Machine Learning Lab 1

Theodor Emanuelsson (theem089), Lisa Goldschmidtböing (lisgo269), Christoforos Spyretos (chrsp415)

date: 17-11-21

Assignment 1

Task 1

Import the data into R and divide into training, validation and testing sets.

First, we import the data from the .csv file. Next, we reorder the data frame to convert the first column's response variable to a factor. We then follow the partitioning principle specified in the lecture slides: 50% training data, 25% validation data and 25% testing data.

```
## 1.1
df <- read.csv("optdigits.csv", header = FALSE)
df <- df[,c(65, 1:64)] # Reorder to make formula pick first column
names(df)[1] <- "digit"
df$digit <- factor(df$digit)
n <- nrow(df)
set.seed(12345)
id_train <- sample(1:n, floor(n*0.5))
df_train <- df[id_train,] # Training data
id_rest <- setdiff(1:n, id_train)
set.seed(12345)
id_vali <- sample(id_rest, floor(n*0.25))
df_vali <- df[id_vali,] # Validation data
id_test <- setdiff(id_rest, id_vali)
df_test <- df[id_test,] # Test data
```

Task 2

Use training data to fit 30-nearest neighbor classifier

We define the model formula automatically using `formula()` so that the first column is defined as the response variable and the rest is defined as the predictor variables. We then use the training data to fit the 30-nearest neighbour classifier using the `kknn()` function and set parameter `kernel = "rectangular"`.

```
##1.2
library(kknn)
form <- formula(df_train) #Automatic formula
fit_train <- kknn(form, train = df_train, test = df_train, k = 30, kernel = "rectangular")
fit_test <- kknn(form, train = df_train, test = df_test, k = 30, kernel = "rectangular")
```

We also define a function to calculate the misclassification rate.

```
# function to calculate misclassification taken from lecture slides
missclass <- function(X,X1){
```

```
  return(1-sum(diag(table(X,X1)))/length(X))
}
```

Using the classifier, we calculate the predicted classifications for the training and testing data.

```
fit_val_train <- fitted(fit_train)
fit_val_test <- fitted(fit_test)
```

Estimate confusion matrices for the training and test data

The rows are the actual labels in the following confusion matrices, and the columns are the predicted labels.

```
conf_mat_train <- table(df_train$digit,fit_val_train)
knitr::kable(conf_mat_train, caption = "Confusion matrix: training data")
```

Table 1: Confusion matrix: training data

	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

```
conf_mat_test <- table(df_test$digit,fit_val_test)
knitr::kable(conf_mat_test, caption = "Confusion matrix: test data")
```

Table 2: Confusion matrix: test data

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

Misclassification errors for the training and test data

```
misclass_train <- missclass(df_train$digit, fit_val_train) #0.045
paste("Misclassification error for training data:", round(misclass_train, 5))
```

```
## [1] "Misclassification error for training data: 0.045"
```

```r
misclass_test <- missclass(df_test$digit, fit_val_test) #0.053
paste("Misclassification error for test data:", round(misclass_test, 5))
```

```
## [1] "Misclassification error for test data: 0.05329"
```

Comment on the quality of predictions for different digits and on the overall prediction quality

The model is best at classifying the digit 0 with only one single misclassification in the testing data. The model also predicts the digit 6 well with only 4 misclassifications in the testing data. However, the classifier specifically struggles with the digits 8 and 9. Interestingly, the classifier has issues differentiating between a true labelled 8 and a 1, predicting the 8 to be a 1 on 7 pictures in the testing data. The same problem can be seen for predictions in the training data as well. Overall, the model performance is acceptable in novel applications, but it is most likely possible to achieve better results using another model. The evaluation of a model's performance should be more determined by its application than a specific rate. If it were to be put into production, a misclassification rate of about 5% would not be good enough.

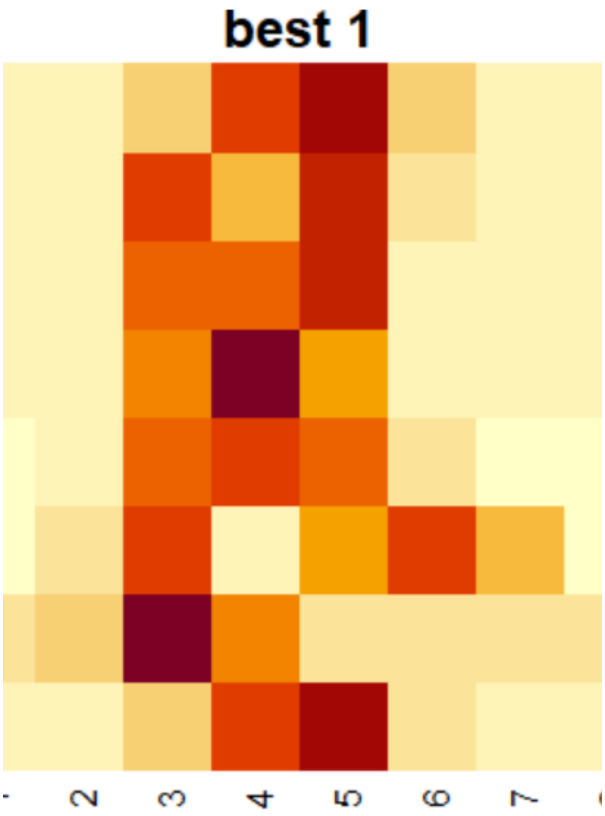
Task 3

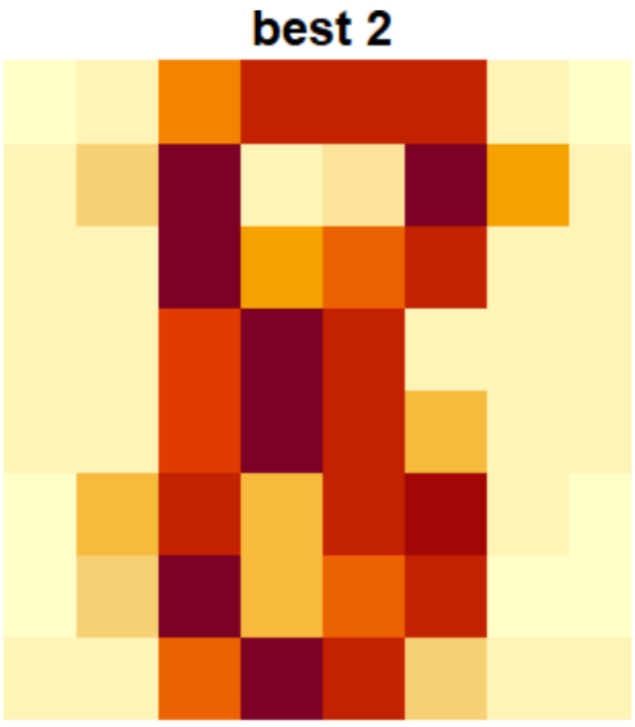
Find any 2 cases of the digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify

```r
## 1.3
library(dplyr)

probs <- df_train %>%
  mutate(prob8 = fit_train$prob[,"8"]) %>%
  filter(digit == 8)

worst <- probs %>% slice_min(prob8, n = 3)
best <- probs %>% slice_max(prob8, n = 40) # 40 digits with prob = 1
set.seed(12345)
randm <- sample(1:nrow(best), 2)
best <- best[randm,]

make_8x8 <- function(inputrow){
  return(matrix(as.numeric(inputrow[2:65]), nrow = 8, ncol = 8, byrow = T))
}

best1 <- make_8x8(best[1,])
best2 <- make_8x8(best[2,])
worst1 <- make_8x8(worst[1,])
worst2 <- make_8x8(worst[2,])
worst3 <- make_8x8(worst[3,])
```

The two heatmaps below are two of the observations that the model find easiest to classify. The probability that these are "8" according to the model is 1. These are two out of 40 observations that the classifier predicts are "8" with a probability of 1. Visually these cases are relatively easy to identify as "8" as we see the lighter pixels between circles of darker pixels.

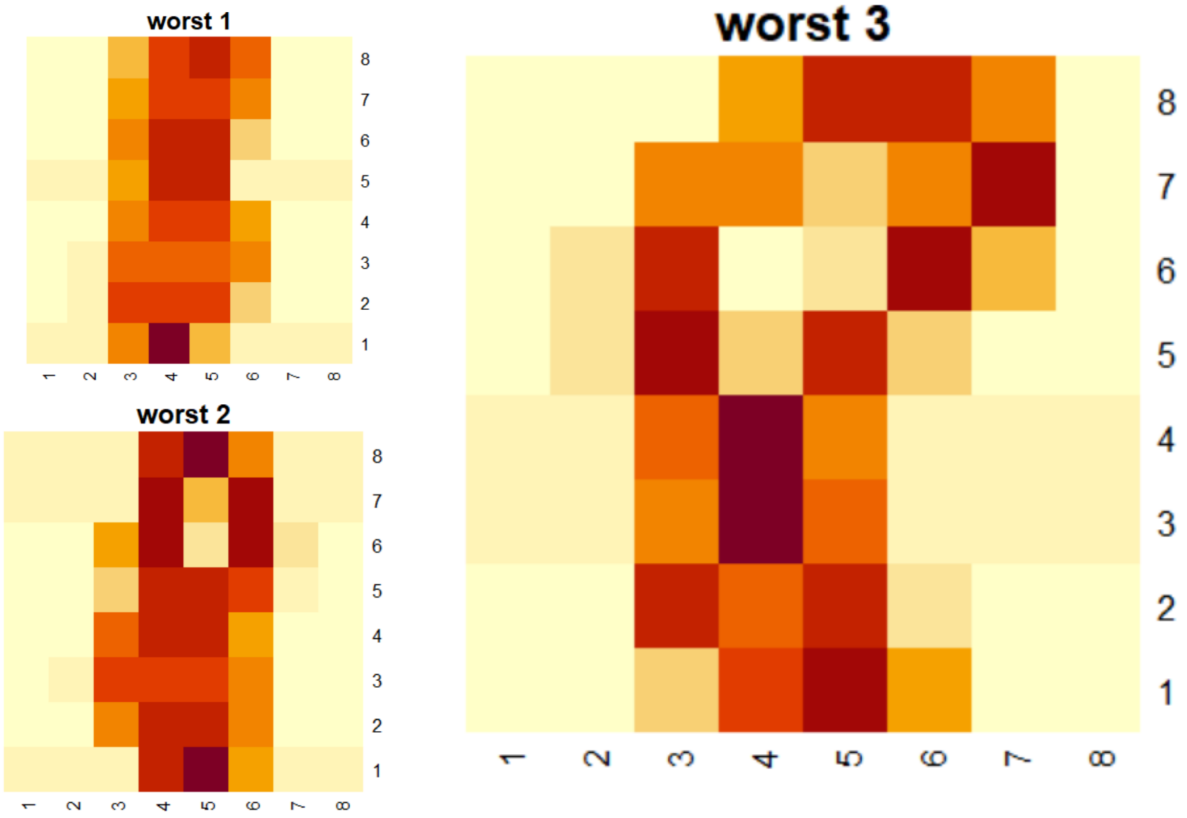
```r
#we do not evaluate these chunks when knitting the document as the
#plots take quite a lot of space and we decided to include them as
#pictures to make it look nicer (same with the next chunk)
p1 <- heatmap(best1, Colv=NA, Rowv=NA, main = "best 1")
p2 <- heatmap(best2, Colv=NA, Rowv=NA, main = "best 2")
```





The three heatmaps below are the three observations that the model found the most difficult to classify to "8". The predicted probabilities range from approximately 0.1 to $\frac{1}{6}$. These cases are hard to identify as "8" as it is impossible to identify any light pixel in either of the two circles that make up the digit 8.

```
p3 <- heatmap(worst1, Colv=NA, Rowv=NA, main = "worst 1")
p4 <- heatmap(worst2, Colv=NA, Rowv=NA, main = "worst 2")
p5 <- heatmap(worst3, Colv=NA, Rowv=NA, main = "worst 3")
```



Task 4

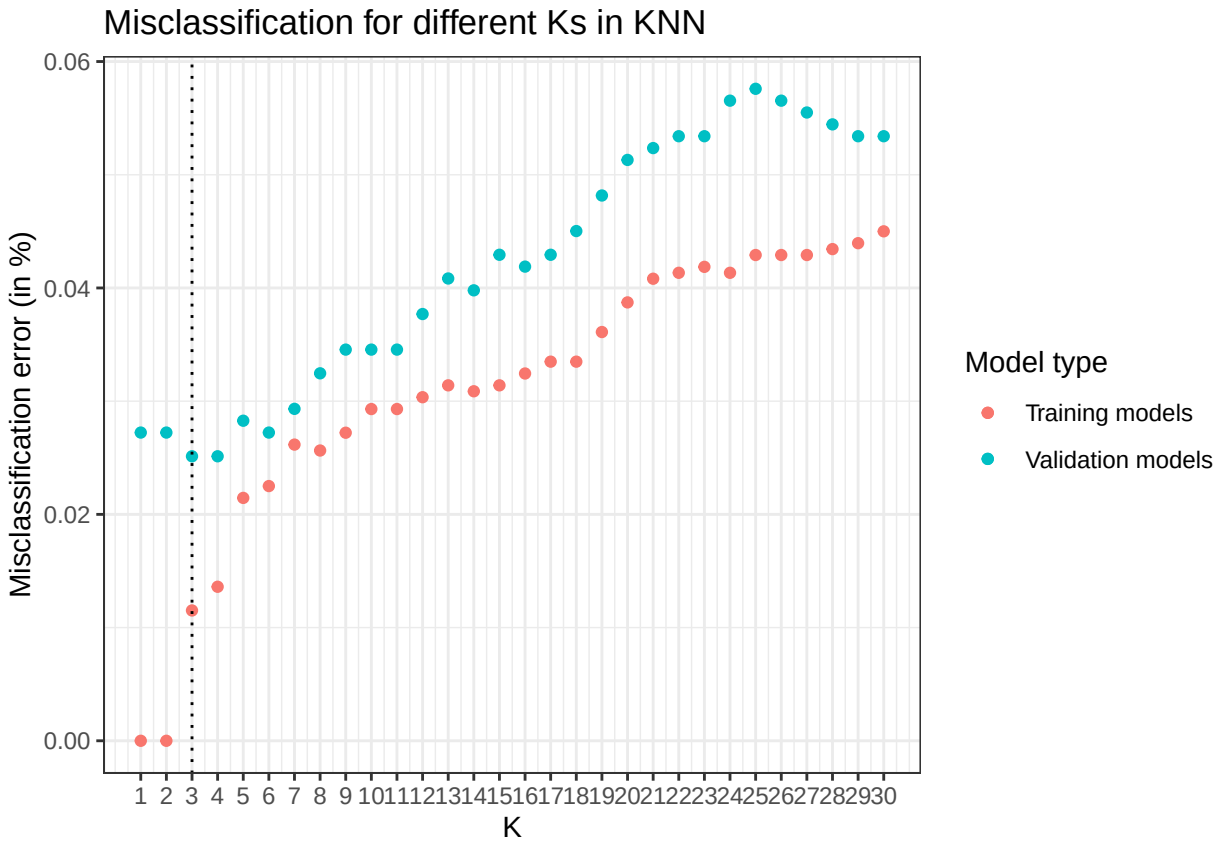
Fit K-nearest neighbor classifiers to the training data for different values of K = 1,2, ... , 30

```
## 1.4
models_missclass <- function(K){
  out <- data.frame(K = numeric(), misclass_train = numeric(), misclass_vali = numeric())
  for (i in 1:K) {
    set.seed(12345)
  fit_training <- kknn(form, train = df_train, test = df_train,
                       k = i, kernel = "rectangular")
  fit_validation <- kknn(form, train = df_train, test = df_vali,
                         k = i, kernel = "rectangular")

  misclass_training <- missclass(df_train$digit, fitted(fit_training))
  misclass_vali <- missclass(df_vali$digit, fitted(fit_validation))

  out[i, ] <- c(i, misclass_training, misclass_vali)
  }
  return(out)
}
misclassifications <- models_missclass(30)
```

Plot the dependence of the training and validation misclassification errors on the value of K

```
library(ggplot2)

ggplot(data = misclassifications) +
```

```
  geom_point(aes(x = K, y = misclass_vali, color = "Validation models")) +
  geom_point(aes(x = K, y = misclass_train, color = "Training models")) +
  labs(title = "Misclassification for different Ks in KNN",
       y = "Misclassification error (in %)",
       colour = "Model type") +
  theme_bw() + scale_x_continuous(breaks = 1:30) +
  geom_vline(xintercept = which.min(misclassifications$misclass_vali), linetype ="dotted")
```



We can see that as K increases both the training and validation models, misclassification errors tend to increase. When K = 1 or K = 2, there is a zero misclassification error for the training data, yet the model generalizes worst to the testing data. After K = 4 we see that training and validation errors become closer to each other. Interestingly as the model becomes more complex, the classifier tends to get worse. An optimal K seem to be at either K = 3 or K = 4 because these models have the lowest misclassification error on the validation data and are relatively simple. We choose to continue with K = 3 since it is an odd number. When K = 4, ties are possible, so if two of the nearest neighbours are one class and two are another, there is an issue that is solved randomly, while with K = 3, there is always a clear decision.

Estimate the test error for the model with optimal K and compare it with the training and validation errors

For the KNN classifier, when $K = 3$, the model has a low misclassification rate for the training data. The classification rate increased by approximately 1% when comparing the model on the training data versus the validation and testing data. The error for the test set is low and even slightly less than for the validation set, indicating that the model generalizes well to new data.

```
fit_train_K3 <- kknn(form, train = df_train, test = df_test, k = 3, kernel = "rectangular")
test_error <- missclass(df_test$digit, fitted(fit_train_K3))
errors_df <- data.frame(TrainingError = misclassifications$misclass_train[3],
                        ValidationError = misclassifications$misclass_vali[3],
```

```
                          TestingError = test_error)
row.names(errors_df) <- "Misclassification error (in %)"

errors_df
```

```
##                                TrainingError ValidationError TestingError
## Misclassification error (in %)     0.0115123      0.02513089   0.02403344
```

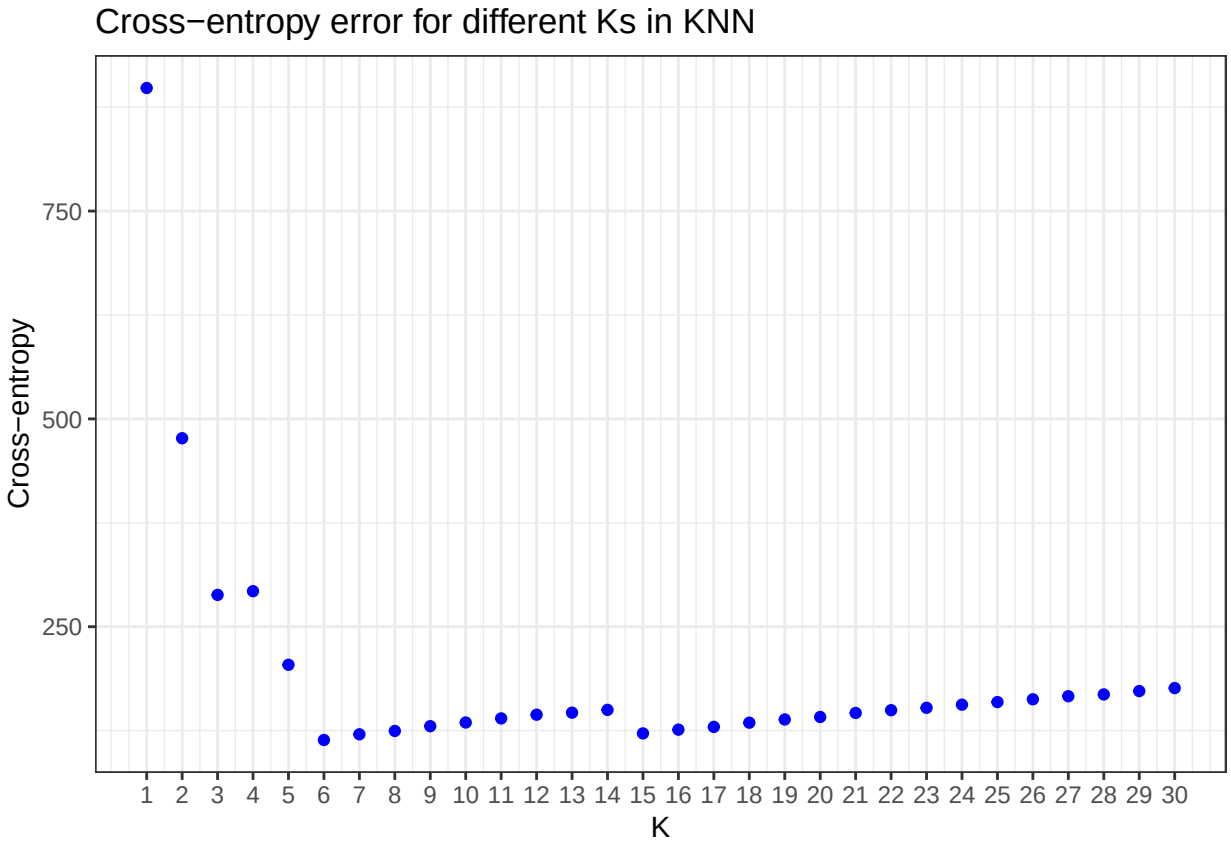
Task 5

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

```
## 1.5
# Since only one digit is possible response variable we do
# not need to compute the -sum(log(p)*d) but can simply take the -log
# of the probability for the true digit. Result will be the same.
cross_entropy <- list()
for (K in 1:30) {
  fit = kknn(form, df_train, df_vali, k=K, kernel="rectangular")

  df_probs <- data.frame(fit$prob)
  df_probs$digit <- df_vali$digit

  entropy <- list()
  for (i in 1:nrow(df_probs)) {
    for (n in 1:10) {
      if (df_probs$digit[i] == n-1) {
        entropy[[i]] = -(log(df_probs[i, n]+ 1e-15))
      }
    }
  }
  cross_entropy[[K]] <- sum(unlist(entropy))
}
cross_entropy_valid <- data.frame(cross_entropy = unlist(cross_entropy), K = 1:30)
```

Plot dependence of the validation cross-entropy error on the value of K

```
ggplot(cross_entropy_valid, aes(x = K, y = cross_entropy)) +
  geom_point(col = "blue") + theme_bw() +
  labs(title = "Cross-entropy error for different Ks in KNN", y = "Cross-entropy")+
  scale_x_continuous(breaks = 1:30)
```



Using cross-entropy to evaluate which K is the best takes a more probability-based perspective than using the misclassification error. In cross-entropy, the predicted probabilities for the true label is what is relevant. Suppose the model predicts that the label is the actual label by a high probability. In that case, the cross-entropy is low, and vice versa if the probability for the true label is low, then the cross-entropy is high. Using this as a loss function would punish uncertain decisions. The model with the lowest cross-entropy is when $K = 6$. We can see here that the models with lower K, for example, $K = 3$, have higher cross-entropy. Meaning that even if there is a slightly lower misclassification rate when $K = 3$, the model estimates the probability for the truly labelled digit lower than a model with $K = 6$ does. It is reasonable to prefer $K = 6$ since the misclassification is slightly higher and has a lot lower cross-entropy.

Assignment 2

Reading & Preparing Data

We start with reading the data set and removing the variables that are not needed.

```
## 2.1
data <- read.csv("parkinsons.csv")

data <- data[,-c(1:4,6)]  # deleting undesirable variables
```

Task 1 (Scaling & splitting data into train & test set.)

We scale and split the data into 60% training data and 40% testing data. We standardise the data for consistency to have the same content and format. In addition, standardised values help track data that is not easy to compare otherwise. We calculate the mean and standard deviation of the variables in the training data and apply the same transformation to the testing data.

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data.frame(data[id,])
test=data.frame(data[-id,])

mean_train <- numeric(ncol(train))
sd_train <- numeric(ncol(train))

for(i in 1:ncol(train)){
  mean_train[i] <- mean(train[,i])
  sd_train[i] <- sd(train[,i])
}

train <- data.frame(scale(train))

test_new <- data.frame(matrix(nrow = nrow(test), ncol = ncol(test)))

for(i in 1:ncol(train)){
  test_new[,i] <- (test[,i] - mean_train[i])/sd_train[i]
}

test <- as.data.frame(test_new)
colnames(test) <- colnames(train)
```

Task 2 (Linear regression model & estimation of train and test MSE.)

Fitting linear regression model.

The variables that contributed to the model are Jitter (Abs), Shimmer: APQ5, Shimmer: APQ11, NHR, HNR, DFA and PPE with a statistical significance of 0.001, and Shimmer and Shimmer: APQ5 with a statistical significance of 0.01.

```
## 2.2
lrm <- lm(motor_UPDRS ~ Jitter... + Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 + Jitter.DDP +
            Shimmer + Shimmer.dB. + Shimmer.APQ3 + Shimmer.APQ5 + Shimmer.APQ11 +
            Shimmer.DDA + NHR + HNR + RPDE + DFA + PPE + 0, data = train)

summary(lrm)
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ Jitter... + Jitter.Abs. + Jitter.RAP +
##     Jitter.PPQ5 + Jitter.DDP + Shimmer + Shimmer.dB. + Shimmer.APQ3 +
##     Shimmer.APQ5 + Shimmer.APQ11 + Shimmer.DDA + NHR + HNR +
##     RPDE + DFA + PPE + 0, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0255 -0.7363 -0.1087  0.7333  2.1960
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## Jitter...       0.186931   0.149561   1.250 0.211431
## Jitter.Abs.    -0.169609   0.040805  -4.157 3.31e-05 ***
```

```
## Jitter.RAP     -5.269544  18.834160  -0.280 0.779658
## Jitter.PPQ5    -0.074568   0.087766  -0.850 0.395592
## Jitter.DDP      5.249558  18.837525   0.279 0.780510
## Shimmer         0.592436   0.205981   2.876 0.004050 **
## Shimmer.dB.    -0.172655   0.139316  -1.239 0.215315
## Shimmer.APQ3   32.070932  77.159242   0.416 0.677694
## Shimmer.APQ5   -0.387507   0.113789  -3.405 0.000668 ***
## Shimmer.APQ11   0.305546   0.061236   4.990 6.34e-07 ***
## Shimmer.DDA   -32.387241  77.158814  -0.420 0.674695
## NHR            -0.185387   0.045567  -4.068 4.84e-05 ***
## HNR            -0.238543   0.036395  -6.554 6.41e-11 ***
## RPDE            0.004068   0.022664   0.179 0.857556
## DFA            -0.280318   0.020136 -13.921  < 2e-16 ***
## PPE             0.226467   0.032881   6.887 6.70e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9394 on 3509 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.25 on 16 and 3509 DF,  p-value: < 2.2e-16
```

Estimating training & test MSE.

The MSE for the training data set is 0.8785431, and for the test data set, it is 0.9354477. As we don't have another model to compare that to, you can not say how good this value is. Nevertheless, one could say that the number is relatively small and it is therefore not too bad.

```
train_MSE <- mean((train$motor_UPDRS - predict(lrm))^2)
train_MSE
```

```
## [1] 0.8785431
```

```
test_MSE <- mean((test$motor_UPDRS - predict(lrm,test))^2)
test_MSE
```

```
## [1] 0.9354477
```

Task 3 (Implementing 4 functions)

Log likelihood function

```
## 2.3
Loglikelihood <- function(theta, sigma, input_data){

  Y <- input_data[,1]
  X <- as.matrix(input_data[,-1])
  n <- nrow(input_data)

  logl <- -n*log(sqrt(2*pi)*abs(sigma)) - (1/(2*(sigma^2)))*sum((Y-X %*% theta)^2)

  return(logl)
}
```

Ridge function

```
Ridge <- function(param, input_data, lambda){

  theta <- param[-1]
```

```
  sigma <- param[1]

  logl <- -Loglikelihood(theta, sigma, input_data) + lambda*sum(theta^2)

  return(-Loglikelihood(theta, sigma, input_data) + lambda*sum(theta^2))
}
```

Ridge optimal

```
RidgeOpt <- function(lambda, input_data){
  optimal <- optim(par = rep(1, 17), fn = Ridge, input_data = input_data, lambda = lambda,
                   method = "BFGS")
  return(optimal$par)
}
```

Degrees of freedom

```
#lamda is 0 if we do not give a penalty
df <- function(input_data, lambda=0){

  X <- as.matrix(input_data[,-1])
  I <- diag(ncol(X))
  hat_matrix <- X %*% solve(t(X) %*% X + lambda*I) %*% t(X)

  degrees_of_freedom <- sum(diag(hat_matrix))

  return(degrees_of_freedom)
}
```

Task 4

Using function RidgeOpt to compute optimal theta parameters for lamda = 1, lamda = 100 and lamda = 1000.

```
## 2.4
theta_hat_1 <- RidgeOpt( lambda = 1, input_data = train)
theta_hat_100 <- RidgeOpt( lambda = 100, input_data = train)
theta_hat_1000 <- RidgeOpt( lambda = 1000, input_data = train)

theta_hat_1[-1]
```

```
##  [1]  0.177927740 -0.168545384 -0.012061419 -0.070163496 -0.004003292
##  [6]  0.534314721 -0.144569663 -0.148993705 -0.374416913  0.306465033
## [11] -0.151941377 -0.183147403 -0.237093006  0.005112815 -0.279307118
## [16]  0.226027448
```

```
theta_hat_100[-1]
```

```
##  [1]  0.04613901 -0.12960433  0.01663553 -0.02564421  0.01671847  0.03654840
##  [7]  0.02066881 -0.07607945 -0.10171112  0.22306258 -0.07611080 -0.14080787
## [13] -0.18181728  0.02776343 -0.24500000  0.21408275
```

```
theta_hat_1000[-1]
```

```
##  [1]  0.005217044 -0.041164077 -0.003517356 -0.006699959 -0.003510768
##  [6]  0.005353007  0.012548573 -0.017317825 -0.009897736  0.072091742
## [11] -0.017318387 -0.036598318 -0.078210762  0.045104230 -0.126634984
## [16]  0.104313954
```

Using the estimating parameters to predict the motor_UPDRS values for training and test data.

```
predicted_values <- function(theta_hat, input_data){

  X <- as.matrix(input_data[,-1])
  y_hat <- X %*% theta_hat[-1]

  return(y_hat)
}

y_hat_1_train <- predicted_values(theta_hat_1, train)
y_hat_100_train <- predicted_values(theta_hat_100, train)
y_hat_1000_train  <- predicted_values(theta_hat_1000, train)

y_hat_1_test <- predicted_values(theta_hat_1, test)
y_hat_100_test <- predicted_values(theta_hat_100, test)
y_hat_1000_test  <- predicted_values(theta_hat_1000, test)
```

Reporting the training and test MSE values.

```
MSE <- function(Y_hat, input_data){

  Y <- input_data[,1]
  n <- nrow(input_data)

  mse <- (1/n) * sum((Y-Y_hat)^2)

  return(mse)
}

MSE_1_train <- MSE(y_hat_1_train, train)
MSE_100_train <- MSE(y_hat_100_train, train)
MSE_1000_train <- MSE(y_hat_1000_train, train)

MSE_1_test <- MSE(y_hat_1_test, test)
MSE_100_test <- MSE(y_hat_100_test, test)
MSE_1000_test <- MSE(y_hat_1000_test, test)

errors_df <- data.frame("lambda=1" = c(MSE_1_train,
                                       MSE_1_test),
                        "lambda=100" = c(MSE_100_train,
                                         MSE_100_test),
                        "lambda=1000" = c(MSE_1000_train,
                                          MSE_1000_test))
row.names(errors_df) <- c("MSE (training data)", "MSE (test data)")
errors_df
```

```
##                       lambda.1 lambda.100 lambda.1000
## MSE (training data) 0.8786272  0.8844114   0.9211166
## MSE (test data)     0.9349976  0.9323317   0.9539456
```

The penalty that seems to be the most appropriate for the training dataset is the theta parameters for lambda = 1. The MSE of the predicted values with the above theta parameters is 0.8786272, which means that predicted values have the slightest error from the observed values compared to the other predicted values.

The theta parameters for lambda = 100 provide the best penalty for the test dataset with an MSE of

0.9323317. Thus, the predicted values based on these thetas are better estimators than the other predicted values.

Compute and compare the degrees of freedom of the training and test models.

Both the train and the test data have 16 degrees of freedom if you look at the normal linear model without penalty because we assume the variables to be independent. Using λ as a penalty term takes away some of the independence, and fewer degrees of freedom remain, as shown in the table below. So it makes sense that the higher the penalty term gets, the lower the degrees of freedom are.

```
freedom_df <- data.frame("no penalty" = c(df(input_data = train),
                                          df(input_data = test)),
                         "lamda=1" = c(df(input_data = train,
                                          lambda = 1),
                                       df(input_data = test,
                                          lambda = 1)),
                         "lamda=100" = c(df(input_data = train,
                                          lambda = 100),
                                       df(input_data = test,
                                          lambda = 100)),
                         "lamda=1000" = c(df(input_data = train,
                                          lambda = 1000),
                                       df(input_data = test,
                                          lambda = 1000)))
row.names(errors_df) <- c("df (training data)", "df (test data)")
freedom_df
```

```
##   no.penalty  lamda.1 lamda.100 lamda.1000
## 1         16 13.86074  9.924887   5.643925
## 2         16 13.77688  9.173182   4.822328
```

Assignment 3

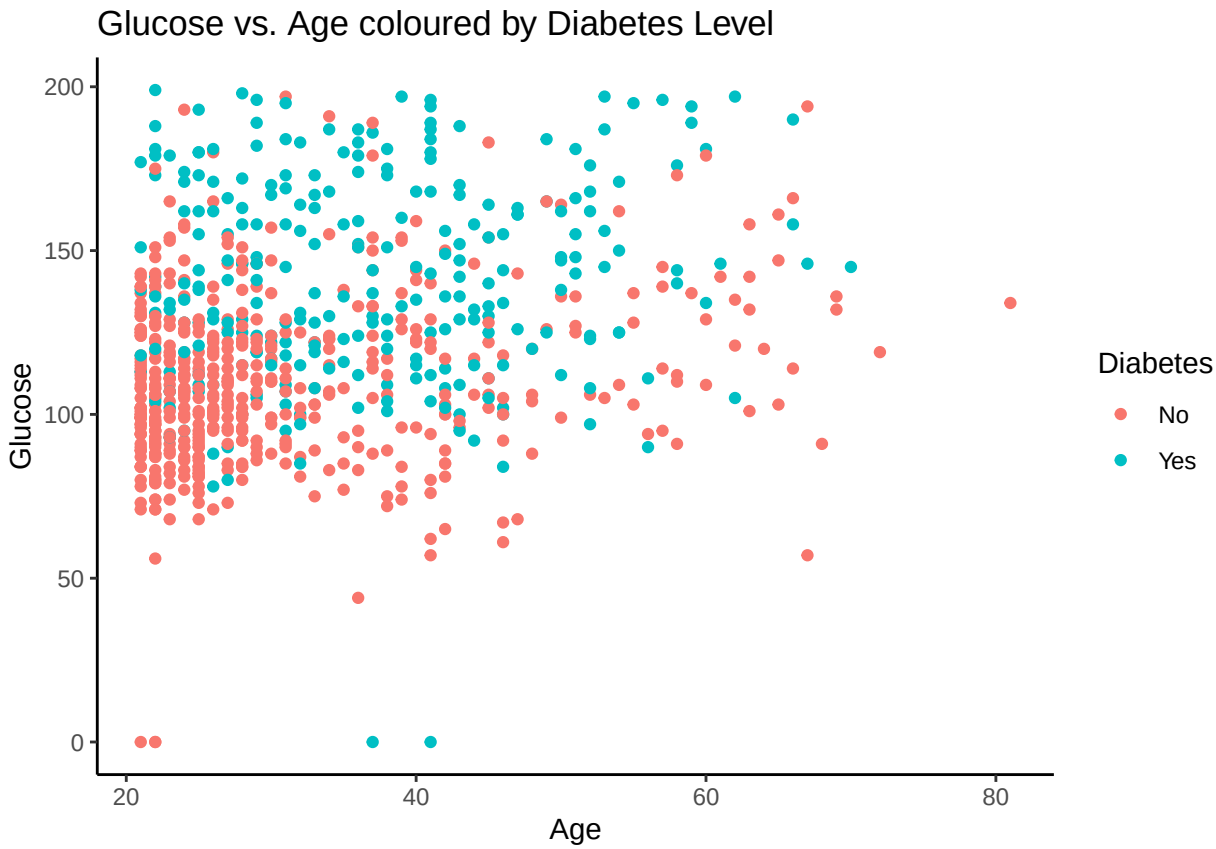
We start with reading the data set and renaming the column names to make working with the data frame easier. Additionally, we change the Diabetes variable to a factor variable with two levels.

```
## 3.1
df <- read.csv("files/pima-indians-diabetes.csv", header = FALSE)
colnames(df) <- c("Pregnancies",
                  "Glucose",
                  "BloodPressure",
                  "SkinThickness",
                  "Insulin",
                  "BMI",
                  "DiabetesPedigreeFunction",
                  "Age",
                  "Diabetes")
df$Diabetes <- as.factor(df$Diabetes)
levels(df$Diabetes) <- c("No","Yes")
```

Task 1

Make a scatter plot showing a Plasma glucose concentration on Age where observations are colored by Diabetes levels.

```r
p1 <- ggplot(df, aes(x = Age, y = Glucose, col = Diabetes)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Diabetes Level")
p1
```



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.

Suppose one is only interested in a rough classification. In that case, these two variables are probably enough as one can definitely spot two different colour areas: Most people that have diabetes have a higher glucose concentration, and additionally, in the group of young people, fewer people have diabetes. Nevertheless, the classification will probably not work that well for all values in the area where both groups are bordering as they have some overlapping areas there. So, in general, the groups are not very well separable, which will be a problem for the logistic regression.

Task 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.

```r
## 3.2
glm_fit <- train(Diabetes ~ Age + Glucose, data = df, method="glm")
df$PredDiabetes <- predict(glm_fit) # predict function uses 0.5 as default value
```

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

$p(y = 1|w, x_1, x_2) = \frac{1}{1+e^{-w_0-w_1x_1-w_2x_2}}$

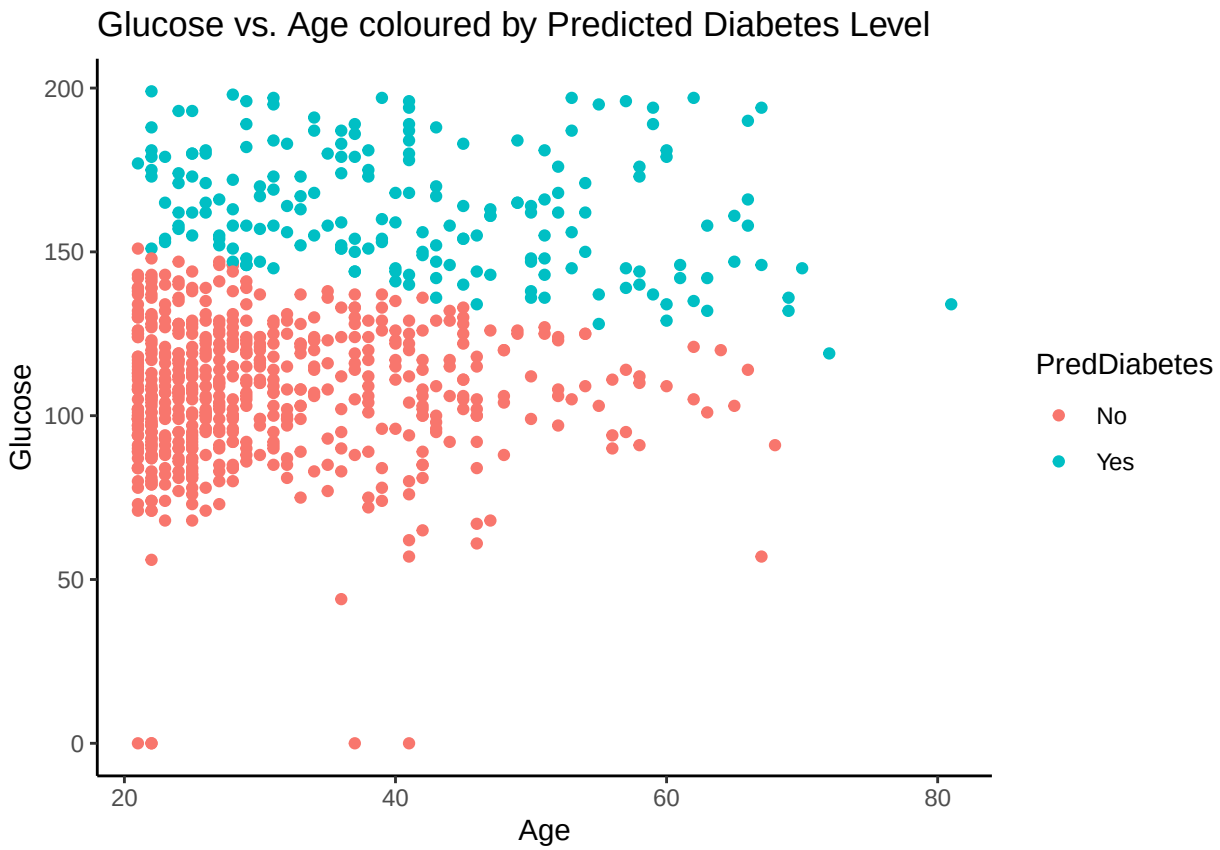
x_1 is the glucose concentration and x_2 is the age.

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.

```
# function to calculate misclassification taken from lecture slides
missclass <- function(X,X1){
  return(1-sum(diag(table(X,X1)))/length(X))
}

missclass(df$Diabetes,df$PredDiabetes)
```

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

```
p2 <- ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetes)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")
p2
```



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

As the classification is only done using the two variables shown in the plot, we knew before that the decision boundary would be better visible after training a model. Therefore, the results are all right if one is only interested in a rough classification using these two variables. Furthermore, the misclassification rate is only around 26%, so three out of four observations are classified right. Nevertheless, one should keep in mind that a higher classification success would definitely be desirable, so it might make sense to include some more variables in the model because there is no rigorous cut visible between these two variables, as mentioned earlier.

Task 3

Use the model estimated in step 2 to report the equation of the decision boundary between the two classes

```
## 3.3
(myIntercept <- glm_fit$finalModel$coefficients[1]/
   -glm_fit$finalModel$coefficients[3])
```

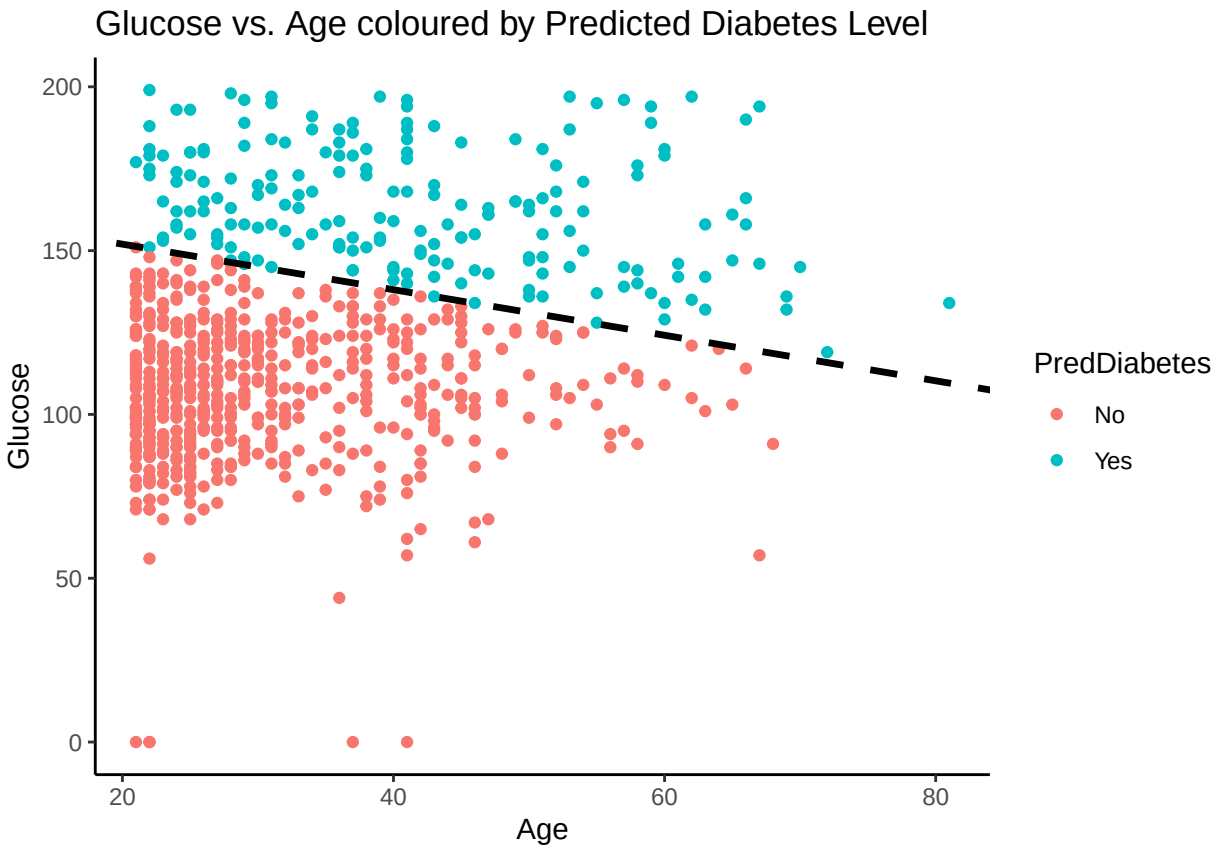
```
## (Intercept)
##    165.8748
```

```
(mySlope <- glm_fit$finalModel$coefficients[2]/
  -glm_fit$finalModel$coefficients[3])
```

```
##        Age
## -0.6951611
```

Use the model estimated in step 2 to add a curve showing this boundary to the scatter plot in step 2.

```
p2 + geom_abline(intercept = myIntercept,
                 slope = mySlope,
                 color = "black",
                 linetype = "dashed",
                 size = 1.2)
```

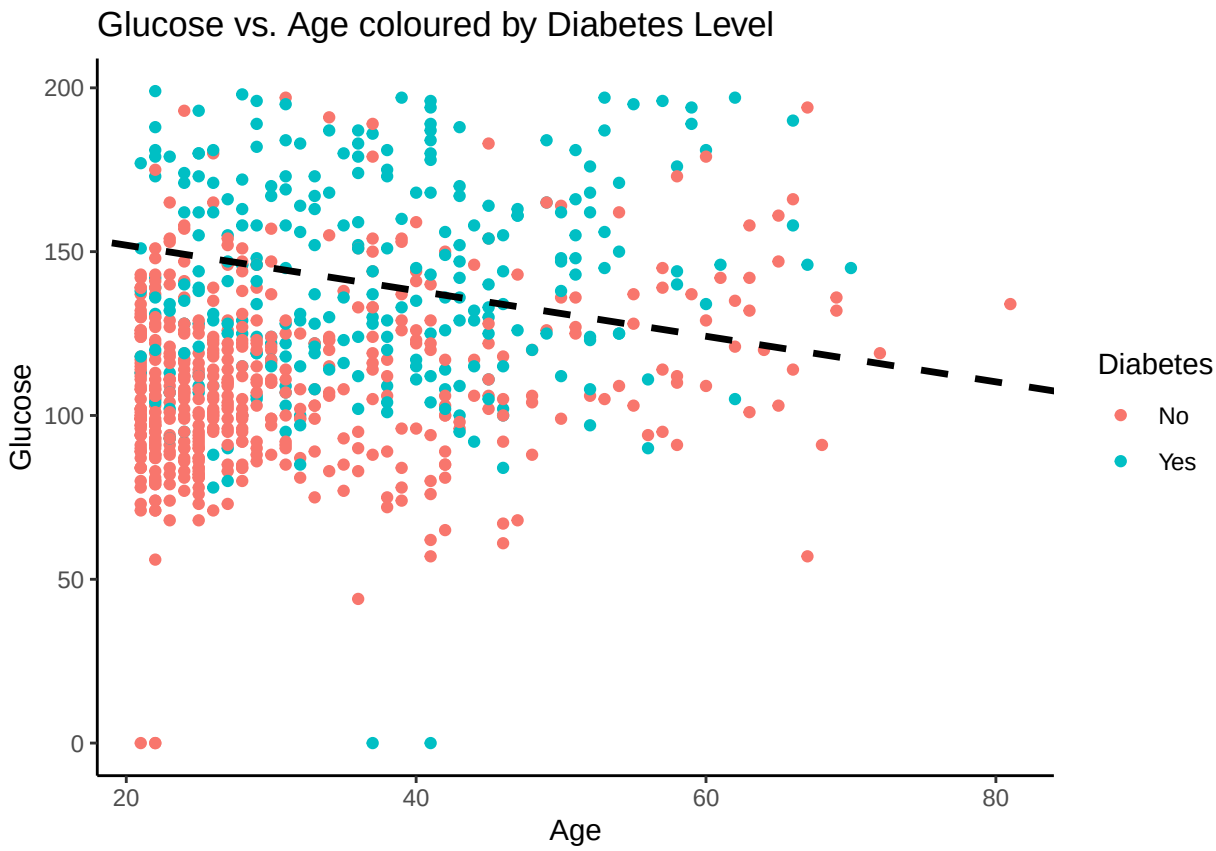


Comment whether the decision boundary seems to catch the data distribution well.

In the plot above, one can see the decision boundary, which splits the predicted data as one would expect if you look at the predicted data. Obviously, the decision boundary catches the data distribution of the predicted data well.

If one actually draws this decision boundary in the original scatter plot from task 1 (see down below), some more interesting things can be observed:

```
p1 + geom_abline(intercept = myIntercept,
                 slope = mySlope,
                 color = "black",
                 linetype = "dashed",
                 size = 1.2)
```



As we have described earlier, one can see that this decision boundary works well for the majority of the data set as both groups have an area where most points are located. Nevertheless, many points of both groups, especially in the border region and all over the plot, are specified wrong. As we calculated earlier, these are around 1/4 of the people and are all blue points underneath the line and all red points above.

Task 4

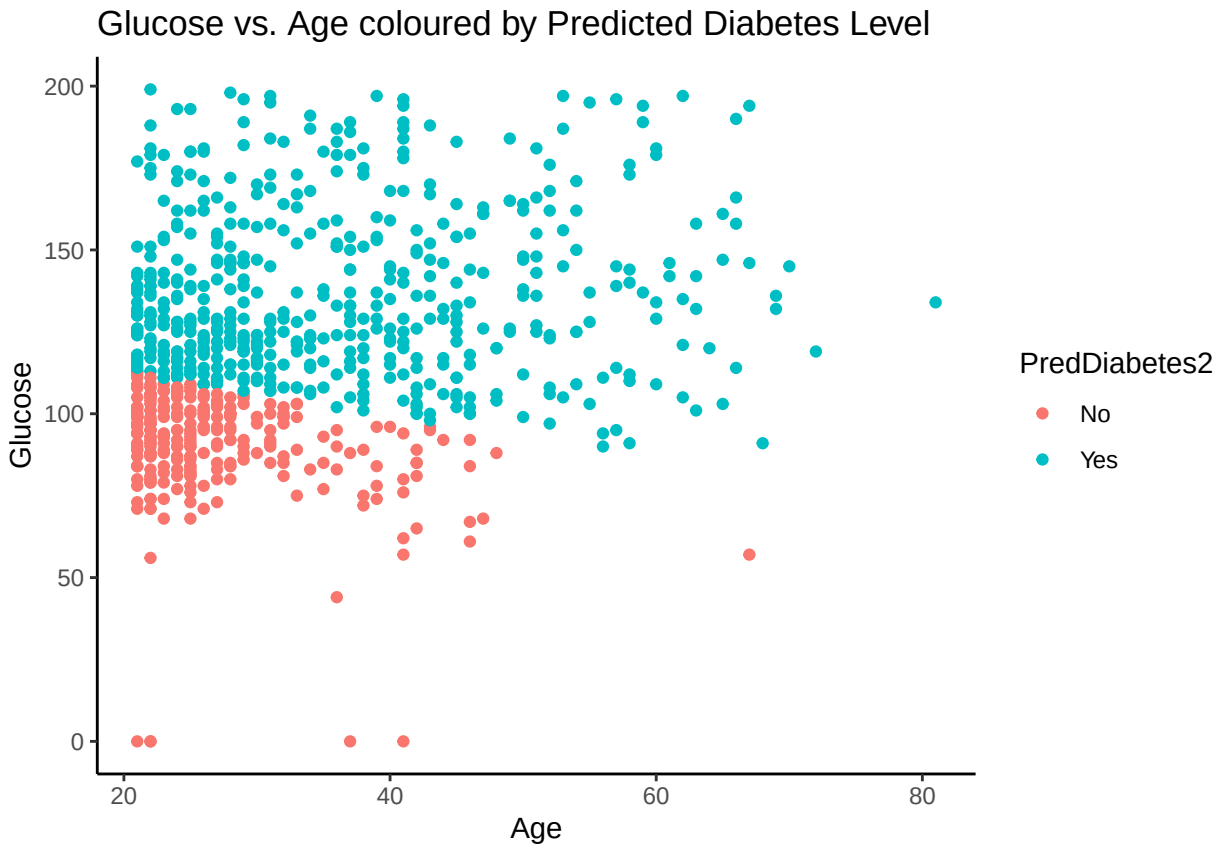
Make same kind of plots as in step 2 but use thresholds $r = 0.2$ and $r = 0.8$.

```
## 3.4
df$PredDiabetes2 <- as.factor(ifelse(predict(glm_fit,
                                             type="prob") < 0.2,
                                     "No","Yes")[,2])

missclass(df$Diabetes,df$PredDiabetes2)
```

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

```
ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetes2)) +
  geom_point() +
```

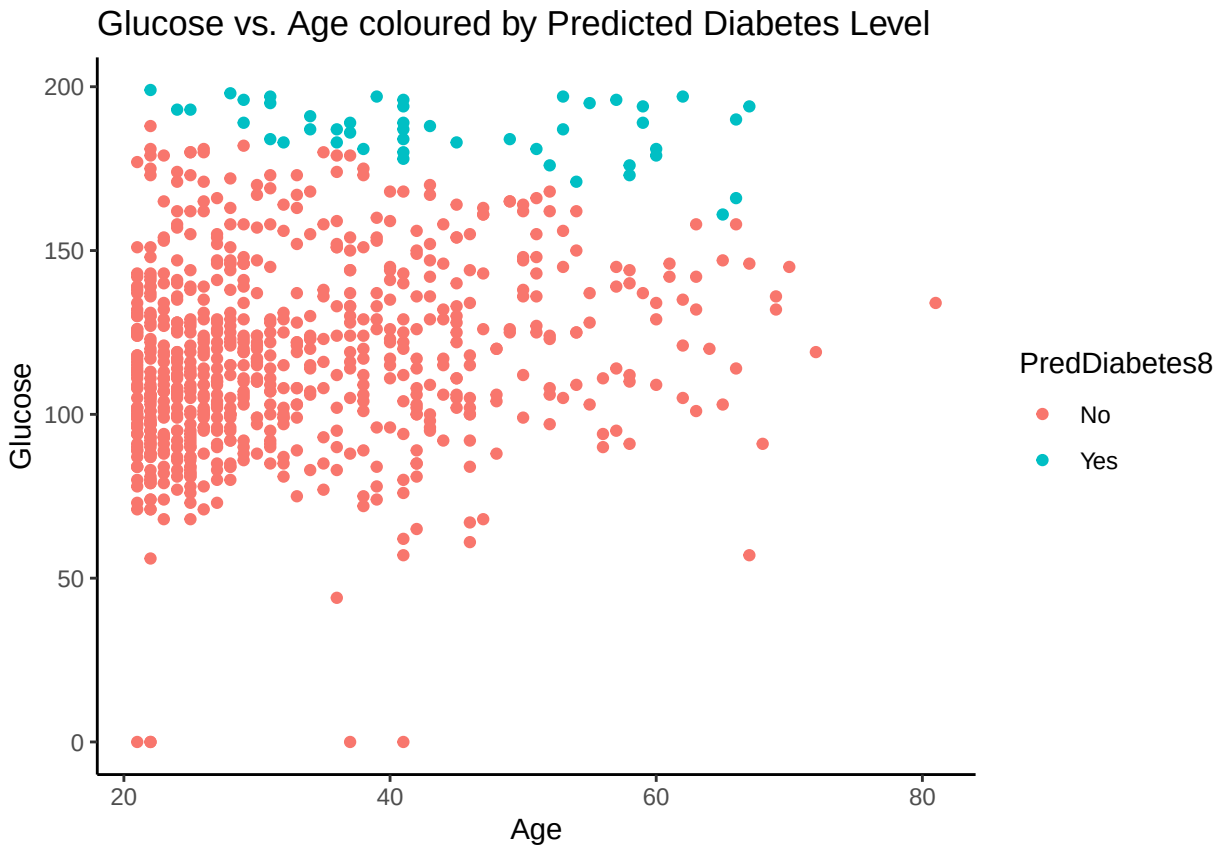
```
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")
```




```
df$PredDiabetes8 <- as.factor(ifelse(predict(glm_fit,
                                             type="prob") < 0.8,
                                     "No","Yes")[,2])


missclass(df$Diabetes,df$PredDiabetes8)
```

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

```
ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetes8)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")
```



By using these plots, comment on what happens with the prediction when r value changes.

If you change the value for r to 0.2 or 0.8, the classification for the data points is estimated with values in the range between changes. So if you change the value from 0.5 to 0.8, fewer values are classified as Diabetes cases because all predicted values between 0.5 and 0.8 are now added to the non-Diabetes group as well. If you change the value to 0.2, many more people are classified as Diabetes cases because all predicted values above 0.2 are put into that group. The misclassification rate is higher than for the 0.5 thresholds, both for 0.2 and 0.8.

Task 5

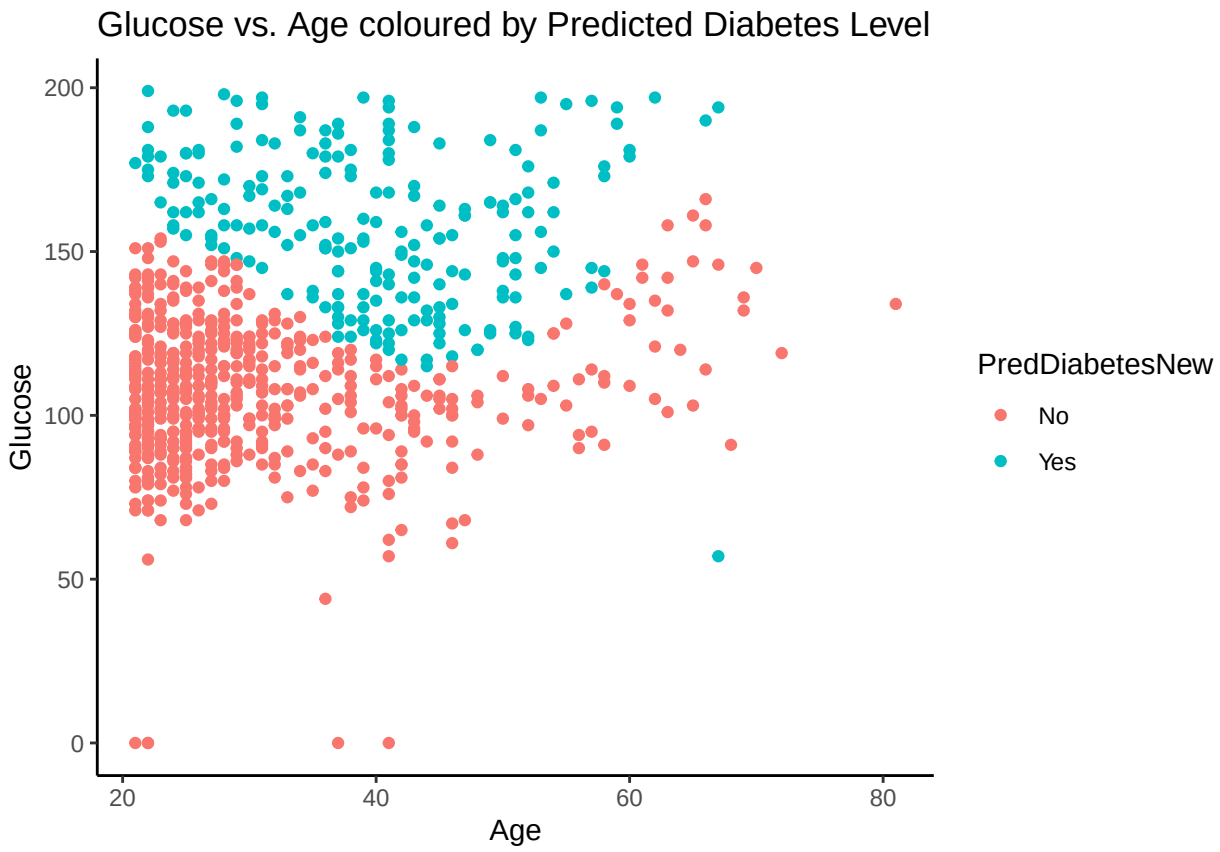
Perform a basis function expansion trick by computing new features $z_1 = x_1^4, z_2 = x_1^3 x_2, z_3 = x_1^2 x_2^2, z_4 = x_1 x_2^3, z_5 = x_2^4$*, adding them to the data set and then computing a logistic regression model with* y *as target and* $x_1, x_2, z_1, ..., z_5$ *as features. Create a scatter plot of the same kind as in step 2 for this model and compute the training misclassification rate.*

```
## 3.5
df$z1 <- df$Glucose^4
df$z2 <- df$Glucose^3*df$Age
df$z3 <- df$Glucose^2*df$Age^2
df$z4 <- df$Glucose^1*df$Age^3
df$z5 <- df$Age^4

glm_fit_new <- train(Diabetes ~ Age + Glucose + z1 + z2 +
                       z3 + z4 + z5, data = df, method="glm")
df$PredDiabetesNew <- predict(glm_fit_new)
```

```
missclass(df$Diabetes,df$PredDiabetesNew)
```

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

```
ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetesNew)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")
```



What can you say about the quality of this model compared to the previous logistic regression model?

If one compares the misclassification rates of the two models, the new model performs a little bit better than the previous one. Before, the misclassification rate was 26% while it now is 24%. As our data set contains 768 observations, this means a difference of 14 right classified data points. So instead of 566 observations classified right before, we know we classify 580 data points correctly.

How have the basis expansion trick affected the shape of the decision boundary and the prediction accuracy?

The shape was affected by the basis expansion trick as we now do not have a straight line that cuts our data, but a parable shaped area that includes all the people predicted not to have diabetes. In contrast, the ones with predicted sickness are above (and one observation is below). What we think is particularly interesting is the blue point in the right bottom corner below the just mentioned red area, which is being classified wrong now but was being classified right before as it actually belongs to the group with people with no diabetes.

This example illustrates nicely that the fitting process obviously concentrates a lot on the more usual data. So we can assume now that if there would be more data points around the just mentioned data point that they would probably belong to the group that is classified as diabetes, even though when looking at the original data, there might be a higher chance that it actually belongs to the people that do not have diabetes.

Statement of Contribution

We first worked on all assignments ourselves, and everyone finished all of them. Theodor was mainly concentrating on assignment 1 and wrote down the interpretations there. Christoforos focused most on assignment 2 and wrote the interpretation for these tasks while Lisa was doing the same for assignment 3. After everyone finished, we discussed our results and adjusted a few things.

Code

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

library(ggplot2)
library(caret)
## 1.1
df <- read.csv("optdigits.csv", header = FALSE)
df <- df[,c(65, 1:64)] # Reorder to make formula pick first column
names(df)[1] <- "digit"
df$digit <- factor(df$digit)
n <- nrow(df)
set.seed(12345)
id_train <- sample(1:n, floor(n*0.5))
df_train <- df[id_train,] # Training data
id_rest <- setdiff(1:n, id_train)
set.seed(12345)
id_vali <- sample(id_rest, floor(n*0.25))
df_vali <- df[id_vali,] # Validation data
id_test <- setdiff(id_rest, id_vali)
df_test <- df[id_test,] # Test data
##1.2
library(kknn)
form <- formula(df_train) #Automatic formula
fit_train <- kknn(form, train = df_train, test = df_train, k = 30, kernel = "rectangular")
fit_test <- kknn(form, train = df_train, test = df_test, k = 30, kernel = "rectangular")
# function to calculate misclassification taken from lecture slides
missclass <- function(X,X1){
  return(1-sum(diag(table(X,X1)))/length(X))
}
fit_val_train <- fitted(fit_train)
fit_val_test <- fitted(fit_test)
conf_mat_train <- table(df_train$digit,fit_val_train)
knitr::kable(conf_mat_train, caption = "Confusion matrix: training data")
conf_mat_test <- table(df_test$digit,fit_val_test)
knitr::kable(conf_mat_test, caption = "Confusion matrix: test data")
misclass_train <- missclass(df_train$digit, fit_val_train) #0.045
paste("Misclassification error for training data:", round(misclass_train, 5))
misclass_test <- missclass(df_test$digit, fit_val_test) #0.053
paste("Misclassification error for test data:", round(misclass_test, 5))
## 1.3
library(dplyr)

probs <- df_train %>%
  mutate(prob8 = fit_train$prob[,"8"]) %>%
  filter(digit == 8)
```

```r
worst <- probs %>% slice_min(prob8, n = 3)
best <- probs %>% slice_max(prob8, n = 40) # 40 digits with prob = 1
set.seed(12345)
randm <- sample(1:nrow(best), 2)
best <- best[randm,]

make_8x8 <- function(inputrow){
  return(matrix(as.numeric(inputrow[2:65]), nrow = 8, ncol = 8, byrow = T))
}

best1 <- make_8x8(best[1,])
best2 <- make_8x8(best[2,])
worst1 <- make_8x8(worst[1,])
worst2 <- make_8x8(worst[2,])
worst3 <- make_8x8(worst[3,])
#we do not evaluate these chunks when knitting the document as the
#plots take quite a lot of space and we decided to include them as
#pictures to make it look nicer (same with the next chunk)
p1 <- heatmap(best1, Colv=NA, Rowv=NA, main = "best 1")
p2 <- heatmap(best2, Colv=NA, Rowv=NA, main = "best 2")
p3 <- heatmap(worst1, Colv=NA, Rowv=NA, main = "worst 1")
p4 <- heatmap(worst2, Colv=NA, Rowv=NA, main = "worst 2")
p5 <- heatmap(worst3, Colv=NA, Rowv=NA, main = "worst 3")
## 1.4
models_missclass <- function(K){
  out <- data.frame(K = numeric(), misclass_train = numeric(), misclass_vali = numeric())
  for (i in 1:K) {
    set.seed(12345)
  fit_training <- kknn(form, train = df_train, test = df_train,
                       k = i, kernel = "rectangular")
  fit_validation <- kknn(form, train = df_train, test = df_vali,
                         k = i, kernel = "rectangular")

  misclass_training <- missclass(df_train$digit, fitted(fit_training))
  misclass_vali <- missclass(df_vali$digit, fitted(fit_validation))

  out[i, ] <- c(i, misclass_training, misclass_vali)
  }
  return(out)
}
misclassifications <- models_missclass(30)
library(ggplot2)

ggplot(data = misclassifications) +
  geom_point(aes(x = K, y = misclass_vali, color = "Validation models")) +
  geom_point(aes(x = K, y = misclass_train, color = "Training models")) +
  labs(title = "Misclassification for different Ks in KNN",
       y = "Misclassification error (in %)",
       colour = "Model type") +
  theme_bw() + scale_x_continuous(breaks = 1:30) +
  geom_vline(xintercept = which.min(misclassifications$misclass_vali), linetype ="dotted")
fit_train_K3 <- kknn(form, train = df_train, test = df_test, k = 3, kernel = "rectangular")
test_error <- missclass(df_test$digit, fitted(fit_train_K3))
```

```r
errors_df <- data.frame(TrainingError = misclassifications$misclass_train[3],
                        ValidationError = misclassifications$misclass_vali[3],
                        TestingError = test_error)
row.names(errors_df) <- "Misclassification error (in %)"

errors_df
## 1.5
# Since only one digit is possible response variable we do
# not need to compute the -sum(log(p)*d) but can simply take the -log
# of the probability for the true digit. Result will be the same.
cross_entropy <- list()
for (K in 1:30) {
  fit = kknn(form, df_train, df_vali, k=K, kernel="rectangular")

  df_probs <- data.frame(fit$prob)
  df_probs$digit <- df_vali$digit

  entropy <- list()
  for (i in 1:nrow(df_probs)) {
    for (n in 1:10) {
      if (df_probs$digit[i] == n-1) {
        entropy[[i]] = -(log(df_probs[i, n]+ 1e-15))
      }
    }
  }
  cross_entropy[[K]] <- sum(unlist(entropy))
}
cross_entropy_valid <- data.frame(cross_entropy = unlist(cross_entropy), K = 1:30)
ggplot(cross_entropy_valid, aes(x = K, y = cross_entropy)) +
  geom_point(col = "blue") + theme_bw() +
  labs(title = "Cross-entropy error for different Ks in KNN", y = "Cross-entropy")+
  scale_x_continuous(breaks = 1:30)
## 2.1
data <- read.csv("parkinsons.csv")

data <- data[,-c(1:4,6)]  # deleting undesirable variables
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data.frame(data[id,])
test=data.frame(data[-id,])

mean_train <- numeric(ncol(train))
sd_train <- numeric(ncol(train))

for(i in 1:ncol(train)){
  mean_train[i] <- mean(train[,i])
  sd_train[i] <- sd(train[,i])
}

train <- data.frame(scale(train))

test_new <- data.frame(matrix(nrow = nrow(test), ncol = ncol(test)))
```

```r
for(i in 1:ncol(train)){
  test_new[,i] <- (test[,i] - mean_train[i])/sd_train[i]
}

test <- as.data.frame(test_new)
colnames(test) <- colnames(train)
## 2.2
lrm <- lm(motor_UPDRS ~ Jitter... + Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 + Jitter.DDP +
            Shimmer + Shimmer.dB. + Shimmer.APQ3 + Shimmer.APQ5 + Shimmer.APQ11 +
            Shimmer.DDA + NHR + HNR + RPDE + DFA + PPE + 0, data = train)

summary(lrm)
train_MSE <- mean((train$motor_UPDRS - predict(lrm))^2)
train_MSE
test_MSE <- mean((test$motor_UPDRS - predict(lrm,test))^2)
test_MSE
## 2.3
Loglikelihood <- function(theta, sigma, input_data){

  Y <- input_data[,1]
  X <- as.matrix(input_data[,-1])
  n <- nrow(input_data)

  logl <- -n*log(sqrt(2*pi)*abs(sigma)) - (1/(2*(sigma^2)))*sum((Y-X %*% theta)^2)

  return(logl)
}
Ridge <- function(param, input_data, lambda){

  theta <- param[-1]
  sigma <- param[1]

  logl <- -Loglikelihood(theta, sigma, input_data) + lambda*sum(theta^2)

  return(-Loglikelihood(theta, sigma, input_data) + lambda*sum(theta^2))
}
RidgeOpt <- function(lambda, input_data){
  optimal <- optim(par = rep(1, 17), fn = Ridge, input_data = input_data, lambda = lambda,
                   method = "BFGS")
  return(optimal$par)
}
#lamda is 0 if we do not give a penalty
df <- function(input_data, lambda=0){

  X <- as.matrix(input_data[,-1])
  I <- diag(ncol(X))
  hat_matrix <- X %*% solve(t(X) %*% X + lambda*I) %*% t(X)

  degrees_of_freedom <- sum(diag(hat_matrix))

  return(degrees_of_freedom)
}
## 2.4
```

```r
theta_hat_1 <- RidgeOpt( lambda = 1, input_data = train)
theta_hat_100 <- RidgeOpt( lambda = 100, input_data = train)
theta_hat_1000 <- RidgeOpt( lambda = 1000, input_data = train)

theta_hat_1[-1]
theta_hat_100[-1]
theta_hat_1000[-1]
predicted_values <- function(theta_hat, input_data){

  X <- as.matrix(input_data[,-1])
  y_hat <- X %*% theta_hat[-1]

  return(y_hat)
}

y_hat_1_train <- predicted_values(theta_hat_1, train)
y_hat_100_train <- predicted_values(theta_hat_100, train)
y_hat_1000_train  <- predicted_values(theta_hat_1000, train)

y_hat_1_test <- predicted_values(theta_hat_1, test)
y_hat_100_test <- predicted_values(theta_hat_100, test)
y_hat_1000_test  <- predicted_values(theta_hat_1000, test)
MSE <- function(Y_hat, input_data){

  Y <- input_data[,1]
  n <- nrow(input_data)

  mse <- (1/n) * sum((Y-Y_hat)^2)

  return(mse)
}

MSE_1_train <- MSE(y_hat_1_train, train)
MSE_100_train <- MSE(y_hat_100_train, train)
MSE_1000_train <- MSE(y_hat_1000_train, train)

MSE_1_test <- MSE(y_hat_1_test, test)
MSE_100_test <- MSE(y_hat_100_test, test)
MSE_1000_test <- MSE(y_hat_1000_test, test)

errors_df <- data.frame("lambda=1" = c(MSE_1_train,
                                       MSE_1_test),
                        "lambda=100" = c(MSE_100_train,
                                         MSE_100_test),
                        "lambda=1000" = c(MSE_1000_train,
                                          MSE_1000_test))
row.names(errors_df) <- c("MSE (training data)", "MSE (test data)")
errors_df
freedom_df <- data.frame("no penalty" = c(df(input_data = train),
                                          df(input_data = test)),
                         "lamda=1" = c(df(input_data = train,
                                          lambda = 1),
                                       df(input_data = test,
```

```
                                        lambda = 1)),
                        "lamda=100" = c(df(input_data = train,
                                        lambda = 100),
                                     df(input_data = test,
                                        lambda = 100)),
                        "lamda=1000" = c(df(input_data = train,
                                        lambda = 1000),
                                     df(input_data = test,
                                        lambda = 1000)))
row.names(errors_df) <- c("df (training data)", "df (test data)")
freedom_df
## 3.1
df <- read.csv("files/pima-indians-diabetes.csv", header = FALSE)
colnames(df) <- c("Pregnancies",
                  "Glucose",
                  "BloodPressure",
                  "SkinThickness",
                  "Insulin",
                  "BMI",
                  "DiabetesPedigreeFunction",
                  "Age",
                  "Diabetes")
df$Diabetes <- as.factor(df$Diabetes)
levels(df$Diabetes) <- c("No","Yes")
p1 <- ggplot(df, aes(x = Age, y = Glucose, col = Diabetes)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Diabetes Level")
p1
## 3.2
glm_fit <- train(Diabetes ~ Age + Glucose, data = df, method="glm")
df$PredDiabetes <- predict(glm_fit) # predict function uses 0.5 as default value
# function to calculate misclassification taken from lecture slides
missclass <- function(X,X1){
  return(1-sum(diag(table(X,X1)))/length(X))
}

missclass(df$Diabetes,df$PredDiabetes)

p2 <- ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetes)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")
p2
## 3.3
(myIntercept <- glm_fit$finalModel$coefficients[1]/
   -glm_fit$finalModel$coefficients[3])
(mySlope <- glm_fit$finalModel$coefficients[2]/
  -glm_fit$finalModel$coefficients[3])
p2 + geom_abline(intercept = myIntercept,
                 slope = mySlope,
                 color = "black",
                 linetype = "dashed",
```

```
                  size = 1.2)
p1 + geom_abline(intercept = myIntercept,
                  slope = mySlope,
                  color = "black",
                  linetype = "dashed",
                  size = 1.2)
## 3.4
df$PredDiabetes2 <- as.factor(ifelse(predict(glm_fit,
                                             type="prob") < 0.2,
                                     "No","Yes")[,2])

missclass(df$Diabetes,df$PredDiabetes2)

ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetes2)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")

df$PredDiabetes8 <- as.factor(ifelse(predict(glm_fit,
                                             type="prob") < 0.8,
                                     "No","Yes")[,2])


missclass(df$Diabetes,df$PredDiabetes8)

ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetes8)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")

## 3.5
df$z1 <- df$Glucose^4
df$z2 <- df$Glucose^3*df$Age
df$z3 <- df$Glucose^2*df$Age^2
df$z4 <- df$Glucose^1*df$Age^3
df$z5 <- df$Age^4

glm_fit_new <- train(Diabetes ~ Age + Glucose + z1 + z2 +
                       z3 + z4 + z5, data = df, method="glm")
df$PredDiabetesNew <- predict(glm_fit_new)


missclass(df$Diabetes,df$PredDiabetesNew)

ggplot(df, aes(x = Age, y = Glucose, col = PredDiabetesNew)) +
  geom_point() +
  theme_classic() +
  ggtitle("Glucose vs. Age coloured by Predicted Diabetes Level")
```