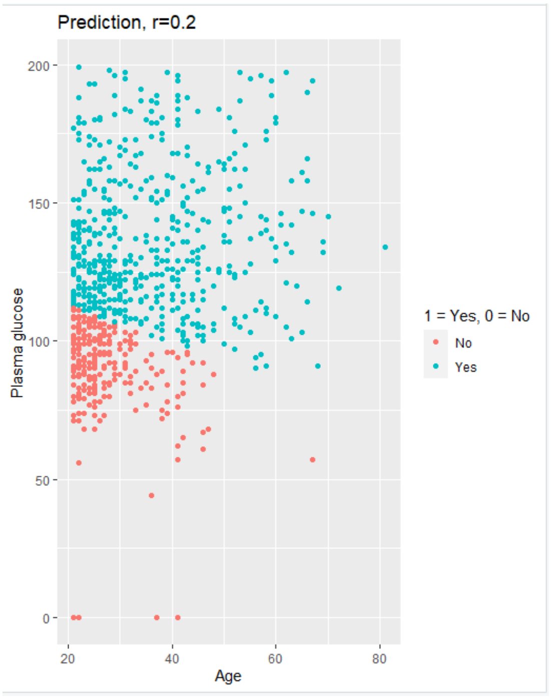
and it can be seen at the graph below:



Looking at the plot we cannot say that the decision boundary catches the data distribution well which is approved by the confusion matrix.

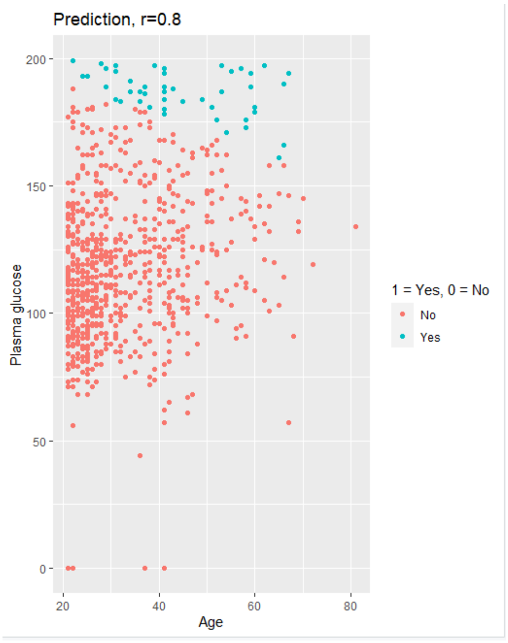
Task 4.

Further on we looked at prediction using r = 0.2:



It can be seen both from the graph and confusion matrix that with r=0.2 the number of FALSE POSITIVEs increased significantly. There are more than 37% errors, i.e. out of 768 objects 286 were false positive or false negative. Hence low threshold is not the best choice.

Then we looked at prediction using r = 0.8:



Although the number of errors slightly decreased, it is still high. There are approx. 32% errors, i.e., out of 768 objects 242 were false positive or false negative. This time there are more FALSE NEGATIVES (which probably is worse than in previous case). High threshold is even worse in this case because it misleads in a detrimental way.

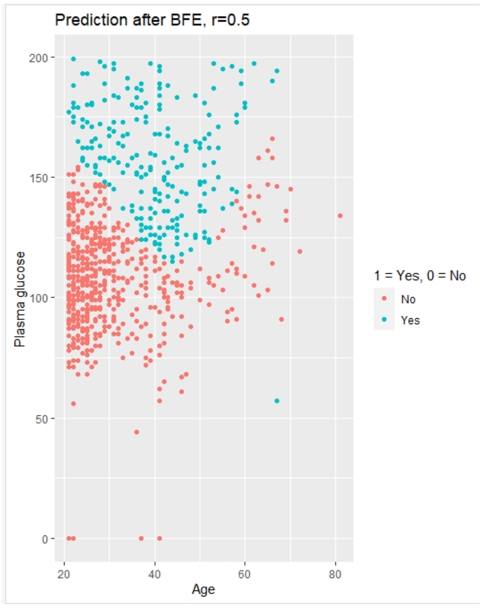
Task 5.

After that we computed new features $z1 = x1^4$, $z2 = x1^3*x2$, $z3 = x1^2*x2^2$, $z4 = x1*x2^3$, $z5 = x2^4$ (basis function expansion). New features were added to the dataset and a new logistic regression model was calculated with features {x1, x2, z1, z1, z3, z4, z5}.

Confusion matrix was calculated:

Prediction/True data	0	1
No	TN = 433	FN = 121
Yes	FT = 67	TP = 147

And the graph was plotted:



Out of 768 objects 188 were false positive or false negative which makes 24% errors. Also, there can be seen that the number of false negatives decreased a bit. From all the iterations we would say the model after BFE + r=0.5 gave the best results because it gave a reduction in the number of errors and the number of false negatives was the least which is crucial in the public health domain.

Appendix. Code.

Assignment1.

```
#----------------------------lab1. assignment1. question1---------------------------

#loading the dataset

optdigit=read.csv("optdigits.csv",header=FALSE)


#factorizing the last column

optdigit$V65=as.factor(optdigit$V65)


#choosing training, validation and test sets

n=dim(optdigit)[1]

set.seed(12345)

id=sample(1:n, floor(n*0.50))

train=optdigit[id,]


id1=setdiff(1:n, id)

set.seed(12345)

id2=sample(id1, floor(n*0.25))

valid=optdigit[id2,]


id3=setdiff(id1,id2)

test=optdigit[id3,]


#----------------------------lab1. assignment1. question2---------------------------
```

```
#install.packages("kknn")
library(kknn)

#building kknn model for training and test data
modelTrain = kknn(V65~., train, train, na.action = na.omit(),k = 30, kernel = "rectangular", scale=TRUE)
modelTest = kknn(V65~., train, test, na.action = na.omit(),k = 30, kernel = "rectangular", scale=TRUE)

#calculating confusion matrices
CMTrain=table(ytrainpred = modelTrain$fitted.values,ytraintrue = train$V65)
CMTest=table(ytrainpred = modelTest$fitted.values,ytraintrue = test$V65)

#calculating miclassification errors
MCETrain=mean(modelTrain$fitted.values != train$V65)
MCETest=mean(modelTest$fitted.values != test$V65)


#----------------------------lab1. assignment1. question3---------------------------

#temporary dataframe for V65 column in initial training dataset and probabilities for digit 8
df <- data.frame(train$V65,modelTrain$prob[,9])

#by looking at this new dataframe we defined that:
#rows 520, 431, 1294 - hardest to predict
#rows 129, 195 - easiest to predict

#reshape row1294 with the probability of 0.16666667 :
# As we can see in the plot it is not easy to recognize it as 8
row1294<- c(train[1294,-65])
row1294df <- data.frame(row1294[1:8])
row1294df[nrow(row1294df)+1,] <- row1294[9:16]
row1294df[nrow(row1294df)+1,] <- row1294[17:24]
row1294df[nrow(row1294df)+1,] <- row1294[25:32]
```

```
row1294df[nrow(row1294df)+1,] <- row1294[33:40]
row1294df[nrow(row1294df)+1,] <- row1294[41:48]
row1294df[nrow(row1294df)+1,] <- row1294[49:56]
row1294df[nrow(row1294df)+1,] <- row1294[57:64]
#build heatmap
heatmap(as.matrix(row1294df),Rowv = NA,Colv = NA)


#reshape row520 with the probability of 0.10000000:
# As we can see in the plot it is not easy to recognize it as 8
row520<- c(train[520,-65])
row520df <- data.frame(row520[1:8])
row520df[nrow(row520df)+1,] <- row520[9:16]
row520df[nrow(row520df)+1,] <- row520[17:24]
row520df[nrow(row520df)+1,] <- row520[25:32]
row520df[nrow(row520df)+1,] <- row520[33:40]
row520df[nrow(row520df)+1,] <- row520[41:48]
row520df[nrow(row520df)+1,] <- row520[49:56]
row520df[nrow(row520df)+1,] <- row520[57:64]
#build heatmap
heatmap(as.matrix(row520df),Rowv = NA,Colv = NA)


#reshape row431 with the probability of 0.13333333:
row431<- c(train[431,-65])
row431df <- data.frame(row431[1:8])
row431df[nrow(row431df)+1,] <- row431[9:16]
row431df[nrow(row431df)+1,] <- row431[17:24]
row431df[nrow(row431df)+1,] <- row431[25:32]
row431df[nrow(row431df)+1,] <- row431[33:40]
row431df[nrow(row431df)+1,] <- row431[41:48]
row431df[nrow(row431df)+1,] <- row431[49:56]
```

```
row431df[nrow(row431df)+1,] <- row431[57:64]

#build heatmap

heatmap(as.matrix(row431df),Rowv = NA,Colv = NA)



#reshape row129 with probability of 1.00000000:

row129<- c(train[129,-65])

row129df <- data.frame(row129[1:8])

row129df[nrow(row129df)+1,] <- row129[9:16]

row129df[nrow(row129df)+1,] <- row129[17:24]

row129df[nrow(row129df)+1,] <- row129[25:32]

row129df[nrow(row129df)+1,] <- row129[33:40]

row129df[nrow(row129df)+1,] <- row129[41:48]

row129df[nrow(row129df)+1,] <- row129[49:56]

row129df[nrow(row129df)+1,] <- row129[57:64]

#build heatmap and comment

heatmap(as.matrix(row129df),Rowv = NA,Colv = NA)




#reshape row195 with the probability of 1.00000000:

row195<- c(train[195,-65])

row195df <- data.frame(row195[1:8])

row195df[nrow(row195df)+1,] <- row195[9:16]

row195df[nrow(row195df)+1,] <- row195[17:24]

row195df[nrow(row195df)+1,] <- row195[25:32]

row195df[nrow(row195df)+1,] <- row195[33:40]

row195df[nrow(row195df)+1,] <- row195[41:48]

row195df[nrow(row195df)+1,] <- row195[49:56]

row195df[nrow(row195df)+1,] <- row195[57:64]


#build heatmap and comment
```

```
heatmap(as.matrix(row195df),Rowv = NA,Colv = NA)




#----------------------------lab1. assignment1. question4---------------------------

#loading the dataset
optdigit=read.csv("optdigits.csv",header=FALSE)

#factorizing the last column
optdigit$V65=as.factor(optdigit$V65)

#choosing training, validation and test sets
n=dim(optdigit)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.50))
train=optdigit[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=optdigit[id2,]

id3=setdiff(id1,id2)
test=optdigit[id3,]

#install.packages("kknn")
library(kknn)
#help("kknn")
```

```
#calculating models for training and validation sets with respect to different K and saving MCEs to
respectful vectors

vectorTrainMCE=c()

vectorValidMCE=c()


i=0

K <- c(1:30)

size <- c(1:nrow(valid))


for (i in 1:30) {


modelTrain = kknn(V65~., train, train, na.action = na.omit(),k = i, kernel = "rectangular", scale=TRUE)

modelValid = kknn(V65~., train, valid, na.action = na.omit(),k = i, kernel = "rectangular", scale=TRUE)


MCETrain=mean(modelTrain$fitted.values != train$V65)

MCEValid=mean(modelValid$fitted.values != valid$V65)


vectorTrainMCE=append(vectorTrainMCE,MCETrain)

vectorValidMCE=append(vectorValidMCE,MCEValid)


}


#building a plot of MCEs


matplot(K,cbind(vectorValidMCE,vectorTrainMCE),type="l",col=c("red","blue"),lty=c(1,1))

legend(15,0.01,legend=c("Validation MCE","Training MCE"),col=c("red","blue"),pch=c(16,1))

#install.packages("Hmisc")

library(Hmisc)

minor.tick(nx = 5,  tick.ratio = 1)




#!!! Find optimal K!!!!

#it seems that 3 can be the optimal K - where the MCE for validation data is minimal
```

```
#estimate the test error for the model with optimal k , compare it with the training and validation
modelTest = kknn(V65~., train, test, na.action = na.omit(),k = 3, kernel = "rectangular", scale=TRUE)
MCETest=mean(modelTest$fitted.values != test$V65)
matpoints(5,MCETest,col="green")



#----------------------------lab1. assignment1. question5---------------------------
#5 calculating cross-entropy for validation data
#plot the dependance
#What is the optimal Kvalue 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?



#introduce vector with our classes
vectorValidCrossEntropy=c()
C <- train$V65
C <- unique(C)
C <- sort(C)

#calculate CrossEntropy for 10 classes while
i=0
m=0
j=1
CrossEntropyValid = 0

for (j in 1:30) {
  modelValid1 = kknn(V65~., train, valid, na.action = na.omit(),k = j, kernel = "rectangular", scale=TRUE)
  #CrossEntropyValid = 0
  for (i in 1:length(valid$V65)){
    for (m in 1:10){
     CrossEntropyValid = CrossEntropyValid + as.numeric(as.numeric(valid$V65[i])==as.numeric(C[m])) *
log(as.double(modelValid1$prob[i,m])+10^-5)


    }
  }
  CrossEntropyValid=-1*CrossEntropyValid/length(valid$V65)
  vectorValidCrossEntropy=append(vectorValidCrossEntropy,CrossEntropyValid)
}


#print(vectorValidCrossEntropy)
# Plot train MCE, valid MCE and valid cross entropy at the same plot
matplot(K,cbind(vectorValidMCE,vectorTrainMCE,vectorValidCrossEntropy),type="l",col=c("red","blue",
"purple"),lty=c(1,1))
#matpoints(5,MCETest,col="green")
legend(15,0.31,legend=c("Valid-MCE","Train-MCE","Valid-
CrossEntrpy"),col=c("red","blue","purple"),pch=c(16,1))
minor.tick(nx = 5,  tick.ratio = 1)
```

<u>Assignment2.</u>

```
#----------------------------lab1. assignment2. linear and ridge regression---------------------------
#setwd("/Users/eli/Desktop/Machine learning/Lab 1/")
#loading the dataset
parkinson=read.csv("parkinsons.csv")
#REMOVE UNNECESSARY COLUMNS FIRST
parkinson$subject.=c()
parkinson$sex=c()
parkinson$test_time=c()
parkinson$age=c()
parkinson$total_UPDRS=c()


#choosing training and test sets
set.seed(12345)
n=nrow(parkinson)
id=sample(1:n, floor(n*0.6))
```

```
train=parkinson[id,]

test=parkinson[-id,]



#scaling

library(caret)

scaler=preProcess(train)

trainS=predict(scaler,train)

testS=predict(scaler,test)


#Compute a linear regression model from the training data, estimate training

#and test MSE and comment on which variables contribute significantly to the model

#no intercept is needed in the modelling



fit = lm(motor_UPDRS~.-1 , data=trainS)


sum=summary(fit)

#calculate MSE for train dataset

MSEtrain=mean(sum$residuals^2)

#run model on test dataset

fitTest=predict(fit,testS,interval = "prediction")

#calculate MSE for test dataset

MSEtest=mean((testS$motor_UPDRS-fitTest)^2)


#comment on which variables contribute significantly to the model.

# Shimmer.DDA

# Shimmer.APQ3

# we scaled the data => resulting coefficients can be compared and we can compare absolute values


#??????question?????????? (significant contribution) - should some other variables be picked? How to
define a threshold?

#answer : look at p-values and ***
```

```
#CHECK IN GOOGLE
summary(fit)



#Implement 4 following functions


#3a) Log-likelihood function that for a given parameter vector θ and
#dispersion σ computes the log-likelihood function log P(T|θ, σ) for
#the stated model and the training data
#Going by that motor_UPDRS is a normal distribution as mentioned in 1.
#Formula log-likelihood F(θ)=log(L(θ))=sum(over i=1 to n) log(fi(yi|θ))


#??????question??????????: is the set of θwe are trying to optimize equal to the vector of coefficients in LR model   - YES
vectorTheta = fit$coefficients
names(vectorTheta) <- NULL
dispersionSigma = summary(fit)$sigma
logLikelihood<-function(theta,sigma){
  #population size n

  n<-length(theta)
  #print(n)
  actualValue<-trainS$motor_UPDRS[1:n] #y in the formula.
  #print(actualValue)
  #print(length(parkinson$motor_UPDRS))
  #ERROR!
  #Expected value, estimated by using the average value.
  #ev=mean(theta)
  #changed it to transpose of the in parametr theta to match p44 book.

  predictedTheta=theta
  #print(ev)
  #Done here because when calculations were done in "sum" in the formula, errors appeared.
  x=(predictedTheta-actualValue)^2 #is vector of
  #print("printing x")
  #print(x)
  #ERROR!

  #Formula
  result=((-1*n/2)*log(2*pi,base=exp(1))-(n/2)*log(sigma^2,base=exp(1))-1/(2*sigma^2)*sum(x,
na.rm=FALSE))

  return (result)
}
print(logLikelihood(vectorTheta,dispersionSigma))
logLik(fit)

#3b) Ridge function that for given vector 𝜽𝜽, scalar 𝜎𝜎 and scalar 𝜆𝜆 uses function from 3a and adds up
a Ridge penalty 𝜆‖𝜽𝜽‖2 to the minus loglikelihood
mylambda=1

myridge<-function(theta,sigma,lambda){

 # print(theta)
  #print(sigma)
  #print(lambda)
  ridgePenalty=lambda*sum(theta^2)

  #print(ridgePenalty)
 # print(logLikelihood(theta,sigma ))
  result =  -logLikelihood(theta,sigma )+ ridgePenalty

 # print(result)
  return (result)

}
```

```
print(myridge(vectorTheta,dispersionSigma,mylambda))



# 3c) 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.
#ANSWER: initiate with 0 and then compare the result of our custom prediction with true values
par1 <- rep(c(0),each=16)
sigmaPar=1
par1 <- append(par1,sigmaPar)

myRidgeOpt=function(lambdaIn) {





  result<- optim(par1,fn=myridge,sigma=sigmaPar,lambda=lambdaIn,method="BFGS")

  #WE SHOULD OPTIMIZE THEM TOGETHER

  return (result)
}

optimal=myRidgeOpt(mylambda)
print(optimal)

#3d) Df function that for a given scalar λ computes the degrees of freedom
#of the Ridge model based on the training data
install.packages("glmnet")
library(glmnet)
install.packages("psych")
library(psych)
```

```
lambdaIn=1
DF=function(lambdaIn) {

  # Compute hat-matrix and degrees of freedom
  ld <- lambdaIn * diag(ncol(trainS))
  trainmatrix<-data.matrix(trainS)
  H <- trainmatrix %*% solve((t(trainmatrix) %*% trainmatrix + ld)) %*% t(trainmatrix)
  #library("psych")
  df <- tr(H)
  return(df)

}

print(DF(lambdaIn))



#----------------------------4. -------
#By using function RidgeOpt, compute optimal 𝜽𝜽 parameters for 𝜆𝜆 = 1, 𝜆𝜆 =
#  100 and 𝜆𝜆 = 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.

#compute
myRidgeOpt(1)$par
myRidgeOpt(100)
myRidgeOpt(1000)

#Predicted motor_UPDRS for train set with lamda=1
data.matrix(trainS[,2:17])%*%myRidgeOpt(1)$par[1:16]
data.matrix(trainS[,2:17])%*%myRidgeOpt(100)$par[1:16]
data.matrix(trainS[,2:17])%*%myRidgeOpt(1000)$par[1:16]


#MSE calculation train set

MSEtrainLambda1 <- mean((data.matrix(trainS[,2:17])%*%myRidgeOpt(1)$par[1:16]-
trainS$motor_UPDRS)^2)

MSEtrainLambda100 <- mean((data.matrix(trainS[,2:17])%*%myRidgeOpt(100)$par[1:16]-
trainS$motor_UPDRS)^2)

MSEtrainLambda1000 <- mean((data.matrix(trainS[,2:17])%*%myRidgeOpt(1000)$par[1:16]-
trainS$motor_UPDRS)^2)


#MSE calculation for test

MSEtestLambda1 <- mean((data.matrix(testS[,2:17])%*%myRidgeOpt(1)$par[1:16]-
testS$motor_UPDRS)^2)

MSEtestLambda100 <- mean((data.matrix(testS[,2:17])%*%myRidgeOpt(100)$par[1:16]-
testS$motor_UPDRS)^2)

MSEtestLambda1000 <- mean((data.matrix(testS[,2:17])%*%myRidgeOpt(1000)$par[1:16]-
testS$motor_UPDRS)^2)


#Pick penalty value lambda 100 it gives the smallest error


#Compute and compare degrees of freedom

DF(1)

DF(100)

DF(1000)


#higher lambda more complex

#higher lambda means fewer independent variables, fewer variables that can vary in value <-

#more degrees of freedom better approximation.


#When we have lambda is 100 error is lowest


#We want low degrees of freedom and the penalty factor lambda high but with higher lambda but

#also low ammounts of errors and after a 100 lambda the errors went up again.
```

<u>Assignment3.</u>

#----------------------------lab1. assignment3. logistic regression and BFE---------------------------

#loading the dataset

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

#---question1---

#1. Make a scatterplot showing a Plasma glucose concentration on Age where observations are colored by Diabetes levels.

#---

library("ggplot2")

pl <- ggplot(diabetes,aes(x=diabetes$V8,y=diabetes$V2,col=diabetes$V9))+geom_point()

pl + labs(x = "Age",y = "Plasma glucose", title = "Diabetes dataset",colour = "1 = Yes, 0 = No")

#---

#Do you think that Diabetes is easy to classify by a standard logistic regression model that uses these two variables as features?

#---

#answer - relatively easy, as there are only two possible outputs (whether a person has diabetes or not) and there is a certain interconnection between

#actually having diabetes and a pair of parameters (Age+Blood sugar levels)

#---question2---

#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.

#---

#building a logistic regression model based on the whole dataset

model=glm(diabetes$V9~diabetes$V2+diabetes$V8,family=binomial())

summary(model)

```
prediction = predict(model,diabetes,type="response")


#building a vector of predictions based on the threshold of 0.5

#Yes - diabetes, No - healthy

vectorPred=rep("No",nrow(diabetes))

vectorPred[prediction>0.5] = "Yes"

View(vectorPred)


#building a confusion matrix

table(vectorPred,diabetes$V9)

#result => more than 25% errors, i.e. out of 768 objects 202 were false positive or false negative


#-----------------------------------------------------------------------------------------

#Report the probabilistic equation of the estimated model (i.e., how the target depends on the features and the estimated model parameters probabilistically).

#-----------------------------------------------------------------------------------------


print(model$coefficients)

#acquired coefficients from the model

#(Intercept) diabetes$V2 diabetes$V8

#-5.91244906  0.03564404  0.02477835


#formula is as following:

#   p(y|X)= 1 / (1+exp(-(-5.91+0.035x1+0.024x2)))


#-----------------------------------------------------------------------------------------

#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.

#-----------------------------------------------------------------------------------------


MSE=mean(model$residuals^2)

pl2 <- ggplot(diabetes,aes(x=diabetes$V8,y=diabetes$V2,col=vectorPred))+geom_point()
```

pl2 + labs(x = "Age",y = "Plasma glucose", title = "Prediction, r=0.5",colour = "1 = Yes, 0 = No")

#--

#Comment on the quality of the classification by using these results.

#--

#The quality is pretty mediocre because as it is seen on the plot, there is no concrete boundary between two domains of decisions so there is a high amount of errors

#--question3--

#3. Use the model estimated in step 2 to

#a) report the equation of the decision boundary between the two classes

#--

# -5.91+0.035x1+0.024x2 = 0

#--

#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

#--

pl3=ggplot(diabetes,aes(x=diabetes$V8,y=diabetes$V2,col=vectorPred))+geom_point()

fun.1 <- function(x1) (5.91244906 - 0.03564404*x1)/ 0.02477835

pl3 + stat_function(fun = fun.1) + xlim(20,100) + labs(x = "Age",y = "Plasma glucose", title = "Prediction, r=0.5",colour = "1 = Yes, 0 = No")

#looking at the plot we would not say that the decision boundary catches the data distribution well which is approved by the confusion matrix

#--question4--

```
#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 rvalue changes.

#-----------------------------------------------------------------------------------------


#building a vector of predictions based on the threshold of 0.2

#Yes - diabetes, No - healthy

vectorPred02=rep("No",nrow(diabetes))

vectorPred02[prediction>0.2] = "Yes"

View(vectorPred02)

#building a confusion matrix

table(vectorPred02,diabetes$V9)

#result => more than 37% errors, i.e. out of 768 objects 286 were false positive or false negative

pl4 <- ggplot(diabetes,aes(x=diabetes$V8,y=diabetes$V2,col=vectorPred02))+geom_point()

pl4 +  labs(x = "Age",y = "Plasma glucose", title = "Prediction, r=0.2",colour = "1 = Yes, 0 = No")

#it can be seen both from the graph and confusion matrix that with r=0.2 the number of FALSE
POSITIVEs increased significantly

#hence low threshold is not the best choice



#building a vector of predictions based on the threshold of 0.8

#Yes - diabetes, No - healthy

vectorPred08=rep("No",nrow(diabetes))

vectorPred08[prediction>0.8] = "Yes"

View(vectorPred08)

#building a confusion matrix

table(vectorPred08,diabetes$V9)

#result => approx 32% errors, i.e. out of 768 objects 242 were false positive or false negative

pl5 <- ggplot(diabetes,aes(x=diabetes$V8,y=diabetes$V2,col=vectorPred08))+geom_point()

pl5 +  labs(x = "Age",y = "Plasma glucose", title = "Prediction, r=0.8",colour = "1 = Yes, 0 = No")

#although the number of error slightly decreased, it is still high. This time there are more FALSE
NEGATIVES (which probably is worse than in previous case)

#high threshold is even worse in this case because it misleads in a detrimental way
```

```
#------------------------------------------------------------question5--------------------------------------------------------

#Perform a basis function expansion trick by computing new features𝑧1 = , adding them to the data set and

#then computing a logistic regression model with 𝑦 as target and #𝑥1, 𝑥2,𝑧1, … , 𝑧5 as features.

#------------------------------------------------------------------------------------------


#x1 = diabetes$V2 = plasma glucose level

#x2=diabetes$V8 = age

BFEdiabetes= diabetes

BFEdiabetes$Z1=BFEdiabetes$V2^4

BFEdiabetes$Z2=BFEdiabetes$V2^3 * BFEdiabetes$V8

BFEdiabetes$Z3=BFEdiabetes$V2^2 * BFEdiabetes$V8^2

BFEdiabetes$Z4=BFEdiabetes$V2 * BFEdiabetes$V8^3

BFEdiabetes$Z5=BFEdiabetes$V8^4


modelBFE=glm(BFEdiabetes$V9~BFEdiabetes$V2+BFEdiabetes$V8+BFEdiabetes$Z1+BFEdiabetes$Z2+BFEdiabetes$Z3+BFEdiabetes$Z4+BFEdiabetes$Z5,family=binomial())

summary(model)



predictionBFE = predict(modelBFE,BFEdiabetes,type="response")


#------------------------------------------------------------------------------------------

#Create a scatterplot of the same kind as in step 2 for this model and compute the training misclassification rate.

#------------------------------------------------------------------------------------------


#building a vector of predictions based on the threshold of 0.5 (after basis function expansion)

#Yes - diabetes, No - healthy

vectorPredBFE=rep("No",nrow(BFEdiabetes))

vectorPredBFE[predictionBFE>0.5] = "Yes"
```

```
View(vectorPredBFE)


#building a confusion matrix

table(vectorPredBFE,BFEdiabetes$V9)




MSEBFE=mean(modelBFE$residuals^2)

pl6 <- ggplot(BFEdiabetes,aes(x=BFEdiabetes$V8,y=BFEdiabetes$V2,col=vectorPredBFE))+geom_point()

pl6 +  labs(x = "Age",y = "Plasma glucose", title = "Prediction after BFE, r=0.5",colour = "1 = Yes, 0 = No")


#-----------------------------------------------------------------------------------------

#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?

#You can visualize the decision boundary by plotting the scatter plots and coloring the samples by your
prediction from this new model.

#You do not need to plot the decision function as such.

#-----------------------------------------------------------------------------------------


#result => out of 768 objects 188  were false positive or false negative which makes 24% errors

#also there can be seen that the amount of false negatives decreased a bit


#take a look at the coefficients

print(modelBFE$coefficients)

#(Intercept)   BFEdiabetes$V2   BFEdiabetes$V8 BFEdiabetes$Z1 BFEdiabetes$Z2  BFEdiabetes$Z3
BFEdiabetes$Z4  BFEdiabetes$Z5

#-9.309821e+00   3.793014e-02   1.456805e-01   1.278015e-08   -1.779600e-07   8.515150e-07   -
1.698011e-06   8.126623e-07

#   1/(1+exp(-(-9.309821e+00 + 3.793014e-02*x1 + 1.456805e-01*x2 + 1.278015e-08*x1^4 - 1.779600e-
07*x1^3*x2 + 8.515150e-07*x1^2*x2^2 - 1.698011e-06*x1*x2^3 +  8.126623e-07*x2^4 )))=0.5
```