TDDE01 : Computer lab 1 report

Group B.15

Charlie Elf - chael086 - chael086@student.liu.se
Christoffer Gärdin - chrga084 - chrga084@student.liu.se
Felix Lindgren - felli675 - felli675@student.liu.se

October 26, 2022

Contents

1 Statement Of Contribution

In this lab the main responsibility of the 3 assignments was divided onto the group members. Assignment 1 was held responsible for by Charlie Elf. Assignment 2 was held responsible for by Christoffer Gärdin. Assignment 3 was held responsible for by Felix Lindgren. The load was in that way divided onto the group members. The assignment responsibility included for each group member making sure that code was working to solve the problem and the writing of each assignment in the lab report.

2 Assignment 1 - Handwritten recognition with K-nearest neighbours

The first assignment is about recognizing handwritten digits between 0 - 9. 43 persons has written digits by hand and the writing is converted into a .csv file containing bitmaps of the handwritten digits. The aim is to create a model, using the method of k - nearest neighbours, that can recognize what digit has been written. The models created in this assignment is also evaluated using two different error calculations.

2.1 Data import

The data for this assignment comes in the form of a .csv file containing 3823 rows and 65 columns. Column 1-64 represents the 8 by 8 pixel image were each element is an integer between 0-16. The element value indicates the intensity of the pixel. 0 being the lowest and 16 the highest. In total there is 3823 different (number of rows) images that could be produced from the data. Each row together with column 1-64 representing a different image. Column 65 has the correct digit for each row.
The data is imported to R-studio and divided into 3 sets. Training, test and validation sets with proportions 50%, 25% and 25%. Dividing the data into the different sets is done using a set seed (12345).

2.2 Confusion matrices and missclassification

The training data is used to train the model for the 30-nearest neighbours. This is done for both the training data itself and also for the test data. The training is done using the function ”kknn” with equal weights and hence the kernel ”rectangular” is used as a parameter. The result is shown in confusion matrices in Table 1 for the training data and in Table 2 for the test data.

In 1 we see that the most common missclassification is happening with number ”1” predicted to be ”2” 11 times. There is also some trouble with wrong predictions of ”1” which is actually ”8”. Number ”9” is miss-classified in total 18 times. Most of the times for ”5” but is spread out on multiple actual numbers. The easiest to classify correctly is ”0”, ”4”, ”5” and ”6” with 2 or less wrong missclassifications each. Total missclassification error for the training model is 4.5% which is very good.

The model fit on the test data in Table 2 shows similar results in terms of missclassifications for ”8” as ”1” and ”9” for several classifications. In comparison to the training fit in Table 1 there is an improvement with number ”1” only being predicted as ”2” twice. It could be reasonable to

Table 1: Table showing the training fit to the training data itself for 30-nearest neighbours. Prediction is on the top horizontal axis and true number on the leftmost vertical axis.

	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

Table 2: Table showing the test fit to the training data for 30-nearest neighbours. Prediction is on the top horizontal axis and true number on the leftmost vertical axis.

	0	1	2	3	4	5	6	7	8	9
0	77	0	0	0	1	0	0	0	0	0
1	0	81	2	0	0	0	0	0	0	3
2	0	0	98	0	0	0	0	0	3	0
3	0	0	0	107	0	2	0	0	1	1
4	0	0	0	0	94	0	2	6	2	5
5	0	1	1	0	0	92	2	1	0	5
6	0	0	0	0	0	0	90	0	0	0
7	0	0	0	1	0	0	0	111	0	0
8	0	7	0	1	0	0	0	0	70	0
9	0	1	1	1	0	0	0	1	0	85

think this is due to the data being split randomly and the amount of data not being big enough. This test model is consistent with the numbers being easy to predict in the training model. Total missclassification error for the test model is 5.3% which indicates a very good model.

2.3 Cases of digit ”8”

To find different cases of correct predictions for the digit ”8” and dividing them into easy and hard the correct predictions were extracted. By then sorting the vector based on probability the 2 easiest were found with probability = 1. The 3 hardest had the lowest probabilities (0.428, 0.428, 0.571). These 5 cases were visualized using heatmap for the 8 by 8 pixel images. The result is shown in Figure 1 and 2.

The digit ”8” is visually not much easier for the models easy cases, see Figure 1, than the cases that were hard for the model, Figure 2. The left in the easy cases could easily be mistaken for a different digit, ”1” for example, while the right one is easy to recognize as an 8. The same could arguably be said about the right one in the hard cases being hard to see ”8”. For the middle one you could see the contours of an ”8” but could be mistaken for a different digit, ”9” for example.

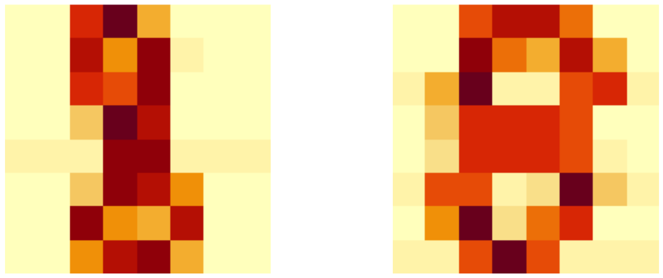

Figure 1: Easy cases of digit ”8”

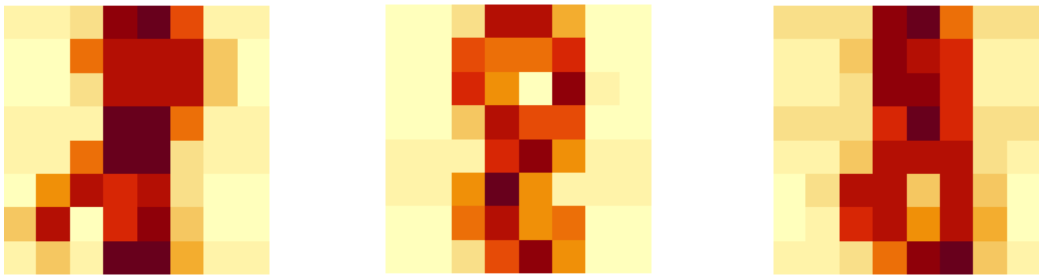

Figure 2: Hard cases of digit ”8”

2.4 Model complexity - missclassification error

To find the optimal k - value for the model both training data and validation data models are created for k - values 1 - 30. Each model for the different data samples result in a missclassification error calculated by using Equation 1. The error is plotted against k - value in Figure 3.

$$Error_{missclass} = 1 - \frac{1}{n}\sum_{i=1}^{n} E(y,\hat{y}), \quad E(y,\hat{y}) = \begin{cases} 0, & y \neq \hat{y} \\ 1, & y = \hat{y} \end{cases} \tag{1}$$

In Figure 3 the training data fits perfect for low k - values since it is fitted to itself. The validation error however is consistent at about an error of 0.03 for k - values up to 8 were it is clearly starting to rise. The training model error rises fast for low k - values and has its lowest validation data missclassification error for k = 3 while still having a low error for training data. Therefore it is reasonable to say that this is the optimal k. A lower k only means that the training data fits better but would lead to a more complex model for the other data samples. A larger k is shown to have larger error and would imply an underfitted and too simple model.

The model with k = 3 is tested against the test data sample and the missclassification error for all 3 data samples are shown in Table 3. The conclusion is that the model works very good for k = 3 since the error is consistent, around 0.025 or below, in between the data samples.

Table 3: Missclassification error-table for optimal k.

	Test	Train	Valid
A	0.0241	0.0115	0.0253

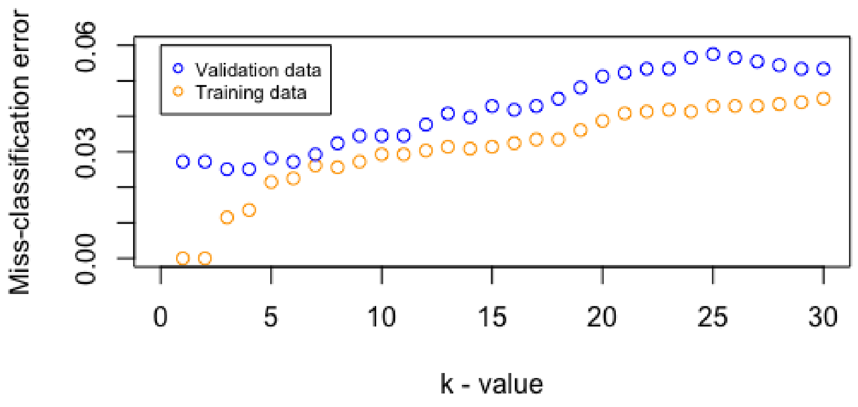

Figure 3: Results for missclassification error with different k - values for both training and validation data samples.

2.5 Cross-entropy error

The cross-entropy error was calculated and with the result the model could be validated for the different k - values. This was done for the validation data and cross-entropy error was calculated using Equation 2. It was then plotted against the k - values 1 to 30.

$$J(y, \hat{p}(y)) = -\sum_{i=1}^{n}\sum_{j=1}^{M} I(y_i = C_m) log(\hat{p}(y_i = C_m)) \tag{2}$$

In Figure 4 we see that the error is very large for small k - values indicating poor models for those k - values. The error is the lowest for k = 6 only to keep rising slowly up to k = 30. It seems reasonable to say that k = 6 is the optimal in this case. Comparing the cross-entropy error in Figure 4 to the missclassification error in Figure 3 it is more clear that a low k - value would not lead to the best model. The cross-entropy error is large while the earlier missclassification error is low for low k. If you only look at the missclassification figure one could argue that a low k - value would be good which is proven with the cross-entropy error to not be the case here.
Cross-entropy would be a better choice for this classification problem since it takes the probabilities into account and makes use of the measure of how likely it is. The way of calculating MSE in this assignment on the other hand is just taking into account whether the prediction is correct or wrong and it doesn't say how much meaning that any wrong prediction is equally bad. For error measurements it would be preferred to have values reflecting the magnitude of the errors in some way which the cross-entropy does better here.

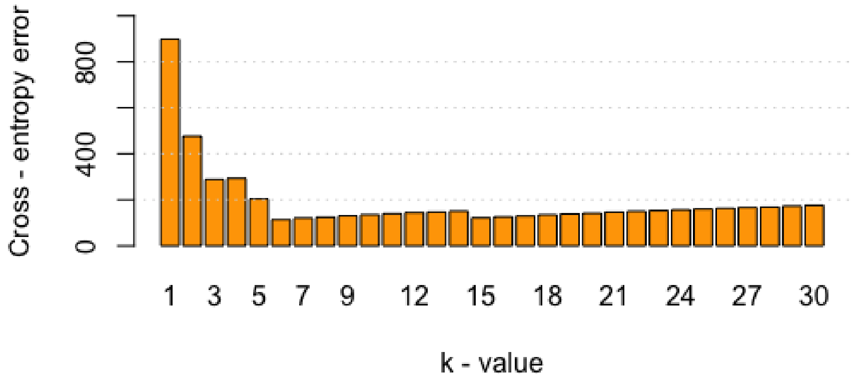

Figure 4: Results for cross - entropy error with different k - values for the validation data sample

3 Assignment 2 - Linear regression and ridge regression

In this assignment, we investigate a range of biomedical voice measurements from 42 people with early Parkinson's disease. The purpose is to predict Parkinson's disease symptom score (motor UPDRS) from different voice characteristics. This is done first by a linear regression model, then with a Ridge model. The Ridge model is based on a log-likelihood estimation that is given a ridge penalty. Features and aversion gets optimized in the log-likelihood function by an optimization function for different values on λ in the ridge penalty. We also compute the degrees of freedom from the Ridge model for a given scalar λ.

3.1 Scaling and partition of data

In the preprocessing part, we started with scaling the data to decrease the spread between the different features. Otherwise, the spread will make the learning process unstable due to large weight values. The data chosen was the different input variables (voice characteristics) and the output variable ($motor\_updrs$) that was going to be predicted. That meant we had to exclude the first four columns: $subject$, age, sex, and $test\_time$. This data was then split into training data(60%) and test data(40%).

3.2 Linear regression model

By creating a regression model based on the training data and the output variable set as $motor\_updrs$ we can generalize and predict the value for $motor\_updrs$ for the test data and training data. The Mean Squared Error (MSE) was calculated by taking the mean square of the real value $-$ the predicted value:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2 \tag{3}$$

In Table 4 it is possible to see the estimated training and test MSE.

Table 4: MSE for the training data and test data.

	Training data	**Test data**
MSE	0.873147	0.9297021

We used the variables p-value to determine which variables contribute significantly to our regression model. The p-value indicates if the variable is not relevant, so we want that value to be as small as possible. In Table 5 we can see the variables that have the smallest p-value, which means that they have a low probability of not being relevant to predict the variable *motor_updrs*. For example, we can see that $Shimmer.APQ11$, DFA and HNR are very significant to predict the *motor_updrs* score.

Table 5: P-values for the regression model.

	P-value	**Significance level**
Jitter(Abs)	8.63e-10	***
Shimmer.APQ5	0.045278	***
Shimmer.APQ11	< 2e-16	***
NHR	1.34e-06	**
HNR	9.06e-09	***
DFA	< 2e-16	***
PPE	3.31e-12	***

3.3 Function implementation

In this task, we have implemented our four own functions to optimize parameters, use ridge regression, and compute degrees of freedom.

3.3.1 Loglikelihood

The parameters for a linear regression model can also be estimated by a maximum likelihood estimation. The logarithm is a monotonically increasing function, that leads to maximizing the log-likelihood is the same thing as maximizing the likelihood. It is preferable to minimize the function rather than maximize it, therefore we use the negative log-likelihood. To minimize the negative log-likelihood given the parameter vector θ and dispersion σ for the stated model and training data we can derive the following equation:

$$-\,lnP(T|\theta,\sigma) = -\left(-\frac{n}{2}ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(\mathbf{X}\theta - \mathbf{Y})^2\right) \tag{4}$$

3.3.2 Ridge

In this function, we have added a Ridge penalty to the minus log-likelihood estimation. It means that we can reduce model complexity by shrinking the parameters. The value on λ determines how large the penalty is going to be: $\lambda||\theta||^2$.

3.3.3 RidgeOpt

The RidgeOpt function uses the Ridge function in section 3.3.2 and the optim function with method=”BFGS” to optimize the parameters θ and σ for the given λ.

3.3.4 DF

In a ridge regression setting, we can calculate the degrees of freedom for a given lambda by the following equation:

$$df(\lambda) = tr(\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda I_p)^{-1}\mathbf{X}^T) \tag{5}$$

3.4 Function result

By using the functions implemented in section 3.3 we can compute the optimal θ and σ for different values on λ and DF for the ridge regression setting. By looking at Table 6 we can see that $\lambda = 1$ gives the lowest MSE for the test data. So by adding this penalty term we get a slightly better result by shrinking the parameters. We can also observe that the degrees of freedom in the ridge regression are monotone decreasing in λ for both the test data and training data. Lower degrees of freedom leads to imprecise estimates and low statistical power, so the ridge penalty for large values on λ makes the predictions worse as the parameters shrinks towards zero.

Table 6: MSE and DF for different values on λ.

	$\boldsymbol{\lambda = 1}$	$\boldsymbol{\lambda = 100}$	$\boldsymbol{\lambda = 1000}$
Training MSE	0.873277	0.8790571	0.915628
Test MSE	0.9108905	0.9191834	0.9579847
Training DF	13.86281	9.939085	5.643351
Test DF	13.78043	9.182898	4.821676

4 Assignment 3 - Logistic regression and basis function expansion

4.1 Plot plasma glucose and age

The data was imported from the given csv file, all the points were used for training. In Figure 5 we can see the plot of ages and glucose concentration, the coloring depends on if the person has diabetes or not. The data seems to be quite mixed, this makes predicting who has diabetes quite tough, since there is no clean way to separate the two groups.

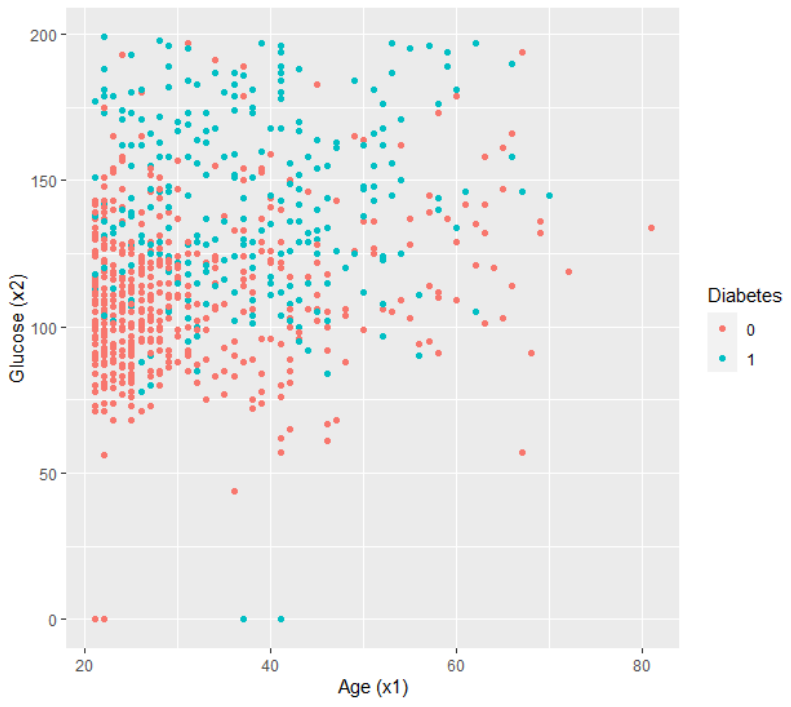

Figure 5: Scatterplot of the raw data showing age and glucose levels, coloring indicates if they have diabetes.

4.2 Train logistic regression model

A logistic regression model was trained on the data with target y =Diabetes and features x_1 =Age and x_2=Plasma glucose. The classification threshold was set to $r = 0.5$. The equation for the probabilities is

$$p = \frac{1}{1 + e^{-(-5.9124+0.0248x_1+0.0356x_2)}} \tag{6}$$

The scatterplot showing the predictions is shown in Figure 6 below. This models gets a 26.302% missclassification error which is not very good, one in four being wrong seems quite high for a potentially serious illness like diabetes.

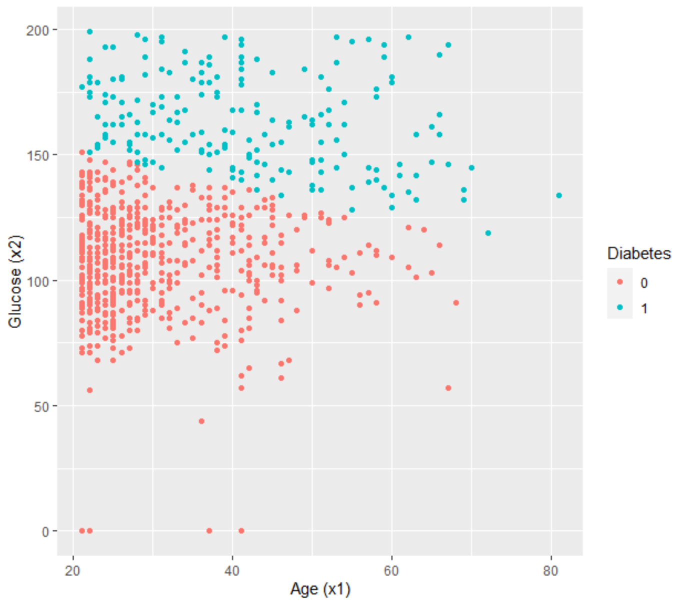

Figure 6: Scatterplot showing prediction of diabetes.

4.3 Showing decision boundary and equation

The decision boundary can be represented by using the parameters from the question, which gives the line $x_2 = -0.695x_1 + 165.874$, this curve is shown in Figure 7. The curve clearly shows where the data is classified.

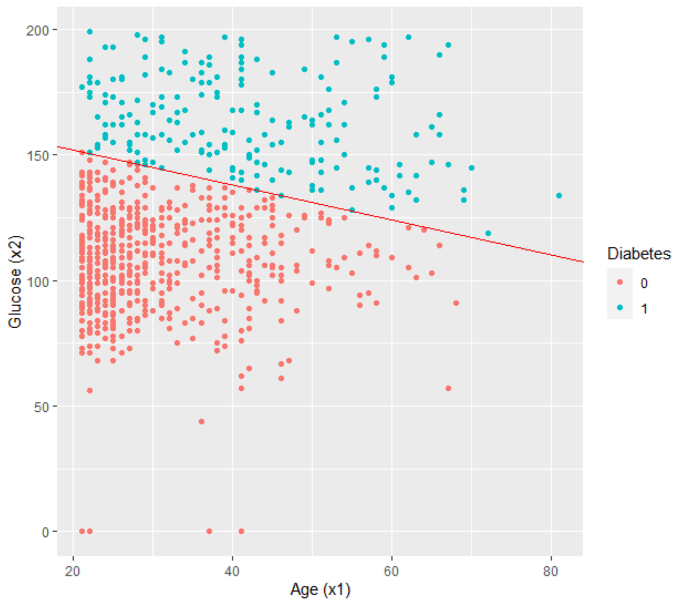

Figure 7: Scatterplot of the data with a line showing the decision boundary.

4.4 Test different thresholds

Figure 8 and 9 shows the classifications for different thresholds of r. Figure 8 shows $r = 0.2$, which increases the amount of points classified as diabetic, while in Figure 9 we see $r = 0.8$ that makes the classification of a diabetic case more strict. We can see that the decision boundary seems to keep it's shape but gets translated along the x_2 axis.

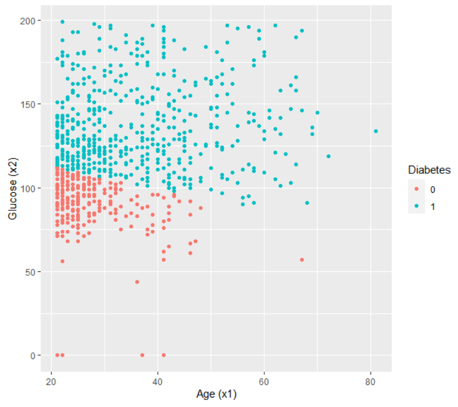

Figure 8: Classification for $r = 0.2$

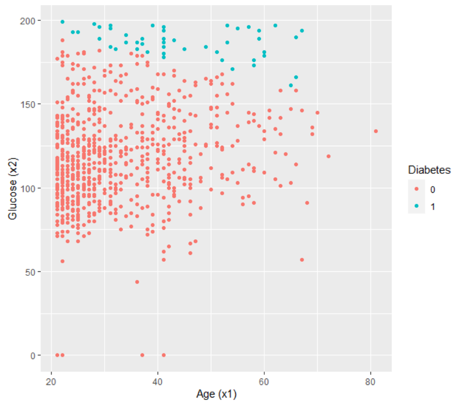

Figure 9: Classification for $r = 0.8$

4.5 Basis function expansion

A basis function expansion was performed to add new features, the new features were as follows:

$$\begin{cases} z_1 = x_1^4 \\ z_2 = x_1^3 x_2 \\ z_3 = x_1^2 x_2^2 \\ z_4 = x_1 x_2^3 \\ z_5 = x_2^4 \end{cases} \tag{7}$$

A new model was trained with the previous features, glucose and age, along with the new ones. Figure 10 shows the scatterplot of the new prediction and the missclassification of the model was 24.479%. The model complexity is increased by adding these new artificial features, however it performs better so the quality seems to be improved. As can be seen in the plot, this decision boundary is no longer linear but has a convex shape. When comparing to the real data it seems to fit a bit better which also shows in the missclassification since it decreased compared to the original model.

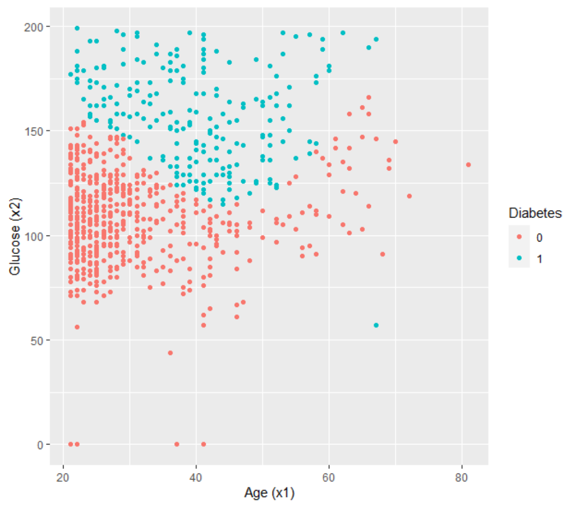

Figure 10: Shows the prediction of the model that used basis expansion.

A Source code

A.1 Assignment 1

```r
#Assignment 1
#missclass function
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n) }
#Task 1.1
library(kknn)
library(ggplot2)
library(xtable)
Dataframe=read.csv("optdigits.csv", header = FALSE)
Dataframe$V65 = as.factor(Dataframe$V65)
n = dim(Dataframe)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = Dataframe[id,]

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

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

#Task 1.2
model_1 = kknn(as.factor(V65) ~., train, test, kernel = "rectangular", k = 30)
model_2 = kknn(as.factor(V65) ~., train, train, kernel = "rectangular", k = 30)
fit <- fitted(model_1)
fit_2 <- fitted(model_2)

conf_test = table(test$V65, fit)
conf_train = table(train$V65, fit_2)
print(xtable(conf_test, type = "latex"), file = "conf_test.tex")
print(xtable(conf_train, type = "latex"), file = "conf_train.tex")
missclass(test$V65, fit)
missclass(train$V65, fit_2)

#Task 1.3

fig_matrix_2 = matrix(as.numeric(train[211,1:64]), 8, 8, byrow=TRUE)
heatmap(fig_matrix_2, Rowv = NA, Colv = NA)

index = 1
prob_vector = c()
index_vector = c()
for (i in fit_2)
{
  if (i == 8 && train[index,65] == 8){
    prob_vector <- c(prob_vector, model_2$prob[index,9])
    index_vector <- c(index_vector, index)
  }
  index = index + 1
}

prob_vector_sorted = sort(prob_vector, index.return=TRUE)
hard_vector = prob_vector_sorted$ix[1:3]
easy_vector = prob_vector_sorted$ix[187:188]

# Easy 8:s
```

```r
fig_matrix_1 = matrix(as.numeric(train[index_vector[easy_vector[1]],1:64]), 8, 8,
    byrow = TRUE)
heatmap(fig_matrix_1, Rowv = NA, Colv = NA)

fig_matrix_2 = matrix(as.numeric(train[index_vector[easy_vector[2]],1:64]), 8, 8,
    byrow = TRUE)
heatmap(fig_matrix_2, Rowv = NA, Colv = NA)
# Hard 8:s
fig_matrix_3 = matrix(as.numeric(train[index_vector[hard_vector[1]],1:64]), 8, 8,
    byrow = TRUE)
heatmap(fig_matrix_3, Rowv = NA, Colv = NA)

fig_matrix_4 = matrix(as.numeric(train[index_vector[hard_vector[2]],1:64]), 8, 8,
    byrow = TRUE)
heatmap(fig_matrix_4, Rowv = NA, Colv = NA)

fig_matrix_5 = matrix(as.numeric(train[index_vector[hard_vector[3]],1:64]), 8, 8,
    byrow = TRUE)
heatmap(fig_matrix_5, Rowv = NA, Colv = NA)

# Task 1.4
error_vector_train = c()
error_vector_valid= c()
k_vector = c(1:30)
for (i in k_vector){
  model_3 = kknn(as.factor(V65) ~., train, train, kernel = "rectangular", k = i)
  fit_3 <- fitted(model_3)
  error_vector_train <- c(error_vector_train, missclass(train$V65, fit_3))

  model_4 = kknn(as.factor(V65) ~., train, valid, kernel = "rectangular", k = i)
  fit_4 <- fitted(model_4)
  error_vector_valid <- c(error_vector_valid, missclass(valid$V65, fit_4))
}

plot(k_vector, error_vector_train, col = "orange",xlim = c(0,30), ylim = c(0,0.06),,
     xlab = "k - value", ylab = "Miss-classification error")
points(k_vector, error_vector_valid, col = "blue")
legend(0, 0.06, legend=c("Validation data", "Training data"),
       col=c( "blue", "orange"), cex=0.7, pch = 1)
# Estimate test error for optimal k and compare to train and valid.
model_5 = kknn(as.factor(V65) ~., train, test, kernel = "rectangular", k = 3)
model_6 = kknn(as.factor(V65) ~., train, train, kernel = "rectangular", k = 3)
model_7 = kknn(as.factor(V65) ~., train, valid, kernel = "rectangular", k = 3)
fit_5 <- fitted(model_5) # test
fit_6 <- fitted(model_6) # train
fit_7 <- fitted(model_7) # valid

test_error = missclass(test$V65, fit_5)
train_error = missclass(train$V65, fit_6)
valid_error = missclass(valid$V65, fit_7)

#Creating error table
error_table <- matrix(c(round(test_error, digits = 4), round(train_error, digits =
    4), round(valid_error, digits = 4)), ncol = 3, byrow = TRUE)
colnames(error_table) <- c("Test", "Train","Valid")
error_table <- as.table(error_table)
print(xtable(error_table, type = "latex"), file = "table_1_4.tex")
# Task 1.5

#cross entropy

crossentropy <- function(p, phat){
  x <- 0
  for (i in 1:length(p)){
    x <- x + log(phat[i,p[i]] + 1e-15)
  }
  return (-x)
}

HP_val= c(NULL)
HP_test= c(NULL)
for (i in 1:30) {
  model_8 <- kknn(V65 ~ ., train, valid, kernel = "rectangular", k = i)
  fit_8 <- fitted(model_8)
  HP_val = append(HP_val,crossentropy(as.numeric(valid$V65), model_8$prob))
}
hp_data = data.frame(
  sup=rep(c("Val"), each=30),
  len=c(HP_val, HP_test),
  val=rep(c(1:30),1))

barplot(hp_data$len, axisnames = TRUE, names.arg = hp_data$val, ylim = c(0, 1000),
    xlab = "k - value", ylab = "Cross - entropy error", col = "orange")
grid(nx = 0, ny = 5)
```

A.2 Assignment 2

```r
#caTools used for splitting data into training and test
library(caTools)
#psych used for matrix operations
library(psych)

setwd("/Users/christoffergardin/Desktop/TDDE01/Lab 1")

#1.
#reads file into data frame
df <- read.csv('parkinsons.csv')

#scale and include the data except from the columns "subject, age, sex, test_time,
    Total_UPDRS"

dfscale <- scale(df[c(-1,-2,-3,-4,-6)])
dfscale <- as.data.frame(dfscale)

#split data into training (60%) and test (40%)
#sample <- sample.split(dfscale$motor_UPDRS, SplitRatio = 0.6)
#train <- subset(dfscale, sample == TRUE)
#test <- subset(dfscale, sample == FALSE)
n=dim(dfscale)[1]
set.seed(12345)
id=sample(1:n,floor(n*0.6))
train=dfscale[id,]
test=dfscale[-id,]

#2.
#create a regression model based on the training data and the predicted variable
    motor_UPDRS
model <- lm(motor_UPDRS ~ ., data = train )
#print(summary(model))
#print(fitted(model))

#predict motor_UPDRS
pred.test <- predict(model,test)
pred.training <- predict(model,train)

#Calculating mean squared error test data
results.test <- cbind(pred.test, test$motor_UPDRS)
```

```r
colnames(results.test) <- c('pred','real')
results.test <- as.data.frame(results.test)
mse.test <- mean((results.test$real - results.test$pred)^2)
cat(paste("MSE test: "), mse.test,"\n")

#Calculating mean squared error training data
results.training <- cbind(pred.training, train$motor_UPDRS)
colnames(results.training) <- c('pred','real')
results.training <- as.data.frame(results.training)
mse.training <- mean((results.training$real - results.training$pred)^2)
cat(paste("MSE training:"), mse.training,"\n")

#3.
#a)
#computes the log-likelihood function for training data given parameters theta1 and
    sigma1
Loglikelihood <- function(training.x, training.y, theta1, sigma1) {
  loglik <- -(length(training.y)/2) * log(2*pi*(sigma1)^2) - (1/(2*(sigma1)^2)) *
    sum((training.x%*%theta1 - training.y)^2)
  return(-loglik)
}

#b)
#adds a ridge penalty to the log-likelihood
Ridge <- function(training.x, training.y, par, lambda) {
  theta1 <- as.matrix(par[1:16])
  sigma1 <- as.matrix(par[17])
  ridge.loglik <- lambda*sum(theta1^2) + Loglikelihood(training.x, training.y,
    theta1, sigma1)
  return(ridge.loglik)
}

#c)
#optimize paramters(par) in the Ridge function
RidgeOpt <- function(training.x, training.y, par, lambda) {
  ridge.opt <- optim(par, Ridge, training.x = training.x, training.y = training.y,
    lambda = lambda, method = c("BFGS"))
  return(ridge.opt)
}

#d)
#compute degrees of freedom
DF <- function(x.data, lambda) {
  df <- tr(x.data%*%solve(t(x.data)%*%x.data + lambda * diag(ncol(x.data)))%*%t(x.
    data))
  return(df)
}

#initializing variables

#training data, all columns except the first one
training.x = as.matrix((train[,-1]))
test.x = as.matrix((test[,-1]))
#predicted column motor_UPDRS
training.y = as.matrix((train[,1]))
test.y = as.matrix((test[,1]))
#init features
#theta = rnorm(ncol(training.x))
theta = sample(c(rep(0,8),rep(1,8)))
#init sigma
sigma = runif(1)
#parameters to optimize
par = c(theta,sigma)

```

```r

#computing optimal parameters, predicting motor_UPDRS and calculating
#degrees of freedoom for lambda = 1, lambda = 100, lambda = 1000
lambdas = c(1,100,1000)
for (lambda in lambdas){
  opt.training <- RidgeOpt(training.x, training.y, par, lambda)
  opt.test <- RidgeOpt(test.x, test.y, par, lambda)
  mse.opt.training <- mean((training.x%*%opt.training$par[1:16] - training.y)^2)
  mse.opt.test <- mean((test.x%*%opt.test$par[1:16] - test.y)^2)
  df.training <- DF(training.x, lambda)
  df.test <- DF(test.x, lambda)

  cat("MSE opt. training, lambda = ",lambda,":", mse.opt.training, "\n")
  cat("MSE opt. test, lambda = ",lambda,":", mse.opt.test, "\n")
  cat("DF training, lambda = ",lambda,":", df.training, "\n")
  cat("DF test, lambda = ",lambda,":", df.test, "\n")
}
```

A.3 Assignment 3

```r
library(ggplot2)
#
data=read.csv2("pima-indians-diabetes.csv", dec=".", sep=",",  header=FALSE,
    stringsAsFactors=FALSE)
d = data.frame(glucos=data$V2, age=data$V8, diabetes=data$V9)

missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#3.1
p <- qplot(glucos, age, data=d, color=factor(diabetes))+
xlab("Glucose (x1)")+
ylab("Age (x2)")
p$labels$colour <- "Diabetes"
p
# 3.2
glm.fit <- glm(diabetes ~ glucos+age, data=d,family=binomial)
glm.probs <- predict(glm.fit, newdata = d, type="response")
glm.pred <- ifelse(glm.probs > 0.5, "1", "0")

p1 <- qplot(glucos, age, data=d, color=glm.pred)+
  xlab("Glucose (x1)")+
  ylab("Age (x2)")
p1$labels$colour <- "Diabetes"
p1
missclass(d$diabetes, glm.pred) # 0.2630208

# 3.3
coeffs = coef(glm.fit)
slope = -coeffs[2]/coeffs[3]
int = -coeffs[1]/coeffs[3]

p2 <- ggplot(d, aes(x=glucos, y=age, color=glm.pred)) + geom_point() +
  geom_abline(intercept = int, slope = slope, color ="red")+
  xlab("Glucose (x1)")+
  ylab("Age (x2)")
p2$labels$colour <- "Diabetes"
p2
#3.4
glm.pred <- ifelse(glm.probs > 0.2, "1", "0")
p4<- qplot(glucos, age, data=d, color=glm.pred)+
  xlab("Glucose (x1)")+
  ylab("Age (x2)")
p4$labels$colour <- "Diabetes"
p4
glm.pred <- ifelse(glm.probs > 0.8, "1", "0")
p5<- qplot(glucos, age, data=d, color=glm.pred)+
  xlab("Glucose (x1)")+
  ylab("Age (x2)")
p5$labels$colour <- "Diabetes"
p5
# We see that the when we change r the slope remains the same
# but the intercept changes, thus requiring the model to be more
# or less sure if the person has diabetes or not.

#3.5
z_data = data.frame(z1=d$glucos**4,
                    z2=(d$glucos**3)*(d$age**1),
                    z3=(d$glucos**2)*(d$age**2),
                    z4=d$glucos*d$age**3,
                    z5=d$age**4)
basis_data <- cbind(d, z_data)

glm.fit <- glm(diabetes ~ glucos+age+z1+z2+z3+z4+z5,
               data=basis_data, family=binomial)
glm.probs <- predict(glm.fit, newdata = basis_data, type="response")
glm.pred <- ifelse(glm.probs > 0.5, "1", "0")

p6 <- qplot(glucos, age, data=basis_data, color=glm.pred)+
  xlab("Glucose (x1)")+
  ylab("Age (x2)")
p6$labels$colour <- "Diabetes"
p6
missclass(basis_data$diabetes, glm.pred)  #  0.2447917
# We get a polynomial regression classification instead,
# the missclassification is increased which could be because of
# overfitting from the high degree of polynomial used.
```