Computer Lab 1 block 1

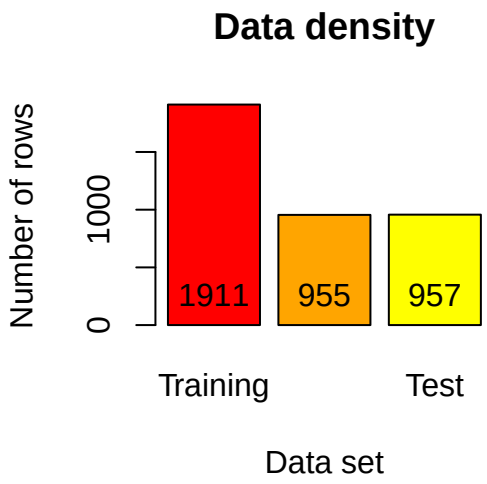
Alan Cacique Tamariz (alaca734) , Tugce Izci (tugiz798) and Kerstin Nilsson (kerni714)

2022-11-20

Assignment 1 Handwritten digit recognition with Knearest neighbors.

1.1 Data set Partition

The original data set **"optdigits.csv"** has a dimension of **3823 rows and 65 columns.** The last column[65] is the target, and represent which number the each row is. After dividing the data set with the proportions indicated the number of rows for each sub set are: **Training: 1911, Validation: 955 and Test: 957**.



1.2 Training the model (Confusion matrices and Misclasification errors)

The Misclassification rate can be calculated as follows:

$$R(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} I(Y_i \neq \hat{Y}_i)$$

The Misclassification Rate of the Training data set is 4.50 %. The Misclassification Rate of the Test data set is 5.33 %.

From the confusion matrices of Testing and Training data sets, we calculated the prediction error for each class. For the Training the numbers 1, 4 and 9 had the highest error rates. For the Testing the numbers 4,

5 and 8, are the ones with the highest error. The error rates for the test set ranges from 0-14%, thus the highest prediction error for the individual digits is almost three times as high as the overall error rate.

The Misclassification error from training and test are similar which indicates that the model might be too simple.

```
#>    pred_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
```

```
#>    pred_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
```

```
#> Misclasification rate per class - Train
```

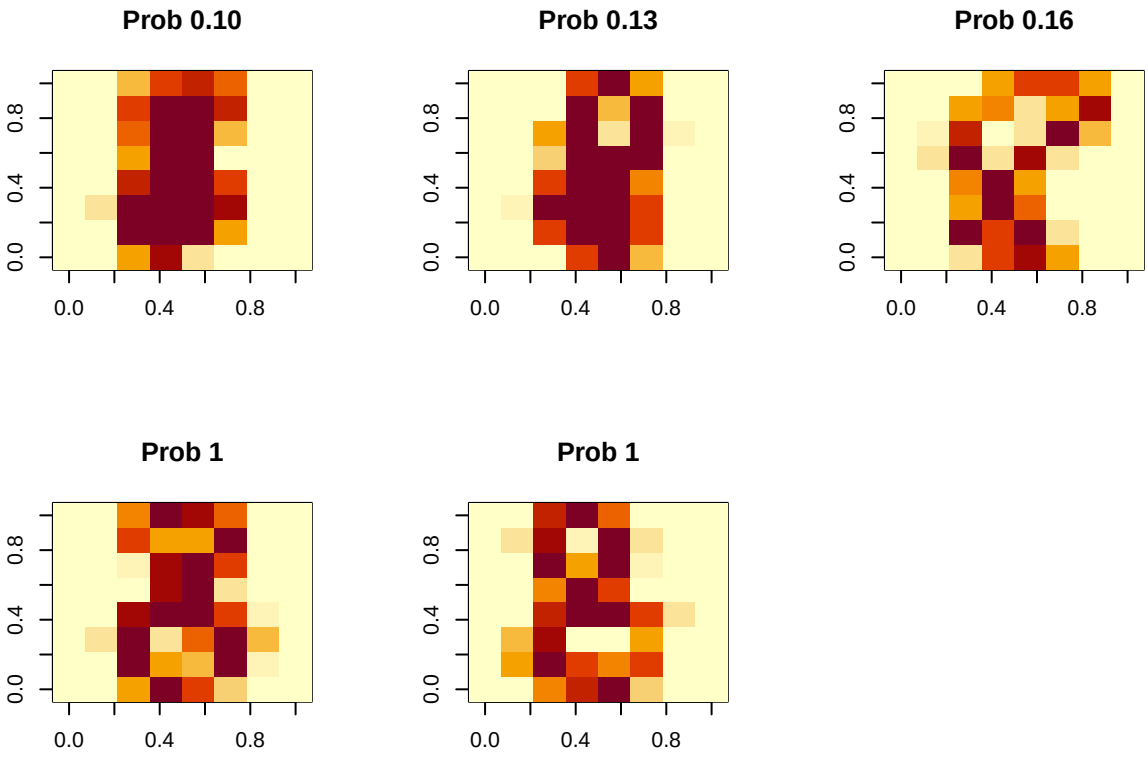
```
#>    0    1    2    3    4    5    6    7    8    9
#> 0.00 8.21 1.04 1.60 9.14 5.52 1.04 2.20 7.84 8.50
```

```
#> Misclasification rate per class- Test
```

```
#>     0     1     2     3     4     5     6     7     8     9
#>  1.28  5.81  2.97  3.60 13.76  9.71  0.00  0.89 10.26  4.49
```

1.3 Easiest and Hardest cases for digit "8"

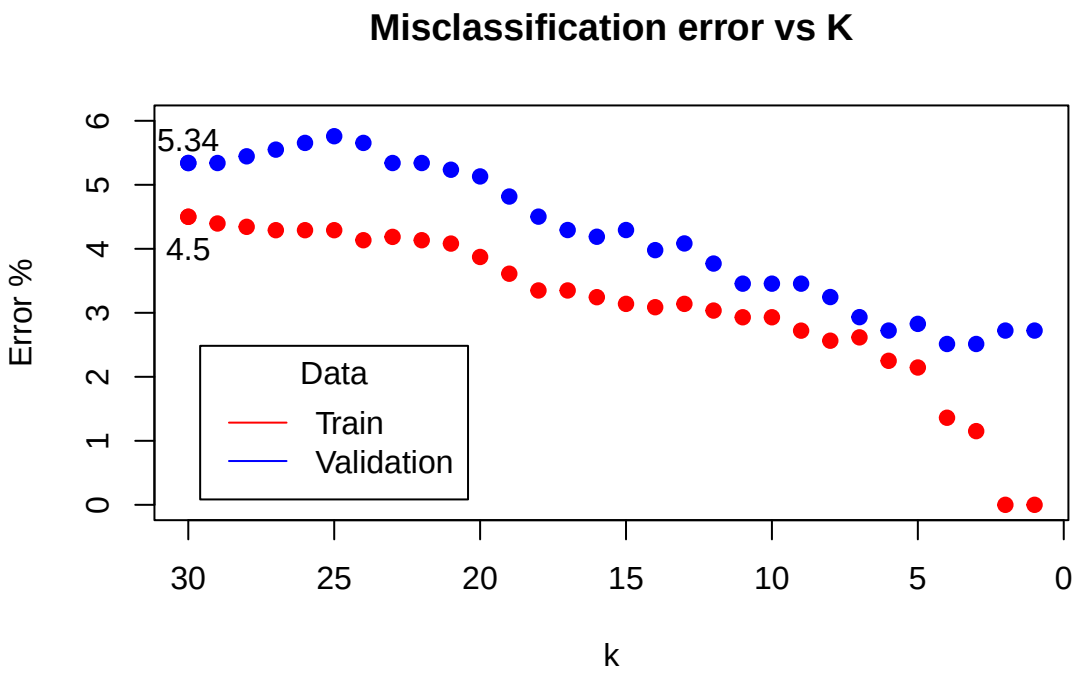
The heatmap plot shows that the three cases with lowest probability or the hardest to predict, have shapes which the two circles of the 8 figure is not define in comparison with the easiest to predict that both circles are well defined. Another detail is that the cases with highest probability have darker colors on the perimeter of the shape, this indicates that there is more pixels in that position and is easier for the model to identify.



1.4 Model Complexity

The complexity of the models increases as the number of K or numbers of nearest neighbors decreases. From the plot it can be seen that there is a substantial difference between the models. For the training set the error decreases with increasing complexity, down to 0 for K equals 1 and 2. The validation error also decreases generally until K equals 4. The Validation error then starts to increase from K equals 2. The optimal K is number 4 because it has the lowest Misclassification Rate and compared with K equals 3, that has the same error we choose K=4 because it has lower complexity.

From the plot it can be seen that for K equal to 1 and 2, for the training data set resulted in a Misclassification rate equal to 0. For K equal to 1 for the training set, the model will just identify for each observation itself as the closest data point, meaning the prediction error will be 0 for the training set, and the model will highly likely lead to overfitting.



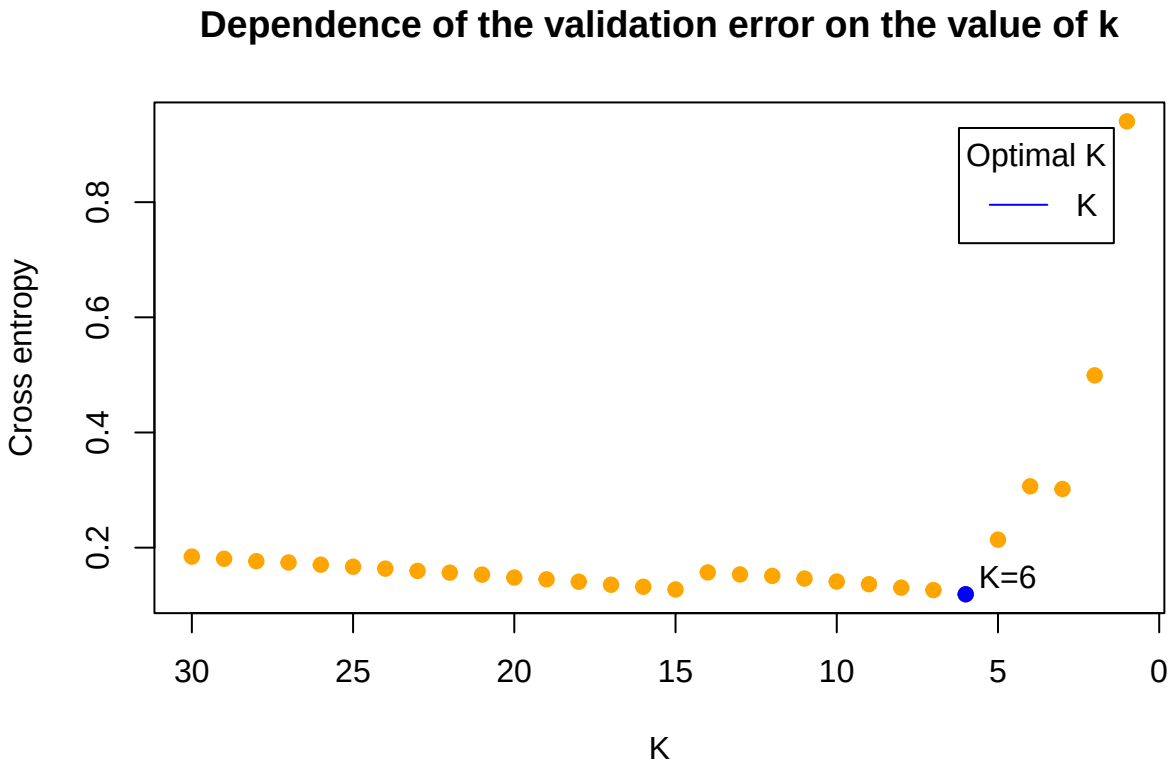
1.4.1 Misclassification error with Test data set, with optimal K

The overall Misclassification error of the test data has a similar value to the model with the validation data, around 2.5%, which is now higher than for training dataset (error 1.4%). Looking at the error rates per class, we can see that the highest error is 5.5%, which might be thought of as a good model (but it also depends on the application).

```
#>  Misclasification rate trainning % Misclasification rate validation %
#>                               1.360                               2.513
#>       Missclassification_rate_Test
#>                               2.510

#> By class error rates:

#>    0    1    2    3    4    5    6    7    8    9
#> 1.28 2.33 0.99 1.80 5.50 1.94 0.00 1.79 5.13 4.49
```

1.5. Cross entropy for training data

The optimal value of K is 6, because it is the one with lowest cross entropy. Cross entropy is a better error performance metric than Misinterpretation rate for multiclass classification; Cross entropy penalize the lower probabilities on each observation for the targets more (penalizes higher loss to predictions), so it gives a better sense of how is performing your model.


Dependence of the validation error on the value of k
Cross entropy
0.2
0.4
0.6
0.8
30
25
20
15
10
5
0
K
Optimal K
K
K=6

Assignment 2. Linear Regression and Ridge Regression

2.1 Pre-processing

We pre-processed the data by performing centering and scaling (to standard deviation of 1) of the variables in the training dataset, and then applied the scaling parameters to the test set. We also scaled the y variable (motor_UPDRS) in addition to centering it, in the hope that it would add stability to the optimisation process and further aid in the task of setting starting values for the parameters. In addition to the voice characteristics variables, we also included the variables for age and sex in the modelling. We further assume that observations are independent.

2.2 Computation of linear model

We computed a linear regression model from the training data and estimated MSE.

```
#> mse_training_lm     mse_test_lm
#>          0.8340          0.8917
```

To understand which variable contribute significantly to the data we need to take absolute value of the coefficients and the highest values are represented as the significant ones. The summary below shows estimated regression coefficients and their standard errors.

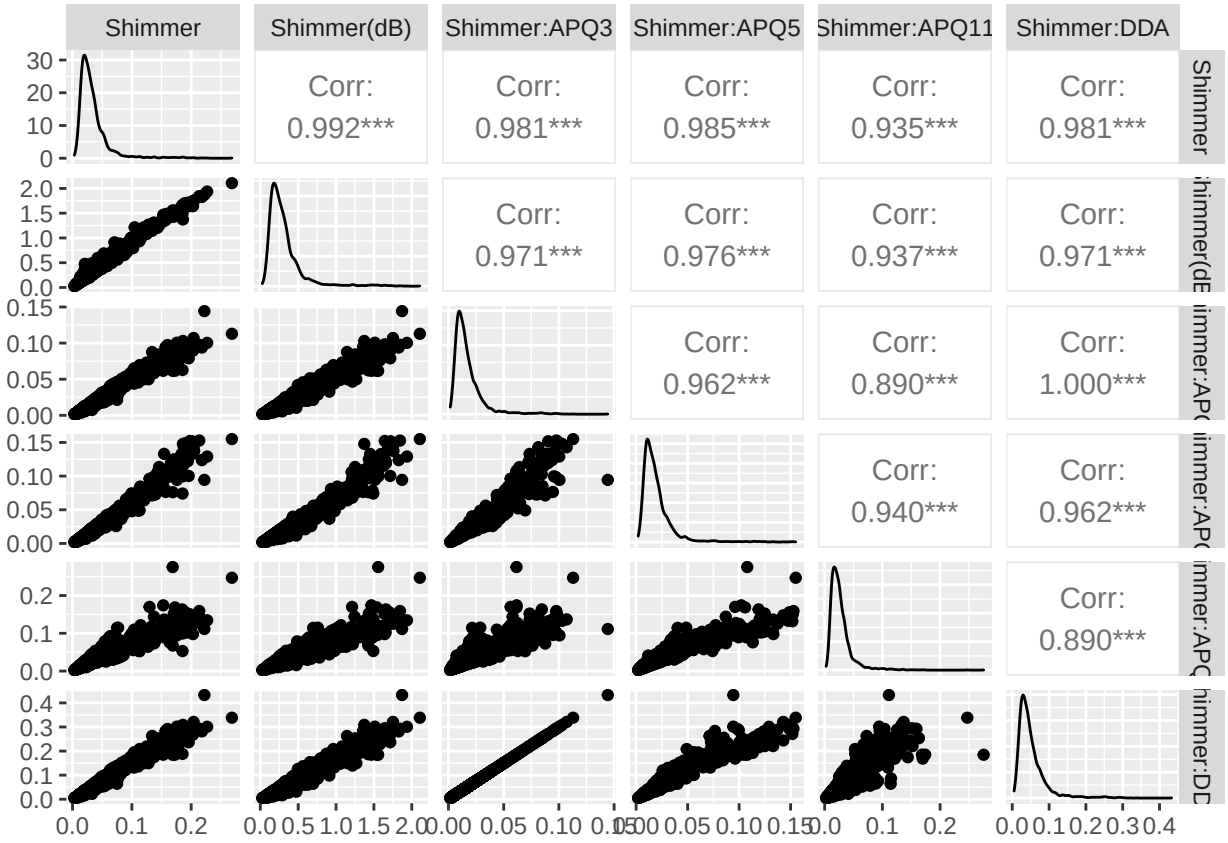
The regression coefficients with the largest (absolute) values were Shimmer:DDA and Shimmer:APQ3, with estimated values of -45.90 and 45.54 respectively. These were followed by Jitter:DDP with an estimated value of 1.57 and Jitter:RAP with an estimated value of -1.49. However, the standard errors for these estimates were large compared to their value, which indicates that the estimated values are not reliable.

```
#>
#> Call:
#> lm(formula = formula, data = data)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -3.457 -0.721 -0.105  0.734  2.430
#>
#> Coefficients:
#>                  Estimate Std. Error t value Pr(>|t|)
#> age               0.21408    0.01609   13.31  < 2e-16 ***
#> sex              -0.05782    0.01780   -3.25   0.0012 **
#> `Jitter(%)`       0.15607    0.14578    1.07   0.2844
#> `Jitter(Abs)`    -0.22396    0.04242   -5.28  1.4e-07 ***
#> `Jitter:RAP`     -1.49817   18.36258   -0.08   0.9350
#> `Jitter:PPQ5`    -0.11078    0.08612   -1.29   0.1984
#> `Jitter:DDP`      1.56805   18.36542    0.09   0.9320
#> Shimmer           0.60474    0.20081    3.01   0.0026 **
#> `Shimmer(dB)`    -0.21258    0.13581   -1.57   0.1176
#> `Shimmer:APQ3`   45.54410   75.21238    0.61   0.5449
#> `Shimmer:APQ5`   -0.28523    0.11115   -2.57   0.0103 *
#> `Shimmer:APQ11`   0.21754    0.06012    3.62   0.0003 ***
#> `Shimmer:DDA`   -45.89624   75.21207   -0.61   0.5418
#> NHR              -0.10410    0.04563   -2.28   0.0226 *
```

```
#> HNR              -0.23193    0.03568   -6.50  9.2e-11 ***
#> RPDE             -0.00368    0.02221   -0.17   0.8683
#> DFA              -0.23052    0.01996  -11.55  < 2e-16 ***
#> PPE               0.20110    0.03212    6.26  4.3e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.916 on 3507 degrees of freedom
#> Multiple R-squared:  0.166,  Adjusted R-squared:  0.162
#> F-statistic: 38.7 on 18 and 3507 DF,  p-value: <2e-16
```

To gain some further insight into the data, correlation plots of the training data were produced.

In the below, pairwise scatterplots for the Shimmer variables are displayed which two of them APQ3 and DDA had the largest regression coefficients (code for displaying further pairwise scatterplots are available in the Appendix).



As can be seen in the plots, the Shimmer variables are highly correlated, with Shimmer:APQ3 and Shimmer:DDA appearing to be perfectly correlated. The values however are not an exact multiple of the other, as can be seen by taking the ratio of the two variables (not shown here, but code is available in the Appendix).

2.3 Implementation of optimisation functions

We implemented Loglikelihood, Ridge, Ridgeopt and DF functions. In the Ridge function, to obtain θ values according to the following criterion (page 63 in The Elements of Statistical Learning by Hastie *et al*):

$$\hat{\theta}^{ridge} = \underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\theta_j)^2 + \lambda \sum_{j=1}^{p} \theta_j^2 \right\}$$

we need to multiply the factor $\frac{1}{2\sigma^2}$ with the ridge penalty term $\lambda||\theta||^2$ (that we then sum with the minus log likelihood obtained from the Loglikelihood function).

2.4 Selection of model

By using the Ridgeopt funtion we compute optimal θ parameters for $\lambda = 1$, $\lambda = 100$ and $\lambda = 1000$. By using the estimated regression coefficients we predicted motor_UPDRS values for train and the test data, then MSE was calculated for training and test data.

Table 1: Degrees of freedom (df) and MSE for training and test set for different λ.

λ	mse train	mse test	df
1	0.8341	0.8911	15.9
100	0.8371	0.8898	11.8
1000	0.8572	0.8996	7.1

-$\lambda = 100$ has the lowest error for the test set, although it seems the differences between the models are quite small. For the training set, the error increase with increasing λ as can be expected. It can be noted that while the MSE for the test set for the linear model is higher than the selected ridge regression model, it is not that far from it, indicating that even though the estimates of the regression coefficients were not reliable, the predictive ability was still quite good in comparison with the ridge regression models computed here.

- Degrees of freedom formula indicates that larger covariance increases degrees of freedom and this result cause to more complex model.

$$df(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^{N} Cov(\hat{y}_i, y_i)$$

When the degrees of freedom was calculated by using the DF function indicated that the increase in λ values decreases the degrees of freedom and thus the complexity of the model.

Finally, to gain some more understanding of the quality of model, a model was computed with the y variable (motor_UPDRS) mean centered only, (i.e. not scaled), to obtain the residuals on the original scale for the selected λ value 100. RMSE was calculated for the training and test data, and for the test data, the RMSE was related to the mean of the y variable.

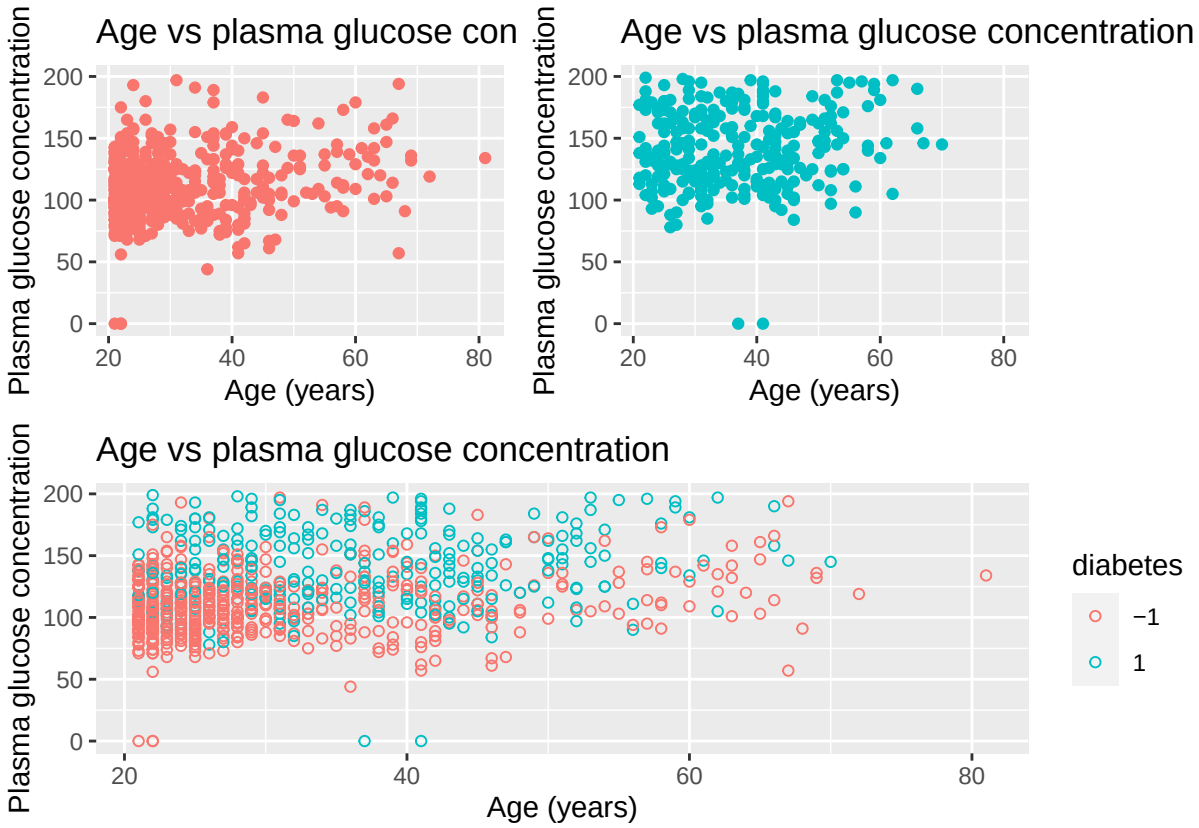
```
#> RMSE for training and test data:

#> [1] 7.415

#> [1] 7.645

#> Mean of motor_UPDRS in the test data: 21.22

#> RMSE as a percentage of the mean: 36
```

The RMSE in relation to the mean for the test set was 36%. This number needs to be put in relation to the application to assess whether it can be considered to be a useful model.

Assignment 3. Logistic regression

3.1 Plot

Plot of age vs glucose plasma concentration, colour by diabetes:



There appears to be considerable overlap between observations classified as diabetes and observations classified as non-diabetes, but it is not so easy from the plot to determine the extent of the overlap. However, it seems it will not be very easy to classify the observations into the two groups.
It can be noted that there are some observations with very low/zero plasma glucose levels, these observations may be erroneous.

3.2 Logistic regression model

A logistic regression model with diabetes as target and glucose plasma concentration and age as features were performed.

Estimated regression coefficients:

```
#> (Intercept)  glucose_pl         age
#>    -5.91245     0.03564     0.02478
```

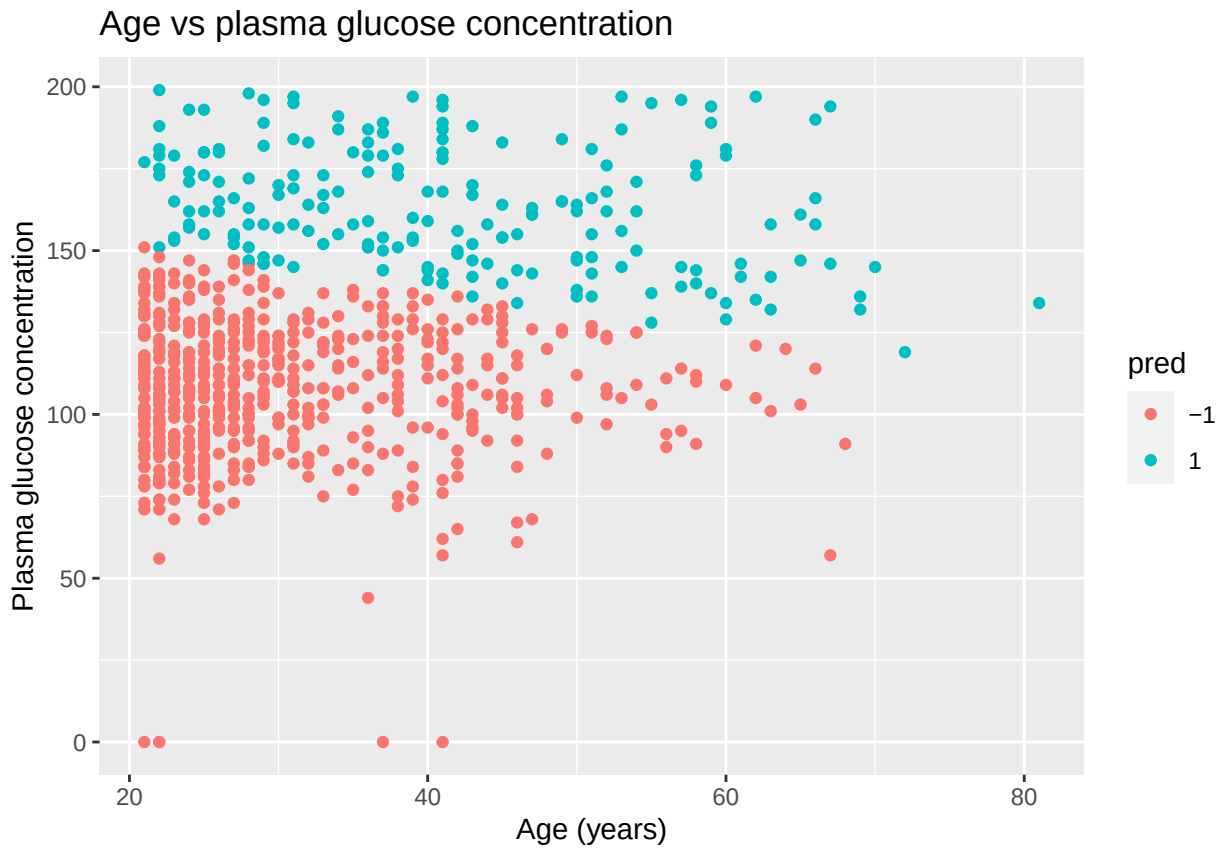
Probabilistic model:

$$p(y=1|\boldsymbol{x}) = g(\boldsymbol{x}) = \frac{e^{\boldsymbol{\theta}^T\boldsymbol{x}}}{1+e^{\boldsymbol{\theta}^T\boldsymbol{x}}} = \frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}} = \frac{1}{1+e^{-(-5.91244906+0.03564404x_1+0.02477835x_2)}}$$

Training misclassification error (percent) and confusion matrix:

```
#> Confusion matrix:

#>       pred
#> target  -1   1
#>     -1 436  64
#>     1  138 130

#> Misclassification error:

#> [1] 26.3

#> Percentage of observations that is non-diabetes:

#> [1] 65.1
```

Plot of age vs glucose plasma concentration, colour by predicted values:



The prediction error is quite high, 26%, while it means that 74% of observations are correctly classified. However, in assessing the discriminative ability of the model one need to consider the imbalance in the number of observations in the different classes, with 65% of the observations belonging to the non-diabetes class. The misclassification rate is larger for the smaller group.

3.3 Decision boundary

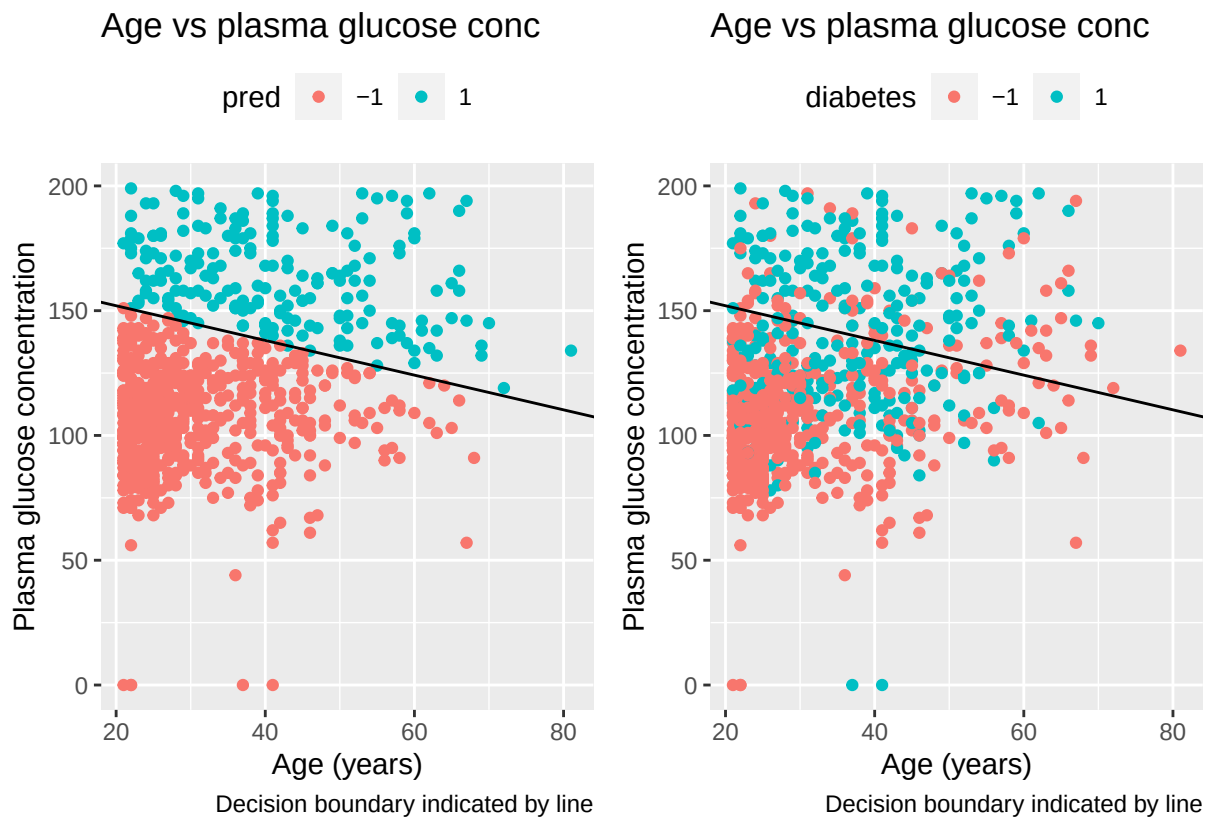
Decision boundary equation:
Setting

$$g(\boldsymbol{x}) = 1 - g(\boldsymbol{x})$$

gives

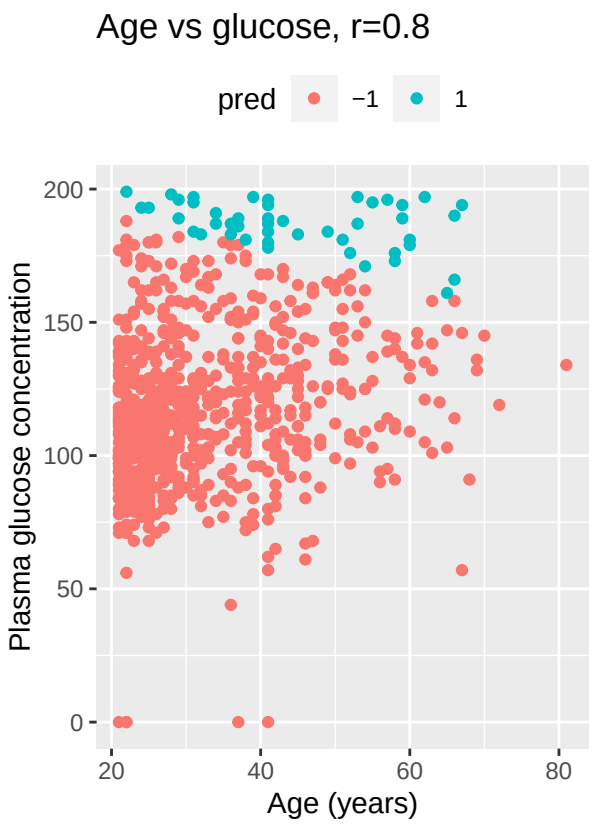
$$x_1 = \frac{\theta_0}{\theta_1} - \frac{\theta_2}{\theta_1} * x_2$$

Plot of data with decision boundary:



The decision boundary appears to be in the center of the smaller group (diabetes), so not really between the groups.

3.4 Plots with thresholds 0.2 and 0.8



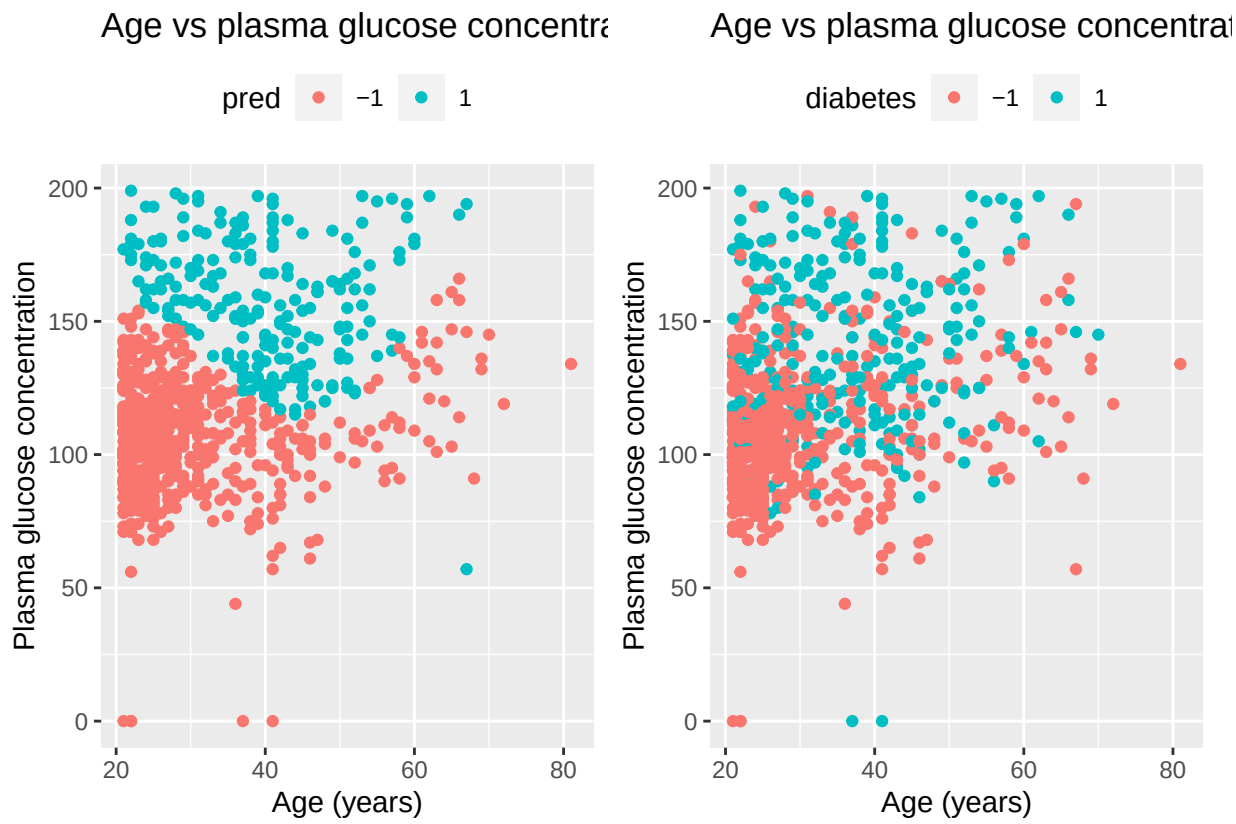

```
#> Confusion matrix r=0.2:

#>        pred
#> target  -1   1
#>     -1 238 262
#>     1   24 244

#> Confusion matrix r=0.8:

#>        pred
#> target  -1   1
#>     -1 490  10
#>     1  232  36

#> Missclassification rate r=0.2:

#> [1] 37.24

#> Missclassification rate r=0.8:

#> [1] 31.51
```

With r = 0.2 compared to using r = 0.5, more observations are being classified as diabetes and the misclassification error for the diabetes group is lower, however the overall misclassification error is higher.

With r = 0.8 compared to using r = 0.5, more observations are being classified as non-diabetes, however the overall misclassification error is higher. Compared to using r = 0.2, the misclassification error is lower.

3.5 Basis function expansion

Scatter plot:



Misclassification rate and confusion matrix:

```
#> Confusion matrix:

#>       pred
#> target  -1   1
#>     -1 433  67
#>     1  121 147

#> Misclassification rate (%):

#> [1] 24.48
```

The misclassification error for the training set is now 24%, which is an improvement compared to the previous model. However, it remains to be seen how the respective models would perform on a validation data set to determine which one is better. With the new model, there is an indication as seen in the plot that individuals

of higher age but with low plasma glucose concentration would be classified as belonging to the diabetes group, which might not be a successful classification. This latter property may be possible to remove if one excludes the observations with plasma glucose = 0, if these measurements can be shown to be erroneous. Code that investigates the influence of these observations is in the the Appendix. However, it may be that even after removal of any erroneous values, the new model could prove to be too sensitive to data points that are deviating due to noise (and not that they are erroneous).

The decision boundar(ies) are no longer linear, and further there is more than one boundary.

Statement of Contribution

Assignment 1 was contributed by Alan Cacique. Assignment 2 was contributed by Tugce Izci. Assignment 3 was contributed by Kerstin Nilsson. All three assignments procedures and results were reviewed and discussed before the creation of the final report.

Appendix: All code for this report

```
knitr::opts_chunk$set(comment = "#>",
                      echo = FALSE)
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 1
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

## Import the data
#df <- read.csv(file = "optdigits.csv", header = FALSE)

file_in <- "C:/Users/kerstin/Documents/LiU/ML/Labs/Lab01/Data/optdigits.csv"
df <- read.csv(file = file_in, header = FALSE)

n <- dim(df)[1]
n1<- nrow(df) # is the same as above

## Train data 50%
set.seed(12345)
id <- sample(1:n, floor(n*0.5))# floor to take round the numbers
train <- df[id,]
## Validation data 25%
id1<- setdiff(1:n, id) # select data that were not selected on the train data
set.seed(12345)
id2<- sample(id1, floor(n*0.25))
validation <- df[id2,]

## Test data
id3<-setdiff(id1,id2)
test <- df[id3,]

# Plot distribution
nt <- nrow(train)
nv <- nrow(validation)
nte<- nrow(test)

n_data<- c(Training = nt, Validation= nv, Test =nte)
barp<- barplot(n_data, main = "Data density", xlab = "Data set", ylab = "Number of rows",
        col = c("red", "orange", "yellow"))
text(barp, 0, round(n_data, 1),cex=1,pos=3)

library(kknn)
## Fit model with test
model_1<- kknn(as.factor(V65)~.,train = train, test = test, k =30, kernel = "rectangular")
## Fit model with train
model_2<- kknn(as.factor(V65)~.,train = train, test = train,k=30 ,kernel = "rectangular")
## Predicted values test
pred_test<- model_1$fitted.values
## Predicted values train
pred_train <- model_2$fitted.values
## Confusion matrix
```

```
confusion_matrix_Test <-table(test[,65],pred_test) ## Test
confusion_matrix_Train <-table(train[,65],pred_train) ## Train
confusion_matrix_Test
confusion_matrix_Train
# Calculate the Misclasification rate by a different method Ignore
#m_test <- mean(test[,65] != pred_test)
#m_train <- mean(train[,65] != pred_train)

# cat("Missclasification rate is the sum of False positives + False negatives / Total predictions","\n"
m_test1 <- (sum(test[,65] != pred_test)/length(test[,65]))*100
m_train1 <- (sum(train[,65]!= pred_train)/length(train[,65]))*100
# m_test1
# m_train1
err_train_by_class_perc <- round(100*((rowSums(confusion_matrix_Train)-diag(confusion_matrix_Train))/
                                      rowSums(confusion_matrix_Train)),2)

names(err_train_by_class_perc) <- 0:9

err_test_by_class_perc <- round(100*((rowSums(confusion_matrix_Test)-diag(confusion_matrix_Test))/
                                      rowSums(confusion_matrix_Test)),2)

names(err_test_by_class_perc) <- 0:9
cat("Misclasification rate per class - Train", "\n")
err_train_by_class_perc
cat("Misclasification rate per class- Test","\n")
err_test_by_class_perc

#{r,fig, fig.height = 3, fig.width = 3 ,echo=FALSE}
## Take the rows which target is the number 8
target_train8 <- which(as.factor(train[,65])== 8)
target_test8 <- which(as.factor(test[,65]) == 8)

## Take the probability values of the prediction model for the number 8

prob8_train <- model_2$prob[target_train8,9]#Train
prob8_test<- model_1$prob[target_test8,9] #Test
#We order the indexes we obtain above to take the two lowest and highest
o<-order(prob8_train)
## Train prob
## Hardest
# cat("Hardest","\n")
# prob8_train[50]
# prob8_train[43]
# prob8_train[136]
# # Easiest
# cat("Easiest","\n")
# prob8_train[179]
# prob8_train[183]

## Taking the row of the subset of 8's matrix which probability of being an 8
## is lowest and highest
low1 <- train[target_train8[50],]
low2 <- train[target_train8[43],]
```

```r
low3 <- train[target_train8[136],]
high1 <- train[target_train8[179],]
high2 <- train[target_train8[183],]

## Reshaping the rows to matrix and taking the last column which is the target or
# nlow1 <- matrix(as.numeric(low1[-65]), nrow = 8, ncol = 8, byrow = TRUE)
# nlow2 <- matrix(as.numeric(low2[-65]), nrow = 8, ncol = 8, byrow = TRUE)
# nlow3 <- matrix(as.numeric(low3[-65]), nrow = 8, ncol = 8, byrow = TRUE)
# nhigh1 <- matrix(as.numeric(high1[-65]), nrow = 8, ncol = 8, byrow = TRUE)
# nhigh2 <- matrix(as.numeric(high2[-65]), nrow = 8, ncol = 8, byrow = TRUE)

##We could have used matrix and then convert the matrix to the transpose matrix
## Or as we did it, we just set to TRUE the byrow parameter.
## We do this to have the correct orientation on the heatmap(), otherwise we would
## see an infinite shape instead of the 8's.

nlow1 <- matrix(as.numeric(low1[-65]), nrow = 8, ncol = 8)
nlow2 <- matrix(as.numeric(low2[-65]), nrow = 8, ncol = 8)
nlow3 <- matrix(as.numeric(low3[-65]), nrow = 8, ncol = 8)
nhigh1 <- matrix(as.numeric(high1[-65]), nrow = 8, ncol = 8)
nhigh2 <- matrix(as.numeric(high2[-65]), nrow = 8, ncol = 8)


## Representation of the numbers 8
par(mfrow=c(2,3))
#cat("Hardest")
image(nlow1, main = "Prob 0.10")
image(nlow2, main = "Prob 0.13")
image(nlow3, main = "Prob 0.16")
#cat("Easiest")
image(nhigh1, main = "Prob 1")
image(nhigh2, main = "Prob 1")
k <- c(1:30)
Knn_models <- matrix(0, nrow = 30, ncol = 4)
colnames(Knn_models) <- c("K","Misclasification rate trainning %",
                          "Misclasification rate validation %",
                          "Misclasification rate test %" )
for (i in k) {
  # Adding values of K to matrix
  Knn_models[i,1] <- i
  # Fit knn models for training and validation data sets with different K <- 1:30
  Knn_train<- kknn(as.factor(V65)~.,train = train, test = train, k =i,
                       kernel = "rectangular")
  knn_validation <- kknn(as.factor(V65)~.,train = train, test = validation,
                         k =i, kernel = "rectangular")
  # Extracting predicted values from the fitted models
  pred_train <- Knn_train$fitted.values
  pred_validation <- knn_validation$fitted.values

  # Calculating the Misclacsifications rates
  ms_train <- (mean(train[,65] != pred_train))*100
  ms_validation <- (mean(validation[,65] != pred_validation))*100
```

```
  # Adding Classifications rates % to the matrix
  Knn_models[i,2] <- ms_train
  Knn_models[i,3] <- ms_validation
  #Knn_models[i,4] <- ms_test
}

plot(k, Knn_models[,2], xlim = rev(range(k)), ylim = c(0,6), col="red", type = "p",
     main = "Misclassification error vs K", ylab = "Error %", pch=19)
points(k,Knn_models[,3], col="blue", pch=19)
points(30,Knn_models[30,2], col = "red", pch = 19)
text(30,4,label= round(Knn_models[30,2], digits = 3))
points(30,Knn_models[30,3], col= "blue", pch = 19)
text(30,5.7,label= round(Knn_models[30,3], digits = 3))
legend('bottomleft',inset=0.05,c("Train","Validation"),lty=1,col=c("red",
                                    "blue"),title="Data")
knn_test <- kknn(as.factor(V65)~.,train = train, test = test,
                         k =4, kernel = "rectangular")
pred_test <- knn_test$fitted.values
ms_test <- (mean(test[,65] != pred_test))*100

#pred_test
options(digits=4)
ms_tests <- c(round(Knn_models[4,2], digits=2),Knn_models[4,3], Missclassification_rate_Test= round(ms_

confusion_matrix_Test4 <-table(test[,65],pred_test) ## Test

err_test_by_class_perc4 <- round(100*((rowSums(confusion_matrix_Test4)-diag(confusion_matrix_Test4))/
                                      rowSums(confusion_matrix_Test4)),2)

names(err_test_by_class_perc4) <- 0:9

ms_tests

cat("By class error rates:")
err_test_by_class_perc4
## Create a matrix to store the K and Cross entropy values
Knn_models1 <- matrix(0, nrow = 30, ncol = 2)
colnames(Knn_models1) <- c("K","CE")
## Vector to store the probability of the actual target
Individual_prob <- c()
for(i in 1:30){
  ## Store the K value
  Knn_models1[i,1]<-i
  ## Fitting the models
  knn_validation <- kknn(as.factor(V65)~.,train = train, test = validation,
                         k =i, kernel = "rectangular")
  # Probability matrix
  knn_validation$prob

   ## Taking probabilities for the value(number) that is supposed to be
  ## Y[target]-column 65 of dataset partition
  for(j in 1:nrow(knn_validation$prob)){
    ## If the Y[target] is 8, we took the prob of being 8 of the current[i] row
```

```
    Individual_prob[j] <- knn_validation$prob[j,validation[j,65]+1]
    }
  ## Cross entropy calculation/ The
  Knn_models1[i,2] <- -mean(log(Individual_prob + 1e-15))
}
k<- c(1:30)
plot(k, Knn_models1[,2],xlim = rev(range(k)),xlab = "K", ylab = "Cross entropy", type = "p", pch= 19,
     main = "Dependence of the validation error on the value of k", col= "orange")
points(6,Knn_models1[6,2], col="blue", pch=19)
text(4.7,0.15,label= "K=6")
legend('topright',inset=0.05,c("K"),lty =1,col=c(
                                    "blue"),title="Optimal K")
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 2
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

#-------------------------------------------------------------------------------#
#- Set libraries
#-------------------------------------------------------------------------------#
library(tidyverse)
library(caret)
library(dplyr)
library(ggplot2)
library(GGally)
#-------------------------------------------------------------------------------#
# 1) Read in data and divide into training, validation and test
#-------------------------------------------------------------------------------#

#file_in <- "/Users/tugceparlakgorur/Desktop/Machine Learning/LABS/parkinson"
#data_in <- read_csv("parkinsons.csv", col_names = TRUE)

file_in <- "C:/Users/kerstin/Documents/LiU/ML/Labs/Lab01/Data/parkinsons.csv"
data_in <- read_csv(file_in, col_names = TRUE)

spec(data_in)

#Exclude variables subject and test time, total_UPDRS

df <- data_in %>% dplyr::select(-`subject#`,-test_time, -total_UPDRS)

summary(df)

#Divide into training and test set
set.seed(12345)
n <- nrow(df)
id <- sample(1:n, floor(n*0.6))
train <- df[id,]
test <- df[-id,]

#Scale data
scaler <- preProcess(train)
```

```r
train_sc <- predict(scaler, train)
test_sc <- predict(scaler, test)

#-----------------------------------------------------------------------------#
# 2) Compute a linear regression model from the training data, estimate training
# and test MSE
#-----------------------------------------------------------------------------#

#- Linear model
formula <- motor_UPDRS ~ . -1
data <- train_sc
m1 <- lm(formula=formula, data=data)

#- Calculate training and test MSE
error_training <- train_sc$motor_UPDRS - predict(m1)
error_test <- test_sc$motor_UPDRS - predict(m1, test_sc)

mse_training_lm <- mean(error_training^2)
mse_test_lm <- mean(error_test^2)

rmse_training_lm <- sqrt(mse_training_lm)
rmse_test_lm <- sqrt(mse_test_lm)

mse_lm <- c(mse_training_lm=mse_training_lm, mse_test_lm=mse_test_lm)
rmse_lm <- c(rmse_training_lm=rmse_training_lm, rmse_test_lm=rmse_test_lm)

#- Print mse for training and test set
(mse_lm)


#- Print estimated coefficients along with standard errors
summary(m1)

#- Obtain order of coefficients
order_coef <- order(abs(coef(m1)), decreasing = TRUE)
(coef(m1)[order_coef])
#- Plot training data (random subset)

set.seed(12345)
#d_plot <- sample_n(train,350)
d_plot <- train
p1 <- ggpairs(dplyr::select(d_plot, starts_with("Shimmer")))
p1

#=> correlation within Jitter and Shimmer variables

p2 <- ggpairs(dplyr::select(d_plot, starts_with("Jitter")))
p2

#- One Jitter and one Shimmer variable selected due to save space
p3 <- ggpairs(dplyr::select(d_plot, age, sex, `Jitter(%)`, Shimmer, NHR:PPE))
p3
```

```r
#- Check if Shimmer:APQ3` is an exact multiple of train$`Shimmer:DDA`
r_Shimmer_APQ3_Shimmer_DDA <- train$`Shimmer:APQ3`/train$`Shimmer:DDA`
unique(r_Shimmer_APQ3_Shimmer_DDA)
#-------------------------------------------------------------------------------#
# 3) Implement 4 following functions by using basic R commands only (no
# external packages):

# a) Loglikelihood function
# b) Ridge function
# c) RidgeOpt function
# d) DF function
#-------------------------------------------------------------------------------#
#- a) log likelihood function
Loglikelihood <- function(formula, data, theta, sigma) {
  n <- nrow(data)

  X <- stats::model.matrix(formula,data)
  y <- data[,all.vars(formula)[1]]

  t1 <- (n/2)*(log(2*pi*sigma^2))
  t2 <- (1/(2*sigma^2))
  ll <- -t1 - t2*sum((t(theta)%*%t(X)-y)^2)
  return(ll)
}

#- b) ridge function
# params: numeric vector that should contain theta for the first positions,
# then sigma in the last position
Ridge <- function(params, data, lambda, formula) {

  theta <- params[-length(params)]
  sigma <- params[length(params)]

  ll_lm <- -Loglikelihood(formula=formula, data=data, theta=theta, sigma=sigma)
  lambda_p <- lambda/(2*sigma^2)
  ll_w_ridge <- ll_lm + lambda_p*sum(theta^2)
  return(ll_w_ridge)
}


#- c) ridge opt function
# params: numeric vector that should contain theta for the first positions,
# then sigma in the last position
RidgeOpt <- function(params, fn, formula, data, lambda, method) {
  res <- optim(par=params, fn=fn, formula=formula, data=data, lambda=lambda,
               method=method)
  return(res)
}

#d) DF function
DF <- function(formula, data, lambda) {

  X <- stats::model.matrix(formula,data)
```

```
  y <- data[,all.vars(formula)[1]]
  p <- dim(X)[2]

  #B_ridge = inv(XTX+lambdaI)XTy
  #y_hat = X%*%B_ridge = X*inv(XTX+lambdaI)XTy =>
  #y_hat = S(X)*Y, where S(X) = X*inv(XTX+lambdaI)XT
  S <- X%*%solve(t(X)%*%X+lambda*diag(p))%*%t(X)

  S_trace <- sum(diag(S))

  return(S_trace)
}

#-------------------------------------------------------------------------------#
# 4) Compute optimal theta parameters för lambda=1, 10 and 1000 respectively.
# Use the estimated parameters to predict the motor_UPDRS values for training
# and test data and report the training and test MSE values.
#-------------------------------------------------------------------------------#
method <- "BFGS"
fn <- Ridge
theta_start <- numeric(dim(train_sc)[2]-1)
names(theta_start) <- names(dplyr::select(data,-motor_UPDRS))
sigma_start <- 1

params <- c(theta=theta_start, sigma=sigma_start)

#- lambda = 1
lambda <- 1
res_optim_l_1 <- RidgeOpt(params, fn, formula, data, lambda, method)

res_optim_l_1$par

#- lambda = 100
lambda <- 100
res_optim_l_100 <- RidgeOpt(params, fn, formula, data, lambda, method)

res_optim_l_100$par

#- lambda = 1000
lambda <- 1000
res_optim_l_1000 <- RidgeOpt(params, fn, formula, data, lambda, method)

res_optim_l_1000$par

#- Calculate errors and MSE
train_sc_X <- as.matrix(dplyr::select(train_sc,- motor_UPDRS))
test_sc_X <- as.matrix(dplyr::select(test_sc,- motor_UPDRS))

#- Lambda = 1 error calc
theta_l_1 <- as.matrix(res_optim_l_1$par[-length(res_optim_l_1$par)])

y_pred_train_l_1 <- t(theta_l_1)%*%t(train_sc_X)
error_train_l_1 <- train_sc$motor_UPDRS-y_pred_train_l_1
```

```
y_pred_test_l_1 <- t(theta_l_1)%*%t(test_sc_X)
error_test_l_1 <- test_sc$motor_UPDRS-y_pred_test_l_1

#- Lambda = 100 error calc
theta_l_100 <- as.matrix(res_optim_l_100$par[-length(res_optim_l_100$par)])

y_pred_train_l_100 <- t(theta_l_100)%*%t(train_sc_X)
error_train_l_100 <- train_sc$motor_UPDRS-y_pred_train_l_100
y_pred_test_l_100 <- t(theta_l_100)%*%t(test_sc_X)
error_test_l_100 <- test_sc$motor_UPDRS-y_pred_test_l_100

#- Lambda = 1000 error calc
theta_l_1000 <- as.matrix(res_optim_l_1000$par[-length(res_optim_l_1000$par)])

y_pred_train_l_1000 <- t(theta_l_1000)%*%t(train_sc_X)
error_train_l_1000 <- train_sc$motor_UPDRS-y_pred_train_l_1000

y_pred_test_l_1000 <- t(theta_l_1000)%*%t(test_sc_X)
error_test_l_1000 <- test_sc$motor_UPDRS-y_pred_test_l_1000

#- MSE calc
# Lambda = 1
mse_train_l_1 <- mean(error_train_l_1^2)
mse_test_l_1 <- mean(error_test_l_1^2)
# Lambda = 100
mse_train_l_100 <- mean(error_train_l_100^2)
mse_test_l_100 <- mean(error_test_l_100^2)
# Lambda = 1000
mse_train_l_1000 <- mean(error_train_l_1000^2)
mse_test_l_1000 <- mean(error_test_l_1000^2)

lambda_v <- c(1L,100L,1000L)
mse_train_ridge <- c(mse_train_l_1,mse_train_l_100,mse_train_l_1000)
mse_test_ridge <- c(mse_test_l_1,mse_test_l_100,mse_test_l_1000)

#- Calculate DF for all lambda
lambda <- 1
trace_l_1 <- DF(formula=formula, data=data, lambda=lambda)

lambda <- 100
trace_l_100 <- DF(formula=formula, data=data, lambda=lambda)

lambda <- 1000
trace_l_1000 <- DF(formula=formula, data=data, lambda=lambda)

trace <- round(c(trace_l_1,trace_l_100,trace_l_1000),1)
mse_df_ridge <- cbind(lambda=lambda_v,train=mse_train_ridge, test=mse_test_ridge,
                      df=trace)
colnames(mse_df_ridge) <- c("$\\lambda$", "mse train", "mse test", "df")

knitr::kable(
  mse_df_ridge,
  caption = "Degrees of freedom (df) and MSE for training and test set for different $\\lambda$."
```

```
)


#- Mean center y
#train_sc_X <- as.matrix(select(train_sc,- motor_UPDRS))
#test_sc_X <- as.matrix(select(test_sc,- motor_UPDRS))

train_y <- dplyr::select(train,motor_UPDRS)
test_y <- dplyr::select(test,motor_UPDRS)

scaler_y <- preProcess(train_y,method = c("center"))

train_sc_y <- predict(scaler_y, train_y)
test_sc_y <- predict(scaler_y, test_y)

train_sc_mcy <- as.data.frame(cbind(train_sc_X,train_sc_y))
test_sc_mcy <- as.data.frame(cbind(test_sc_X,test_sc_y))

#- Estimate ridge model

#- Calculate training and test MSE
method <- "BFGS"
fn <- Ridge
data <- train_sc_mcy
formula <- motor_UPDRS ~ . -1
lambda = 100
theta_start <- numeric(dim(train_sc)[2]-1)
names(theta_start) <- names(dplyr::select(data,-motor_UPDRS))
sigma_start <- 7

params <- c(theta=theta_start, sigma=sigma_start)

res_optim_l_100_mcy <- RidgeOpt(params, fn, formula, data, lambda, method)

#res_optim_l_100_mcy$par
#res_optim_l_100$par

#- Lambda = 100 error calc
theta_l_100_mcy <- as.matrix(res_optim_l_100_mcy$par[-length(res_optim_l_100_mcy$par)])

y_pred_train_l_100_mcy <- t(theta_l_100_mcy)%*%t(train_sc_X)
error_train_l_100_mcy <- train_sc_mcy$motor_UPDRS-y_pred_train_l_100_mcy
y_pred_test_l_100_mcy <- t(theta_l_100_mcy)%*%t(test_sc_X)
error_test_l_100_mcy <- test_sc_mcy$motor_UPDRS-y_pred_test_l_100_mcy

# Lambda = 100
cat(paste0("RMSE for training and test data:\n"))
(rmse_train_l_100 <- sqrt(mean(error_train_l_100_mcy^2)))
(rmse_test_l_100 <- sqrt(mean(error_test_l_100_mcy^2)))

cat(paste0("Mean of motor_UPDRS in the test data: ", round(mean(test$motor_UPDRS),2)))

rmse_rel_to_mean_l_100 <- round(100*(rmse_test_l_100/mean(test$motor_UPDRS)),1)
```

```
cat(paste0("RMSE as a percentage of the mean: ", rmse_rel_to_mean_l_100))
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 3
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

#-------------------------------------------------------------------------------#
#- Set libraries
#-------------------------------------------------------------------------------#
library(tidyverse)
library(caret)
library(ggplot2)
library(patchwork)
library(scales)
library(tinytex)
#-------------------------------------------------------------------------------#
#- 0) Read in data and divide into training, validation and test
#-------------------------------------------------------------------------------#

file_in <- "C:/Users/kerstin/Documents/LiU/ML/Labs/Lab01/Data/pima-indians-diabetes.csv"
data_in <- read_csv(file_in, col_names = FALSE)

spec(data_in)

names(data_in) <- c(
  "pregnant_num",
  "glucose_pl",
  "bp_dia",
  "triceps_skin_thick",
  "insulin_serum",
  "bmi",
  "diabetes_pedigree",
  "age",
  "diabetes")

#- Set factors
data_1 <- mutate(data_in,diabetes=ifelse(diabetes<1,-1,1),
                 diabetes=as.factor(diabetes))

data <- data_1

#-------------------------------------------------------------------------------#
#- 1) Make a plot of age vs plasma glucose concentration
#-------------------------------------------------------------------------------#
ylim_min <- min(data$glucose_pl)
ylim_max <- max(data$glucose_pl)

xlim_min <- min(data$age)
xlim_max <- max(data$age)

#- Obtain colour codes for plot
hex <- hue_pal()(2)
```

```r
p1 <- data %>% ggplot() + aes(x=age,y=glucose_pl, colour=diabetes)+ geom_point(shape=21, fill = NA) +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + xlim(xlim_min, xlim_max)

p2 <- data %>% filter(diabetes==-1) %>% ggplot() + aes(x=age,y=glucose_pl, colour=diabetes)+
  geom_point(colour= "#F8766D") +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + xlim(xlim_min, xlim_max)

p3 <- data %>% filter(diabetes==1) %>% ggplot() + aes(x=age,y=glucose_pl, colour=diabetes)+
  geom_point(colour= "#00BFC4") +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + xlim(xlim_min, xlim_max)

(p2 + p3)/p1

#- There are observations with glucose levels = 0, identify those
order_glucose <- order(data$glucose_pl)
data$glucose_pl[order_glucose[1:10]]
obs_with_glucose_eq_0 <- which(data$glucose_pl==0)

#-------------------------------------------------------------------------------#
#- 2) Train a logistic regression model with y = Diabetes as target and
# 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
#-------------------------------------------------------------------------------#

train <- data%>%select(glucose_pl,age,diabetes)

m1 <- glm(diabetes~., train, family = "binomial")

#- Regression coefficients:
(coef(m1))


#- Model with observations with plasma glucose excluded
train_b <- dplyr::filter(train, glucose_pl > 0)

m1_b <- glm(diabetes~., train_b, family = "binomial")

#- Regression coefficients:
(coef(m1_b))

#- Training misclassification error (percent) and confusion matrix
prob <- predict(m1, type="response")
pred <- ifelse(prob>0.5, 1, -1)

cat("Confusion matrix:")
(cm_train <- table(train$diabetes, pred, dnn=c("target","pred")))
```

```r
cat("Misclassification error:")
(err_train_perc <- round(100*((dim(train)[1]-sum(diag(cm_train)))/dim(train)[1]),2))

#cat("Number of observations in the different groups:")
no_obs_grp <- table(train$diabetes)

no_obs_grp_b <- table(train_b$diabetes)

cat("Percentage of observations that is non-diabetes:")
(round(100*unname(no_obs_grp[1]) /sum(no_obs_grp),2))

#- Model excluding obs with glucose = 0
prob_b <- predict(m1_b, type="response")
pred_b <- ifelse(prob_b>0.5, 1, -1)

(cm_train_b <- table(train_b$diabetes, pred_b, dnn=c("target","pred")))

(err_train_perc_b <- round(100*((dim(train_b)[1]-sum(diag(cm_train_b)))/dim(train_b)[1]),2))


#- Plot of age vs glucose plasma concentration, colour by predicted values:
data_plot <- add_column(data,pred)

data_plot <- mutate(data_plot,pred=as.factor(pred))

ylim_min <- min(data_plot$glucose_pl)
ylim_max <- max(data_plot$glucose_pl)

p1 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=pred)+ geom_point() +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max)

p1

#-------------------------------------------------------------------------------#
# 3) Use the model estimated in 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
#-------------------------------------------------------------------------------#

#- Calculate intercept and slope for the decision boundary
theta_0 <- coef(m1)[1]
theta_v <- as.matrix(coef(m1)[-1])

intercept <- -theta_0/theta_v[1]
slope <- -theta_v[2]/theta_v[1]

ylim_min <- min(data_plot$glucose_pl)
ylim_max <- max(data_plot$glucose_pl)

#- Plot decision boundary  ----#
```

```r
#- Colour by pred
p1 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=pred)+ geom_point() +
  geom_abline(slope=slope, intercept = intercept, show.legend = TRUE) +
  labs(title="Age vs plasma glucose conc",
       y="Plasma glucose concentration", x="Age (years)", caption =
         "Decision boundary indicated by line") +
  ylim(ylim_min, ylim_max) + theme(legend.position="top")

#- Colour by target
p2 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=diabetes)+ geom_point() +
  geom_abline(slope=slope, intercept = intercept, show.legend = TRUE) +
  labs(title="Age vs plasma glucose conc",
       y="Plasma glucose concentration", x="Age (years)", caption =
         "Decision boundary indicated by line") +
  ylim(ylim_min, ylim_max) + theme(legend.position="top")

(p1+p2)
#-------------------------------------------------------------------------------#
# 4) Make same kind of plots as in 2) but use thresholds r = 0.2 and r = 0.8.
#-------------------------------------------------------------------------------#
#- r: 0.2
pred <- ifelse(prob>0.2, 1, -1)

#- Calculate confusion matrix and misclassification error
cm_train_0_2 <- table(train$diabetes, pred, dnn=c("target","pred"))
err_train_perc_0_2 <- round(100*((dim(train)[1]-sum(diag(cm_train_0_2)))/dim(train)[1]),2)

#- Make plot
data_plot <- add_column(data,pred)

data_plot <- mutate(data_plot,pred=as.factor(pred))

ylim_min <- min(data_plot$glucose_pl)
ylim_max <- max(data_plot$glucose_pl)

p1_0_02 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=pred)+ geom_point() +
  labs(title="Age vs glucose, r=0.2",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + theme(legend.position="top")

#- r: 0.8
pred=ifelse(prob>0.8, 1, -1)

#- Calculate confusion matrix and misclassification error
cm_train_0_8 <- table(train$diabetes, pred, dnn=c("target","pred"))
err_train_perc_0_8 <- round(100*((dim(train)[1]-sum(diag(cm_train_0_8)))/dim(train)[1]),2)

#- Make plot
data_plot <- add_column(data,pred)

data_plot <- mutate(data_plot,pred=as.factor(pred))

ylim_min <- min(data_plot$glucose_pl)
```

```r
ylim_max <- max(data_plot$glucose_pl)

p1_0_08 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=pred)+ geom_point() +
  labs(title="Age vs glucose, r=0.8",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + theme(legend.position="top")

(p1_0_02 + p1_0_08)

#- Print confusion matrices and misclassification errors
cat("Confusion matrix r=0.2:")
(cm_train_0_2)
cat("Confusion matrix r=0.8:")
(cm_train_0_8)

cat("Missclassification rate r=0.2:")
(err_train_perc_0_2)
cat("Missclassification rate r=0.8:")
(err_train_perc_0_8)
#-------------------------------------------------------------------------------#
# 5) Perform a basis function expansion trick by computing new features
# z1 = x1^4, z2 = x1^3*x2, z3= x1^2*x2^2, z4 = x1*x2^3, z5=x2^4, adding them
# to the data set and then computing a logistic regression model with y as target
# and x1,x2,z1, ... z5 as features. Create a scatterplot of the same kind as in
# 2) for this model and compute the training misclassification rate.
#-------------------------------------------------------------------------------#

#- Compute new variables and train model
data_2 <- data %>% mutate(z1=glucose_pl^4,
                          z2=glucose_pl^3*age,
                          z3=glucose_pl^2*age^2,
                          z4=glucose_pl*age^3,
                          z5=age^4)


train <- data_2 %>%select(glucose_pl,age,diabetes,z1:z5)

m2 <- glm(diabetes~., train, family = "binomial")
prob <- predict(m2, type="response")
pred <- ifelse(prob>0.5, 1, -1)

#- Model without observations with glucose = 0
data_2_b <- filter(data_2, glucose_pl> 0)
train_b <- data_2_b %>%select(glucose_pl,age,diabetes,z1:z5)

m2_b <- glm(diabetes~., train_b, family = "binomial")
prob_b <- predict(m2_b, type="response")
pred_b <- ifelse(prob_b>0.5, 1, -1)

#- Plot of data

data_plot <- add_column(data,pred)
```

```r
data_plot <- mutate(data_plot,pred=as.factor(pred))

ylim_min <- min(data_plot$glucose_pl)
ylim_max <- max(data_plot$glucose_pl)

p1 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=pred)+ geom_point() +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + theme(legend.position="top")

p2 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=diabetes)+ geom_point() +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max)+ theme(legend.position="top")

(p1 + p2)


#- Print confusion matrix and misclassification rate
cat("Confusion matrix:")
(cm_train_2 <- table(train$diabetes, pred, dnn=c("target","pred")))

cat("Misclassification rate (%):")
(err_train_perc_2 <- round(100*((dim(train)[1]-sum(diag(cm_train_2)))/dim(train)[1]),2))
#- Model without observations with plasma glucose = 0 ------------#
#- Plot of data

data_b <- filter(data, glucose_pl > 0)
data_plot <- add_column(data_b,pred_b)

data_plot <- mutate(data_plot,pred=as.factor(pred_b))

ylim_min <- min(data_plot$glucose_pl)
ylim_max <- max(data_plot$glucose_pl)

p1 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=pred)+ geom_point() +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max) + theme(legend.position="top")

p2 <- data_plot %>% ggplot() + aes(x=age,y=glucose_pl, colour=diabetes)+ geom_point() +
  labs(title="Age vs plasma glucose concentration",
       y="Plasma glucose concentration", x="Age (years)") +
  ylim(ylim_min, ylim_max)+ theme(legend.position="top")

(p1 + p2)

#- Print confusion matrix and misclassification rate

cat("Confusion matrix:")
(cm_train_2_b <- table(train_b$diabetes, pred_b, dnn=c("target","pred")))

cat("Misclassification rate (%):")
```

```
(err_train_perc_2_b <- round(100*((dim(train_b)[1]-sum(diag(cm_train_2_b)))/dim(train_b)[1]),2))
```