Lab 1 TDDE01

Caspian Süsskind, Rasmus Jonsson & Ludvig Eriksson Knast

November 21, 2021

1 Assignments

1.1 Handwritten digit recognition with K-nearest neighbor

The task in assignment 1 was to, given a dataset, use the R package "kknn" to train and fit a model to be able to perform digit recognition of handwritten digits using the K-nearest neighbor algorithm. There was also some model and error function evaluation performed.

1.1.1 Question 1

The first step in this task was to divide the given data into training (50%), validation (25%) and test (25%) sets.

1.1.2 Question 2

Overall the model was able to predict the training and test data fairly good, meaning the model was able to correctly predict about 95% of the handwritten digits. The most interesting observations were that the model wrongly predicted the number 5 to be a 9 about 5% of all the occurring 5s in the training data (see Figure 1), similarly this same miss-prediction (predicting a 5 to be a 9) also happened with around 5% of the occurring 5s in the test data (see Figure 2). There were also some repeating mix-up between the number 8 and 1 in the training data, where the model predicted a 8 to be a 1 about 5% of the occurring 8s in the test data, this mix-up is kind of understandable when you look at the images produced in task 3 below.

```
df.kknn.fit   0   1   2   3   4   5   6   7   8   9
          0 177   0   0   0   0   0   0   0   0   0
          1   0 174   0   0   1   0   0   1  10   3
          2   0   9 170   0   0   0   0   0   0   0
          3   0   0   0 197   0   0   0   1   1   4
          4   1   0   0   0 166   0   0   0   0   2
          5   0   0   0   2   0 183   0   1   0   0
          6   0   1   0   0   2   1 200   0   2   0
          7   0   0   1   1   6   2   0 192   0   2
          8   0   1   2   0   2   0   0   0 190   4
          9   0   3   0   0   2  11   0   0   2 181
Miss rate for (K=30, training data) =  0.04238619
```

Figure 1: Confusion matrix and misclassification rate for the training data.

```
df.kknn.fit  0  1  2  3  4  5  6  7  8  9
          0 97  0  0  0  1  0  0  0  0  0
          1  0 91  0  0  0  1  0  2  3  0
          2  0  3 93  0  0  0  0  0  0  0
          3  0  0  1 95  0  1  0  0  1  4
          4  0  0  0  0 89  0  0  0  0  0
          5  0  0  0  0  0 79  0  1  0  0
          6  1  0  0  0  1  1 94  0  1  0
          7  0  0  0  2  5  0  0 91  0  2
          8  0  0  1  1  1  0  0  1 86  1
          9  0  3  0  0  3  5  0  0  0 94
Miss rate for (K=30, test data) =  0.04916318
```

Figure 2: Confusion matrix and misclassification rate for the test data.

1.1.3 Question 3

Looking at the 5 images of the digits you can clearly see the difference between the 2 easiest and 3 hardest ones. What visually is the biggest factor is that the 2 easy ones has 2 loops with empty middles (see Figure 3), compared to the 3 hard ones where there is basically no clear loops which are empty (see Figure 4). This makes the hard cases look more like a broad 1 which then makes the results from task 2 more understandable. From seeing the images it is completely reasonable that the model would have problems with the hard ones, likewise no problem with easy.

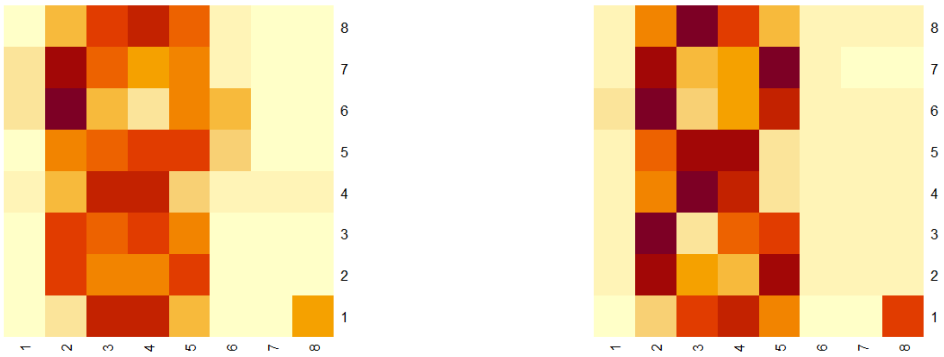

Figure 3: Two of the easiest cases of the digit 8 in the training data for the model to classify.

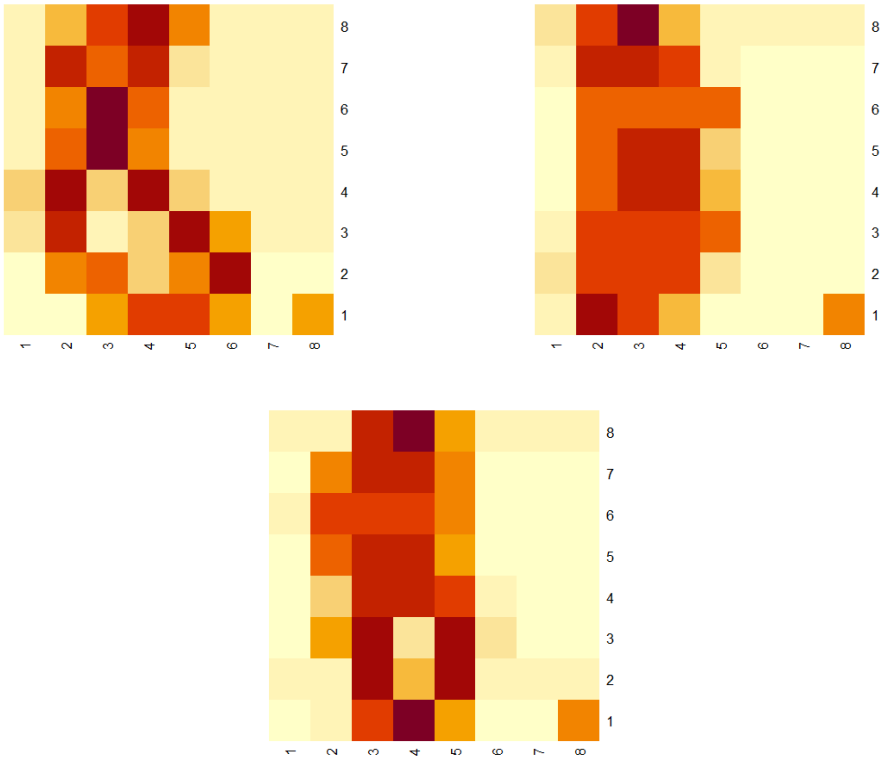

Figure 4: The three hardest cases of the digit 8 in the training data for the model to classify.

1.1.4 Question 4

The lower the value of K is, the higher the complexity gets and vice versa. This means that with really low Ks the model has very high variance and is overfitting towards the training data which results in almost perfect results when predicting the training data and the performance with predicting the validation data is not really showing the accuracy of the model but instead indicates how similar the data sets (training and validation) are to each other. Likewise high Ks results in a underfitted model with low complexity that can't really improve to the extent we need it to. It means that the model will be far from optimal as well as uncertain when predicting which will lead to both high training and validation errors. From our results K=7 is the optimal K for this model (see Figure 5). However when comparing the model by using the test data set there is a difference in performance between the training data and test data (about 2%) but between the validation data and test data (about 1%) (see Figure 6). It isn't too bad but it can indicate that there is a close by K which could be better as we preferably would want to see that the model could perform as well on the test data as the validation data, however not if this change in K would lead to a overall worse performance of the model. We want a consistent, high performing **and** confident model.

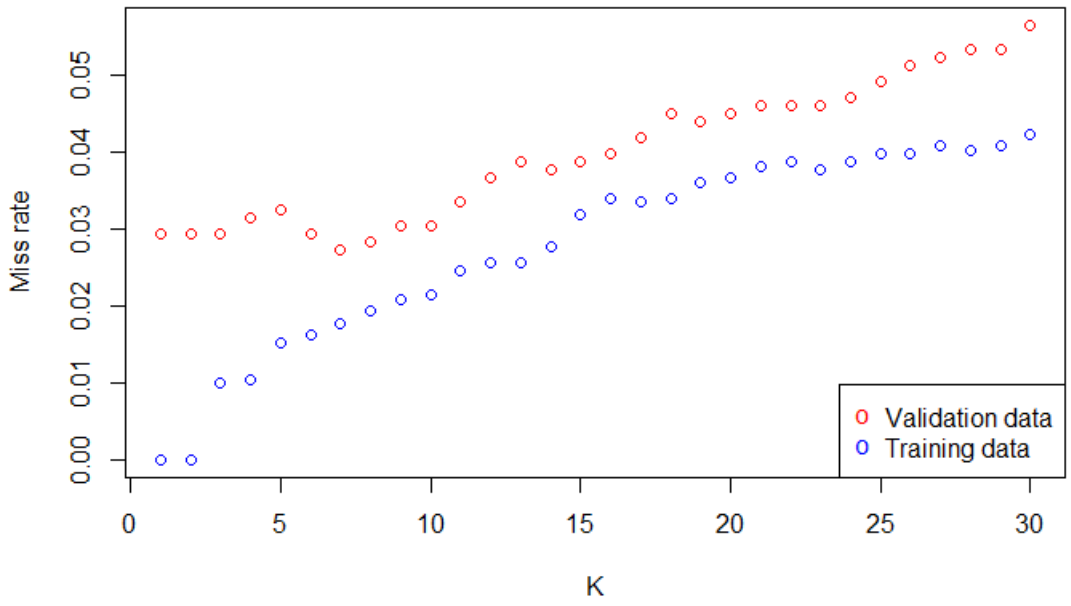

Figure 5: Dependence of he training and validation misclassification errors on the value of K.

```
For the optimal K from task 4 the following results can be retrieved:
Optimal K = 7
Miss rate with training data: 0.01779173
Miss rate with validation data: 0.02722513
Miss rate with test data: 0.03870293
```

Figure 6: Resulting optimal K with misclassification rates with the different data sets.

1.1.5 Question 5

The optimal value of K using cross entropy as error function was 8 (see Figure 7). In this problem, where we have a relatively small data set as well as response with multinomial distribution, cross entropy is more suitable as it enables us to utilize and take in to consideration how confident the model was in its' predication as well as how wrong the worse predictions were, not only correct/wrong. The binary nature of misclassification errors does not suit the multinomial distribution equally well. Once again, we want a consistent, high performing **and** confident model.

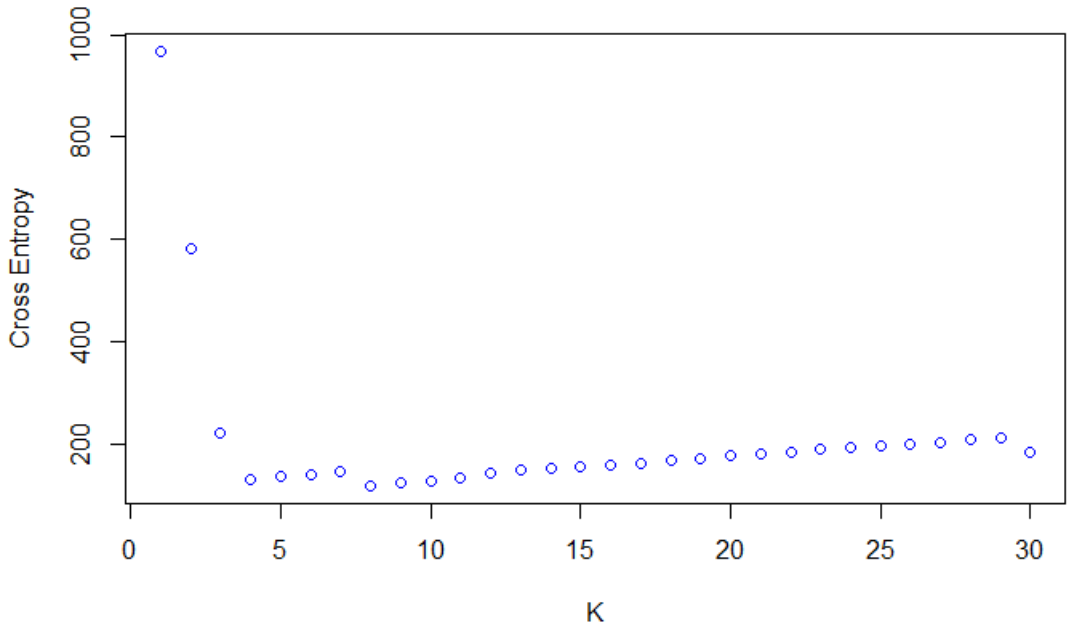

Figure 7: Dependence of the validation error on the value of K (using cross entropy error function).

1.2 Linear regression and ridge regression

The task in assignment 2 was to, given a dataset, use linear regression to predict a Parkinson's disease score from voice characteristics. Implementing functions for loglikelihood, ridge and to calculate optimal θ and degrees of freedom given a λ was also a part of the task.

1.2.1 Question 1

In the first question the task was to scale (and center) the data, and then split it into a training and test set using a 60/40 split. The scaling was simply done by subtracting the mean of each column and dividing the result by the standard deviation. The scaling was done on the whole dataset, but better training may be given by scaling only the training data and then scale the test data using the same values for mean and standard deviation.

1.2.2 Question 2

Question 2 involved training a linear regression model using the training data and then estimating the training and test Mean Squared Error (MSE) using the trained model. The MSE on the training data was estimated to approximately 0.8731931 and the MSE on the test data was estimated to approximately 0.9294911 In Figure 8 the coefficients of the trained linear regression model can be seen. Judging by the coefficients it seems as if the variables Jitter.DDP, Jitter.RAP, Shimmer.APQ3 and Shimmer.DDA contribute the most to the model, because their coefficients are the largest.

```
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
Jitter...       0.181065   0.144249   1.255 0.209481
Jitter.Abs.    -0.169830   0.040851  -4.157 3.30e-05 ***
Jitter.RAP     -5.098809  18.184783  -0.280 0.779196
Jitter.PPQ5    -0.071777   0.084701  -0.847 0.396816
Jitter.DDP      5.079056  18.188164   0.279 0.780069
Shimmer         0.590992   0.205286   2.879 0.004015 **
Shimmer.dB.    -0.172860   0.139380  -1.240 0.214983
Shimmer.APQ3   32.213852  77.012847   0.418 0.675759
Shimmer.APQ5   -0.386846   0.113713  -3.402 0.000677 ***
Shimmer.APQ11   0.310256   0.062270   4.982 6.58e-07 ***
Shimmer.DDA   -32.529915  77.012630  -0.422 0.672761
NHR            -0.186755   0.045741  -4.083 4.55e-05 ***
HNR            -0.239777   0.036565  -6.558 6.27e-11 ***
RPDE            0.003958   0.022611   0.175 0.861052
DFA            -0.277038   0.019888 -13.930  < 2e-16 ***
PPE             0.229006   0.033264   6.885 6.84e-12 ***
```

Figure 8: Coefficients of the trained linear regression model.

1.2.3 Question 3

The third question was to implement a number of functions: *Loglikelihood*, *Ridge*, *RidgeOpt* and *DF*. The *Loglikelihood* function takes a parameter vector θ, a dispersion σ as well as data X and Y. It computes $ln(p(Y|X,\theta))$, as in equation 1. The *Ridge* function takes a vector θ, and scalars σ and λ. It uses the *Loglikelihood* function to add a ridge penalty, $\lambda||\theta||$, to the minus *Loglikelihood* function. The *RidgeOpt* function takes a scalar λ and uses the *Ridge* function and the optim(...) function to find the optimal θ and σ for a given λ. Finally the *DF* function computes the degrees of freedom (dof) for a given λ.

$$ln(p(Y|X,\theta)) = -\frac{n}{2}ln(2\pi\sigma_\epsilon^2)) - \frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^{n}(\theta^{\mathbf{T}}\mathbf{x_i} - y_i)^2 \tag{1}$$

1.2.4 Question 4

The final question of assignment 2 involved computing the optimal θ for $\lambda \in \{1, 100, 1000\}$. The estimated parameters were then supposed to predict MotorUPDRS values and then estimate the MSE training and test errors. The degrees of freedom were also computed for the different values of λ. The results can be seen in Table 1. Judging by the MSE errors a penalty parameter λ with a value of 100 seems most appropriate. By looking at the degrees of freedom it can be seen that as λ increases the degrees of freedom decreases. Which makes sense since the penalty parameter "pushes" variables to zero.

Table 1: Results of assignment 2 question 4.

Lambda, λ	**MSE train**	**MSE test**	**DOF**
1	0.8732611	0.9292190	13.862811
100	0.8790072	0.9263081	9.939085
1000	0.9762517	0.9939754	5.643351

1.3 Logistic regression and basis function expansion

The task in assignment 3 was to, given a dataset, use logistic regression and basis function expansion to predict the onset of diabetes in pima indians within five years. The data contained 768 entries, all of which with 9 variables, including the target.

1.3.1 Question 1

The first question involved making a scatter plot showing the "Plasma glucose concentration 2 hours in an oral glucose tolerance test" and the "Age". In the scatter plot the observations were colored by the diabetes levels corresponding to each point. The resulting scatter plot can be seen in Figure 9. Judging by the distribution of the points in Figure 9, it seems as if it would be hard to separate the points and classify diabetes with a standard logistic regression model using only the two parameters: "Plasma glucose concentration 2 hours in an oral glucose tolerance test" and the "Age". This is because there is no straight line decision boundary that can separate the points very well, since the points are quite mixed.

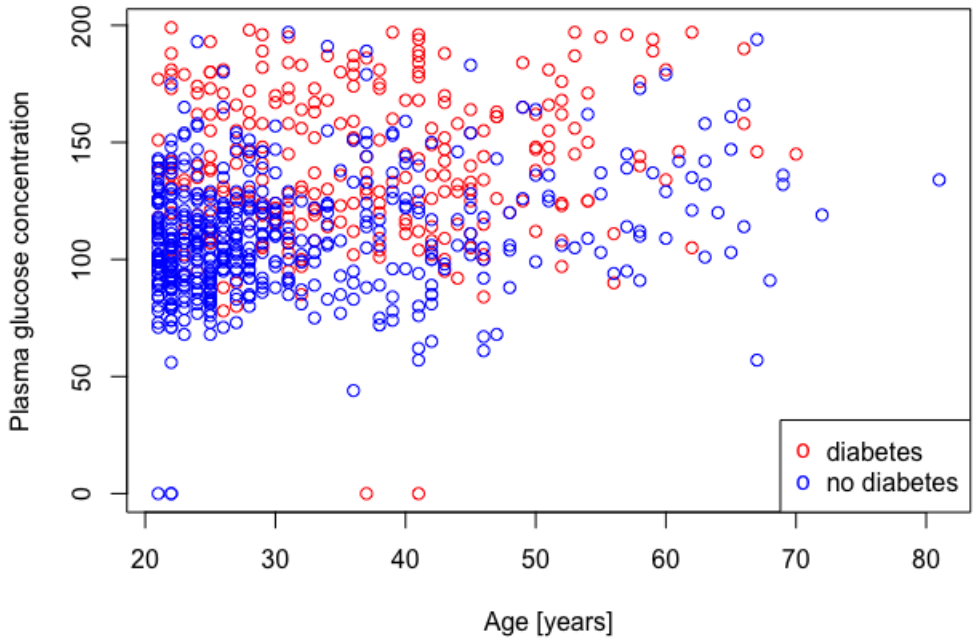

Figure 9: Scatter plot of plasma glucose concentration and age with diabetes levels indicated by different colors.

1.3.2 Question 2

Question 2 involved training a logistic regression model using "Diabetes" as the target y, "Plasma glucose concentration 2 hours in an oral glucose tolerance test" as feature x_1 and "Age" as feature x_2. Then predictions were to be made for all observations using a classification threshold, r, with a value of 0.5. The probabilistic equation of the fitted model can be formed by plugging in $g(\mathbf{x})$ as in equation 2 into equation 3. The training misclassification rate was calculated to be approximately 0.2630208, and the predictions can be seen in Figure 10. Judging by the misclasssification rate, and the scatter plot in Figure 10 compared to Figure 9, the predictions are not so bad but they are not great either.

$$g(\mathbf{x}) = \frac{1}{1 + e^{-(-5.91245 + 0.03564 \cdot x_1 + 0.02478 \cdot x_2)}} \tag{2}$$

$$p(y = C|x, \theta) = \begin{cases} g(x), & \text{if } C = 1 \\ 1 - g(x), & \text{if } C = 0 \end{cases} \tag{3}$$

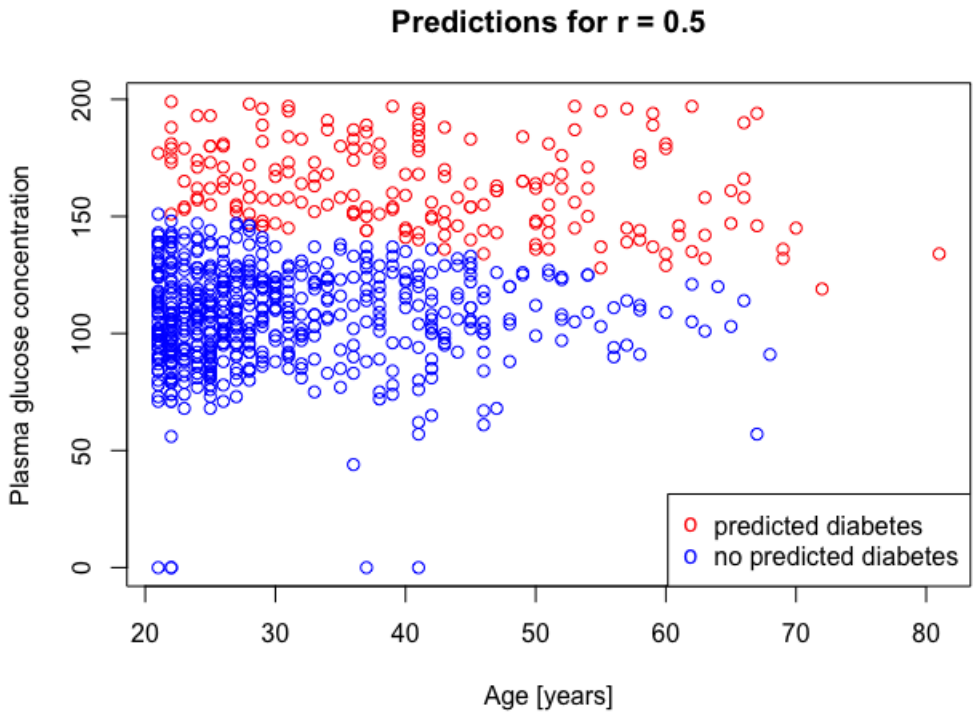

Figure 10: Scatter plot of plasma glucose concentration and age with predicted diabetes levels indicated by different colors.

1.3.3 Question 3

The decision boundary for the model estimated in question 2 is given by equation 4. A plot including the aforementioned decision boundary is present in Figure 11, and to no real surprise it catches the predictions perfectly. With regards to the actual data distribution shown in Figure 9, the decision boundary doesn't catch the data distribution perfectly but rather medium well.

$$g(\mathbf{x}) = \frac{1}{1 + e^{-(-5.91245 + 0.03564 \cdot x_1 + 0.02478 \cdot x_2)}} = 0.5 \tag{4}$$

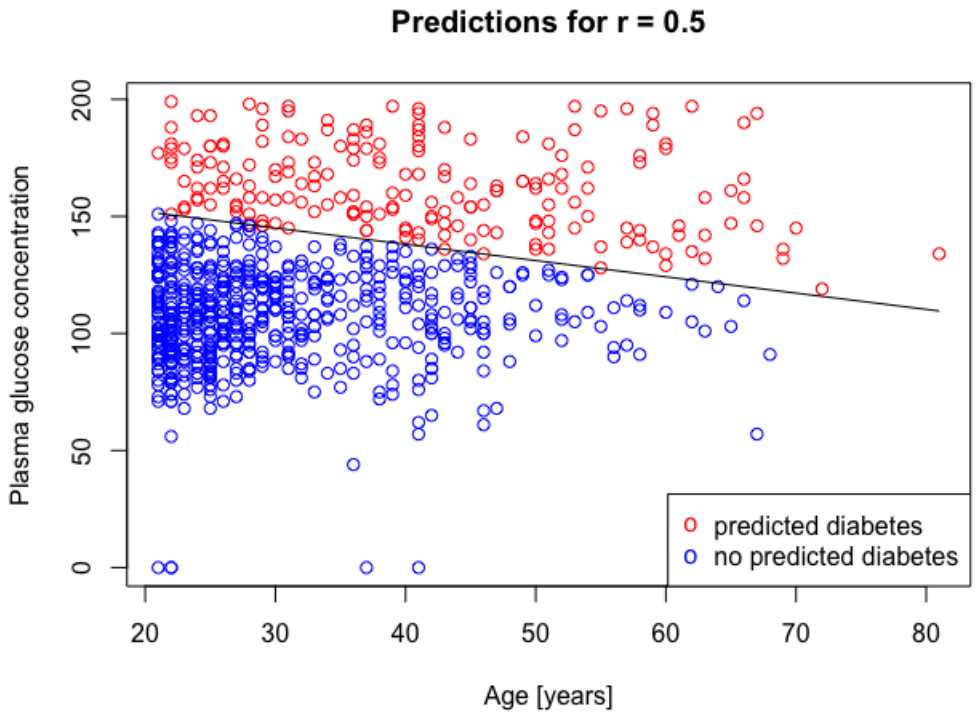

Figure 11: Scatter plot of plasma glucose concentration and age with predicted diabetes levels indicated by different colors. Decision boundary included as black curve.

1.3.4 Question 4

In question 4 the aim was to investigate what happens when the classification threshold, r, was changed. In Figure 12 the predictions when r was set to 0.2 can be seen. By comparing this to the predictions in Figure 10, where r was 0.5, it can be deduced that there seem to be more predictions of diabetes when r is decreased, which makes sense. In the same way the predictions of diabetes can be seen to decrease as r is increased, judging by Figure 13, where r was set to 0.8. This makes perfect sense since the prediction is done as in 5.

$$\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } g(\mathbf{x}) > r \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

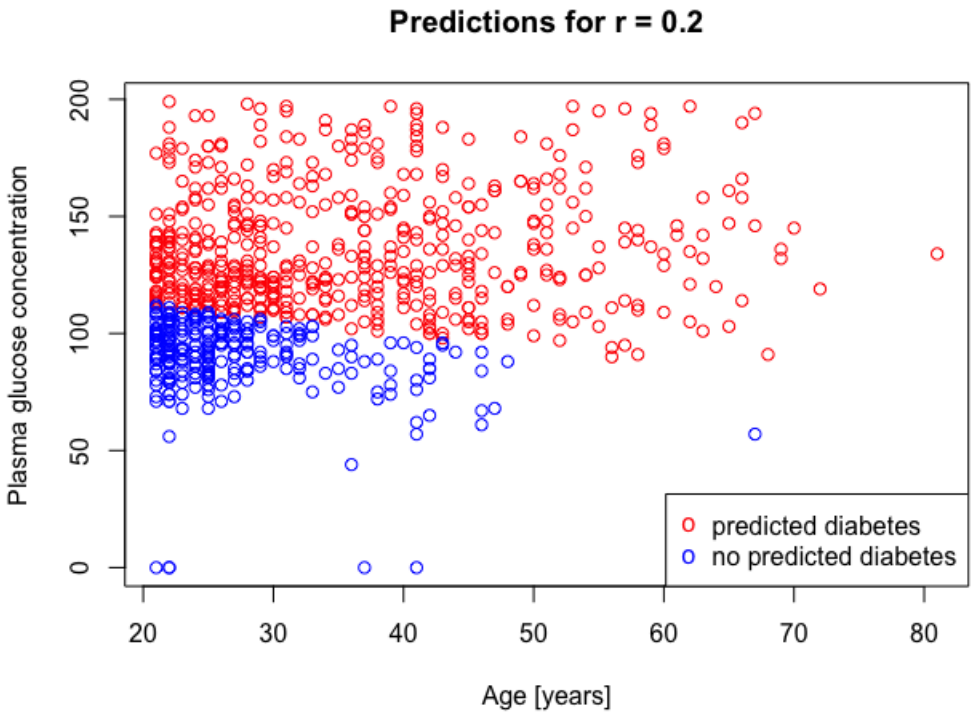

Figure 12: Scatter plot of plasma glucose concentration and age with predicted diabetes levels indicated by different colors.

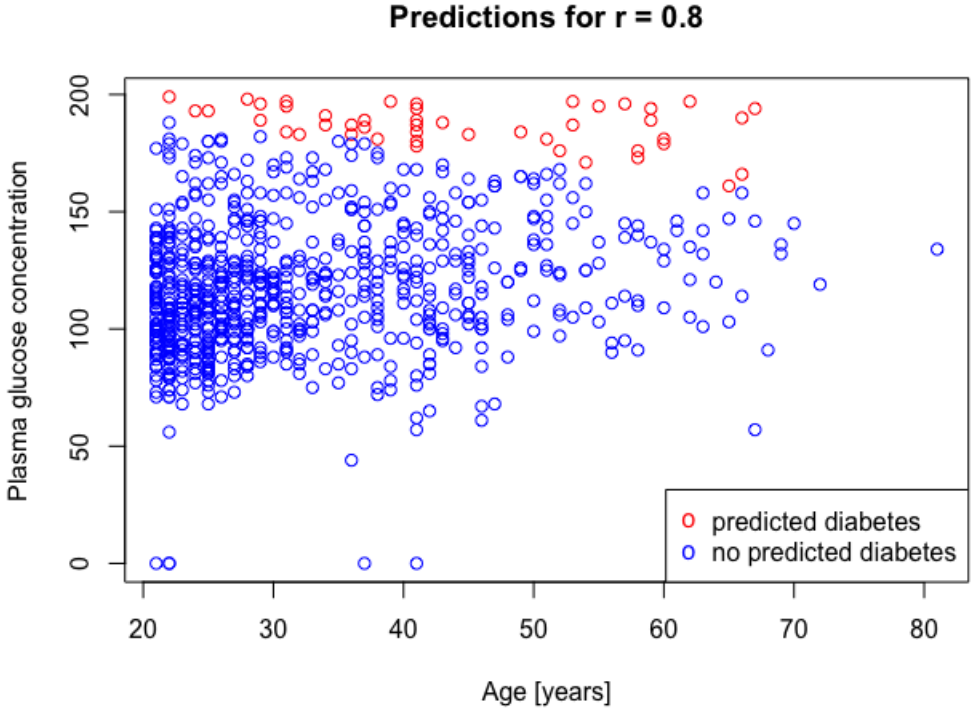

Figure 13: Scatter plot of plasma glucose concentration and age with predicted diabetes levels indicated by different colors.

1.3.5 Question 5

In question 5 the task is to perform a basis expansion trick by creating new features as: $z_1 = x_1^4$, $z_2 = x_1^3 x_2$, $z_3 = x_1^2 x_2^2$, $z_4 = x_1 x_2^3$ and $z_5 = x_2^4$. These features were then added to the data set along with the previously used x_1 and x_2, and a logistic regression model was fitted using these features. The resulting predictions on the data by the fitted model can be seen in Figure 14. The misclassification rate was computed to be approximately 0.2447917, which is slightly better than what was achieved in question 2 of assignment 3. Just judging by this the quality of this model is better than the previous model. However, as can be seen in Figure 14 the decision boundary now has a curved shape instead, and there seems to be a weird outlier prediction in the bottom right corner. This might indicate that the model is now too complex for the data and has been overfitted. So even if the prediction accuracy is better on the data used to fit the model, the model itself might not generalize very well, and it is not necessarily better than the previous model.

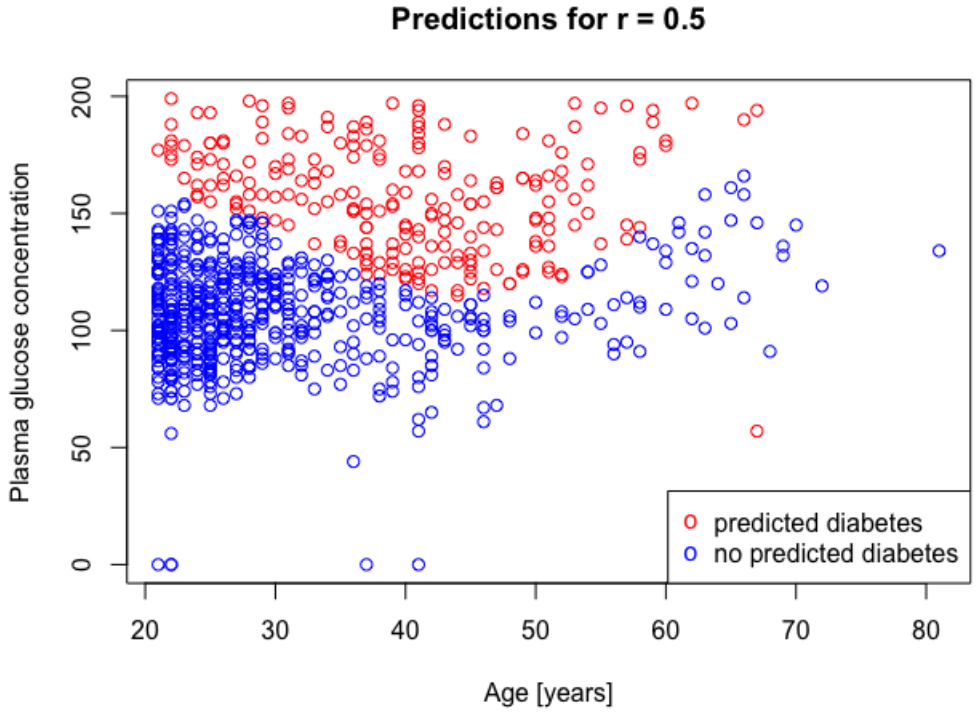

Figure 14: Scatter plot of plasma glucose concentration and age with predicted diabetes levels indicated by different colors. Using basis expansion trick.

2 Statement of contribution

Caspian Süsskind
Mainly responsible for assigment 3 in terms of both code and report writing. Wrote all code for assignment 3 and helped write code for assignment 2 as well. Wrote everything about assignment 3 and assisted a bit in writing assignment 2 in the report.

Ludvig Eriksson Knast
Mainly responsible for assigment 1 in terms of both code and report writing. Wrote all code for assignment 1, and wrote about assignment 1 in the report.

Rasmus Jonsson
Mainly responsible for assigment 2 in terms of both code and report writing. Wrote part of the code for assignment 2 and part of the report for assignment 2.

A Code

A.1 Assignment 1

Below is the code written for assignment 1 of the lab.

```
library(kknn)
#Read data file to data frame
df = read.csv("optdigits.csv")


################################################
##----------------- Task 1 -----------------##
################################################

#Divide data into training/validation/test sets.
n = dim(df)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = df[id,]

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

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



################################################
##----------------- Task 2 -----------------##
################################################

#Confusion matrix and misclassification error for test data

#Fitting with kknn
df.kknn = kknn(formula = as.factor(X0.26) ~ ., train = train, test = test,
               kernel = "rectangular", k = 30)
df.kknn.fit <- fitted(df.kknn)

#Confusion matrix
conf_matrix.test <- table(df.kknn.fit, test$X0.26)
print(conf_matrix.test)

#Missclassification rate
miss_rate <- 1-sum(diag(conf_matrix.test)/sum(conf_matrix.test))
cat("Miss␣rate␣for␣(K=30,␣test␣data)␣=␣", miss_rate, "\n")

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

#Confusion matrix and misclassification error for training data

#Fitting with kknn
df.kknn = kknn(formula = as.factor(X0.26) ~ ., train = train, test = train,
               kernel = "rectangular", k = 30)
df.kknn.fit <- fitted(df.kknn)

#Confusion matrix
conf_matrix.train <- table(df.kknn.fit, train$X0.26)
print(conf_matrix.train)
```

```r
#Missclassification rate
miss_rate <- 1-sum(diag(conf_matrix.train)/sum(conf_matrix.train))
cat("Miss rate for (K=30, training data) = ", miss_rate, "\n")



##############################################
##----------------- Task 3 -----------------##
##############################################

#Creating dataframe "summary" for test cases with 8 and to see
#prob from the model for those cases, then determine the
#3 hardest and 2 easiest cases
truth <- train$X0.26
prob_8 <- predict(df.kknn, train, type = "prob")[,9]
summary <- data.frame(truth, prob_8)
summary <- subset(summary, truth == 8)
summary <- summary[order(summary$prob_8),]
hard.8 <- c(strtoi(row.names(summary[1,])))
hard.8 <- c(hard.8, strtoi(row.names(summary[2,])))
hard.8 <- c(hard.8, strtoi(row.names(summary[3,])))
easy.8 <- c(strtoi(row.names(summary[nrow(summary),])))
easy.8 <- c(easy.8, strtoi(row.names(summary[nrow(summary) -1,])))

cat("Easiest 8s to predict:", easy.8, "\n")
cat("Hardest 8s to predict:", hard.8, "\n")

# Easiest 8s to predict: 1864 1811
# Hardest 8s to predict: 1624 1663 229

#Visualize given digit (x)
rotate <- function(mat){
  return(t(apply(mat,2, rev)))
}
x <- 229 #change x to visualize wanted entry
hmMatrix = matrix(as.numeric(train[x,0:-1]), ncol = 8, nrow = 8)
hmMatrix = rotate(rotate(rotate(hmMatrix)))
hm <- heatmap(hmMatrix, Colv = NA, Rowv = NA)



##############################################
##----------------- Task 4 -----------------##
##############################################

#Missclass function
missclass=function(X,Y)
{
  return(1-sum(diag(table(X,Y)))/length(X))
}

#Collecting missclassification and cross entropy for each K (1..30)
K.missclass.valid = c()
K.missclass.train = c()
K.optimal = c(0,1,1) #(K, miss_rate.valid, miss_rate.train)

for(KVal in 1:30) {
  df.kknn <- kknn(formula = as.factor(X0.26) ~ ., train=train,
                  test=valid, kernel="rectangular", k = KVal)
  df.kknn.fit <- fitted(df.kknn)
  miss_rate.valid <- missclass(df.kknn.fit, valid$X0.26)
```

```
  K.missclass.valid <- c(K.missclass.valid, miss_rate.valid)

  df.kknn <- kknn(formula = as.factor(X0.26) ~ ., train=train,
                  test=train, kernel="rectangular", k = KVal)
  df.kknn.fit <- fitted(df.kknn)
  miss_rate.train <- missclass(df.kknn.fit, train$X0.26)
  K.missclass.train <- c(K.missclass.train, miss_rate.train)

  if (miss_rate.valid < K.optimal[2]) {
    K.optimal <- c(KVal, miss_rate.valid, miss_rate.train)
  }

}
plot(K.missclass.valid,
     xlab = "K",
     ylab = "Miss rate",
     col ="red",
     ylim = c(0,max(K.missclass.valid)),
     )
points(K.missclass.train, col = "blue")
legend(x="bottomright",
       legend = c("Validation data","Training data"),
       col=c("red", "blue"),
       pch = "oo")

#Testing optimal K from task 4
df.kknn = kknn(formula = as.factor(X0.26) ~ ., train = train, test = test,
               kernel = "rectangular", k = K.optimal[1])
df.kknn.fit <- fitted(df.kknn)
miss_rate <- missclass(df.kknn.fit, test$X0.26)
cat("For the optimal K from task 4 the following results can be retrieved:",
    "\nOptimal K =", K.optimal[1], "\nMiss rate with training data:",
    K.optimal[3],"\nMiss rate with validation data:",K.optimal[2],
    "\nMiss rate with test data:", miss_rate)



################################################
##----------------- Task 5 -----------------##
################################################

#Collecting cross entropy for each K = 1..30

K.crossEntropy = c()
for(KVal in 1:30) {
  df.kknn <- kknn(formula = as.factor(X0.26) ~ ., train=train,
                  test=valid, kernel="rectangular", k = KVal)
  df.kknn.fit <- fitted(df.kknn)

  x <- 0
  for (i in 1:length(valid$X0.26)){
    for (j in 0:9) {
      if (valid$X0.26[i] == j) {
        x <- x + (log(df.kknn$prob[i, j+1] + 1e-15))
      }
    }
  }

  K.crossEntropy <- c(K.crossEntropy, -x)
}
plot(K.crossEntropy, xlab = "K", ylab = "Cross Entropy", col = "blue")
```

A.2 Assignment 2

```
################################################
# ----------------- Task 1 ----------------- #
################################################

df = read.csv("parkinsons.csv")

## Scaling the data using mean and standard deviation.
sdf = df

sdf[2] = (sdf[2]-mean(sdf[[2]]))/sd(sdf[[2]])
for (i in 4:22)
{
  sdf[i] = (sdf[[i]]-mean(sdf[[i]]))/sd(sdf[[i]])
}

# Splitting data to training and test data
n = dim(sdf)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.6))
train=sdf[id,]
test=sdf[-id,]


################################################
# ----------------- Task 2 ----------------- #
################################################
# Fit using all listed voice characteristics on training data
fit = lm(formula=motor_UPDRS~0+.-subject.-age-sex-test_time-total_UPDRS, data=train)

# Show summary of fitted model
summary(fit)

# Print coefficients of the model
coeff <- coefficients(fit)
print(coeff)

# Get predictions on training data
y_hat_train = fitted(fit)

# Estimate training MSE
n_train <- nrow(train)
mse_train <- sum((y_hat_train-train$motor_UPDRS)^2)/n_train

# Get predictions on test data
y_hat_test <- predict(fit, test)

# Estimate test MSE
n_test <- nrow(test)
mse_test <- sum((y_hat_test-test$motor_UPDRS)^2)/n_test


################################################
# ----------------- Task 3 ----------------- #
################################################

# Loglikelihood, according to 3.20 in MLFC (course book)
loglikelihood <- function(theta, sigma, X, Y)
{
  n = length(Y)
```

```
  res = -(n*log(2*pi*sigma^2)/2) - sum((X%*%theta-Y)^2)/(2*sigma^2)
  return(res)
}

# Ridge function, using norm(type=2) to get svd / 2-norm.
ridge <- function(params, lambda, X, Y)
{
  theta <- params[1:ncol(X)]
  sigma <- params[ncol(X)+1]
  return (lambda*norm(theta, type="2") - loglikelihood(theta, sigma, X, Y))
}

# RidgeOpt function,
ridgeopt <- function(lambda, initTheta, initSigma, X, Y)
{

  optimRes <- optim(par=c(initTheta, initSigma), fn=ridge, gr=NULL, lambda, X, Y, method="BFGS
  return(optimRes)
}

# Degrees of freedom function
DF <- function(lambda, X)
{
  P = X %*% solve((t(X)%*%X + lambda*diag(ncol(X)))) %*% t(X)
  return(sum(diag(P)))
}


################################################
# ----------------- Task 4 ----------------- #
################################################

# Setting lambda to 0 gives us almost identical results as the linear
# regression, as it should. So most likely correct solution.

# Use RidgeOpt
lambdas = c(1, 100, 1000)
initTheta <- matrix(0, length(coeff), 1)
initSigma <- 1
X_train <- as.matrix(train[, 7:ncol(train)])
Y_train <- as.matrix(train$motor_UPDRS)
X_test <- as.matrix(test[, 7:ncol(test)])
Y_test <- as.matrix(test$motor_UPDRS)

# Init empty result data frame
res <- data.frame(matrix(ncol = 5, nrow = 0))
colnames(res) <- c("mse_train",
                   "mse_test",
                   "theta_hat",
                   "sigma_hat",
                   "dof")


for(lambda in lambdas) {
  # Find optimal theta and sigma using RidgeOpt
  ridgeRes <- ridgeopt(lambda, initTheta, initSigma, X_train, Y_train)

  # Extract theta and sigma
  theta_hat <- ridgeRes$par[1:length(initTheta)]
  sigma_hat <- ridgeRes$par[length(initTheta)+1]

  # Predict on training data
```

```
  y_hat_train <- X_train%*%theta_hat

  # Compute estimated training MSE
  n_train <- nrow(X_train)
  mse_train <- sum((y_hat_train-Y_train)^2)/n_train

  # Predict on test data
  y_hat_test <- X_test%*%theta_hat

  # Compute estimated test MSE
  n_test <- nrow(X_test)
  mse_test <- sum((y_hat_test-Y_test)^2)/n_test

  # Calculate dof
  dof <- DF(lambda, X_train)

  # Append results
  row <- list(mse_train, mse_test, list(theta_hat), sigma_hat, dof)
  res[nrow(res)+1,] <- row

}

# Print result and set row names
rownames(res) <- lambdas
print(res)
```

A.3 Assignment 3

Below is the code written for assignment 3 of the lab.

```
# Load data and init
df = read.csv("pima-indians-diabetes.csv", header=FALSE)

################################################
# ----------------- Task 1 ----------------- #
################################################

# Split data into diabetes or no diabetes
diabetes = df[df$V9 == 1, ]
no_diabetes = df[df$V9 == 0,]

# Scatter plot of age (x-axis) and plasma glucose concentration (y-axis)
plot(x = diabetes$V8,
     y = diabetes$V2,
     type = "p",
     xlab = "Age␣[years]",
     ylab = "Plasma␣glucose␣concentration",
     xlim = c(min(df$V8), max(df$V8)),
     ylim = c(min(df$V2), max(df$V2)),
     col = "red",
     )
points(x = no_diabetes$V8, y = no_diabetes$V2, col = "blue")
legend(x="bottomright",
       legend = c("diabetes","no␣diabetes"),
       col=c("red", "blue"),
       pch = "oo")



################################################
# ----------------- Task 2 ----------------- #
################################################
```

```r
# Data and parameters
r = 0.5
y = df$V9
n = length(y)
x1 = df$V2
x2 = df$V8
X = data.frame(x1 = x1, x2 = x2)

# Fit model
glm_logistic = glm(formula = y ~ x1 + x2, family = "binomial")

# Make predictions
y_hat = as.numeric(predict(object = glm_logistic, type = "response") > r)

# Create confusion matrix
conf_matrix = table(y_hat, y)

# Misclassification rate
miss_rate = (conf_matrix[1, 2] + conf_matrix[2, 1])/n

# Scatter plot of predictions
plot(x = x2[y_hat == 1],
     y = x1[y_hat == 1],
     type = "p",
     xlab = "Age␣[years]",
     xlim = c(min(df$V8), max(df$V8)),
     ylim = c(min(df$V2), max(df$V2)),
     ylab = "Plasma␣glucose␣concentration",
     main = "Predictions␣for␣r␣=␣0.5",
     col = "red",
)
points(x = x2[y_hat == 0], y = x1[y_hat == 0], col = "blue")
legend(x="bottomright",
       legend = c("predicted␣diabetes","no␣predicted␣diabetes"),
       col=c("red", "blue"),
       pch = "oo")


################################################
# ----------------- Task 3 ----------------- #
################################################

# Plot decision boundary
boundary = (5.91245 - log(1/r - 1) - 0.02478*x2)/0.03564
x2_sorted = sort(x = x2, index.return = TRUE)
boundary_ordered = boundary[x2_sorted$ix]
lines(x = x2_sorted$x, y = boundary_ordered, col = "black")



################################################
# ----------------- Task 4 ----------------- #
################################################

# Set r
r = 0.2

# Make predictions
y_hat = as.numeric(predict(object = glm_logistic, type = "response") > r)

# Create confusion matrix
conf_matrix = table(y_hat, y)
```

```r
# Misclassification rate
miss_rate = (conf_matrix[1, 2] + conf_matrix[2, 1])/n

# Scatter plot of predictions
plot(x = x2[y_hat == 1],
     y = x1[y_hat == 1],
     type = "p",
     xlab = "Age [years]",
     xlim = c(min(df$V8), max(df$V8)),
     ylim = c(min(df$V2), max(df$V2)),
     ylab = "Plasma glucose concentration",
     main = "Predictions for r = 0.2",
     col = "red",
)
points(x = x2[y_hat == 0], y = x1[y_hat == 0], col = "blue")
legend(x="bottomright",
       legend = c("predicted diabetes","no predicted diabetes"),
       col=c("red", "blue"),
       pch = "oo")

# Plot decision boundary
boundary = (5.91245 - log(1/r - 1) - 0.02478*x2)/0.03564
x2_sorted = sort(x = x2, index.return = TRUE)
boundary_ordered = boundary[x2_sorted$ix]
lines(x = x2_sorted$x, y = boundary_ordered, col = "black")


# Set r
r = 0.8

# Make predictions
y_hat = as.numeric(predict(object = glm_logistic, type = "response") > r)

# Create confusion matrix
conf_matrix = table(y_hat, y)

# Misclassification rate
miss_rate = (conf_matrix[1, 2] + conf_matrix[2, 1])/n

# Scatter plot of predictions
plot(x = x2[y_hat == 1],
     y = x1[y_hat == 1],
     type = "p",
     xlab = "Age [years]",
     xlim = c(min(df$V8), max(df$V8)),
     ylim = c(min(df$V2), max(df$V2)),
     ylab = "Plasma glucose concentration",
     main = "Predictions for r = 0.8",
     col = "red",
)
points(x = x2[y_hat == 0], y = x1[y_hat == 0], col = "blue")
legend(x="bottomright",
       legend = c("predicted diabetes","no predicted diabetes"),
       col=c("red", "blue"),
       pch = "oo")

# Plot decision boundary
boundary = (5.91245 - log(1/r - 1) - 0.02478*x2)/0.03564
x2_sorted = sort(x = x2, index.return = TRUE)
boundary_ordered = boundary[x2_sorted$ix]
lines(x = x2_sorted$x, y = boundary_ordered, col = "black")
```

```
################################################
# ----------------- Task 5 ----------------- #
################################################

# Create new features
z1 = x1^4
z2 = x1^3 * x2
z3 = x1^2 * x2^2
z4 = x1 * x2^3
z5 = x2^4

# Set r
r = 0.5

# Fit model
glm_logistic = glm(formula = y ~ z1 + z2 + z3 + z4 +z5 + x1 + x2,
                   family = "binomial")

# Make predictions
y_hat = as.numeric(predict(object = glm_logistic, type = "response") > r)

# Create confusion matrix
conf_matrix = table(y_hat, y)

# Misclassification rate
miss_rate = (conf_matrix[1, 2] + conf_matrix[2, 1])/n

# Scatter plot of predictions
plot(x = x2[y_hat == 1],
     y = x1[y_hat == 1],
     type = "p",
     xlab = "Age␣[years]",
     xlim = c(min(df$V8), max(df$V8)),
     ylim = c(min(df$V2), max(df$V2)),
     ylab = "Plasma␣glucose␣concentration",
     main = "Predictions␣for␣r␣=␣0.5",
     col = "red",
)
points(x = x2[y_hat == 0], y = x1[y_hat == 0], col = "blue")
legend(x="bottomright",
       legend = c("predicted␣diabetes","no␣predicted␣diabetes"),
       col=c("red", "blue"),
       pch = "oo")
```