Lab1 Report

jonal155, marfr380, davhe786

Statement of Contribution

The responsibility for the assignments were divided between the group members. jonal155 were responsible for assignment 1, davhe786 were responsible for assignment 2 and marfr380 were responsible for assignment 3.

marfr380 checked the work on assignment 1 by jonal155, jonal155 checked the work on assignment 2 by davhe786 and davhe786 checked the work on assignment 3 by marfr380.

All members of the group worked on assignment 4 together.

Results from all assignments have been discussed afterwards between all members of the group and the report was created based on this discussion.

Assignment 1

The data used for this assignment are bitmaps of handwritten digits. Each row in the data file is a sequence corresponding to a generated 8x8 matrix where each pixel has a value between 0-16 depending on how much of the pixel is filled with writing. Each row is labeled with the last row as the correct class for the image.

This assignment uses the K-nearest algorithm to make a predictive model from the data set.

Exercise 1

The data is prepared by dividing it into three parts, training, test and validation. This is done by generating a random vector with numbers corresponding to the row index in the data set. The number of indicies in the vector is 50% of the number of entries in the data for the training set and 25% for both the test and validation sets.

```r
data <- read.csv("optdigits.csv", header = FALSE)

n <- nrow(data)

set.seed(12345)
train_ids <- sample(1:n, floor(n * 0.5))
train <- data[train_ids, ]

remaining_ids <- setdiff(1:n, train_ids)

set.seed(12345)
valid_ids <- sample(remaining_ids, floor(n * 0.25))
valid <- data[valid_ids, ]

test_ids <- setdiff(remaining_ids, valid_ids)
test <- data[test_ids, ]
```

Exercise 2

Two kknn prediction runs are performed, **m_train** and **m_test**. **m_train** uses the training data to make predictions on the training data while **m_test** uses the training data to make predictions on the test data, meaning that **m_test** is evaluated on unseen data.

The row entry with the correct classification for the row needs to be converted into a categorical label for it to be used as an input in the kknn models. The formula V65~. specifies that all other columns are used as features when making the predictions.

```
train$V65 <- as.factor(train$V65)
test$V65 <- as.factor(test$V65)

# Predict the training data itself.
m_train <- kknn(V65~., train = train, test = train, k = 30, kernel = "rectangular")

# Predict on unseen data(test).
m_test <- kknn(V65~., train = train, test = test, k = 30, kernel = "rectangular")
```

The models generate predicted values which then are used to construct confusion matrices. The rows in the confusion matrix correspond to the predicted values and the columns correspond to the real values from the data. The resulting confusion matrices has the correctly predicted values along the diagonal and the misclassified values above and below the diagonal.

```
pred_train_values <- predict(m_train)
pred_test_values <- predict(m_test)

cm_train <- table(pred_train_values, train$V65)
cm_test <- table(pred_test_values, test$V65)

cm_train
```

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

```
cm_test
```

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

```r
NumberOfMisCalc <- function(cm) {
  cat("\n")
  numbers <- rownames(cm)
  for (num in numbers) {
    misclassified <- sum(cm[num,]) - cm[num, num]
    cat("The number", num, "was misclassified", misclassified, "times\n")
  }
}

# NumberOfMisCalc(cm_train)
# NumberOfMisCalc(cm_test)
```

The resulting confusion matrices shows that the numbers 0, 4, 5 and 6 are almost always correctly classified while the numbers 1, 7 and 9 are more frequently misclassified. This is most likely caused by the visual similarities between the numbers which causes the features to be closer together.

A misclassification error rate can then be calculated to evaluate the different models. The error rate is computed by taking the numbers of misclassifications divided by the total number of predictions.

```r
CalcMisError <- function(cm) {
  correct <- sum(diag(cm))       # The correctly predicted values.
  total <- sum(cm)               # total values.
  return (1 - correct/total)     # 1 - rate of correct classification = misclassification rate.
}

train_mis_error <- CalcMisError(cm_train)
test_mis_error <- CalcMisError(cm_test)

train_mis_error
```

```
## [1] 0.04500262
```

```r
test_mis_error
```

```
## [1] 0.05329154
```

The resulting error rate is **4.5%** for the **m__train** model and **5.3%** for the **m__test** model. The model testing on unseen data only shows a small increase in the error rate, **0.8%**, which indicates that the model is not overfitting and therefore has a good prediction quality.

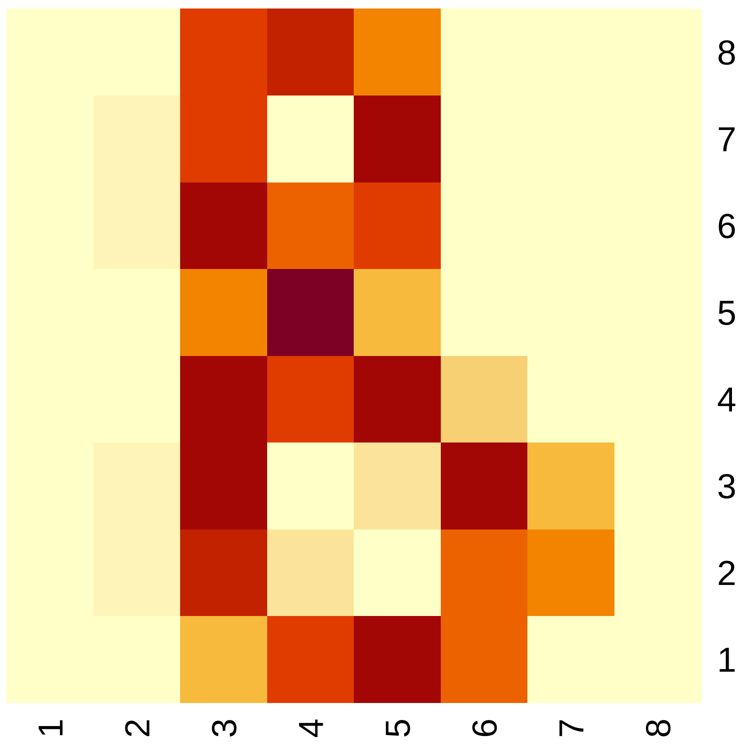
Exercise 3

We can visually evaluate the result by reshaping the features into a 8x8 matrix and represent the values in a heatmap. A heatmap shows different colors depending on the values in the matrix, higher number values will be more red than lower value numbers.

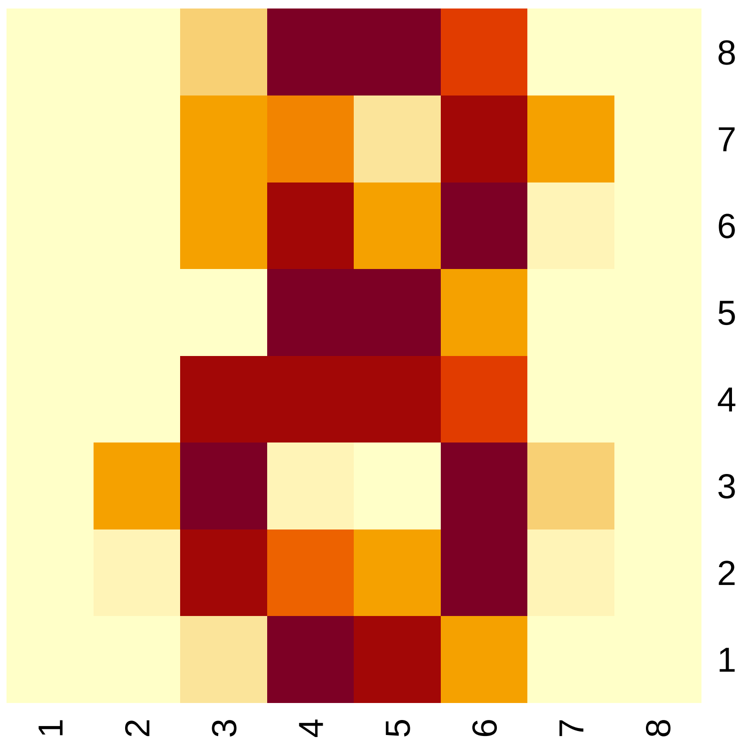
To generate the heatmap we form a vector from the probability matrix containing all of the probabilities for the correct classification of that image. We use the real values in the test data set to get the indices for every 8. These indices are then used to query the probabilities for each 8 in the previously mentioned probability vector. The probabilities are then order from highest to lowest and are converted from probabilities to indices so that the features(rows) can be queried from the test data set for generation of the heatmap.

```r
# Probability matrix.
train_probs <- m_train$prob

# True labels.
```

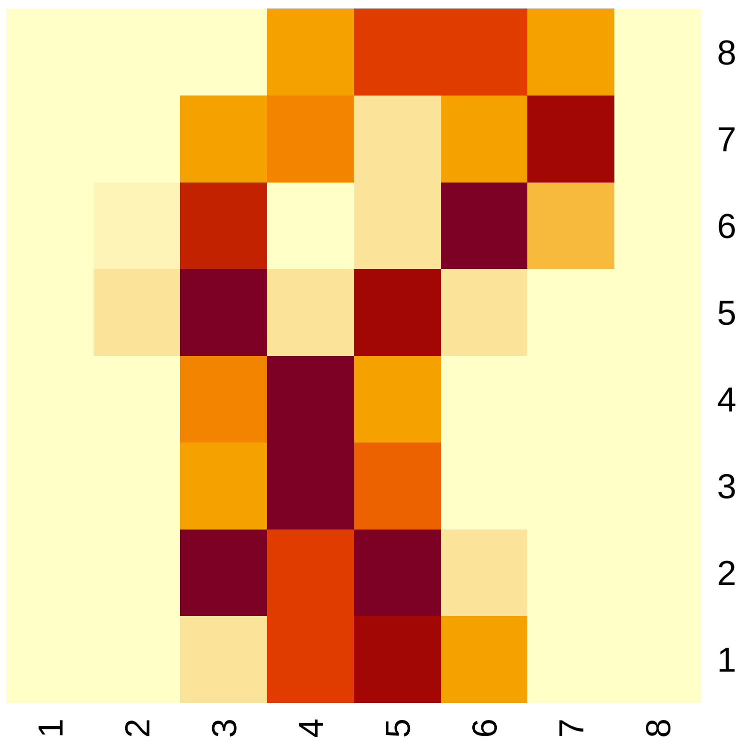
```r
y_true_num <- as.numeric(as.character(train$V65))

# Correct probability per row.
correct_train_prob <- train_probs[cbind(1:nrow(train_probs), y_true_num + 1)]

# Indices of all true "8" rows.
idx_8 <- which(y_true_num == 8)

# Correct probabilities for "8"s.
prob_8 <- correct_train_prob[idx_8]

# Sort by probability, highest to lowest.
order_idx <- order(prob_8, decreasing = TRUE)

# Easiest 2 and hardest 3.
easiest <- idx_8[order_idx[1:2]]
hardest <- idx_8[tail(order_idx, 3)]

# Heatmap function.
plot_digit <- function(row) {
  m <- matrix(as.numeric(row[1:64]), nrow = 8, byrow = TRUE)
  heatmap(m, Colv = NA, Rowv = NA, scale = "none")
}

# Plot easiest 2.
plot_digit(train[easiest[1], ])
```



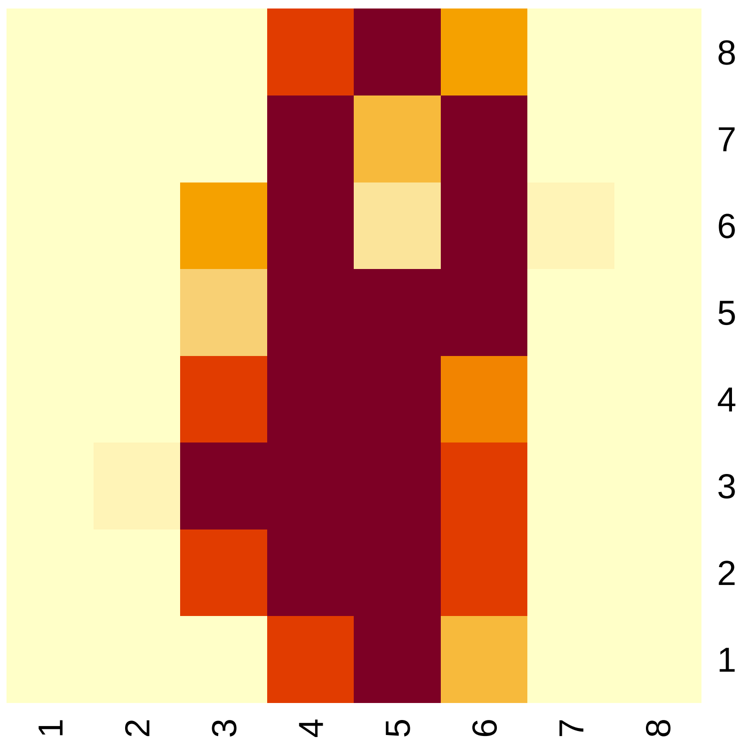
```r
plot_digit(train[easiest[2], ])
```



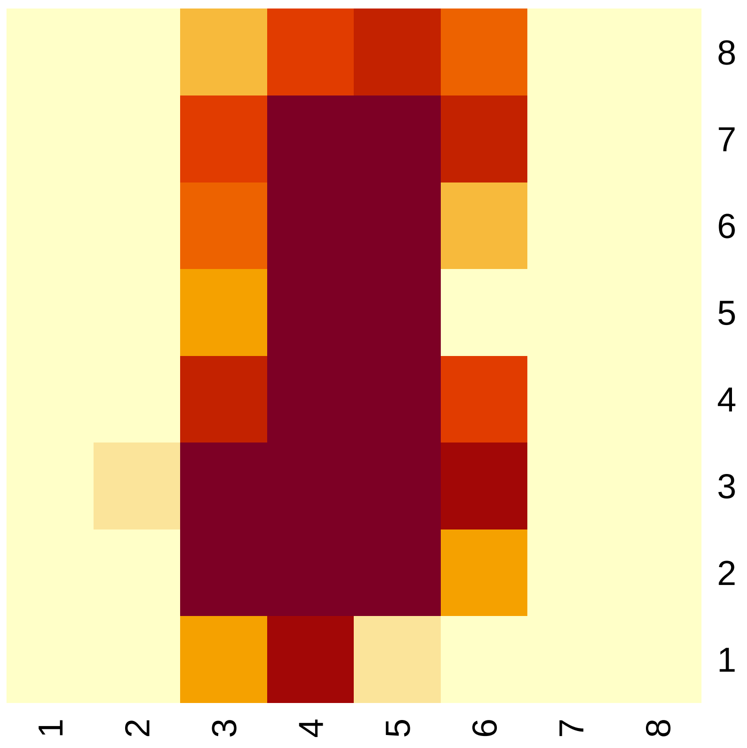
```r
# Plot hardest 3.
plot_digit(train[hardest[1], ])
```



```r
plot_digit(train[hardest[2], ])
```



```r
plot_digit(train[hardest[3], ])
```



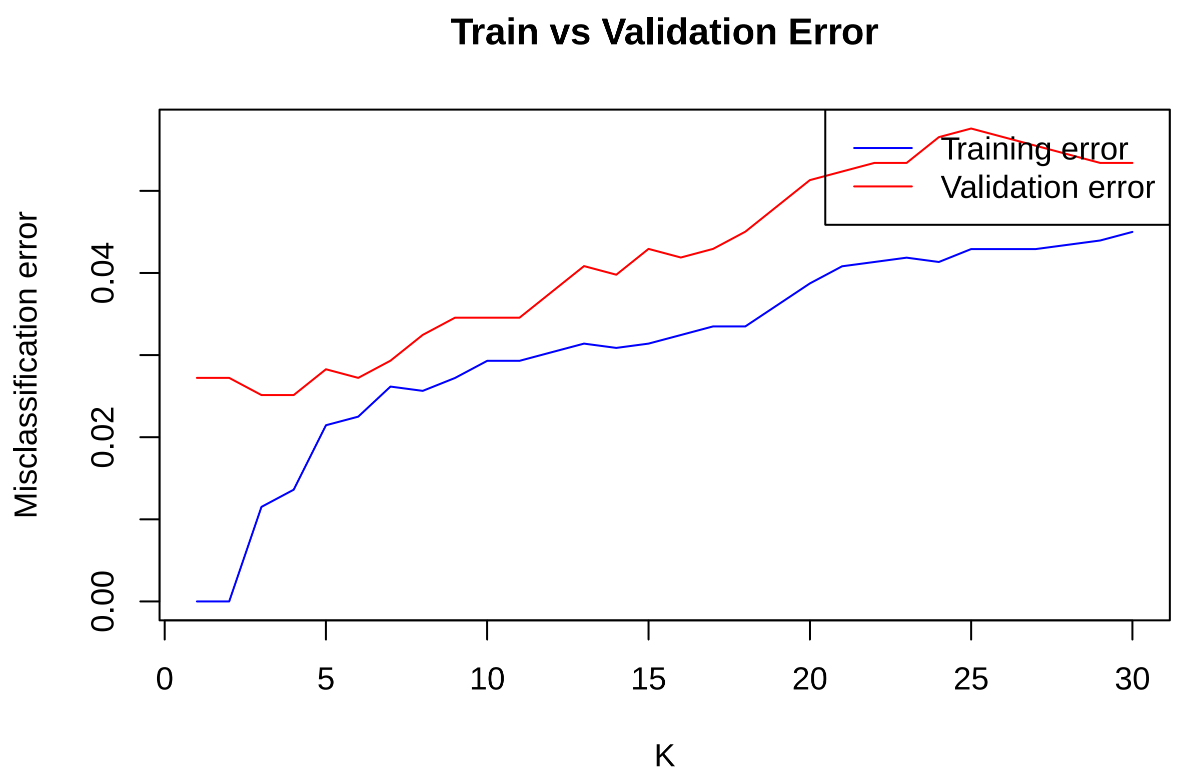
The first two heatmaps are both easily recognizable as the number 8 while the last 3 become significantly harder to recognize. The last heatmaps features are very close together which makes it hard to determine which number it is.

Exercise 4

In this next section different values for K are tested were the misclassification error is calculated for each choice of K, on one model tested on training data and another model tested on validation data. The K values tried are between the values 1 and 30.

```
Ks <- 1:30
train_errors <- numeric(length(Ks))
valid_errors <- numeric(length(Ks))

for (i in seq_along(Ks)) {
  k <- Ks[i]

  # Predict on training
  model_train <- kknn(V65~., train = train, test = train, k = k, kernel = "rectangular")
  pred_train <- predict(model_train)
  cm_train_k <- table(pred_train, train$V65)
  train_errors[i] <- CalcMisError(cm_train_k)

  # Predict on validation
  model_valid <- kknn(V65 ~ ., train = train, test = valid, k = k, kernel = "rectangular")
  pred_valid <- predict(model_valid)
  cm_valid_k <- table(pred_valid, valid$V65)
```

```r
  valid_errors[i] <- CalcMisError(cm_valid_k)
}

plot(Ks, train_errors, type = "l", col = "blue", ylim = c(0, max(valid_errors)),
     xlab = "K", ylab = "Misclassification error", main = "Train vs Validation Error")
lines(Ks, valid_errors, col = "red")
legend("topright", legend = c("Training error", "Validation error"),
       col = c("blue", "red"), lty = 1)
```



As seen in the graph the training error increases with K but the validation error initially drops and then starts to increase with K. In KNN the model complexity refers to how flexible the classification is and how detailed its decision boundaries can become. A low K value will result in a model with overfitting and high complexity. The decision boundaries become more jagged and are more prone to follow noise. A high K value means that more neighbors have an impact of the final classification, a K value that is too high will lead to underfitting and will have a very low complexity. We want to find the optimal K which does not over- or underfitt.

The optimal K can be found by taking the index of the lowest validation error and extracting the corresponding K from the list. The validation error is used instead of the test error because we want to pick the optimal value for unseen data.

```r
# Find optimal K.
optimal_K <- Ks[which.min(valid_errors)]
optimal_K
```

```
## [1] 3
```

```r
# Calc misclassification error.
m_opt_test <- kknn(V65 ~ ., train = train, test = test, k = optimal_K, kernel = "rectangular")
pred_opt_test <- predict(m_opt_test)

cm_opt_test <- table(pred_opt_test, test$V65)
error_opt_test <- CalcMisError(cm_opt_test)

opt_train_error <- train_errors[optimal_K]
opt_valid_error <- valid_errors[optimal_K]

opt_train_error
```

```
## [1] 0.0115123
```

```r
opt_valid_error
```

```
## [1] 0.02513089
```

```r
error_opt_test
```

```
## [1] 0.02403344
```

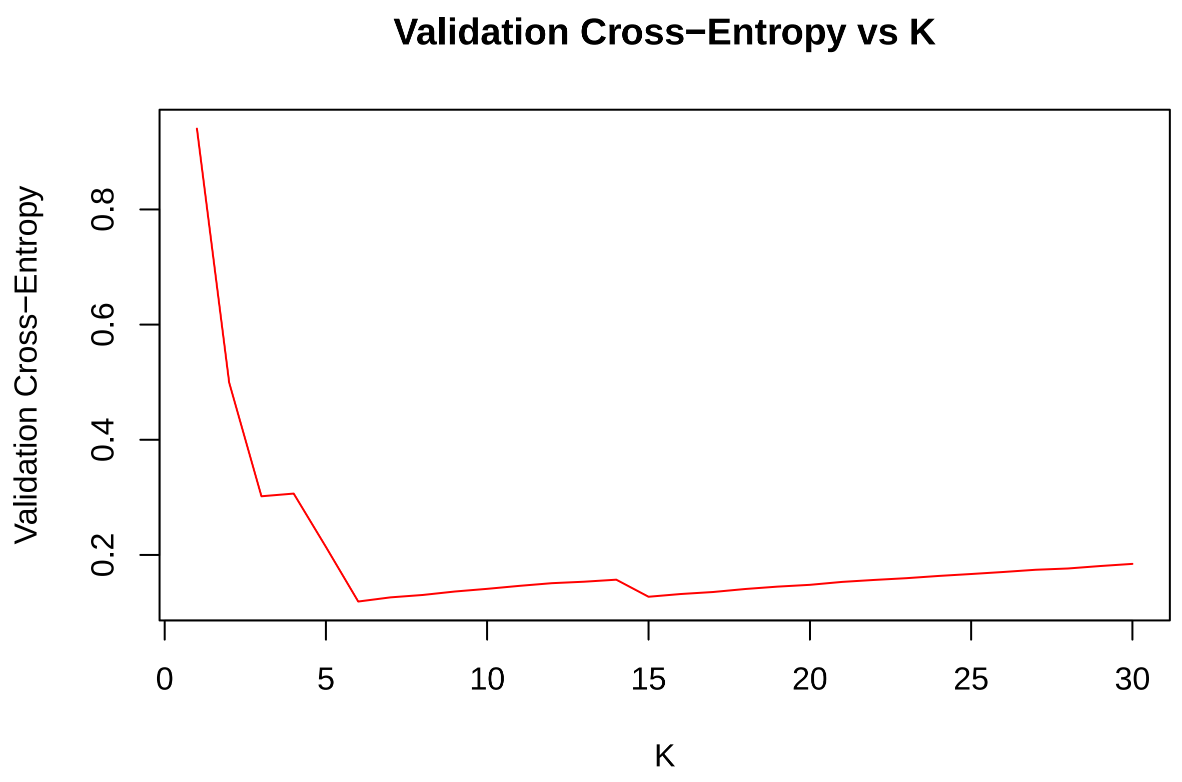
The optimal K for our model is 3 and has a training error rate of **1.2%**, a validation error rate of **2.5%** and a test error rate of **2.4%**. The training error is very low which is expected because a lower K value in KNN will closely fit the training data. The validation and test error rate are roughly the same which shows that the model generalizes well to unseen data.

Exercise 5

We can use the cross-entropy to evaluate our model instead of the misclassification error rate. The misclassification error rate only considers if a prediction is exactly correct and ignores how confident the model is. Cross-entropy looks at the probability distribution. If the prediction has a high probability for the correct class it has low cross-entropy and high if not.

```r
Ks <- 1:30
valid_ce <- numeric(length(Ks))

epsilon <- 1e-15

for (i in seq_along(Ks)) {
  k <- Ks[i]

  model_valid <- kknn(V65~., train = train, test = valid, k = k, kernel = "rectangular")

  prob_valid <- model_valid$prob

  y_true <- as.numeric(as.character(valid$V65))

  correct_prob <- prob_valid[cbind(1:nrow(prob_valid), y_true + 1)]

  # Cross-entropy calculation
  # By using -log we penalize very low probabilities for the correct class,
  # log(p) closer to 1, and very high probabilities wont effect the entropy as much.
  valid_ce[i] <- -mean(log(correct_prob + epsilon))
}

# Plot cross-entropy vs K.
```

```r
plot(Ks, valid_ce, type = "l", col = "red",
     xlab = "K", ylab = "Validation Cross-Entropy",
     main = "Validation Cross-Entropy vs K")
```



```r
optimal_K <- Ks[which.min(valid_ce)]
optimal_K
```

```
## [1] 6
```

The optimal K was found to be 6.

Cross-entropy evaluation is better for categorical problems with more than two classes because it takes into account how confident the model is in its predictions for the correct class. This is useful in our case, where some digits are visually similar, like 1, 7, and 9. Cross-entropy penalizes the model more when it assigns low probability to the true class, providing a more detailed measure of performance compared to misclassification error.

Assignment 2

librarys used:

Assignment 2. Task 1

```r
data <- read.csv("parkinsons.csv")
set.seed(12345)
n <- dim(data)[1]
id <- sample(1:n, floor(n*0.6))
training <-data[id,]
```

```
test <- data[-id,]

scaler <- preProcess(training)
training_scaled <- predict(scaler,training)
test_scaled <- predict(scaler,test)
```

from now on assume that MOTOR_UPDRS is normal distriubuted and a function of voice characterist, no need for scaling aswell.

Making a linear regression estimating values of motor_UDPS obtainging this by using lm to get estimated parameters and then using these parameters to check the value of new

```
# train the model on training data
estimated = lm(formula = motor_UPDRS~., data = training_scaled)

coefficients(estimated)
```

```
##   (Intercept)      subject.           age           sex     test_time
##  2.031899e-15 -2.652108e-02 -3.529144e-02  5.917090e-02 -2.521447e-03
##    total_UPDRS      Jitter...    Jitter.Abs.    Jitter.RAP   Jitter.PPQ5
##  9.600329e-01  1.695278e-01 -7.945877e-02  7.343894e-01 -3.548609e-02
##    Jitter.DDP       Shimmer   Shimmer.dB.  Shimmer.APQ3  Shimmer.APQ5
## -8.338121e-01  1.672351e-01 -2.789139e-02  9.504993e+00 -1.431157e-01
## Shimmer.APQ11   Shimmer.DDA           NHR           HNR          RPDE
##  7.730738e-02 -9.562490e+00  1.453999e-03 -4.297925e-03 -3.056560e-02
##           DFA           PPE
## -6.872099e-03  5.457535e-02
```

```
predicted_train = predict(estimated)
predicted_test = predict(estimated, newdata = test_scaled)
mse_train = mean((training_scaled$motor_UPDRS-predicted_train )^2)
mse_test = mean((test_scaled$motor_UPDRS-predicted_test )^2)

mse_train
```

```
## [1] 0.09665287
```

```
mse_test
```

```
## [1] 0.09250051
```

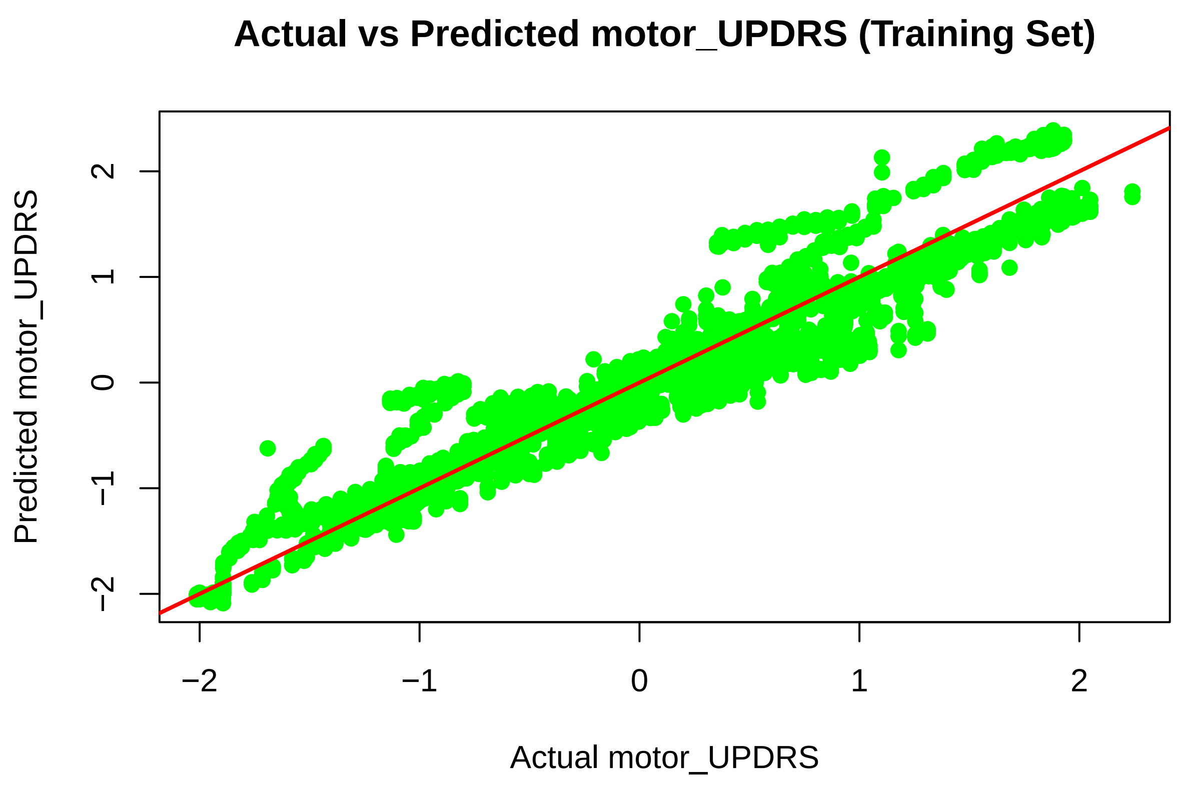
```
### to find the variables that contribute significantly to the model
summary(estimated)
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ ., data = training_scaled)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.06784 -0.16937  0.02134  0.20942  0.87136
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.032e-15  5.253e-03   0.000 1.000000
## subject.      -2.652e-02  6.089e-03  -4.356 1.36e-05 ***
## age           -3.529e-02  5.725e-03  -6.164 7.89e-10 ***
## sex            5.917e-02  6.468e-03   9.149  < 2e-16 ***
## test_time     -2.521e-03  5.308e-03  -0.475 0.634817    
## total_UPDRS    9.600e-01  6.117e-03 156.933  < 2e-16 ***
## Jitter...      1.695e-01  4.972e-02   3.410 0.000657 ***
## Jitter.Abs.   -7.946e-02  1.449e-02  -5.484 4.45e-08 ***
## Jitter.RAP     7.344e-01  6.256e+00   0.117 0.906553    
## Jitter.PPQ5   -3.549e-02  2.938e-02  -1.208 0.227127    
## Jitter.DDP    -8.338e-01  6.257e+00  -0.133 0.893988    
## Shimmer        1.672e-01  6.867e-02   2.435 0.014929 *  
## Shimmer.dB.   -2.789e-02  4.635e-02  -0.602 0.547387    
## Shimmer.APQ3   9.505e+00  2.563e+01   0.371 0.710735    
## Shimmer.APQ5  -1.431e-01  3.801e-02  -3.766 0.000169 ***
## Shimmer.APQ11  7.731e-02  2.056e-02   3.760 0.000173 ***
## Shimmer.DDA   -9.562e+00  2.563e+01  -0.373 0.709064    
## NHR            1.454e-03  1.558e-02   0.093 0.925651    
## HNR           -4.298e-03  1.227e-02  -0.350 0.726048    
## RPDE          -3.057e-02  7.589e-03  -4.028 5.75e-05 ***
## DFA           -6.872e-03  6.999e-03  -0.982 0.326256    
## PPE            5.458e-02  1.098e-02   4.970 7.02e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3119 on 3503 degrees of freedom
## Multiple R-squared:  0.9033, Adjusted R-squared:  0.9027 
## F-statistic:  1559 on 21 and 3503 DF,  p-value: < 2.2e-16
```

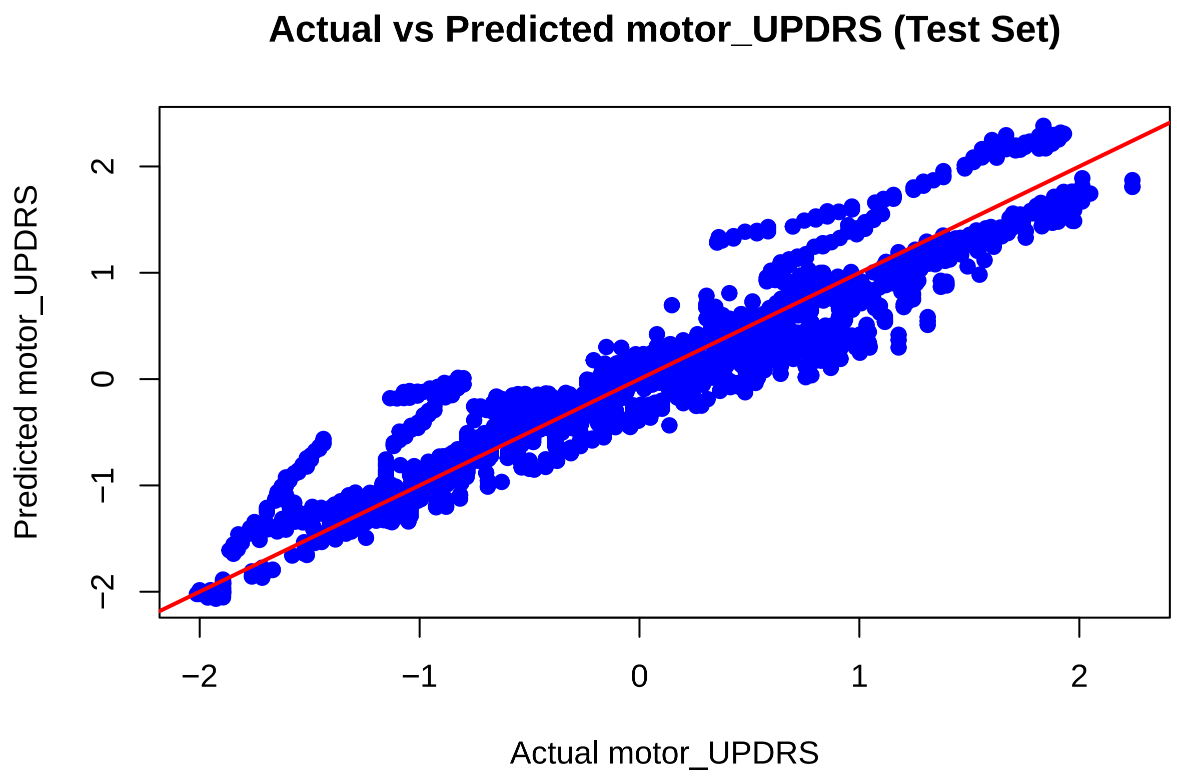
columns with the lowest p value is age (7.89e-10 ***), sex(< 2e-16***) and total updrs (< 2e-16 ***). The columns with*** has the highest significance.

Below is for ploting the difference between model. estimation and actual value

```r
# Plot for training data
plot(
  training_scaled$motor_UPDRS,
  predicted_train,
  xlab = "Actual motor_UPDRS",
  ylab = "Predicted motor_UPDRS",
  main = "Actual vs Predicted motor_UPDRS (Training Set)",
  pch = 19, col = "green"
)
abline(a = 0, b = 1, col = "red", lwd = 2)
```



```r
# Plot for test data
plot(
  test_scaled$motor_UPDRS,
  predicted_test,
  xlab = "Actual motor_UPDRS",
  ylab = "Predicted motor_UPDRS",
  main = "Actual vs Predicted motor_UPDRS (Test Set)",
  pch = 19, col = "blue"
)
abline(a = 0, b = 1, col = "red", lwd = 2)
```



Formula for loglikelihood: (from lecture slides)

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ln \left(1 + e^{-y_i \theta^T x_i}\right)$$

so, X is theta and y is sigma

$$\ell(\theta, \sigma) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - x_i^T \theta)^2$$

This is from class, derivation in notebook.

So we have loglike(theta,sigma) where theta is regression coefficients and sigma is the standard deviation

For task 3d we have

$$\hat{y} = X\hat{\theta} = X(X^\top X)^{-1} X^\top y = Py$$

from class, where P is the hat matrix and wich we produce in the function. since were using the ridge function we have to add that and we obtain

$$\hat{\theta} = (X^\top X + \lambda I)^{-1} X^\top y$$

```r
### values of theta and sigma from the model
theta_start <- coef(estimated)[-1]
sigma_start <- summary(estimated)$sigma
y_training <- as.matrix(training_scaled$motor_UPDRS)
x_training <- as.matrix(training_scaled[ , -which(names(training_scaled) == "motor_UPDRS")])
```

```r
loglike <- function(theta, sigma){
  n <- nrow(x_training)
  return ((-n/2)*log(2*pi*sigma ^2) - (1/(2*sigma^2))* sum((y_training-x_training%*%theta)^2))
}

ridge <- function(sigma, theta, lambda){
  first <- lambda * sum((theta)^2)
  second <- -loglike(theta, sigma)
  return (first + second)
  #return first
  #return (sigma)
}

ridge_opt <- function(lambda) {

  p <- ncol(x_training)                 # number of coefficients
  start <- c(rep(0, p), 0)      # [theta (p zeros), log_sigma = 0]
  minimize <- function(values) {
    theta <- values[1:p]
    sigma <- exp(values[p + 1])   # keep sigma > 0
    ridge(sigma, theta, lambda)
  }

  optim(par = start, fn = minimize, method = "BFGS")
}

#### X training has 21 zero as do values 1:p.
#loglike(theta_start,sigma_start)
#ridge(sigma = sigma_start, theta = theta_start, lambda = 10)
identity <- diag(ncol(x_training))

df <- function(lambda){
  lambda_i <- lambda*identity
  H = x_training%*%(solve(t(x_training)%*%x_training + lambda_i)%*%t(x_training))
  degree_free <- sum(diag(H))
  return(degree_free)
}
```

```r
lambdas <- c(1, 100, 1000)
x_test <- as.matrix(test_scaled[ , -which(names(test_scaled) == "motor_UPDRS")])
length_training <- ncol(x_training)
length_testing <- ncol(x_test)
for(i in lambdas){
  model <- ridge_opt(i)
  x_1 = as.matrix(x_training) ## training x
  x_2 <- as.matrix(x_test) ### testing x
  theta_hat_training <- as.matrix(model$par[1:length_training])
  theta_hat_testing <- as.matrix(model$par[1:length_testing])
  y_hat_training <- (x_1%*%theta_hat_training)
  y_hat_testing <- (x_2%*%theta_hat_testing)
  mse_training <- mean((training_scaled$motor_UPDRS-y_hat_training )^2)
  mse_testing <- mean((test_scaled$motor_UPDRS-y_hat_testing )^2)
  print(paste("RESULTS FOR LAMBDA =", i))
  print(paste("MSE FOR TRAINING DATA: ", mse_training))
  print(paste("MSE FOR TESTING DATA: ", mse_testing))
  }
```

```
## [1] "RESULTS FOR LAMBDA = 1"
## [1] "MSE FOR TRAINING DATA:  0.0966572618001955"
## [1] "MSE FOR TESTING DATA:  0.0925048944104364"
## [1] "RESULTS FOR LAMBDA = 100"
## [1] "MSE FOR TRAINING DATA:  0.0968043816283262"
## [1] "MSE FOR TESTING DATA:  0.0927424592375212"
## [1] "RESULTS FOR LAMBDA = 1000"
## [1] "MSE FOR TRAINING DATA:  0.100829537626891"
## [1] "MSE FOR TESTING DATA:  0.0972505348475704"
```

```
for(y in lambdas){
  dfr <- df(y)
  print(paste("THE DEGREE OF FREEDOM FOR LAMBDA : " ,y, "IS: ", dfr))
}
```

```
## [1] "THE DEGREE OF FREEDOM FOR LAMBDA :  1 IS:  18.8571965923825"
## [1] "THE DEGREE OF FREEDOM FOR LAMBDA :  100 IS:  14.7017171938565"
## [1] "THE DEGREE OF FREEDOM FOR LAMBDA :  1000 IS:  9.32147295417459"
```

When comparing the three Ridge models (lambda = 1, 100, 1000), the model with lambda = 1 gives the lowest test MSE (0.0925), indicating the best predictive performance on unseen data. As the penalty increases, both training and test MSE worsen, with lambda = 1000 producing the highest errors.

The degrees of freedom decrease as lambda increases (df(1) = 18.86, df(100) = 14.70, df(1000) = 9.32), which reflects stronger shrinkage and a less flexible model. While higher penalties reduce model complexity, they also introduce more bias, which is visible in the increased test MSE for lambda = 100 and lambda = 1000.

Overall, lambda = 1 is the most appropriate choice among the tested values, providing the best balance between model complexity and predictive accuracy.

Assignment 3. Logistic regression and basis function expansion

```
knitr::opts_chunk$set(echo = TRUE)
set.seed(1234)
# Load data
raw_data = read.csv("pima-indians-diabetes.csv", header=FALSE)
data = as_tibble(raw_data)
# The data has no header, meaning no names for the data.
# We give the columns names based on the lab instructions
colnames(data) <- c(
  "Pregnancies",
  "Glucose",
  "BloodPressure_mmHg",
  "SkinThickness_mm",
  "Insulin_muUml",
  "BMI_kgperm2",
  "Pedigree",
  "Age",
  "Diabetes"
)

# Some of the data have glucose = 0 for example, i think this
```

```r
# is just missing data, so will filter them out
data = data %>% filter(Glucose > 0)
```

We start by lading the data as a tibble, and assigning names to the columns based on the lab instructions: 1. Number of times pregnant.
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skinfold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index (weight in kg/(height in m)^2).
7. Diabetes pedigree function.
8. Age (years).
9. Diabetes (0=no or 1=yes).

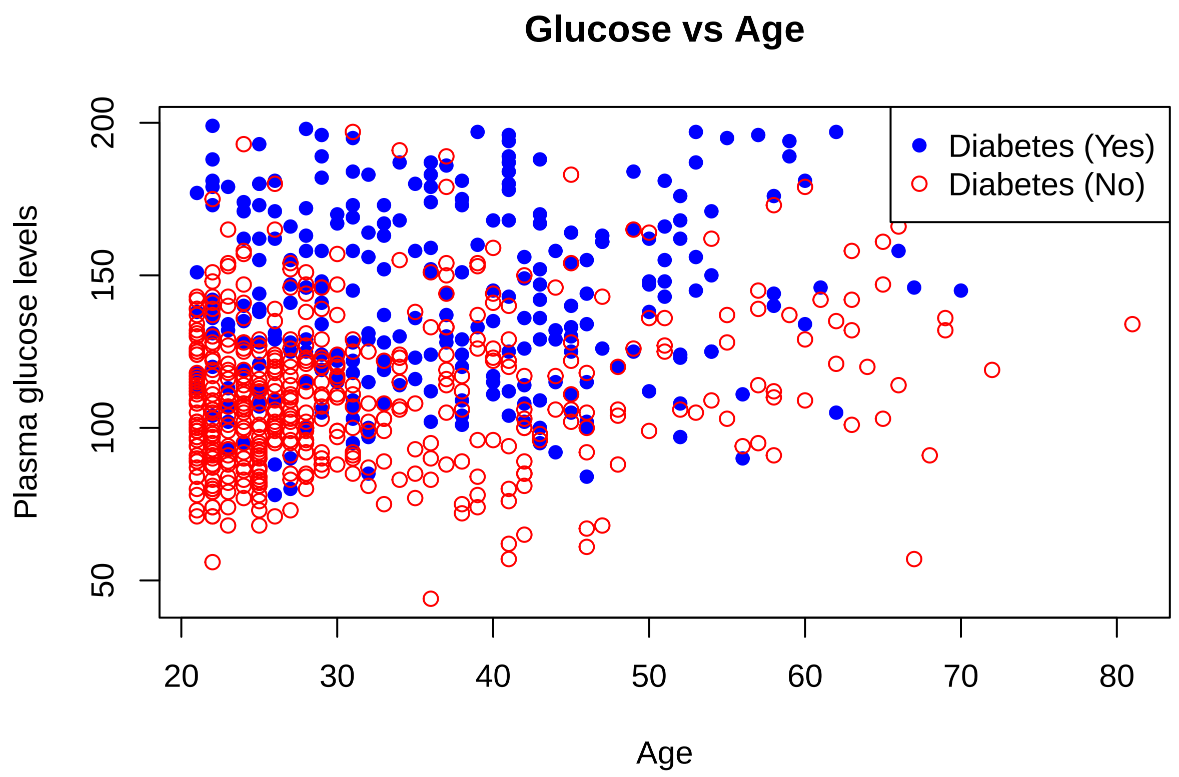
Lets visualise some of the data:

```r
head(data)
```

```
## # A tibble: 6 x 9
##   Pregnancies Glucose BloodPressure_mmHg SkinThickness_mm Insulin_muUml
##         <int>   <int>              <int>            <int>         <int>
## 1           6     148                 72               35             0
## 2           1      85                 66               29             0
## 3           8     183                 64                0             0
## 4           1      89                 66               23            94
## 5           0     137                 40               35           168
## 6           5     116                 74                0             0
## # i 4 more variables: BMI_kgperm2 <dbl>, Pedigree <dbl>, Age <int>,
## #   Diabetes <int>
```

1. Scatterplot

Lets make a scatterplot showing Plasma glucose concentration on Age, where the observations are also colored by Diabetes levels, **blue** for Diabetes and **red** otherwise. Additionally, those with 0 glucose is filtered away, since they are most likely just errors or not captured data.



Do you think that Diabetes is easy to classify by a standard logistic regression model that uses these two variables as features?

Based on the plot alone, it does not seems like diabetes can be easily classified with these two variables alone, at least not to a very high degree. At the extremes, say for **glucose $>$ 150**, there seems to be a majority of diabetics. However, since the question is in regards to standard logistic regression and the fact that there is no boundary that can be drawn that separates the data in good way.

2. Training a logistic regression model

We can train a logistic regression model by using **glm()**, with the formula being **Diabetes** ~ **Age** + **Glucose** and using the **"binomial"** family since we have a binary target 1 or 0.

```
fit = glm(Diabetes ~ Age + Glucose, family = "binomial", data=data)
summary(fit)
```

```
##
## Call:
## glm(formula = Diabetes ~ Age + Glucose, family = "binomial",
##     data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.24035    0.47873 -13.035  < 2e-16 ***
## Age          0.02333    0.00747   3.123  0.00179 **
## Glucose      0.03850    0.00342  11.257  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 986.70  on 762  degrees of freedom
## Residual deviance: 776.77  on 760  degrees of freedom
## AIC: 782.77
##
## Number of Fisher Scoring iterations: 4
```

The probabilistic equation of the estimated model is given by the sigmoid function:

$$p = \frac{1}{1+e^{-n}}\,,$$

where n is given by the linear prediction equation:

$$n = \beta_0 + \beta_1 x_1 + \beta_2 x_2\,.$$

Plugging our coeeficients into the equation, we get the final probabilisstic equation:

$$\begin{aligned} p &= \frac{1}{1+e^{-(-5.912449+0.024778\cdot x_1+0.035644\cdot x_2)}} \\ p &\approx \frac{1}{1+e^{(6-0.025\cdot x_1-0.036\cdot x_2)}}\,. \end{aligned}$$

We can now check if our equation is correct.

```r
sample = data[1,]
cf = coef(fit)
n = cf["(Intercept)"] + cf["Age"] * sample$Age + cf["Glucose"] * sample$Glucose
p = 1 / (1+exp(-n))
# Compare our equation to the predict function. Should be TRUE.
all.equal(p, predict(fit, type="response", sample), check.attributes = FALSE)
```

```
## [1] TRUE
```

Our equation matches the prediction function, which means it should be correct.

Lets now predict the whole dataset, and compute the models **error rate**.

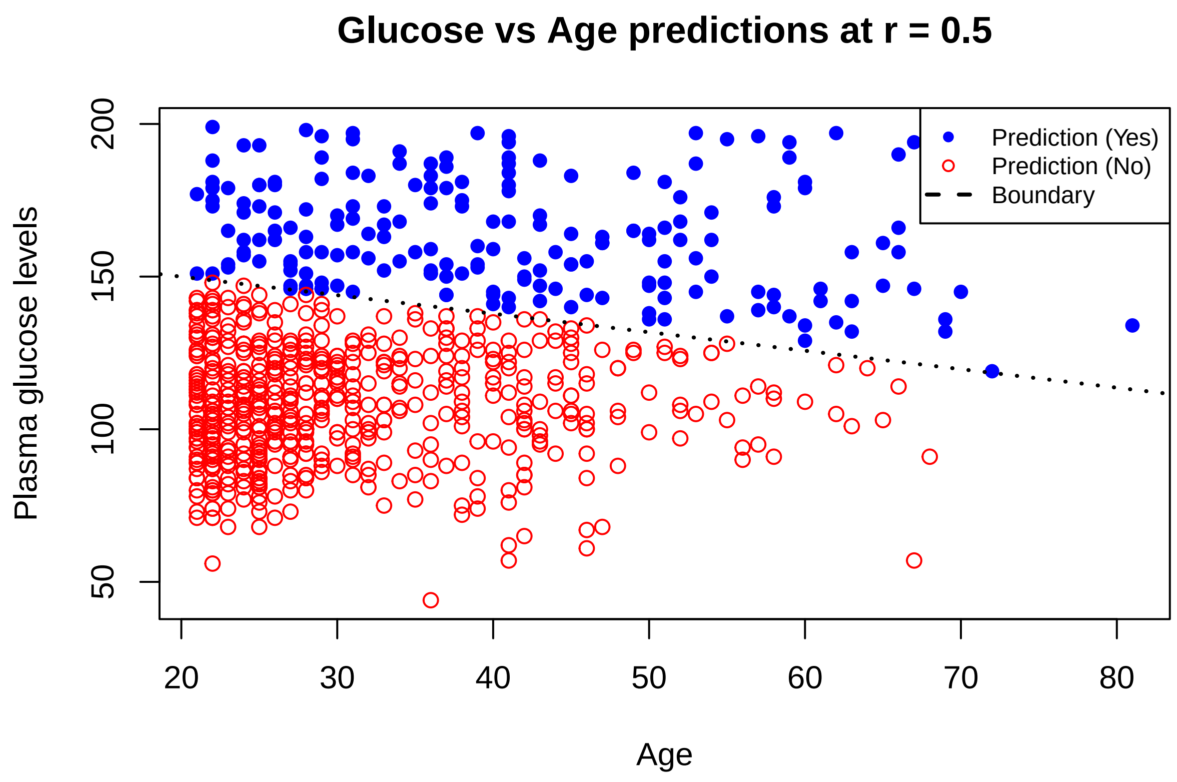
```r
probs = predict(fit, type="response") # Probabilities for each data point
# We use r = 0.5 as the classification threshold.
preds = ifelse(probs >= 0.5, 1, 0) # Convert to predictions 1 or 0

# We can now calculate the error rate as the number of incorrect
# predictions divided by the amount of predictions
sum(preds != data$Diabetes) / nrow(data)
```

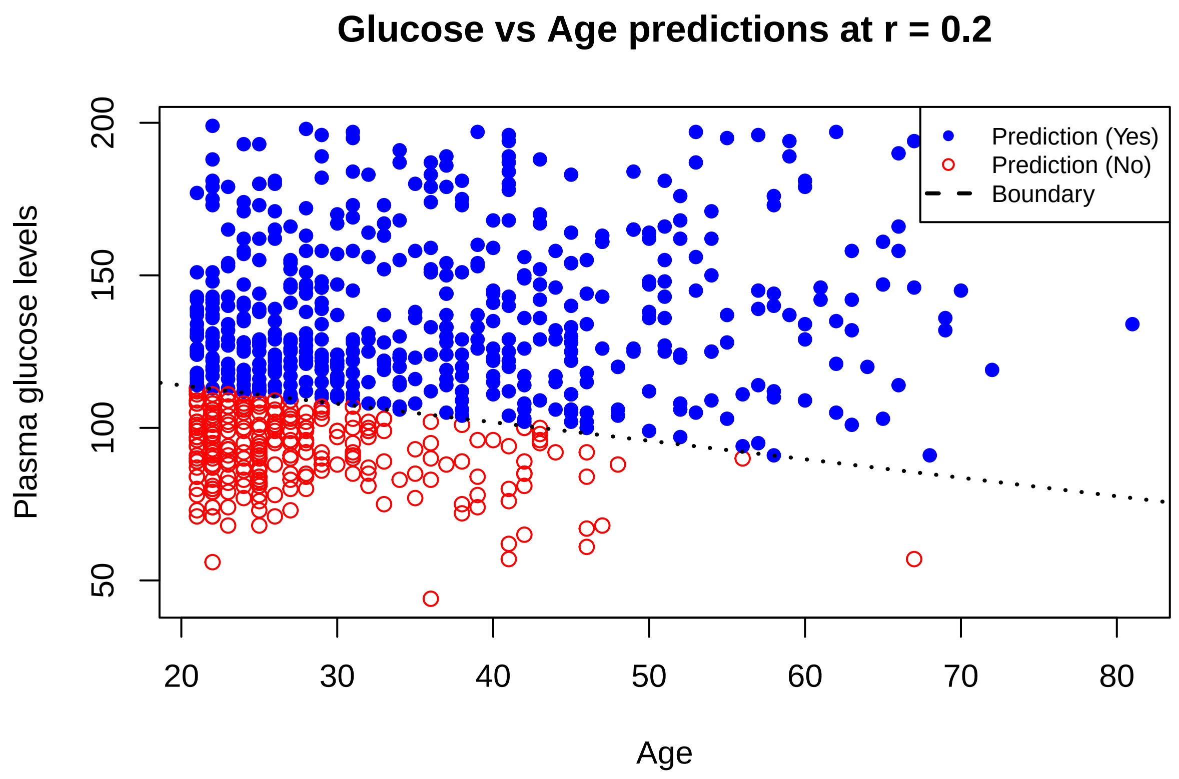
```
## [1] 0.259502
```

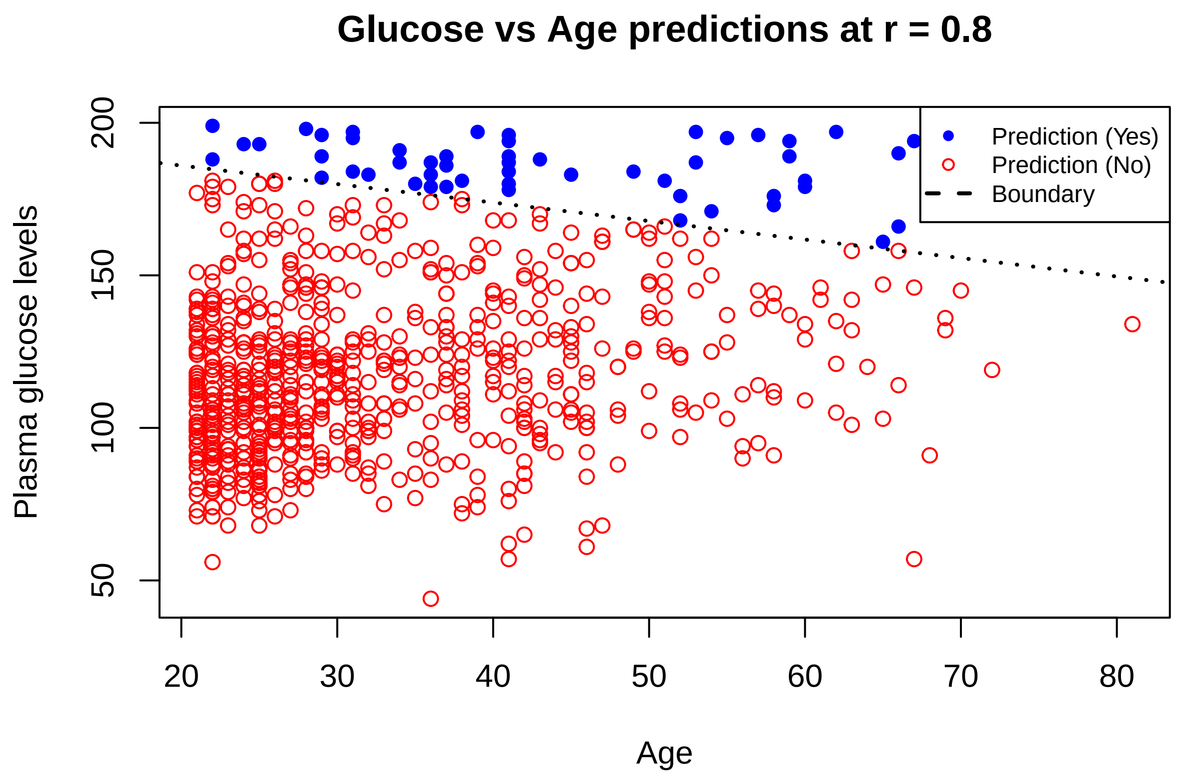
$\approx 26.0\%$ error rate.

We can now use the same predictions to create a new scatterplot showing the models predictions

```r
thresholds = c(0.5, 0.2, 0.8)
names(thresholds) = c("r = 0.5", "r = 0.2", "r = 0.8")
age_range = seq(0, 100) # Range that covers all ages in the plot

for (r in thresholds) {
  preds = ifelse(probs >= r, 1, 0) # compute the actual predictions based on the r value

  plot(x = data$Age,
       y = data$Glucose,
       main = paste("Glucose vs Age predictions at r =", r),
       ylab="Plasma glucose levels", xlab = "Age",
       col=ifelse(preds == 1, "blue", "red"),
       pch = ifelse(preds == 1, 16, 1))

  legend("topright",
         legend = c("Prediction (Yes)", "Prediction (No)", "Boundary"),
         col = c("blue", "red", "black"),
         pch = c(16, 1, NA),
         lty = c(NA, NA, 2),
         lwd = c(NA, NA, 2),
         bg="white",
         cex=0.75)


  # Calculate boundary
  n = qlogis(r) # Computes required n for getting the threshold probability
  y_boundary = (n - cf["(Intercept)"] - cf["Age"] * age_range) / cf["Glucose"]
  lines(x = age_range, y = y_boundary, col="black", lwd=2, lty="dotted")
}
```




Glucose vs Age predictions at r = 0.2
Plasma glucose levels
Age
Prediction (Yes)
Prediction (No)
Boundary
50
100
150
200
20
30
40
50
60
70
80



```
## Overall Diabetes Distribution
## Diabetic     : 266  (34.86%)
## Non-diabetic : 497  (65.14%)
## Total        : 763
## Age < 30 Distribution
## Diabetic     :  84  (21.37%)
## Non-diabetic : 309  (78.63%)
## Total <30    : 393
```

Quality of classification

Given the results I would say the model performed decently, While the error rate looks pretty good at a first glance, there are some things that has to be taken into consideration.

- The data is not evenly distributed ($\approx 65\%$ non diabetic).
  - Predicting every person as not having diabetes would therefor have $\approx 35\%$ error rate
  - Additionally, the distribution is even more skewed for ages under 30, which majority of the data is in.
- The model does however perform better than a null model (Intercept only).
  - Both the **Age** and **Glucose** coefficients show statistical significance when it comes to predicting Diabetes.
  - Increases in Age and Glucose levels both increase the likelihood of having diabetes.

3. Decision boundary

To find the decision boundary for

$$r = 0.5$$

we can use our models equation:

$$p = \frac{1}{1 + e^{-n}},$$

which will be at the boundary when $n = 0$:

$$\beta_0 + \beta_1 \cdot \text{Age} + \beta_2 \cdot \text{Glucose} = 0\,,$$

and since we are plotting it over Glucose vs Age, we need to rearrange the equation to find the where on the y-axis (Glucose) the model has its boundary, so we get:

$$\text{Glucose} = \frac{-\beta_0 - \beta_1 \cdot \text{Age}}{\beta_2}$$

See Glucose vs Age prediction plot for boundary. The decision boundary does not seem to catch the data distribution very well, this is because of the fact that overlap is very high in the data, which means a linear boundary cannot possibly fit the data very well.

4. Different thresholds (See plots from Step 2)

Using different threshold values such as $r = 0.2$ and $r = 0.8$ basically only changes the intercept, the slope of the boundary line remains the same, this in turn changes the predictions. For low r values, false positives increase, since a lower probability is needed in order to predict diabetes, while for high r values, the opposite happens, false negatives increase, since a high probability is required in order to predict that a person has diabetes.

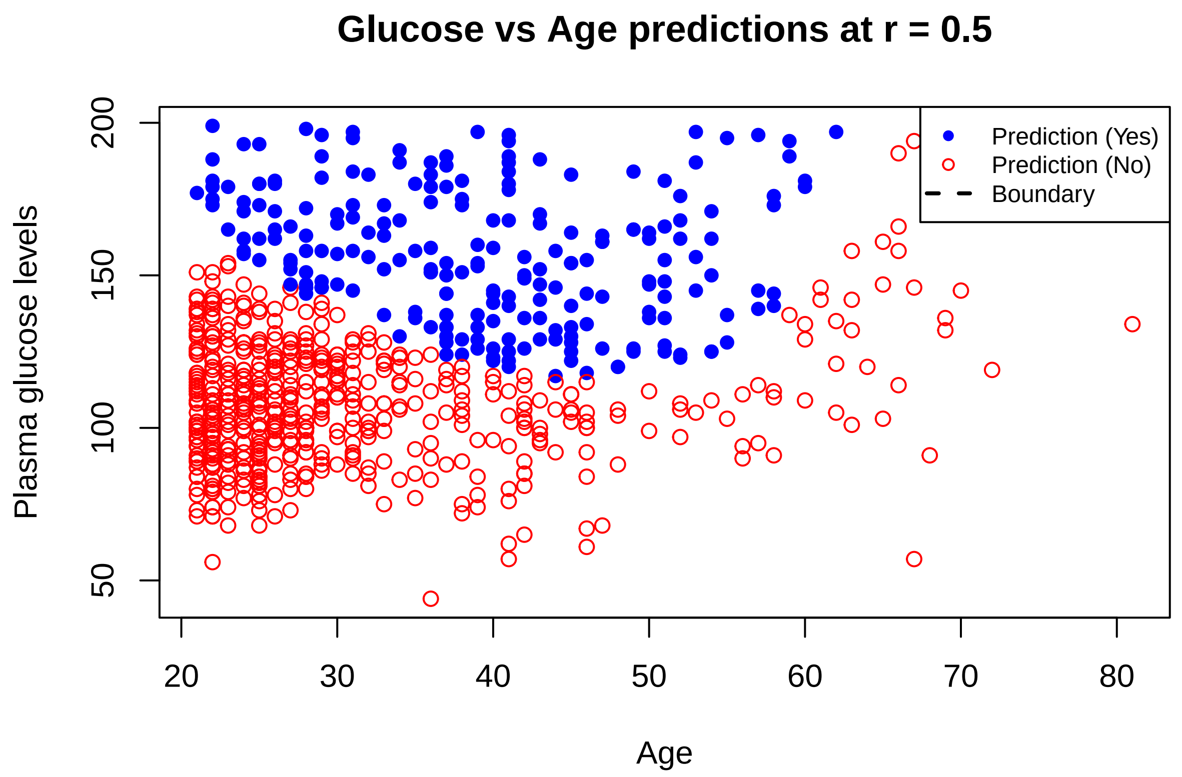
5. Basis function expansion

$$z_1 = x_1^4, \quad z_2 = x_1^3 x_2, \quad z_3 = x_1^2 x_2^2, \quad z_4 = x_1 x_2^3, \quad z_5 = x_2^4$$

```
data = data %>%
  mutate(z1 = Age^4,
         z2 = Age^3 * Glucose,
         z3 = Age^2 * Glucose^2,
         z4 = Age * Glucose^3,
         z5 = Glucose^4)

bfe = glm(Diabetes ~ Age+Glucose+z1+z2+z3+z4+z5, family = "binomial", data=data)
summary(bfe)
```

```
##
## Call:
## glm(formula = Diabetes ~ Age + Glucose + z1 + z2 + z3 + z4 +
##     z5, family = "binomial", data = data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.047e+01  1.222e+00  -8.568  < 2e-16 ***
## Age          1.422e-01  2.076e-02   6.847 7.52e-12 ***
## Glucose      5.010e-02  1.051e-02   4.768 1.86e-06 ***
## z1          -7.931e-08  9.375e-07  -0.085    0.933
## z2          -3.972e-07  1.210e-06  -0.328    0.743
## z3            2.133e-07  5.584e-07   0.382    0.702
## z4           -5.141e-08  1.080e-07  -0.476    0.634
## z5            3.296e-09  7.136e-09   0.462    0.644
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 986.70  on 762  degrees of freedom
## Residual deviance: 729.21  on 755  degrees of freedom
## AIC: 745.21
##
## Number of Fisher Scoring iterations: 6
```

```
probs = predict(bfe, type="response")
preds = ifelse(probs >= 0.5, 1, 0) # compute the actual predictions based on the r value

plot(x = data$Age,
     y = data$Glucose,
     main = paste("Glucose vs Age predictions at r = 0.5"),
     ylab="Plasma glucose levels", xlab = "Age",
     col=ifelse(preds == 1, "blue", "red"),
     pch = ifelse(preds == 1, 16, 1))

legend("topright",
       legend = c("Prediction (Yes)", "Prediction (No)", "Boundary"),
       col = c("blue", "red", "black"),
       pch = c(16, 1, NA),
       lty = c(NA, NA, 2),
       lwd = c(NA, NA, 2),
       bg="white",
       cex=0.75)
```



```r
# Error rate
sum(preds != data$Diabetes) / nrow(data)
```

```
## [1] 0.2385321
```

The error rate of $\approx 23.9\%$ is slightly lower than that of the model from step 2 which had an error rate of $\approx 26.0\%$. The decision boundary is no longer linear, this is probably because we added non-linearities to the formula, which in turn allows the model to take on more complex shapes for its decision boundary. However, while the error rate might be lower, the complexity of this model is also higher, as it has more coefficients. The added complexity might have made the model overfitted to the data.

Assignment 4

- Why can it be important to consider various probability thresholds in the classification problems, according to the book?

According to the coursebook, page 49-50, you should consider various tresholds when dealing with asymmetric categories. For example, in medical diagnosis its much worse to predict a sick patient being healthy (false negative) than a healthy patient being sick (false positive). Therefore, when choosing treshold you should choose a treshold that takes into consideration the importance of various categories being classified as false or true. Another intuitive example could be for self driving cars. Say you are trying to predict if entering a road from a stop sign is going to lead to a collision, or not. In this scenario, missing a potential gap and having to wait it way less destructive than thinking you have a gap, but then colliding with another car. A high threshold would then be appropriate for categorizing a scenario as having no collision.

- What ways of collecting correct values of the target variable for the supervised learning problems are mentioned in the book?

According to the book, there are many different ways of collecting target variables, which basically means the value or category you are trying to predict. One common way is to label data with the help of human experts in specific field, where they use available data and create targets based on the data. Such as cardiologist and ECG data (Page 14 in the coursebook). Data does not necesarily have to be labeled by experts, images are often also hand labeled, which does not require any expertise. Historical data can also be used to collect targets, the course book (Page 6) mentions predicting outcomes of soccer matches, in such a case, we do not have to label the data manually, as the historical matches themselves already provide what we are trying to predict, that is the outcome of the matches. Additionally, as sensors become more prevalent, they are a great way of collecting data, by having multiple sensors across both time and space. The target variables is then directly observed by sensors (Page 9 in course book).

- How can one express the cost function of the linear regression in the matrix form, according to the book?

$$J(\theta) = \frac{1}{n}\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$$

and with y hat = Xtheta

$$J(\theta) = \frac{1}{n}\|X\theta - \mathbf{y}\|_2^2$$

theta is the vector of model parameters (the weights) y hat is the vector of predicted values from the model. y is the vector of true target values. n is the number of data points. y hat - y is the error between predictions and true values. (y hat - y)^2 is the sum of squared errors.

found on page 41.

Source code

```r
####################
### Assignment 1 ###
####################
### Exercise 1
data <- read.csv("optdigits.csv", header = FALSE)

n <- nrow(data)

set.seed(12345)
train_ids <- sample(1:n, floor(n * 0.5))
train <- data[train_ids, ]

remaining_ids <- setdiff(1:n, train_ids)

set.seed(12345)
valid_ids <- sample(remaining_ids, floor(n * 0.25))
valid <- data[valid_ids, ]

test_ids <- setdiff(remaining_ids, valid_ids)
test <- data[test_ids, ]

### Exercise 2
library(kknn)

train$V65 <- as.factor(train$V65)
test$V65 <- as.factor(test$V65)
```

```r
# Predict the training data itself.
m_train <- kknn(V65~., train = train, test = train, k = 30, kernel = "rectangular")

# Predict on unseen data(test).
m_test <- kknn(V65~., train = train, test = test, k = 30, kernel = "rectangular")

pred_train_values <- predict(m_train)
pred_test_values <- predict(m_test)

cm_train <- table(pred_train_values, train$V65)
cm_test <- table(pred_test_values, test$V65)

cm_train
cm_test

NumberOfMisCalc <- function(cm) {
  cat("\n")
  numbers <- rownames(cm)
  for (num in numbers) {
    misclassified <- sum(cm[num,]) - cm[num, num]
    cat("The number", num, "was misclassified", misclassified, "times\n")
  }
}

# NumberOfMisCalc(cm_train)
# NumberOfMisCalc(cm_test)

CalcMisError <- function(cm) {
  correct <- sum(diag(cm))       # The correctly predicted values.
  total <- sum(cm)               # total values.
  return (1 - correct/total)     # 1 - rate of correct classification = misclassification rate.
}

train_mis_error <- CalcMisError(cm_train)
test_mis_error <- CalcMisError(cm_test)

train_mis_error
test_mis_error

### Exercise 3
# Probability matrix.
train_probs <- m_train$prob

# True labels.
y_true_num <- as.numeric(as.character(train$V65))

# Correct probability per row.
correct_train_prob <- train_probs[cbind(1:nrow(train_probs), y_true_num + 1)]

# Indices of all true "8" rows.
idx_8 <- which(y_true_num == 8)

# Correct probabilities for "8"s.
```

```r
prob_8 <- correct_train_prob[idx_8]

# Sort by probability, highest to lowest.
order_idx <- order(prob_8, decreasing = TRUE)

# Easiest 2 and hardest 3.
easiest <- idx_8[order_idx[1:2]]
hardest <- idx_8[tail(order_idx, 3)]

# Heatmap function.
plot_digit <- function(row) {
  m <- matrix(as.numeric(row[1:64]), nrow = 8, byrow = TRUE)
  heatmap(m, Colv = NA, Rowv = NA, scale = "none")
}

# Plot easiest 2.
plot_digit(train[easiest[1], ])
plot_digit(train[easiest[2], ])

# Plot hardest 3.
plot_digit(train[hardest[1], ])
plot_digit(train[hardest[2], ])
plot_digit(train[hardest[3], ])

### Exercise 4
Ks <- 1:30
train_errors <- numeric(length(Ks))
valid_errors <- numeric(length(Ks))

for (i in seq_along(Ks)) {
  k <- Ks[i]

  # Predict on training
  model_train <- kknn(V65~., train = train, test = train, k = k, kernel = "rectangular")
  pred_train <- predict(model_train)
  cm_train_k <- table(pred_train, train$V65)
  train_errors[i] <- CalcMisError(cm_train_k)

  # Predict on validation
  model_valid <- kknn(V65 ~ ., train = train, test = valid, k = k, kernel = "rectangular")
  pred_valid <- predict(model_valid)
  cm_valid_k <- table(pred_valid, valid$V65)
  valid_errors[i] <- CalcMisError(cm_valid_k)
}

plot(Ks, train_errors, type = "l", col = "blue", ylim = c(0, max(valid_errors)),
     xlab = "K", ylab = "Misclassification error", main = "Train vs Validation Error")
lines(Ks, valid_errors, col = "red")
legend("topright", legend = c("Training error", "Validation error"),
       col = c("blue", "red"), lty = 1)

# Find optimal K.
optimal_K <- Ks[which.min(valid_errors)]
```

```r
optimal_K

# Calc misclassification error.
m_opt_test <- kknn(V65 ~ ., train = train, test = test, k = optimal_K, kernel = "rectangular")
pred_opt_test <- predict(m_opt_test)

cm_opt_test <- table(pred_opt_test, test$V65)
error_opt_test <- CalcMisError(cm_opt_test)

opt_train_error <- train_errors[optimal_K]
opt_valid_error <- valid_errors[optimal_K]

opt_train_error
opt_valid_error
error_opt_test

### Exercise 5
Ks <- 1:30
valid_ce <- numeric(length(Ks))

epsilon <- 1e-15

for (i in seq_along(Ks)) {
  k <- Ks[i]

  model_valid <- kknn(V65~., train = train, test = valid, k = k, kernel = "rectangular")

  prob_valid <- model_valid$prob

  y_true <- as.numeric(as.character(valid$V65))

  correct_prob <- prob_valid[cbind(1:nrow(prob_valid), y_true + 1)]

  # Cross-entropy calculation
  valid_ce[i] <- -mean(log(correct_prob + epsilon))
}

# Plot cross-entropy vs K.
plot(Ks, valid_ce, type = "l", col = "red",
     xlab = "K", ylab = "Validation Cross-Entropy",
     main = "Validation Cross-Entropy vs K")

optimal_K <- Ks[which.min(valid_ce)]
optimal_K

library(caret)
library(dplyr)

####################
### Assignment 2 ###
####################

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

```r
set.seed(12345)
n <- dim(data)[1]
id <- sample(1:n, floor(n*0.6))
training <-data[id,]
test <- data[-id,]

scaler <- preProcess(training)
training_scaled <- predict(scaler,training)
test_scaled <- predict(scaler,test)

### ass . 2

# train the model on training data
estimated = lm(formula = motor_UPDRS~., data = training_scaled)

coefficients(estimated)

predicted_train = predict(estimated)
predicted_test = predict(estimated, newdata = test_scaled)
mse_train = mean((training_scaled$motor_UPDRS-predicted_train )^2)
mse_test = mean((test_scaled$motor_UPDRS-predicted_test )^2)

mse_train
mse_test

### to find the variables that contribute significantly to the model
summary(estimated)
### columns with the lowest p value is age (7.89e-10 ***), sex(< 2e-16 ***) and total updrs (< 2e-16 **


### values of theta and sigma from the model
theta_start <- coef(estimated)[-1]
sigma_start <- summary(estimated)$sigma
y_training <- as.matrix(training_scaled$motor_UPDRS)
x_training <- as.matrix(training_scaled[ , -which(names(training_scaled) == "motor_UPDRS")])

loglike <- function(theta, sigma){
  n <- nrow(x_training)
  return ((-n/2)*log(2*pi*sigma ^2) - (1/(2*sigma^2))* sum((y_training-x_training%*%theta)^2))
}

ridge <- function(sigma, theta, lambda){
  first <- lambda * sum((theta)^2)
  second <- -loglike(theta, sigma)
  return (first + second)
  #return first
  #return (sigma)
}

ridge_opt <- function(lambda) {

  p <- ncol(x_training)                 # number of coefficients
  start <- c(rep(0, p), 0)      # [theta (p zeros), log_sigma = 0]
  minimize <- function(values) {
    theta <- values[1:p]
    sigma <- exp(values[p + 1])   # keep sigma > 0
    ridge(sigma, theta, lambda)
  }

  optim(par = start, fn = minimize, method = "BFGS")
}

#### X training has 21 zero as do values 1:p.
#loglike(theta_start,sigma_start)
#ridge(sigma = sigma_start, theta = theta_start, lambda = 10)
identity <- diag(ncol(x_training))

df <- function(lambda){
  lambda_i <- lambda*identity
  H = x_training%*%(solve(t(x_training)%*%x_training + lambda_i)%*%t(x_training))
  degree_free <- sum(diag(H))
  return(degree_free)
}

lambdas <- c(1, 100, 1000)
x_test <- as.matrix(test_scaled[ , -which(names(test_scaled) == "motor_UPDRS")])
length_training <- ncol(x_training)
length_testing <- ncol(x_test)
for(i in lambdas){
  model <- ridge_opt(i)
  x_1 = as.matrix(x_training) ## training x
  x_2 <- as.matrix(x_test) ### testing x
  theta_hat_training <- as.matrix(model$par[1:length_training])
  theta_hat_testing <- as.matrix(model$par[1:length_testing])
  y_hat_training <- (x_1%*%theta_hat_training)
  y_hat_testing <- (x_2%*%theta_hat_testing)
  mse_training <- mean((training_scaled$motor_UPDRS-y_hat_training )^2)
  mse_testing <- mean((test_scaled$motor_UPDRS-y_hat_testing )^2)
  print(paste("RESULTS FOR LAMBDA =", i))
  print(paste("MSE FOR TRAINING DATA: ", mse_training))
  print(paste("MSE FOR TESTING DATA: ", mse_testing))
}

for(y in lambdas){
  dfr <- df(y)
  print(paste("THE DEGREE OF FREEDOM FOR LAMBDA : " ,y, "IS: ", dfr))
}

####################
### Assignment 3 ###
####################
set.seed(1234)
library(dplyr)
library(tidyr)
# Load data
raw_data = read.csv("pima-indians-diabetes.csv", header=FALSE)
```

```r
data = as_tibble(raw_data)
# The data has no header, meaning no names for the data.
# We give the columns names based on the lab instructions
colnames(data) <- c(
  "Pregnancies",
  "Glucose",
  "BloodPressure_mmHg",
  "SkinThickness_mm",
  "Insulin_muUml",
  "BMI_kgperm2",
  "Pedigree",
  "Age",
  "Diabetes"
)

# Some of the data have glucose = 0 for example, i think this is just missing data, so will filter them
data = data %>% filter(Glucose > 0)

head(data)

xlim = range(data %>% filter(Glucose > 0) %>% select(Age))
ylim = range(data %>% filter(Glucose > 0) %>% select(Glucose))

diabetes = data %>% filter(Diabetes==1, Glucose > -1)
not_diabetes = data %>% filter(Diabetes==0, Glucose > -1)
plot(x = diabetes$Age, y = diabetes$Glucose, main="Glucose vs Age", ylab="Plasma glucose levels", xlab =
points(x = not_diabetes$Age, y=not_diabetes$Glucose,  col="red")
legend("topright", legend = c("Diabetes (Yes)", "Diabetes (No)"), col = c("blue", "red"), pch = c(16, 1)

fit = glm(Diabetes ~ Age + Glucose, family = "binomial", data=data)
summary(fit)

sample = data[1,]
cf = coef(fit)
n = cf["(Intercept)"] + cf["Age"] * sample$Age + cf["Glucose"] * sample$Glucose
p = 1 / (1+exp(-n))
# Compare our equation to the predict function. Should be TRUE.
all.equal(p, predict(fit, type="response", sample), check.attributes = FALSE)

probs = predict(fit, type="response") # Probabilities for each data point
# We use r = 0.5 as the classification threshold.
preds = ifelse(probs >= 0.5, 1, 0) # Convert to predictions 1 or 0

# We can now calculate the error rate as the number of incorrect predictions divided by the amount of p
sum(preds != data$Diabetes) / nrow(data)

thresholds = c(0.5, 0.2, 0.8)
names(thresholds) = c("r = 0.5", "r = 0.2", "r = 0.8")
age_range = seq(0, 100) # Range that covers all ages in the plot

for (r in thresholds) {
  preds = ifelse(probs >= r, 1, 0) # compute the actual predictions based on the r value
  plot(x = data$Age,
       y = data$Glucose,
       main = paste("Glucose vs Age predictions at r =", r),
       ylab="Plasma glucose levels", xlab = "Age",
       col=ifelse(preds == 1, "blue", "red"),
       pch = ifelse(preds == 1, 16, 1))

  legend("topright",
         legend = c("Prediction (Yes)", "Prediction (No)", "Boundary"),
         col = c("blue", "red", "black"),
         pch = c(16, 1, NA),
         lty = c(NA, NA, 2),
         lwd = c(NA, NA, 2),
         bg="white",
         cex=0.75)


  # Calculate boundary
  n = qlogis(r) # Computes required n for getting the threshold probability
  y_boundary = (n - cf["(Intercept)"] - cf["Age"] * age_range) / cf["Glucose"]
  lines(x = age_range, y = y_boundary, col="black", lwd=2, lty="dotted")
}

total_n = dim(data)[1]
diabetic_n = sum(data$Diabetes == 1)
nondiabetic_n = sum(data$Diabetes == 0)

cat("Overall Diabetes Distribution\n")
cat(sprintf("Diabetic     : %3d  (%.2f%%)\n", diabetic_n,   100 * diabetic_n / total_n))
cat(sprintf("Non-diabetic : %3d  (%.2f%%)\n", nondiabetic_n, 100 * nondiabetic_n / total_n))
cat(sprintf("Total        : %3d\n\n", total_n))

under30 <- data$Age < 30
diab_under30    <- sum(data$Diabetes == 1 & under30)
nondiab_under30<- sum(data$Diabetes == 0 & under30)
total_under30  <- diab_under30 + nondiab_under30

cat("Age < 30 Distribution\n")
cat(sprintf("Diabetic     : %3d  (%.2f%%)\n", diab_under30,   100 * diab_under30 / total_under30))
cat(sprintf("Non-diabetic : %3d  (%.2f%%)\n", nondiab_under30, 100 * nondiab_under30 / total_under30))
cat(sprintf("Total <30    : %3d\n", total_under30))


data = data %>%
  mutate(z1 = Age^4,
         z2 = Age^3 * Glucose,
         z3 = Age^2 * Glucose^2,
         z4 = Age * Glucose^3,
         z5 = Glucose^4)

bfe = glm(Diabetes ~ Age+Glucose+z1+z2+z3+z4+z5, family = "binomial", data=data)
summary(bfe)
```

```r
probs = predict(bfe, type="response")
preds = ifelse(probs >= 0.5, 1, 0) # compute the actual predictions based on the r value

plot(x = data$Age,
     y = data$Glucose,
     main = paste("Glucose vs Age predictions at r = 0.5"),
     ylab="Plasma glucose levels", xlab = "Age",
     col=ifelse(preds == 1, "blue", "red"),
     pch = ifelse(preds == 1, 16, 1))

legend("topright",
       legend = c("Prediction (Yes)", "Prediction (No)", "Boundary"),
       col = c("blue", "red", "black"),
       pch = c(16, 1, NA),
       lty = c(NA, NA, 2),
       lwd = c(NA, NA, 2),
       bg="white",
       cex=0.75)

# Error rate
sum(preds != data$Diabetes) / nrow(data)
```