TDDE01 - Lab 1

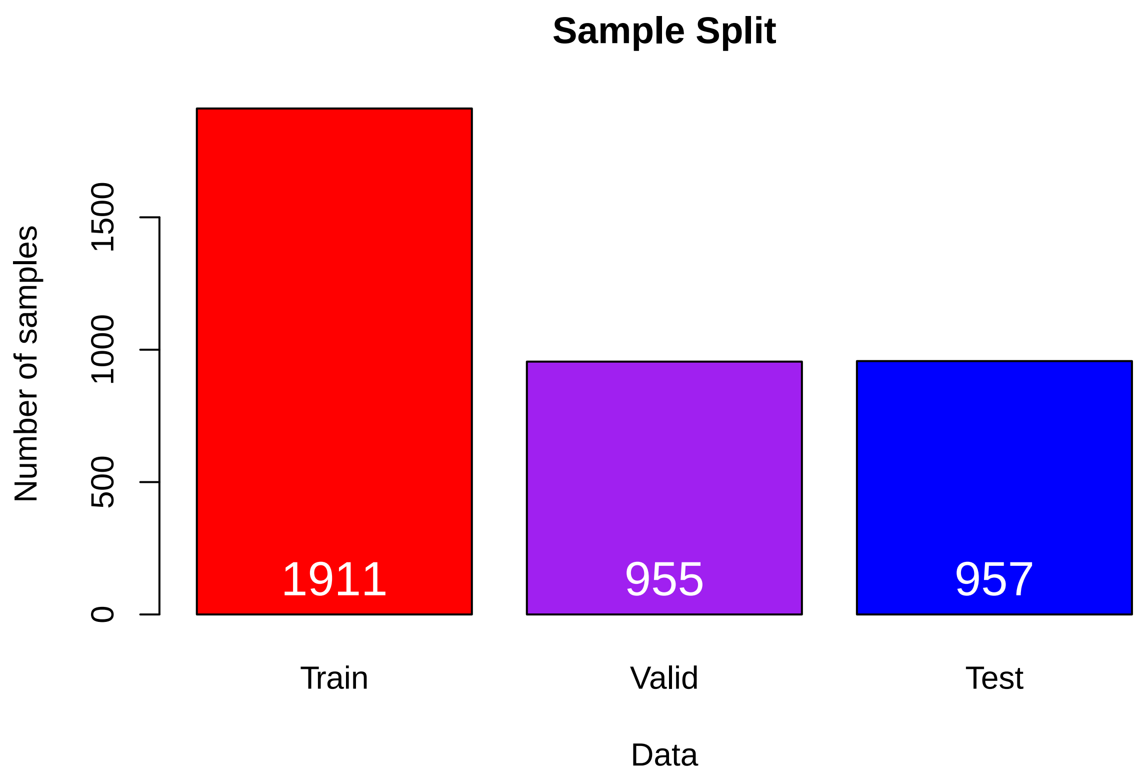
Olof Simander (olosi122), Gustaf Lindgren (gusli281), Anton Jervebo (antje840)

Statement of contribution: All members of the group assisted with the programming and the writing for the first 3 tasks of assignment 1. Then we split the tasks so Olof took assignment 1, Gustaf assignment 2, and Anton assignment 3. We assisted each other during the lab sessions with coding and problem solving. Each member has done the writing of the report on respective assignment. All results have been discussed and adjusted based on input from the group before hand in. Olof did the revision with regard to coding and writing after the first remark and have discussed the results with the group before hand in.

Assignment 1:

Step 1

- Import the data into R and divide it into training, validation and test sets (50%/25%/25%)



Step 2

- Use training data to fit 30-nearest neighbor classifier with function kknn() and kernel="rectangular" from package kknn and estimate .
- Confusion matrices for the training and test data.
- Misclassification errors for the training and test data.
- Comment on the quality of predictions for different digits and on the overall prediction quality.

Comparing the number of faults is easiest done by the absolute predictions the algorithm gets wrong and the proportion of error compared to number of samples.

The largest number of errors are found in predicting the training data. However, this is to be expected since it has a larger sample size. **The more important point is that the rate of misclassifications is slightly higher for the testing data: 5.329154% compared to 4.500262% for training data.** These are both relatively good at predicting the outcome.

The largest difference in proportional performance for a specific outcome with regard to **misclassifications on training data is "4" with 9.14%, and on the test data is "5" is 9.709%**. The model has some variation, as both the training and test data is predicted perfectly for some outcomes in both data sets. For example, the number "0" has low misclassification rates for both data sets. The conclusion is that the model does predict certain outcomes better, such as "0", than others, such as "4" or "5". This might be because of the varying number of data points for each outcome or because of the difficulty of predicting the actual data in the bitmap.

```
## The confusion matrix for the training data is:

##    Pred_train
##        0   1   2   3   4   5   6   7   8   9
##   0 202   0   0   0   0   0   0   0   0   0
##   1   0 179  11   0   0   0   0   1   1   3
##   2   0   1 190   0   0   0   0   1   0   0
##   3   0   0   0 185   0   1   0   1   0   1
##   4   1   3   0   0 159   0   0   7   1   4
##   5   0   0   0   1   0 171   0   1   0   8
##   6   0   2   0   0   0   0 190   0   0   0
##   7   0   3   0   0   0   0   0 178   1   0
##   8   0  10   0   2   0   0   2   0 188   2
##   9   1   3   0   5   2   0   0   3   3 183

##
##  The misclassification rate for each number is:
##  0.04500262

## The misclassification for all numbers is:
##  0.0000 0.0821 0.0104 0.0160 0.0914 0.0552 0.0104 0.0220 0.0784 0.0850

## The confusion matrix for the test data is:

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

```
##
##  The misclassification rate for each number is:
##  0.05329154
```

```
## The misclassification for all numbers is:
##  0.01282 0.05814 0.02970 0.03604 0.13761 0.09709 0.00000 0.00893 0.10256 0.04494
```

```
## The difference in missclassification for each number in rising order is:
##   0.0128 -0.0239  0.0193  0.0201  0.0462  0.0418 -0.0104 -0.0130  0.0241 -0.0401
```

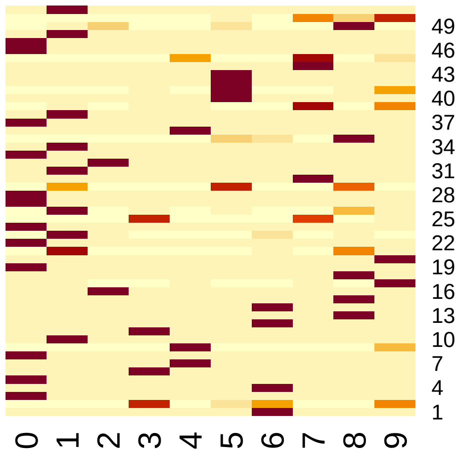
Step 3

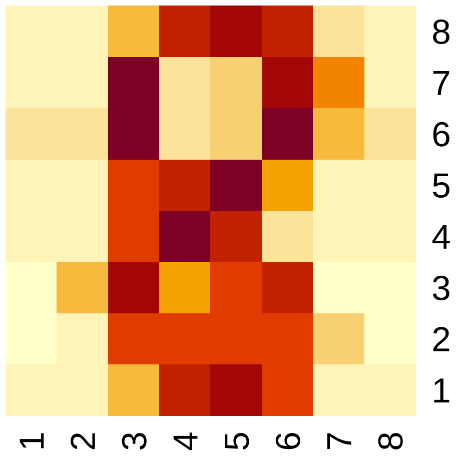
- Find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify.
- Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits.
- Comment on whether these cases seem to be hard or easy to recognize visually.

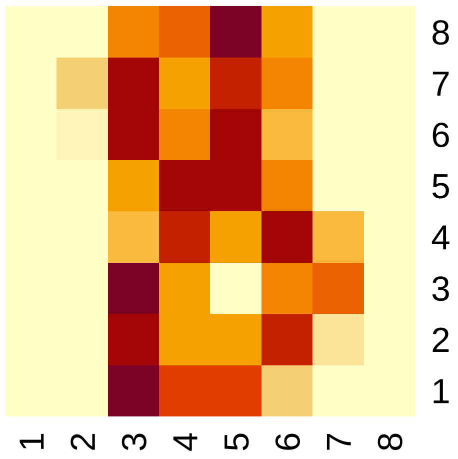
As seen in the pictures below, the first two pictures does have much more of the resemblance of the number eight. This mainly comes down to the look of having two circles on top of each other with lower pixel vales in the centers of them.

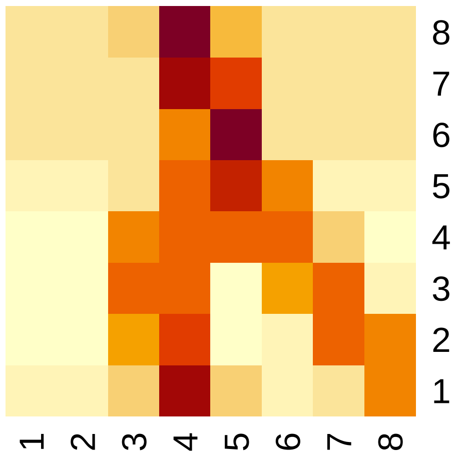
The latter three pictures do not resemble the number eight.

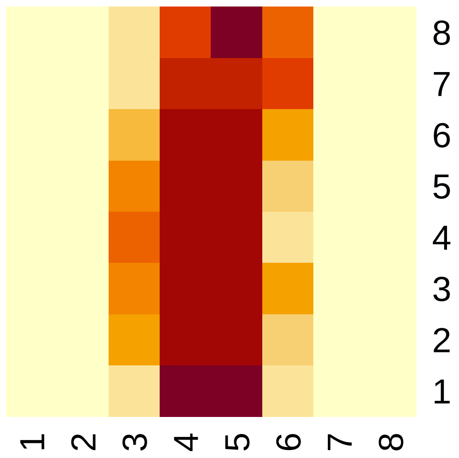
- The first of which takes on more of a lambda shape, not having any circles.
- The second to last number correlates strongly with the number 1. The difference in the picture is the clear lack of two distinct circles as well as now hollow center in the middle of the number.
- The very last number of which much more resemble the number three. It should be noticed that the number three and eight have somewhat similar characteristics, the oversimplified difference being the the number three is made of stacked half circles. The reason for the model not predicting these numbers to be an eight is because the strong correlation with the number 3.

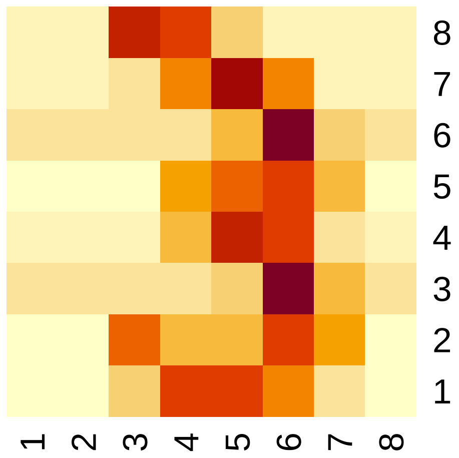

49
46
43
40
37
34
31
28
25
22
19
16
13
10
7
4
1
0 1 2 3 4 5 6 7 8 9


8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8


8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8


8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8





Step 4

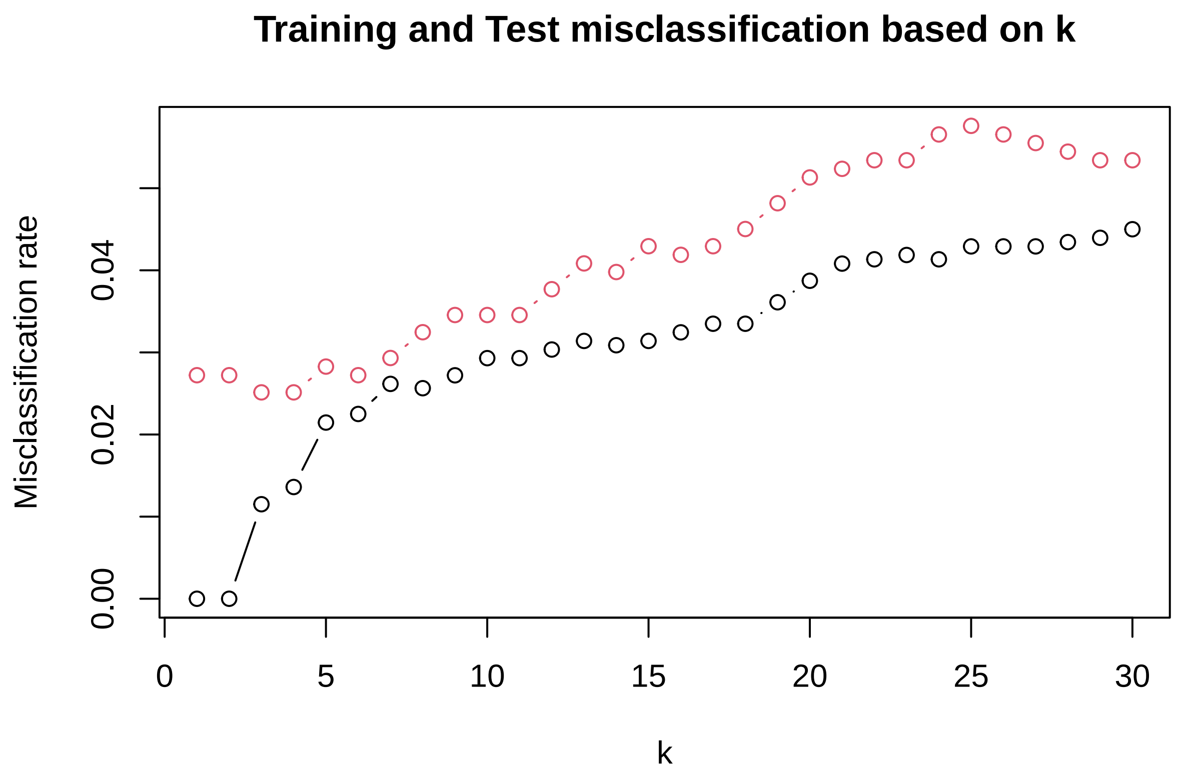
- Fit a K-nearest neighbor classifiers to the training data for different values of K = 1,2, ... , 30 and plot the dependence of the training and validation misclassification errors on the value of K.
- How does the model complexity change when K increases and how does it affect the training and validation errors?
- Report the optimal K according to this plot.
- Finally, estimate the test error for the model having the optimal K, compare it with the training and validation errors and make necessary conclusions about the model quality.

The best result for the validation data is found in both k = 3 and k = 4 where the lowest number of misclassifications exists. They have an error rate of 2.513089% against the validation data.

Since we want to minimize unnecessary complexity and the bias of the model, k = 4 will be the superior choice. **The missclassification rates are: 1.360544% for the training data and 2.513089% for the validation data, and 0.02507837 for the test data.**

The higher number for K means that more data points will be included in the calculation and the result and the model will be less sensitive to variation in the data - higher bias. Naturally, for very small number of k such as 1 and 2, the model will predict the training data with no misclassifications, as it only predicts the correct number according to the training data - high variance. After about k = 5, the misclassification increases approximately linearly because of the inclusion of different outcomes in the prediction. The misclassification for the validation data is 0.02722513 for low numbers of k, such as 1 and 2. It does not go 0 because of the inherent misclassification even if only considering a single data point at a time. Beyond that, it behaves like the training data for the same reason.

The models aptitude depends highly on the application of it. Presumably, a 0.02507837 misclassification rate, based of the test data, with this relatively simple model could be useful in some applications where the given misclassification rate is tolerable. A ~2.5% misclassification rate might even be considered small for interpreting numbers written by hand.



```
## Training data:

## k = 4: 0.01360544

## Validation data:

## k = 3: 0.02513089
##  k = 4: 0.02513089

## Test data:
##  k = 4:  0.02507837
```

Step 5

For the cross-entropy loss function we naturally want to minimize the minus log likeleyhood.

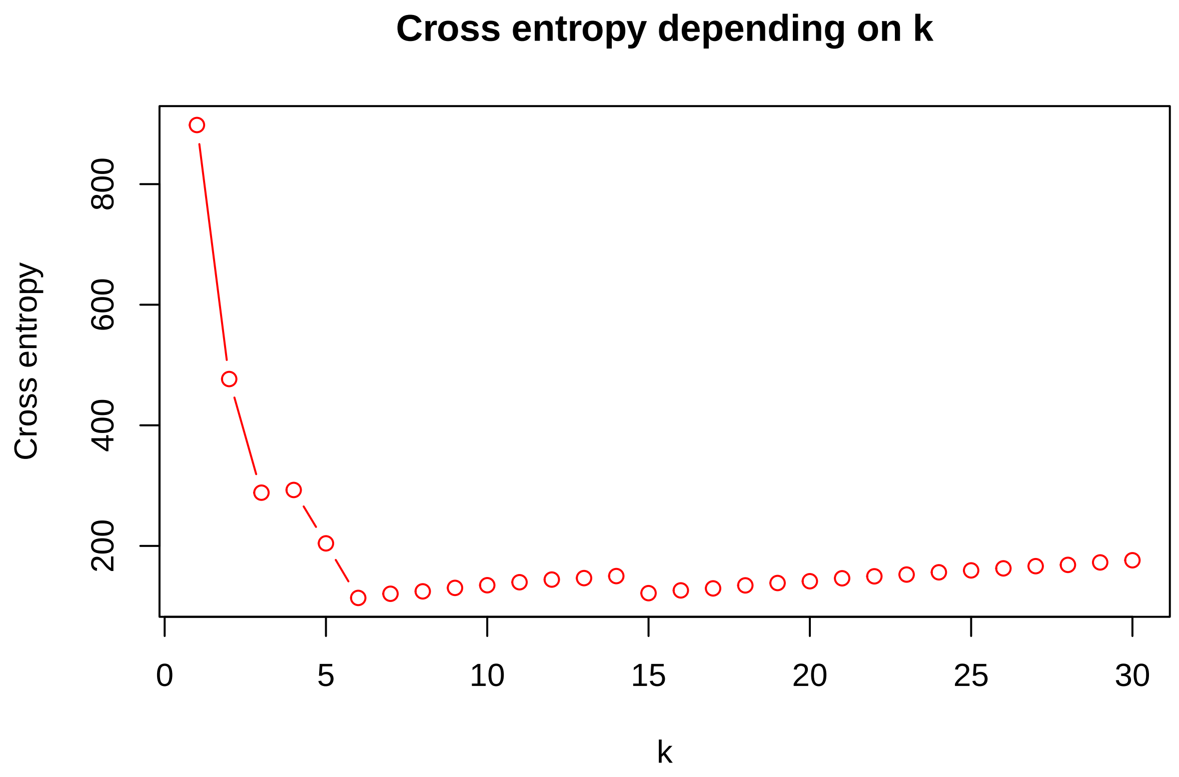
$$R\big(Y, \hat{p}(Y)\big) = -\sum_{i=1}^{N}\sum_{m=1}^{M} I(Y_i = C_m)\log\hat{p}(Y_i = C_m)$$

The best result is now found to be k = 6, which is close but not the same as the previous conclusion from the misclassification rate

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

The reason cross entropy might be more suitable to use as the error function is essentially that it is more exact. With a multinomial distribution, there could be many solutions for the model where the misclassifications are exactly the same the same for either the training or the validation data. By using cross-entropy as the error function, we can choose the best of the viable alternatives.

Another important difference is the value of the probabilities. With cross entropy, we consider the "conviction" of the model in a certain outcome. What can happen in the case of misclassification is a situation where all probabilities are low, but the model still gets the correct outcome. In this case, the model is not robust as much as it was lucky.



```
## The best k is:  6 with a value of  113.768
```

Appendix 1

```
library(knitr)
<style>
body {
text-align: justify}
</style>
# Read the csv file
optdigits = read.csv("optdigits.csv", header = FALSE)
n=dim(optdigits)[1]

# Divide the data 50/25/25 for train, validation, and test
set.seed(12345)
id=sample(1:n, floor(n*0.5))
```

```r
train=optdigits[id,]
id1=setdiff(1:n, id)

set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=optdigits[id2,]

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

barpl = barplot(c(Train = nrow(train),Valid = nrow(valid) , Test = nrow(test)),
                main = "Sample Split",
                xlab = "Data", ylab = "Number of samples",
                col = c("Red", "Purple", "Blue"))
text(barpl,0, c(Train = nrow(train), Valid = nrow(valid), Test = nrow(test)),
     cex = 1.5, pos = 3, col = "white")
library(kknn)
#Training data
m1 = kknn(as.factor(V65)~., train, train, k=30, kernel = "rectangular")

Pred_train = m1$fitted.value

T_train = table(train$V65, Pred_train)
cat("The confusion matrix for the training data is: ",
    "\n")
T_train

# The rate of misclassification(s) for all outcomes
cat("\n",
    "The misclassification rate for each number is: ",
    "\n", sum(rowSums(T_train) - diag(T_train)) / sum(T_train),
    "\n")

# The rate of misclassification for each outcome
misclassr_train = 1 - (diag(T_train) / rowSums(T_train))
cat("The misclassification for all numbers is: ",
    "\n", format(misclassr_train, digits = 3))
#Test data
m2 = kknn(as.factor(V65)~., train, test, k=30, kernel = "rectangular")

Pred_test = m2$fitted.value

T_test = table(test$V65, Pred_test)
cat("The confusion matrix for the test data is: ",
    "\n")
T_test

# The rate of misclassification(s) for all outcomes
cat("\n",
    "The misclassification rate for each number is: ",
    "\n", sum(rowSums(T_test) - diag(T_test)) / sum(T_test),
    "\n")
```

```r
# The rate of misclassification for each outcome
misclassr_test = 1 - (diag(T_test)/ rowSums(T_test))
cat("The misclassification for all numbers is: ",
    "\n", format(misclassr_test, digits = 3))
# The difference in performance on each digit
cat("The difference in missclassification for each number in rising order is: ",
    "\n", (format(misclassr_test - misclassr_train, digits = 3)))
# Subset of the data
heatmap(as.matrix(m1$prob[950:1000,]), Colv = NA, Rowv = NA)

# High prob
p1 = matrix(as.numeric(train[964, 1:64]), ncol = 8, byrow = T)

p2 = matrix(as.numeric(train[962, 1:64]), ncol = 8, byrow = T)

#Low prob
p3 = matrix(as.numeric(train[994, 1:64]), ncol = 8, byrow = T)

p4 = matrix(as.numeric(train[987, 1:64]), ncol = 8, byrow = T)

p5 = matrix(as.numeric(train[974, 1:64]), ncol = 8, byrow = T)

# Plot results
heatmap(p1,  Colv = NA, Rowv = NA)
heatmap(p2,  Colv = NA, Rowv = NA)
heatmap(p3,  Colv = NA, Rowv = NA)
heatmap(p4,  Colv = NA, Rowv = NA)
heatmap(p5,  Colv = NA, Rowv = NA)
results_train = vector(mode = "numeric", length = 30)
results_valid = vector(mode = "numeric", length = 30)

#Training data
for(i in 1:30) {
  m1 = kknn(as.factor(V65) ~ .,
            train,
            train,
            k = i,
            kernel = "rectangular")

  Pred1 = m1$fitted.value
  T1 = table(train$V65, Pred1)

  # The number of misclassifications for each outcome
  misclass1 = rowSums(T1) - diag(T1)

  # The number and rate of misclassification(s) for all outcomes
  results_train[i] = (sum(misclass1) / sum(T1))
}

#Validation data
for(i in 1:30) {
  m2 = kknn(as.factor(V65) ~ .,
            train,
            valid,
            k = i,
            kernel = "rectangular")

  Pred2 = m2$fitted.value
  T2 = table(valid$V65, Pred2)

  # The number of misclassifications for each outcome
  misclass2 = rowSums(T2) - diag(T2)

  # The number and rate of misclassification(s) for all outcomes
  results_valid[i] = (sum(misclass2) / sum(T2))
}

#results_valid[1]
#results_valid[2]

mat = matrix(c(results_train, results_valid), nrow = 30, byrow = FALSE)
matplot(mat, type = "b", pch = 1,
        main = "Training and Test misclassification based on k",
        xlab = "k", ylab = "Misclassification rate")
# Test data
m1 = kknn(as.factor(V65) ~ .,
          train,
          test,
          k = 4,
          kernel = "rectangular")

Pred_test = m1$fitted.value
T_test = table(test$V65, Pred_test)

# The number of misclassifications for each outcome
misclass_test = rowSums(T_test) - diag(T_test)

cat("Training data:", "\n")
cat("k = 4:", results_train[4],"\n")

cat("Validation data:", "\n")
cat("k = 3:", results_valid[3],"\n" , "k = 4:", results_valid[4], "\n")

# The number and rate of misclassification(s) for all outcomes
cat("Test data: ",
    "\n","k = 4: ",(sum(misclass_test) / sum(T_test)))
CE_arr = c()

for(i in 1:30){
  m1 = kknn(as.factor(V65) ~ .,
            train,
            valid,
            k = i,
            kernel = "rectangular")

  probs = m1$prob
  CE = 0

  for (j in 1:nrow(valid)) {
    # The cross entropy loss function can be implemented as
    # The +1 comes is for identifying the correct column
    # since r counts from 1,2,3,...
    CE = (CE - (log((probs[j, valid[j,65]+1]) + 1e-15)))
  }
  CE_arr = c(CE_arr, CE)
}

plot(CE_arr, col = "red", type = "b", main = "Cross entropy depending on k",
     xlab = "k", ylab = "Cross entropy")
cat("The best k is: ", which.min(CE_arr), "with a value of ",
    CE_arr[which.min(CE_arr)])
```

Assignment 2:

Step 1

In this step, the data is split into a training data set (60%) and a test data set (40%). The data is then scaled by using the preProcess function. This is necessary to construct good regression models.

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Loading required package: lattice

## # A tibble: 3,525 x 22
##    subject.     age    sex test_time motor_UP~1 total~2 Jitte~3 Jitte~4 Jitter~5
##       <dbl>   <dbl>  <dbl>     <dbl>      <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
##  1   -1.66   0.816  -0.682    -1.50       0.880  0.549  -0.573   -0.785 -0.636
##  2   -1.33   1.16   -0.682     1.15       1.47   1.42   -0.0967   0.122 -0.172
##  3   -1.33   1.16   -0.682    -0.642      1.19   1.11   -0.314   -0.118 -0.393
##  4    1.41   0.135  -0.682     0.393      1.32   1.28    0.163    0.199  0.436
##  5   -0.121  0.249  -0.682     1.23      -1.21  -1.15    0.134    0.277  0.161
##  6    1.17  -0.318   1.47     -1.47      -0.144 -0.192   4.36     2.78   3.48
##  7   -1.33   1.16   -0.682    -0.290      1.19   1.17   -0.459   -0.345 -0.412
##  8   -0.444  0.0221 -0.682     0.311     -1.58  -0.904   0.191    0.521 -0.0151
##  9   -0.687  0.929   1.47      0.502     -0.116 -0.0990 -0.146   -0.480  0.00410
## 10   -1.66   0.816  -0.682    -1.63       0.849  0.503  -0.708   -0.906 -0.681
## # ... with 3,515 more rows, 13 more variables: Jitter.PPQ5 <dbl>,
## #   Jitter.DDP <dbl>, Shimmer <dbl>, Shimmer.dB. <dbl>, Shimmer.APQ3 <dbl>,
## #   Shimmer.APQ5 <dbl>, Shimmer.APQ11 <dbl>, Shimmer.DDA <dbl>, NHR <dbl>,
## #   HNR <dbl>, RPDE <dbl>, DFA <dbl>, PPE <dbl>, and abbreviated variable names
## #   1: motor_UPDRS, 2: total_UPDRS, 3: Jitter..., 4: Jitter.Abs., 5: Jitter.RAP
```

Step 2

In this step, the linear model is constructed. The Parkinson disease symptom score (motor_UPDRS) is predicted based on variables representing voice characteristics. The mean square error for the training and test data is computed.

```
##
## 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, data = trainS)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0162 -0.7340 -0.1084  0.7310  2.1892
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.006789   0.015790   0.430 0.667252
## Jitter...       0.180301   0.144277   1.250 0.211496
## Jitter.Abs.    -0.169795   0.040856  -4.156 3.32e-05 ***
## Jitter.RAP     -5.087728  18.186914  -0.280 0.779688
## Jitter.PPQ5    -0.071963   0.084712  -0.850 0.395659
## Jitter.DDP      5.068468  18.190294   0.279 0.780541
## Shimmer         0.590434   0.205314   2.876 0.004055 **
## Shimmer.dB.    -0.172730   0.139396  -1.239 0.215380
## Shimmer.APQ3   32.009848  77.023255   0.416 0.677738
## Shimmer.APQ5   -0.387249   0.113730  -3.405 0.000669 ***
## Shimmer.APQ11   0.310751   0.062288   4.989 6.37e-07 ***
## Shimmer.DDA   -32.325644  77.023042  -0.420 0.674739
## NHR            -0.186169   0.045766  -4.068 4.85e-05 ***
## HNR            -0.239653   0.036570  -6.553 6.45e-11 ***
## RPDE            0.004059   0.022615   0.179 0.857576
## DFA            -0.276898   0.019893 -13.919  < 2e-16 ***
## PPE             0.229100   0.033268   6.886 6.75e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9367 on 3508 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.24 on 16 and 3508 DF,  p-value: < 2.2e-16
```

```
## [1] 0.873147
```

```
## [1] 0.9297021
```

By looking at the p-value (forth column)for the estimated values of the coefficients, we can determine that the independent variables *Jitter.Abs, Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA,* and *PPE* contribute significantly to the model.

Task 3

The formulas for completing this were found in the lecture slides unless something else is stated.

Task 4

In this task, ridge regression is used with different lambda on the training data. The accuracy of the model is then evaluated by calculating the MSE of the test data and degrees of freedom for each lambda is computed.

```
## [1] "lambda: 1"
## [1] "train MSE: 0.873276950507478"
## [1] "test_MSE: 0.929035685734743"
## [1] "df_test: 13.7804331534807"
## [1] "df_train 13.8628107030632"
```

```
## [1] "lambda: 100"
## [1] "train MSE: 0.879059936030921"
## [1] "test_MSE: 0.926316386508651"
## [1] "df_test: 9.18289840204118"
## [1] "df_train 9.93908507636864"

## [1] "lambda: 1000"
## [1] "train MSE: 0.915626779289396"
## [1] "test_MSE: 0.947916644425209"
## [1] "df_test: 4.82167557512052"
## [1] "df_train 5.64335089965714"
```

We get the lowest test MSE for lambda = 100. This tells us that lambda = 100 is closer to the optimum than lambda = 1 or lambda = 1000 (after a certain threshold, adding a higher shrinkage penalty will result in higher variance, which in turn will result in a higher MSE).

We can see that as lambda increases, the degrees of freedom decrease. This makes sense since if there is no penalization, we have as many parameters as there are in the model. On the contrary, when lambda approaches infinity, the degrees of freedom will approach zero.

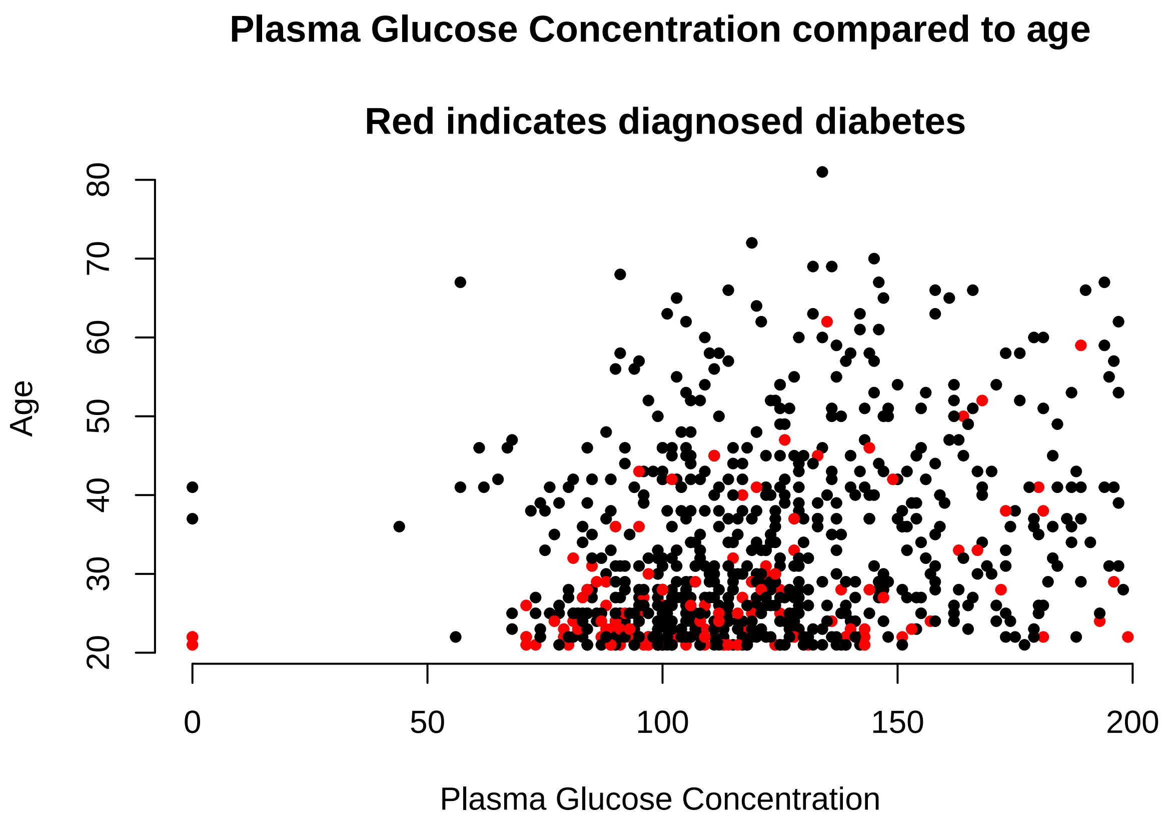
```r
library(knitr)
library(dplyr)
library(tidyr)
library(caret) #for preprocessing of data
set.seed(12345)
data = read.csv("parkinsons.csv")

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

scaler=preProcess(data)
trainS=predict(scaler,train)
testS=predict(scaler,test)
trainS
# Computing the linear model and the MSE for the training data
m1 = 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, data = trainS)
summary(m1)
preds = predict(m1, trainS)
MSE = sum((trainS$motor_UPDRS - preds)^2) / length(preds)
MSE
# MSE for the test data
preds = predict(m1, testS)
MSE = sum((testS$motor_UPDRS - preds)^2) / length(preds)
MSE
# The purpose of this function is to find values for the thetas and
# sigmas which maximizes the probability for our model to generate the data.
LogLikelihood <- function(data, theta, sigma){
  x = as.matrix(data[ ,7:22])
  y = as.matrix(data[, 5])
  n = length(y) # observations
  logLik = -(n/2)*(log(2*pi) + log(sigma^2)) - (2*sigma^2)^-1 * sum((x%*%theta - y)^2) #squared loss is
  return(logLik)
}
# The purpose of ridge regression is to reduce the variance when the independent
# variables have high correlarity with one another.This is accomplished by
# adding a shrinkage penalty which increases the bias a little bit but
# in turn reduce the variance significantly, which leads to a lower MSE.
Ridge <- function(sigmaThetas, lambda, data){
  sigma = sigmaThetas[1]
  theta = sigmaThetas[-1] #removes first element (sigma)
  logLik = LogLikelihood(data, theta, sigma)
  ridge = lambda * sum(theta^2) - logLik
  return(ridge)
}
RidgeOpt <- function(lambda, data){
  initSigma = 1
  initThetas = rep(1, 16)
  optRes = optim(par=c(initSigma, initThetas), fn=Ridge, lambda=lambda,
                 data=data, method="BFGS")
  return(optRes)
}
# formula found at https://online.stat.psu.edu/stat508/lesson/5/5.1
DF <- function(lambda, X){
  I = diag(dim(X)[2])
  #Sum of diagonal is the same as the trace function (tr)
  temp = X %*% solve(t(X)%*%X + lambda*I)%*%t(X)
  dt = sum(diag(temp))
  return(dt)
}

lambda = c(1, 100, 1000)
#lambda[1] = 1
#lambda[2] = 100
#lambda[3] = 1000
testOpt <- function(lambda) {
  opt = RidgeOpt(lambda, trainS)
  sigma = opt$par[1]
  thetas = as.matrix(opt$par[-1])
  X_train = as.matrix(trainS[, 7:22])
  X_test = as.matrix(testS[, 7:22])
  Y_train = as.matrix(trainS[, 5])
  Y_test = as.matrix(testS[, 5])

  train_predict = X_train %*% thetas
  test_predict = X_test %*% thetas

  #calculate MSE for train and test data
  train_MSE = (length(Y_train))^-1 * sum((Y_train - train_predict)^2)
  test_MSE = (length(Y_test))^-1 * sum((Y_test - test_predict)^2)

  #calculate degrees of freedom
  df_train = DF(lambda, X_train)
  df_test = DF(lambda, X_test)

  print(paste("lambda:", lambda))
  print(paste("train MSE:", train_MSE))
  print(paste("test_MSE:", test_MSE))
  print(paste("df_test:", df_test))
  print(paste("df_train", df_train))

}
opt1 = testOpt(lambda[1])
opt2 = testOpt(lambda[2])
opt3 = testOpt(lambda[3])
```

Assignment 3:

Step 1

Using the data from the csv file, a scatter plot is created representing the age and glucose level are plotted against each other. The red color of the plot indicates that the patient has been diagnosed with diabetes, while black means the patient is not diagnosed with diabetes.

The outcomes are binary and the variables doesn't seem to be correlated, which means that the data is suitable to use with logistic regression. However, the presence of diabetes seems to scattered across the entire plot, which would make it hard to predict the outcome based on the 2 selected variables.



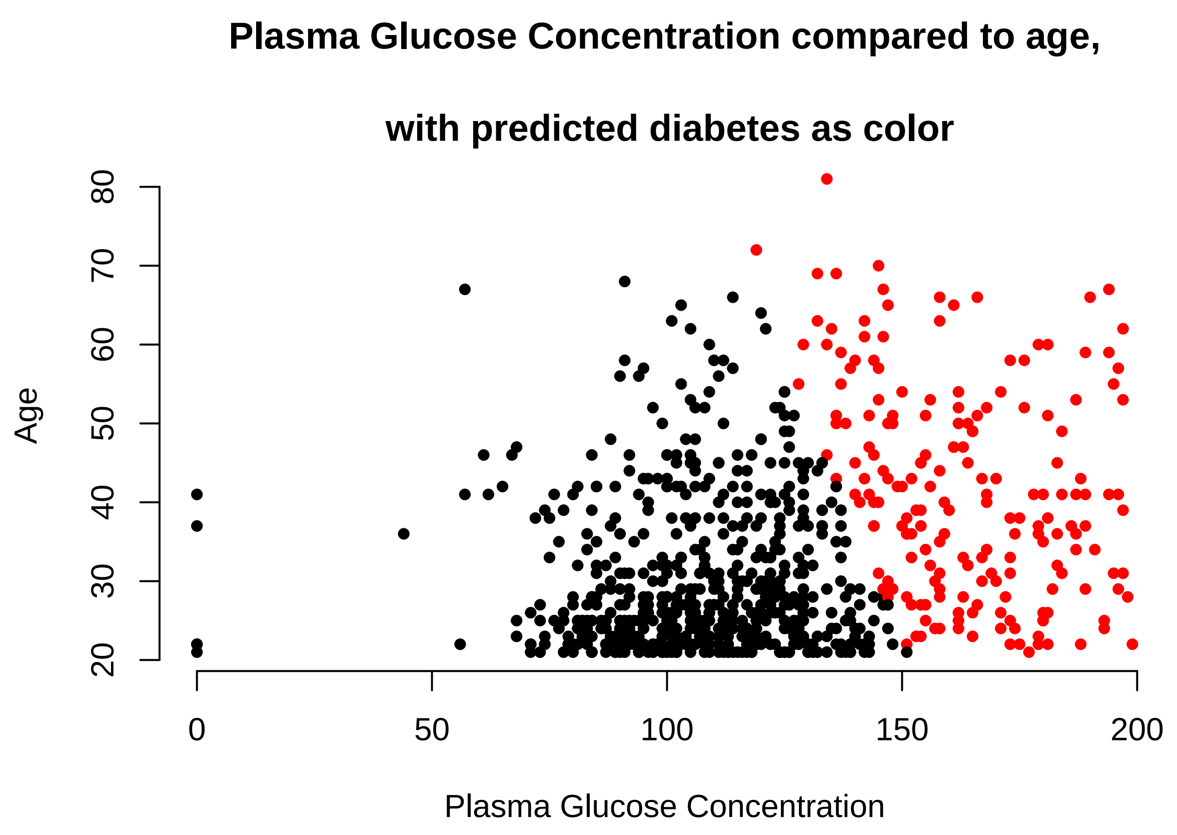
Step 2

Based on the outcome of the training of the logistic model seen below, the probabilistic model can be determined to be:

$$P(Y = 1) = \frac{1}{1 + e^{5.9124 - 0.0356 * X1 - 0.0248 * X2}}$$

The misclassification rate of 26,4% means that more than a quarter of the time, the predicted outcome is not correct. A high rate of misclassifications is understandable based on the fact that the presence of diabetes is spread out in the plot in step 1. Without any clear correlation between diagnosed diabetes and the 2 variables, It's hard to get a classification rate that is good. Therefore, the predicted values lead to a lot scatter plot that looks like the one below, containing a lot of misclassifications.

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

##    pred
##       0   1
##   0 436  64
##   1 138 130

##
## Call:
## glm(formula = as.factor(V9) ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3367  -0.7775  -0.5087   0.8367   3.1630
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.912449   0.462620  -12.78  < 2e-16 ***
## V2           0.035644   0.003290   10.83  < 2e-16 ***
## V8           0.024778   0.007374    3.36 0.000778 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 797.36  on 765  degrees of freedom
## AIC: 803.36
##
## Number of Fisher Scoring iterations: 4

## Misclassification rate:  0.2630208
```



Step 3

In step 3, the decision boundary is calculated and inserted into the plot from step 2.

The decision boundary is decided with:

$$\beta_0 + \beta 1 * x_1 + \beta_2 * x_2 = 0$$

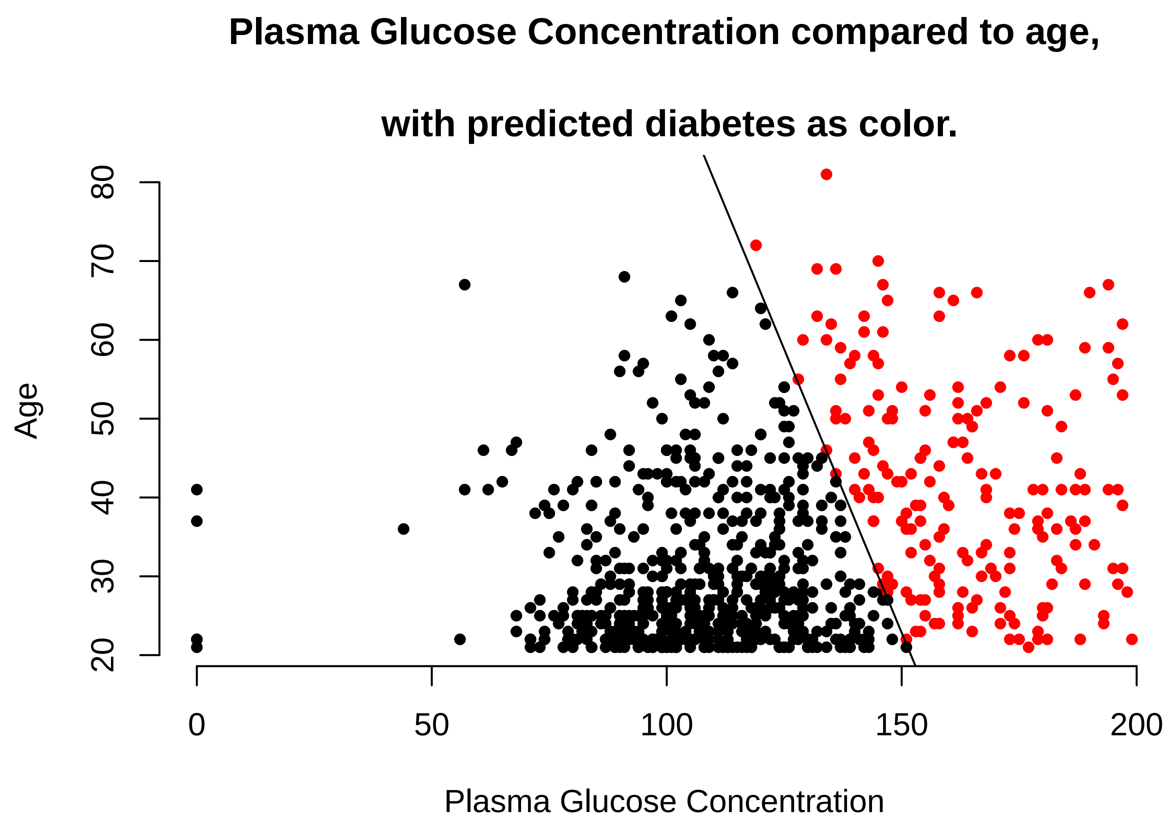
This means that the the line on the $y = kx + m$ format can be calculated with:

$$x_2 = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2} x_1$$

Using the coefficient values from the glm function, we get the function:

$$x_2 = -\frac{-5.912}{0.025} - \frac{0.036}{0.025} x_1 \approx 239 - 1.44 x_1$$

The decision boundary clearly separates the predicted cases of diabetes from the ones where diabetes is predicted to be false. However, when comparing to the actual outcomes, a lot of the observations to the left of the decision boundary are really cases of diabetes but classified as non-diabetes. This outcome is because of the linear decision boundary, which splits the data into 2 with a straight line. Since the actual cases of diabetes can't be split between true and false with a straight line, there will be misclassifications.



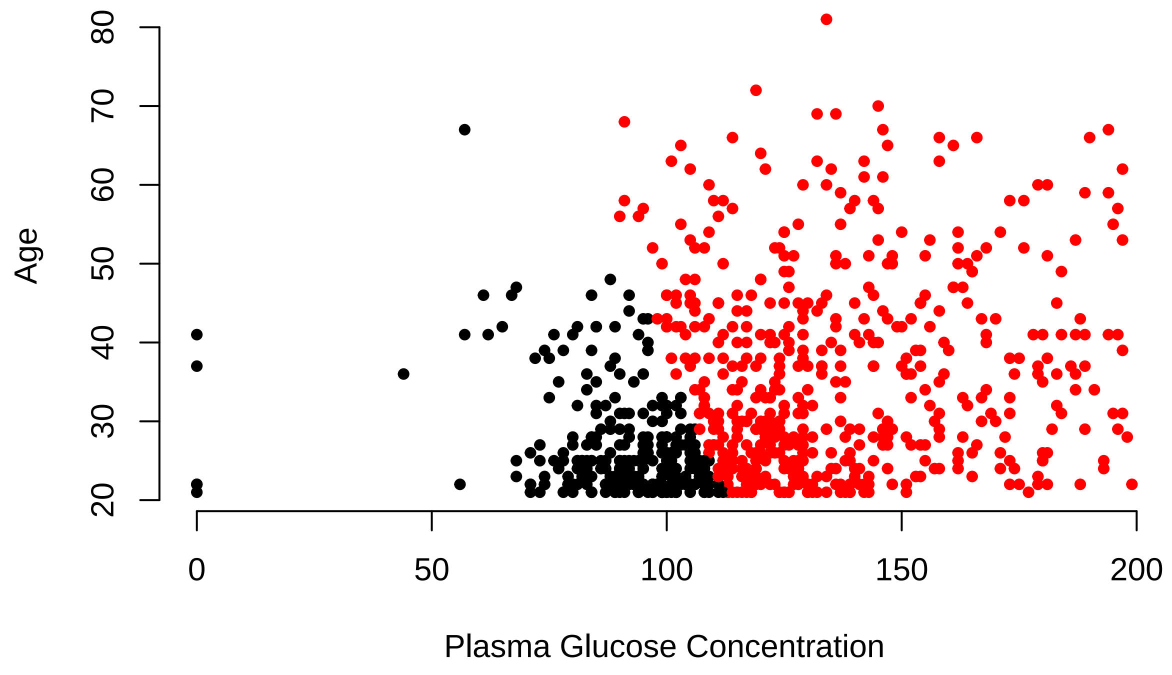
Step 4

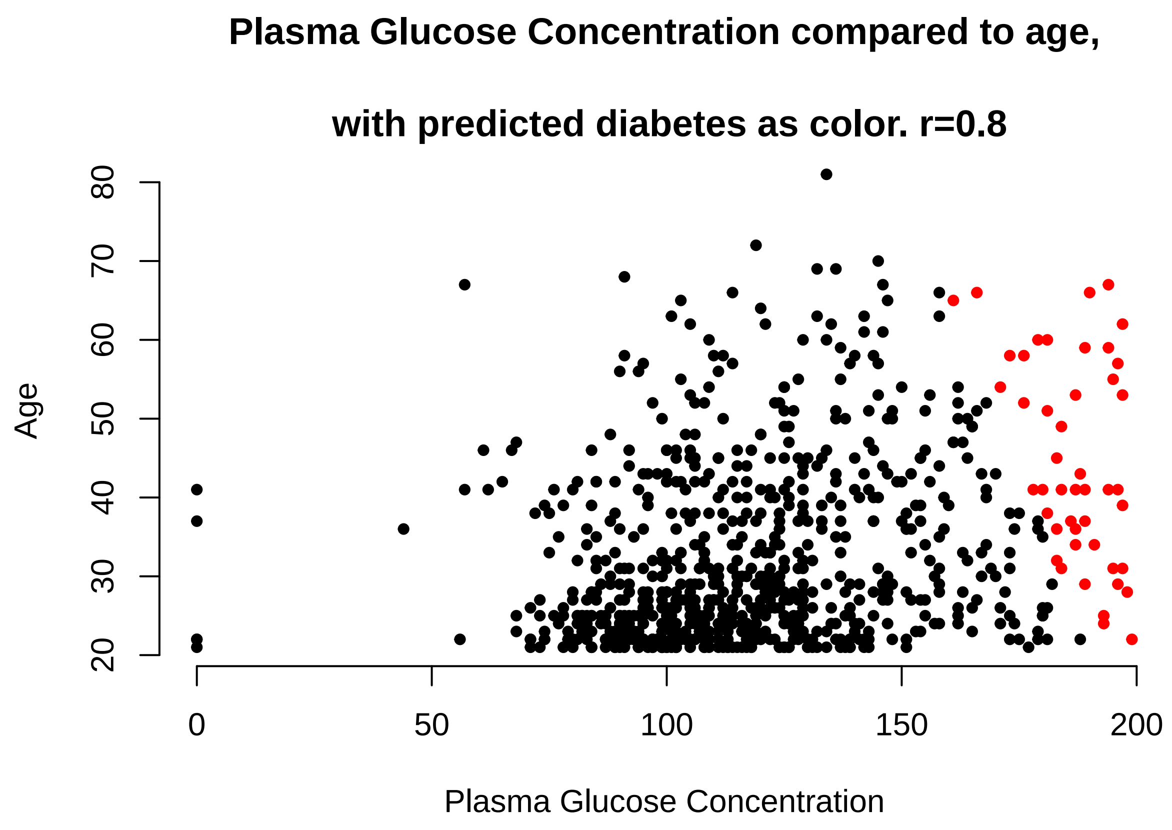
Changing the threshold to 0.8 means that the probability of diabetes has to be higher in order for the model to classify it as true. This will lead to a smaller number of observations being classified as diabetes compared to r=0.5.

The opposite happens if changing the threshold to 0.2, where the probability now doesn't need to be as high in order for the model to classify it as diabetes. This leads to a higher number of observations being classified as diabetes compared to r=0.5.

Plasma Glucose Concentration compared to age,

with predicted diabetes as color. r=0.2





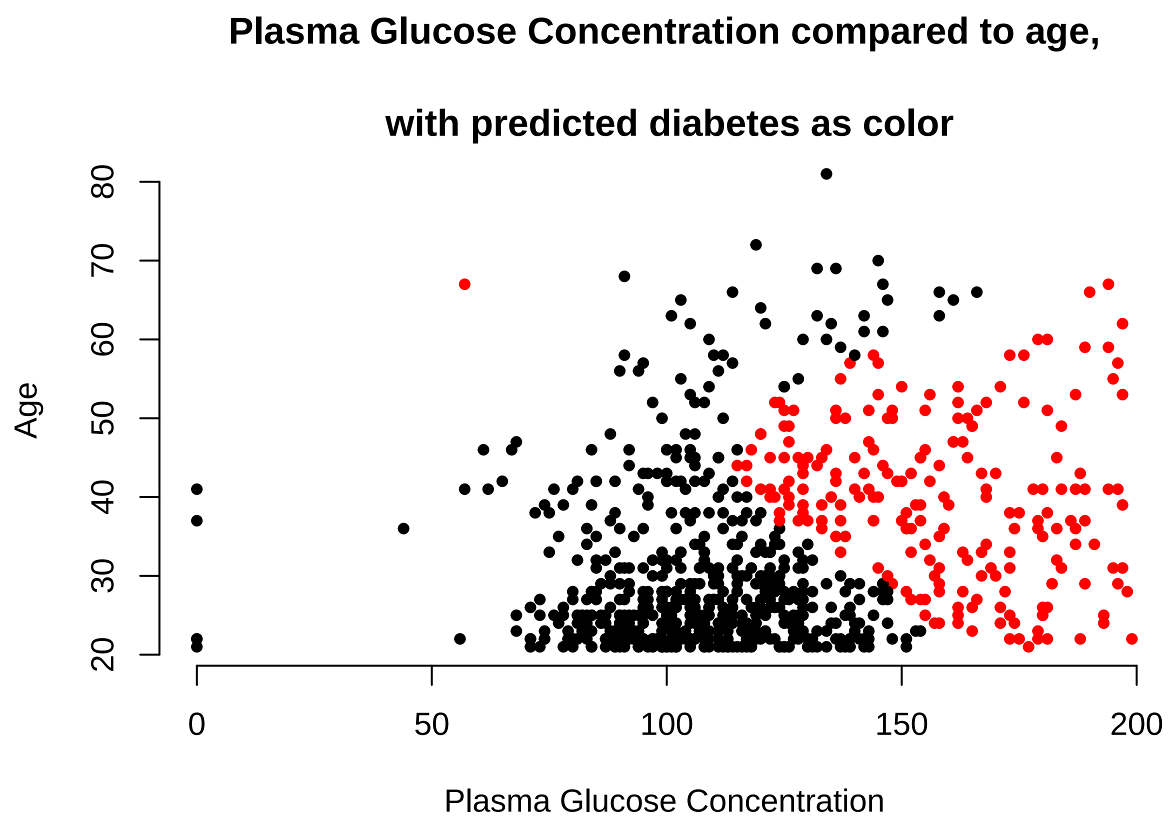
Step 5

The new variables introduced means that the decision boundary is no longer linear, since the polynomial degree is now 4. The misclassification rate has decreased a little bit which would indicate that the model is slightly better than before. This is also indicated by the AIC being lower than before.

```
##    pred
##       0   1
##   0 436  64
##   1 138 130
```

```
##
## Call:
## glm(formula = as.factor(V9) ~ ., family = "binomial", data = train2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2887  -0.7258  -0.4257   0.7451   2.5280
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.310e+00  1.129e+00  -8.243  < 2e-16 ***
## V2           3.793e-02  9.473e-03   4.004 6.23e-05 ***
## V8           1.457e-01  2.072e-02   7.031 2.05e-12 ***
## z1           1.278e-08  5.610e-09   2.278  0.02271 *
## z2           -1.780e-07  7.635e-08  -2.331  0.01976 *
## z3            8.515e-07  3.437e-07   2.478  0.01322 *
## z4           -1.698e-06  6.313e-07  -2.690  0.00715 **
## z5            8.127e-07  4.054e-07   2.004  0.04503 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.48  on 767  degrees of freedom
## Residual deviance: 741.61  on 760  degrees of freedom
## AIC: 757.61
##
## Number of Fisher Scoring iterations: 5
```

```
## Misclassification rate:  0.2447917
```



```
library(knitr)
<style>
body {
text-align: justify}
</style>
pima_indians_diabetes=read.csv("pima-indians-diabetes.csv", header=FALSE)

plot(pima_indians_diabetes$V2, pima_indians_diabetes$V8,
```

```r
     main = "Plasma Glucose Concentration compared to age
     \n Red indicates diagnosed diabetes", xlab="Plasma Glucose Concentration",
     ylab="Age", pch=20, frame=FALSE,
     col=ifelse(pima_indians_diabetes == 1, "red", "black"))
library(dplyr)
library(tidyr)

train=tibble(pima_indians_diabetes)
train=train%>%select(V2,V8,V9)

r=0.5
m1=glm(as.factor(V9)~., train, family = "binomial")
prob=predict(m1, type = "response")
pred=ifelse(prob>r, 1, 0)

table(train$V9, pred)

summary(m1)
misclass=function(X,X1){
  n=length(X)
return(1-sum(diag(table(X,X1)))/n) }

mc=misclass(train$V9, pred)
cat("Misclassification rate: ", mc)

plot(pima_indians_diabetes$V2, pima_indians_diabetes$V8,
     main = "Plasma Glucose Concentration compared to age,
     \n with predicted diabetes as color", xlab="Plasma Glucose Concentration",
     ylab="Age", pch=20, frame=FALSE, col=ifelse(pred==1, "red", "black"))
slope=coef(m1)[2]/(-coef(m1)[3])
intercept=coef(m1)[1]/(-coef(m1)[3])

plot(pima_indians_diabetes$V2, pima_indians_diabetes$V8,
     main = "Plasma Glucose Concentration compared to age,
     \n with predicted diabetes as color.", xlab="Plasma Glucose Concentration",
     ylab="Age", pch=20, frame=FALSE, col=ifelse(pred==1, "red", "black"))
abline(intercept, slope)
r=0.2
pred2=ifelse(prob>r, 1, 0)

plot(pima_indians_diabetes$V2, pima_indians_diabetes$V8,
     main = "Plasma Glucose Concentration compared to age,
     \n with predicted diabetes as color. r=0.2",
     xlab="Plasma Glucose Concentration", ylab="Age", pch=20,
     frame=FALSE, col=ifelse(pred2==1, "red", "black"))

r=0.8
pred3=ifelse(prob>r, 1, 0)
plot(pima_indians_diabetes$V2, pima_indians_diabetes$V8,
     main = "Plasma Glucose Concentration compared to age,
     \n with predicted diabetes as color. r=0.8",
     xlab="Plasma Glucose Concentration", ylab="Age", pch=20,
     frame=FALSE, col=ifelse(pred3==1, "red", "black"))
```

```r
pima_indians_diabetes$z1=sapply(pima_indians_diabetes$V2, function(x) x^4)
pima_indians_diabetes$z2=sapply(pima_indians_diabetes$V2,
                                function(x) x^3) * pima_indians_diabetes$V8
pima_indians_diabetes$z3=sapply(pima_indians_diabetes$V2,
                                function(x) x^2) * sapply(pima_indians_diabetes$V8,
                                                          function(x) x^2)
pima_indians_diabetes$z4=pima_indians_diabetes$V2 * sapply(
  pima_indians_diabetes$V8, function(x) x^3)
pima_indians_diabetes$z5=sapply(pima_indians_diabetes$V8, function(x) x^4)

train2=tibble(pima_indians_diabetes)
train2=train2%>%select(V2,V8,V9,(z1:z5))

r=0.5
m2=glm(as.factor(V9)~., train2, family = "binomial")
prob4=predict(m2, type = "response")
pred4=ifelse(prob4>r, 1, 0)

table(train$V9, pred)

summary(m2)

mc2=misclass(train$V9, pred4)
cat("Misclassification rate: ", mc2)

plot(pima_indians_diabetes$V2, pima_indians_diabetes$V8,
     main = "Plasma Glucose Concentration compared to age,
     \n with predicted diabetes as color",
     xlab="Plasma Glucose Concentration", ylab="Age", pch=20,
     frame=FALSE, col=ifelse(pred4==1, "red", "black"))
```