Lab 1 Report

Marijn Jaarsma & Simon Jorstedt & Hugo Morvan

2023-11-19

Libraries

```
## Loading required package: ggplot2

## Loading required package: lattice

## Warning: package 'kknn' was built under R version 4.3.2

##
## Attaching package: 'kknn'

## The following object is masked from 'package:caret':
##
##     contr.dummy
```

Assignment 1

Assignment 1.1

> Import the data into R and divide it into training, validation and test sets (50%/25%/25%) by using the partitioning principle specified in the lecture slides.

```
# Read data
digit_data <- read.csv("optdigits.csv", header=FALSE)

# Partition data (according to Oleg)
n = dim(digit_data)[1]

set.seed(12345)
id = sample(1:n, floor(n*0.5))
digits_train = digit_data[id,]
id1 = setdiff(1:n, id)

set.seed(12345)
id2 = sample(id1, floor(n*0.25))
digits_valid = digit_data[id2,]
id3 = setdiff(id1,id2)
digits_test = digit_data[id3,]
```

Assignment 1.2

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 (use table()) • Misclassification errors for the training and test data

```
#Fitting the classifier
digits_kknn_test <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = 30, kernel = "rectangular")
kknn_fit_test <- fitted(digits_kknn_test)
conf_mat_test <- table(obs = digits_valid$V65, pred = kknn_fit_test)
conf_mat_test
```

```
##    pred
## obs  0  1  2  3  4  5  6  7  8  9
##   0 95  0  0  0  0  0  1  0  0  0
##   1  0 99  6  1  0  0  1  0  0  1
##   2  0  1 84  0  0  0  0  1  0  1
##   3  0  0  0 85  0  0  0  3  2  0
##   4  0  2  0  0 94  0  1  1  0  5
##   5  0  0  0  1  0 81  0  0  0 10
##   6  0  0  0  0  0  0 95  0  0  0
##   7  0  1  0  0  0  0  0 91  0  1
##   8  0  5  0  0  0  0  0  0 93  0
##   9  0  1  0  1  1  0  0  2  1 87
```

```
digits_kknn_train <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_train, k = 30, kernel = "rectangular")
kknn_fit_train <- fitted(digits_kknn_train)
conf_mat_train <- table(obs = digits_train$V65, pred = kknn_fit_train)
conf_mat_train
```

```
##    pred
## obs   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
```

```
#Misclassification error
get_mean_misc_err <- function(conf_mat){
  #Given a confusion matrix, return the misclassification error
  (sum(conf_mat)-sum(diag(conf_mat)))/sum(conf_mat)
}
get_indiv_misc_err <- function(conf_mat){
  #Given a confusion matrix, return the misclassification error for each digit
```

```
  err = c()
  for(i in 1:10){
    #correct this
    err[i] <- (sum(conf_mat[i,]-conf_mat[i,i])/sum(conf_mat[i,]))
  }
  return(Misc_error = err)
}
print("Misclassification error for test data :")
```

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

```
get_mean_misc_err(conf_mat_test)
```

```
## [1] 0.05340314
```

```
get_indiv_misc_err(conf_mat_test)
```

```
##  [1] -8.895833 -8.166667 -8.655172 -8.444444 -8.126214 -7.804348 -9.000000
##  [8] -8.784946 -8.489796 -8.354839
```

```
print("Misclassification error for train data :")
```

```
## [1] "Misclassification error for train data :"
```

```
get_mean_misc_err(conf_mat_train)
```

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

```
get_indiv_misc_err(conf_mat_train)
```

```
##  [1] -9.000000 -8.179487 -8.895833 -8.840426 -8.085714 -8.447514 -8.895833
##  [8] -8.780220 -8.215686 -8.150000
```

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

Some digits have a really high quality of prediction (6 -> 0% error, 7 -> 0.9% error while others have a relatively worst quality (8 -> 10.3% error, 4 -> 13.8% error). Overall, the quality is quite good, with a mean misclassification error of 5.3% for the test data and 4.5% for the train data.

Assignment 1.3

> Find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class).

```r
#Find all fitted values classified as 8 (CL = 8), find the max and min probabilities in
 ↪  prob

#Fitted values
fitted_eight <- predict(digits_kknn_train, data = digits_train)
#True values
actual_eight <- digits_train$V65
#Probabilities
probs <- digits_kknn_train[["prob"]]
#Probabilities for 8
probs_eight <- probs[,9]

my_df <- data.frame("actual" = actual_eight, "fitted" = fitted_eight , "prob" =
 ↪  probs_eight)
#filter my_df to only keep the 8s correctly fitted
my_df <- my_df[my_df$actual == 8,]
my_df <- my_df[my_df$fitted == 8,]

#Find the max and min probabilities
maxp <- 0
minp <- 1
for(i in 1:length(probs_eight)){
  if(factor(digits_kknn_train$fitted.values[i]) == 8){
   ↪  #https://www.tutorialspoint.com/how-to-extract-the-factor-levels-from-factor-column-in-an-r-data-
    maxp <- max(probs_eight[i], maxp)
    minp <- min(probs_eight[i], minp)
  }
}
# Find all the index where p=maxp and p= minp :
high_idx = c()
low_idx = c()
for(i in 1:length(probs_eight)){
  if(probs_eight[i] == maxp){
    high_idx <- append(high_idx, i)
  }
  if(probs_eight[i] == minp){
    low_idx <- append(low_idx, i)
  }
}
high_idx
```

```
##  [1]  129  195  211  233  292  294  515  601  650  679  684  693  726  729  752
## [16]  763  768  779  855  864  899  929 1006 1092 1134 1216 1227 1261 1295 1318
## [31] 1355 1380 1387 1397 1419 1472 1533 1607 1646 1686
```

```r
# 129  195  211  233  292  294  515  601  650  679  684  693  726  729  752  763  768
 ↪  779 855  864  899  929 1006 1092 1134 1216 1227 1261 1295 1318 1355 1380 1387 1397
 ↪  1419 1472 1533 1607 1646 1686
low_idx
```

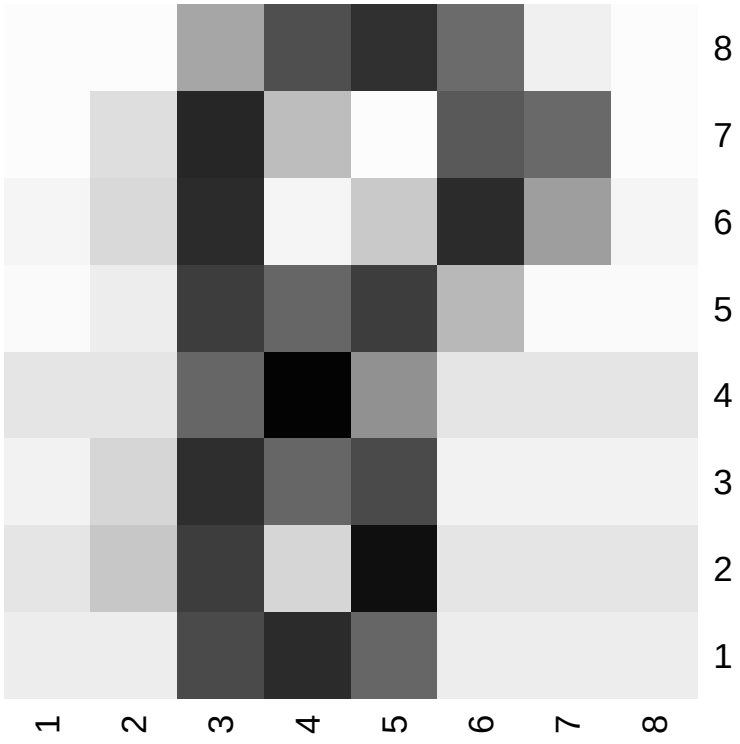
```
## [1]  141  258  469  560  629  881 1274 1716
```

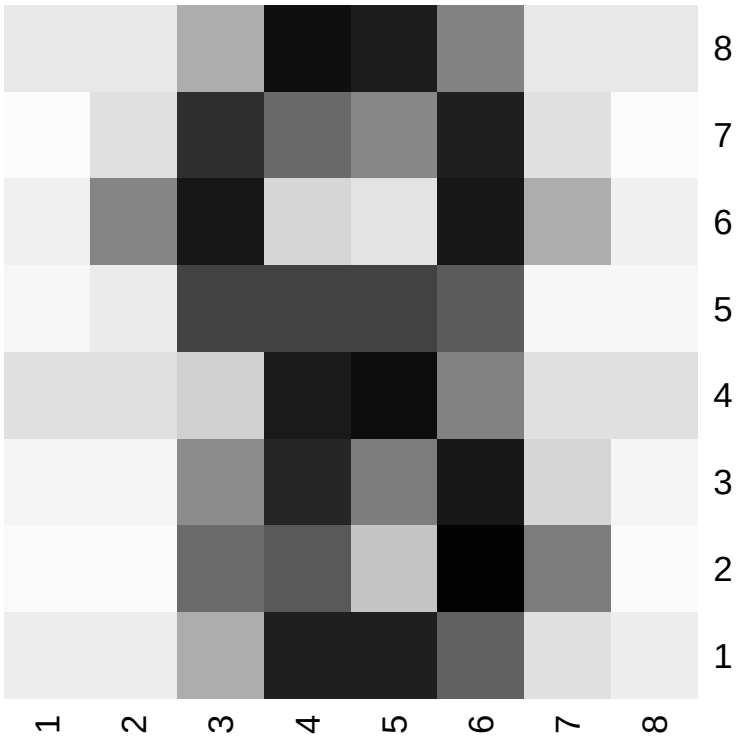
```
# 141  258  469  560  629  881 1274 1716
```

Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits (by using e.g. heatmap() function with parameters Colv=NA and Rowv=NA) and comment on whether these cases seem to be hard or easy to recognize visually.

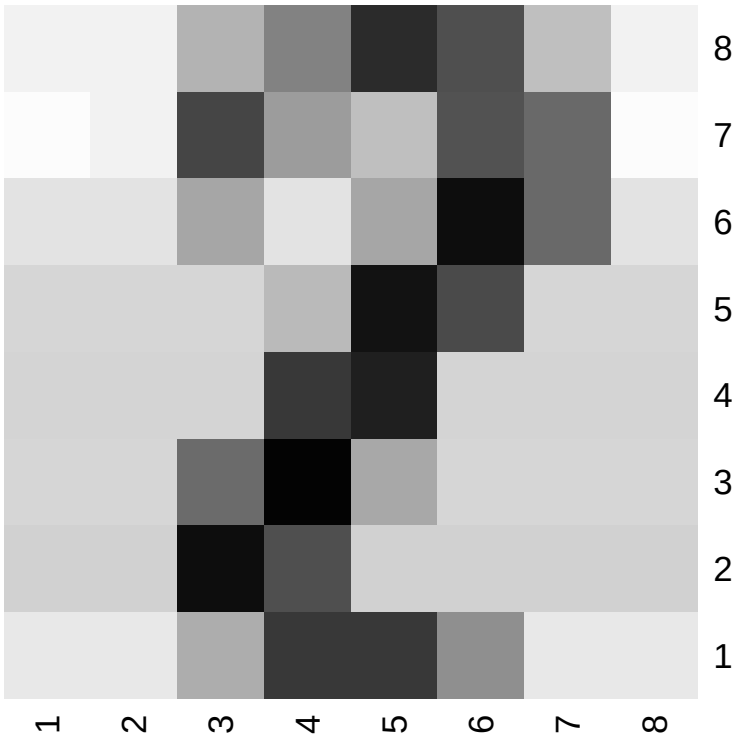
```
visualise_dig <- function(idx, data){
  #Given an index (idx) and a dataframe (data), visualize the digit at data[idx]
  raw_dig <- data[idx,][-65]
  mat = matrix(as.numeric(raw_dig), nrow = 8)
  heatmap(apply(t(mat),2,rev), Colv=NA, Rowv=NA, col=paste("gray",99:1,sep=""))
}
```

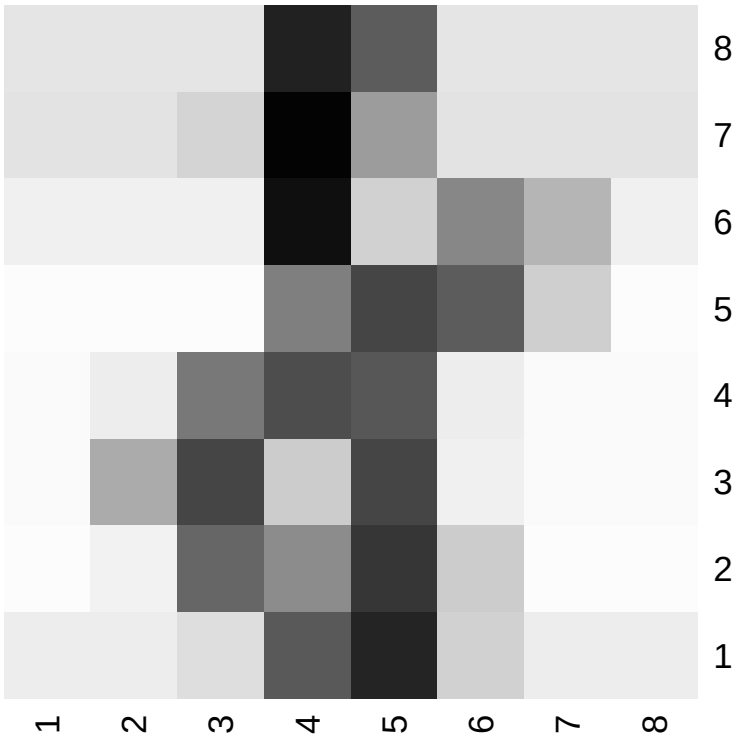
```
#High probabilities
for(i in 1:2){
  visualise_dig(high_idx[i], digits_train)
}
```

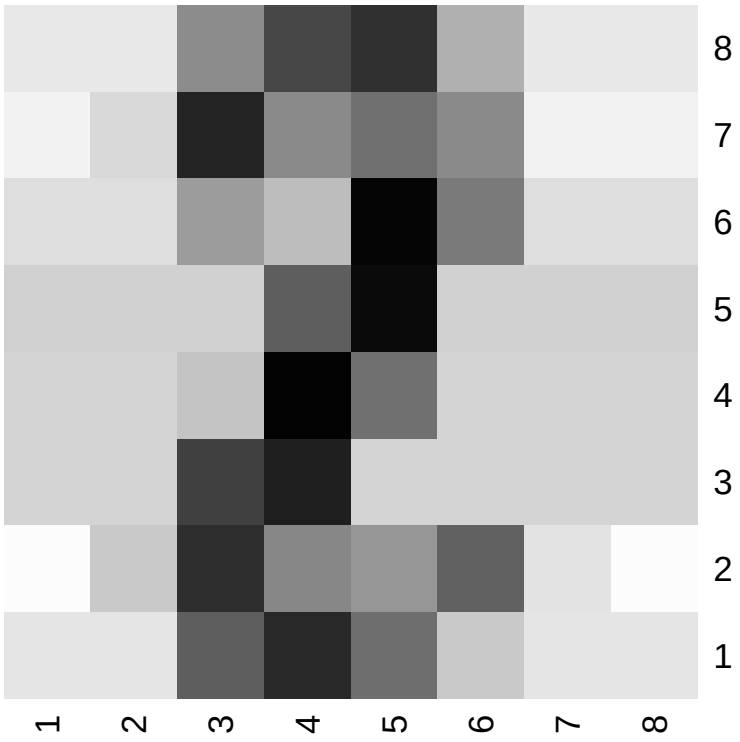





```
#Low probabilities
for(i in 1:3){
  visualise_dig(low_idx[i], digits_train)
}
```


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



Assignment 1.4

Fit a K-nearest neighbor classifiers to the training data for different values of K K = 1,2, … , 30 and plot the dependence of the training and validation misclassification errors on the value of K (in the same plot). How does the model complexity change when K increases and how does it affect the training and validation errors? Report the optimal K 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.

```
k_values <- c()
misc_errs_test <- c()
misc_errs_train <- c()

for(i in c(1:30)){
  #Fitting the classifier
  #Test:
  digits_kknn_test_i <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = i, kernel = "rectangular")
  #Train:
  digits_kknn_train_i <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_train, k = i, kernel = "rectangular")

  #kbetter <- train.kknn(formula = as.factor(V65) ~ . , data = digits_train, ks=i, kernel
   ↪  = "rectangular")
  #Predicting the values
```

```
  fitted_values_valid <- predict(digits_kknn_test_i, data = digits_valid)
  fitted_values_train <- predict(digits_kknn_train_i, data = digits_train)
  #Confusion matrix
  conf_mat_test <- table(obs = digits_valid$V65, pred = fitted_values_valid)
  conf_mat_train <- table(obs = digits_train$V65, pred = fitted_values_train)
  #Misclassification error
  misc_err_test <- get_mean_misc_err(conf_mat_test)
  misc_err_train <- get_mean_misc_err(conf_mat_train)

  #Append to the vectors
  k_values <- append(k_values, i)
  misc_errs_test <- append(misc_errs_test, misc_err_test)
  misc_errs_train <- append(misc_errs_train, misc_err_train)
}
df_ks <- data.frame("k" = k_values, "misc_err_test" = misc_errs_test, "misc_err_train" =
 ↪  misc_errs_train)
```

```
#Plotting misc_errs_test and misc_errs_train as a function of k
ggplot(df_ks, aes(x = k, y = misc_err_test)) + geom_line() + geom_point() +
 ↪  geom_line(aes(y = misc_err_train), color = "red") + geom_point(aes(y =
 ↪  misc_err_train), color = "red") + ggtitle("Misc error as a function of k") +
 ↪  xlab("k") + ylab("Misc error")
```

It would seems like k=3 and k=4 have the lowest valid misclassification rate. Since k=3 have a lower training misclassification rate, we choose k=3 as the optimal k. The model complexity increases with k but it does

not seem to affect the training and validation errors after k=3.

```
#Estimation of the model for k=3

#Fitting the classifier
digits_kknn_test_3 <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_test, k = 3, kernel = "rectangular")
digits_kknn_train_3 <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_train, k = 3, kernel = "rectangular")
digits_kknn_valid_3 <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = 3, kernel = "rectangular")

#Predicting the values
fitted_values_test_3 <- predict(digits_kknn_test_3, data = digits_test)
fitted_values_train_3 <- predict(digits_kknn_train_3, data = digits_train)
fitted_values_valid_3 <- predict(digits_kknn_valid_3, data = digits_valid)
#Validation error
conf_mat_test_3 <- table(obs = digits_test$V65, pred = fitted_values_test_3)
conf_mat_train_3 <- table(obs = digits_train$V65, pred = fitted_values_train_3)
conf_mat_valid_3 <- table(obs = digits_valid$V65, pred = fitted_values_valid_3)
#Misclassification error
misc_err_test_3 <- get_mean_misc_err(conf_mat_test_3)
misc_err_train_3 <- get_mean_misc_err(conf_mat_train_3)
misc_err_valid_3 <- get_mean_misc_err(conf_mat_valid_3)

print(paste("Test error for k=3: ", misc_err_test_3))
```

```
## [1] "Test error for k=3:  0.0240334378265413"
```

```
print(paste("Train error for k=3: ", misc_err_train_3))
```

```
## [1] "Train error for k=3:  0.0115122972265829"
```

```
print(paste("Valid error for k=3: ", misc_err_valid_3))
```

```
## [1] "Valid error for k=3:  0.025130890052356"
```

Assignment 1.5

> Fit K-nearest neighbor classifiers to the training data for different values of K K = 1,2, … , 30, compute the error for the validation data as cross-entropy (when computing log of probabilities add a small constant within log, e.g. 1e-15, to avoid numerical problems) and plot the dependence of the validation error on the value of K K.

Cross Entropy :

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

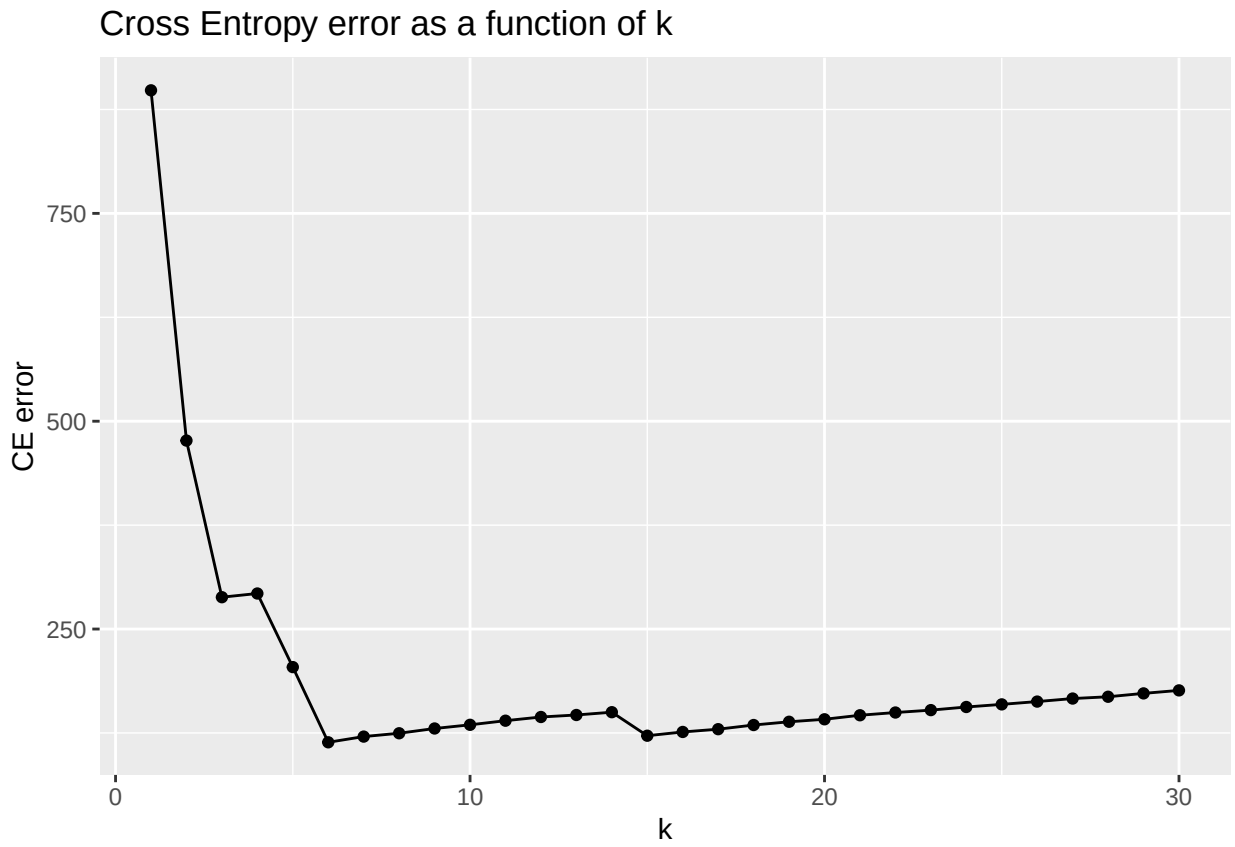
```
get_cross_entropy <- function(prob, true_vals){
  ce_sum <- 0
  for(i in 1:length(true_vals)){ #for each prediction
    truth <- true_vals[i] #Cm
    ce_sum <- ce_sum + log(prob[i,truth+1] + 1e-15)
  }
  return (-1*ce_sum)
}
k_values <- c()
x_ent_errs <- c()
for(i in 1:30){
  #Fitting the classifier
  cat(i, " ")
  #Valid:
  digits_kknn_ce <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = i, kernel = "rectangular")

  #Cross entropy Error:
  ce_err <- get_cross_entropy(digits_kknn_ce[["prob"]], digits_valid$V65)
  #Append to the vectors
  k_values <- append(k_values, i)
  x_ent_errs <- append(x_ent_errs, ce_err)
}
```

```
## 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28
```

```
df_CE <- data.frame("k" = k_values, "CE_err" = x_ent_errs)
```

Plot:

```
#Plotting misc_errs_test and misc_errs_train as a function of k
ggplot(df_CE, aes(x = k, y = CE_err)) + geom_line() + geom_point() + ggtitle("Cross
 ↪  Entropy error as a function of k") + xlab("k") + ylab("CE error")
```



What is the optimal K K value here? Assuming that response has multinomial distribution, why might the cross-entropy be a more suitable choice of the error function than the misclassification error for this problem?

Assignment 2

2.1

Divide it into training and test data (60/40) and scale it appropriately. In the coming steps, assume that motor_UPDRS is normally distributed and is a function of the voice characteristics, and since the data are scaled, no intercept is needed in the modelling.

```
# Read in data
df_park <- read.csv("parkinsons.csv")

# Split training/test and scale
set.seed(12345)
train_ind <- sample(1:nrow(df_park), floor(nrow(df_park) * 0.6))
df_train <- df_park[train_ind,]
df_test <- df_park[-train_ind,]
```

```r
scaler <- preProcess(df_train)
df_train_scaled <- predict(scaler, df_train)
df_test_scaled <- predict(scaler, df_test)
```

2.2

> Compute a linear regression model from the training data, estimate training and test MSE and comment on which variables contribute significantly to the model.

```
## MSE training data: 0.8785431
## MSE test data: 0.9354477
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ . + 0 - subject. - age - sex - test_time -
##     total_UPDRS, data = df_train_scaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0255 -0.7363 -0.1087  0.7333  2.1960
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## Jitter...       0.186931   0.149561   1.250 0.211431
## Jitter.Abs.    -0.169609   0.040805  -4.157 3.31e-05 ***
## Jitter.RAP     -5.269544  18.834160  -0.280 0.779658
## Jitter.PPQ5    -0.074568   0.087766  -0.850 0.395592
## Jitter.DDP      5.249558  18.837525   0.279 0.780510
## Shimmer         0.592436   0.205981   2.876 0.004050 **
## Shimmer.dB.    -0.172655   0.139316  -1.239 0.215315
## Shimmer.APQ3   32.070932  77.159242   0.416 0.677694
## Shimmer.APQ5   -0.387507   0.113789  -3.405 0.000668 ***
## Shimmer.APQ11   0.305546   0.061236   4.990 6.34e-07 ***
## Shimmer.DDA   -32.387241  77.158814  -0.420 0.674695
## NHR            -0.185387   0.045567  -4.068 4.84e-05 ***
## HNR            -0.238543   0.036395  -6.554 6.41e-11 ***
## RPDE            0.004068   0.022664   0.179 0.857556
## DFA            -0.280318   0.020136 -13.921  < 2e-16 ***
## PPE             0.226467   0.032881   6.887 6.70e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9394 on 3509 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.25 on 16 and 3509 DF,  p-value: < 2.2e-16
```

The total_UPDRS variables has by far the biggest impact on the model. If this variable is included in the model, the MSE on both the training and test data are very good at about 0.09. The MSE on the test data is slightly higher than it is on the training data, which suggests the model was not overfit to the training data. However, when this variable is removed from the model, the MSE rises from 0.09 to about 0.8-0.9. There is no description of the variable included in the assignment, but it may be very strongly, if not fully

correlated with motor_UPDRS, causing it to be the main predictor in the model. If this is the case, it is probably a bad idea to include this variable in the model, as it pretty much represents the same thing as the y variable.

Either way, though, there are quite a few variables that are not significant. When including total_UPDRS in the model, Jitter.Abs., Jitter.DDP, Shimmer.APQ3, Shimmer.DDA, NHR, and HNR all have p-values above 0.7, and test_time, Jitter.PPQ5, Shimmer.dB., and DFA are also not significant. Not much pre-exploring of the data was done in the assignment before putting together the model, but it would be good to check if there is any multicollinearity happening within the Jitter and Shimmer categories. Similarly, NHR and HNR are similar measures and may be capturing the same information. When total_UPDRS is removed, Jiter.RAP, Jitter.DDP, and RPDE have very high p-values, and Jitter..., Jitter.PPQ5, Shimmer.dB., Shimmer.APQ3, and Shimmer.DDA are also not significant. While not being significant, Shimmer.APQ3 has the largest coefficient value, meaning it has the biggest impact on the prediction.

2.3a

Loglikelihood function that for a given parameter vector θ and dispersion computes the log-likelihood function $logP(T|\theta, \sigma)$ for the stated model and the training data.

```
loglikelihood <- function(data, formula, theta, sigma) {
  # Get variable names and y and x matrices
  y_var <- all.vars(formula)[1]
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }

  y <- data[, y_var]
  x <- data[, x_var]

  # Get n observations
  n <- nrow(data)

  # Compute log likelihood
  log_likelihood <- -n * log(sqrt(2 * pi * sigma^2)) - 1 / (2 * sigma^2) * sum((y -
↪   t(theta) * x)^2)

  return(log_likelihood)
}
```

2.3b

Ridge function that for given vector θ, scalar σ and scalar λ uses function from 3a and adds up a Ridge penalty $\lambda||\theta||^2$ to the minus log-likelihood.

```
ridge <- function(v_theta_sigma, data, formula, lambda) {
  theta <- v_theta_sigma[1:length(v_theta_sigma) - 1]
  sigma <- v_theta_sigma[length(v_theta_sigma)]
```

```
  penalty <- lambda * sum(theta^2) #
 ↪  https://stackoverflow.com/questions/10933945/how-to-calculate-the-euclidean-norm-of-a-vector-in-r
  min_loglikelihood <- -loglikelihood(data, formula, theta, sigma)
  penalized_min_loglikelihood <- min_loglikelihood + penalty

  return(penalized_min_loglikelihood)
}
```

2.3c

RidgeOptfunction that depends on scalar λ, uses function from 3b and function optim() with method="BFGS" to find the optimal θ and σ for the given λ.

```
ridge_opt <- function(data, formula, lambda) {
  # #
   ↪  https://stackoverflow.com/questions/24623488/how-do-i-use-a-function-with-parameters-in-optim-in-
  #
   ↪  https://stackoverflow.com/questions/59517244/r-optim-can-i-pass-a-list-to-parameter-par
  y_var <- all.vars(formula)[1]
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }

  opt <- optim(c(rep(0, length(x_var)), 1), ridge, data=data, formula=formula,
 ↪  lambda=lambda, method="BFGS")
  opt_theta <- opt$par[1:length(opt$par) - 1]
  opt_sigma <- opt$par[length(opt$par)]

  return(list(theta=opt_theta, sigma=opt_sigma))
}
```

2.3d

Df function that for a given scalar λ computes the degrees of freedom of the Ridge model based on the training data.

```
df <- function(data, formula, lambda) {
  # Get x matrix
  y_var <- all.vars(formula)[1]
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }
```

```
  x <- as.matrix(data[, x_var])

  # Compute trace of hat matrix
  sum(diag((x %*% solve(t(x) %*% x + lambda * diag(ncol(x))) %*% t(x)))) #
    ↪  https://online.stat.psu.edu/stat508/lesson/5/5.1
}
```

2.4

By using function RidgeOpt, compute optimal θ parameters for $\lambda = 1$, $\lambda = 100$ and $\lambda = 1000$. Use the estimated parameters to predict the motor_UPDRS values for training and test data and report the training and test MSE values. Which penalty parameter is most appropriate among the selected ones? Compute and compare the degrees of freedom of these models and make appropriate conclusions.

```
## lambda = 1
## MSE train: 1.418124
## MSE test: 1.416343
## df train: 13.86074
## df test: 13.77688
##
## lambda = 100
## MSE train: 1.082856
## MSE test: 1.09509
## df train: 9.924887
## df test: 9.173182
##
## lambda = 1000
## MSE train: 0.9914506
## MSE test: 1.007587
## df train: 5.643925
## df test: 4.822328
```

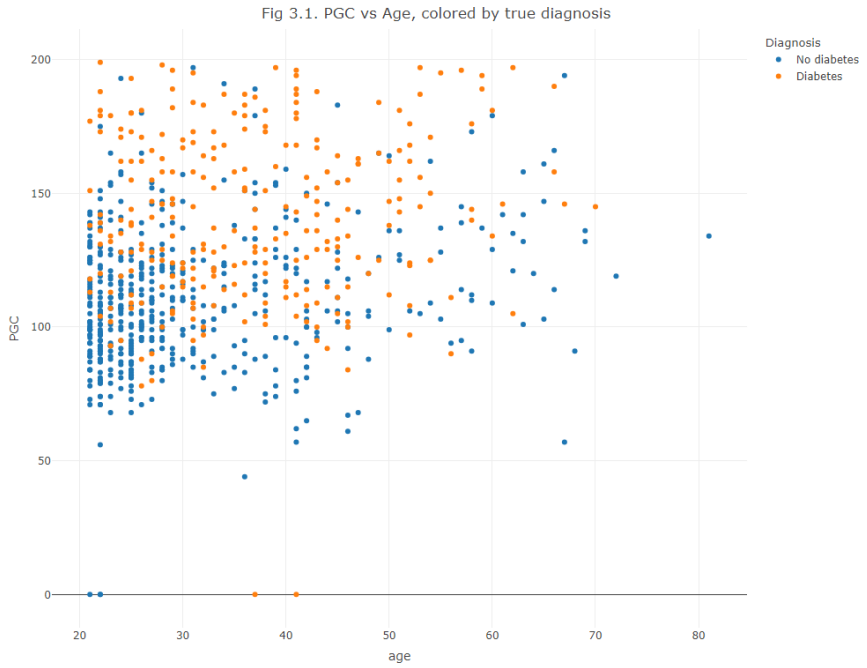
Interestingly, the test MSE is a little bit better than the train MSE when lambda = 1. This could be due to a fortunate choice in train/test split, but it at least suggests this model is not overfitting to the data. However, this is not a reason to pick this model over the others, as both other models only show a small difference between the train and test MSE. This could be a signifier that the model could be more complex, but, again, it does show that neither model is overfitted to the data. The difference in MSE between the lambda = 100 and lambda = 1000 models is relatively small - only about 0.1 - compared to the difference between the lambda = 1 and lambda = 100 models - about 0.32. Between the lambda = 100 and lambda = 1000 models, the degrees of freedom go down a lot from around 9/10 to 5/5.5. These models perform almost equally in terms of MSE, but show a big difference in degrees of freedom. The most appropriate choice would probably be the model with lambda = 100, as its performance is good compared to the others, but the higher degrees of freedom give more statistical power.

Reference: https://sites.utexas.edu/sos/degreesfreedom/

Assignment 3

We are provided data covering the onset of diabetes within a five year period for a group of individuals. The data consists of nine variables including a binary response variable indicating diagnosis: presence of diabetes

or not. In Figure 3.1, we plot Plasma Glucose Concentration (PGC) against age in years. Datapoints are colored by diagnosis.



Analysing Figure 3.1 we observe a large cluster of young people (ages ~20-30) that do not have diabetes, along with a significant number of outliers (among the non-diabetes people.) The people with diabetes however are much more spread out, with no clear clusters. It appears as though people with diabetes tend to have slightly larger Plasma Glucose Concentration (PGC) values than people without diabetes. Thus it does appear as though there is some explanatory power in the PGC values and Age, but it is likely not enough to achieve a highly accurate logistic regression (classification) predictor.

Assignment 3.2 and 3.3

We will now fit a logistic regression (classification) model using the PGC and age features to predict the presence of diabetes. See *Machine Learning - A first course for engineers and scientists* (pp. 45-52) for a discussion on logistic regression. We will initially use a classification threshold of $r = 0.5$. Mathematically, our model predictor $g(x)$ can be represented in the following way, where $\hat{y}(x) = 1$ indicates presence of diabetes, and $\hat{y}(x = 0)$ indicates absence.

$$\hat{y}(x) = \begin{Bmatrix} 1 & \text{if} & g(x) \geq 0.5 \\ 0 & \text{if} & g(x) < 0.5 \end{Bmatrix}$$

where

$$g(x) = \frac{1}{1 + e^{-z}}$$

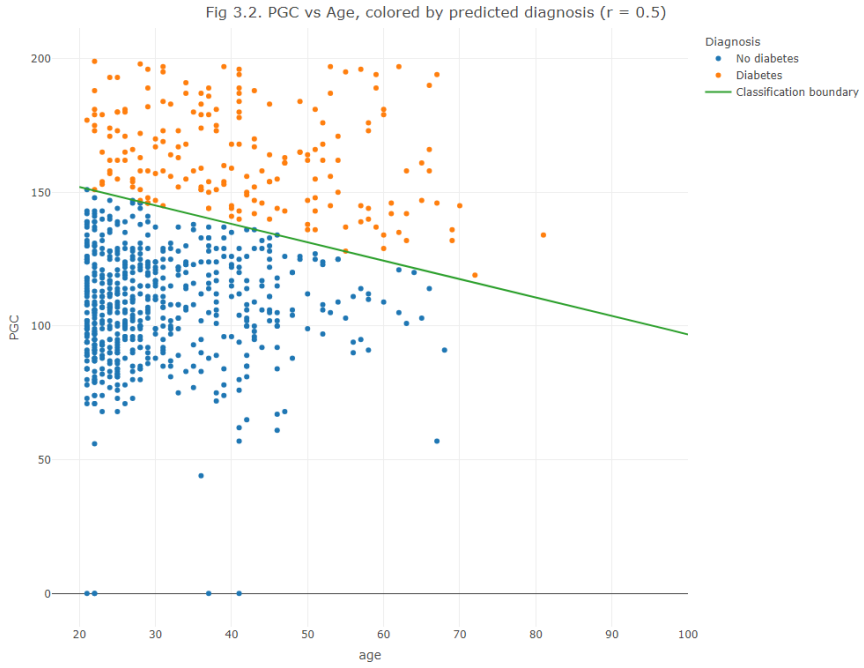
where

$$z = \theta^T x = (\theta_0, \theta_1, \theta_2) \cdot (1, x_{\text{pgc}}, x_{\text{age}})^T.$$

The decision boundary for this model will be the set of points that satisfy the following equation:

$$g(x) = \frac{e^{\theta^T x}}{1 + e^{\theta^T x}} = \frac{1}{2} \implies \theta^T x = 0 \implies \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 = 0 \implies x_2 = \frac{-\theta_0}{\theta_2} - \frac{\theta_1}{\theta_2} x_1.$$

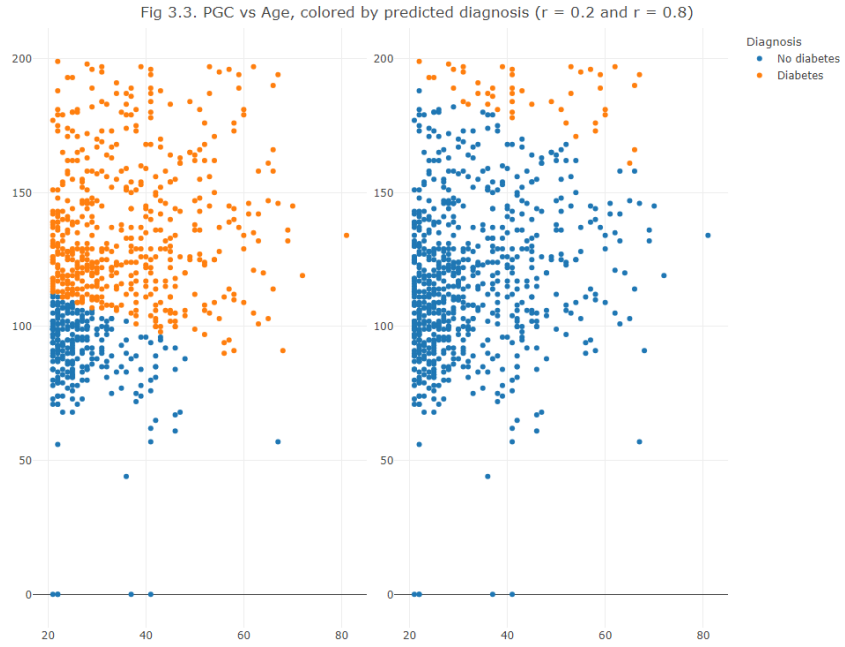
In Figure 3.2 below, we plot again PGC values against Age, but datapoints are colored by predicted diagnosis (using $r = 0.5$). The discussed linear decision boundary of the model is also included.



When comparing Figure 3.1 and 3.2 we see that the quality of the classification is decent at best. The resulting predictions are visually precisely what could be expected when analysing Figure 3.1. The model essentially implements what was discussed previously: diabetes is common among people with high PGC values. The missclassification error 0.266 reflects our model assessment, as it is good, but far from excellent. The decision boundary echoes this, as it clearly divides the data into high and low PGC values, but it also increases the "diabetes region" as Age increases.

Assignment 3.4

In Figure 3.3 we plot again PGC against Age values, but this time color points based on classification thresholds $r = 0.2$ and $r = 0.8$. To clarify, this is *almost* the same model as discussed previously, except that the final classification threshold is changed in the definition of $\hat{y}(x)$.



When analysing Figure 3.3, we find that the linear classification threshold simply has been moved down for $r = 0.2$, and up for $r = 0.8$. Specifically, it has been shifted by a constant $\frac{\ln(1/r-1)}{-\theta_2}$. This has the effect that many more, or many fewer datapoints respectively are classified as positive for diabetes.

Assignment 3.5

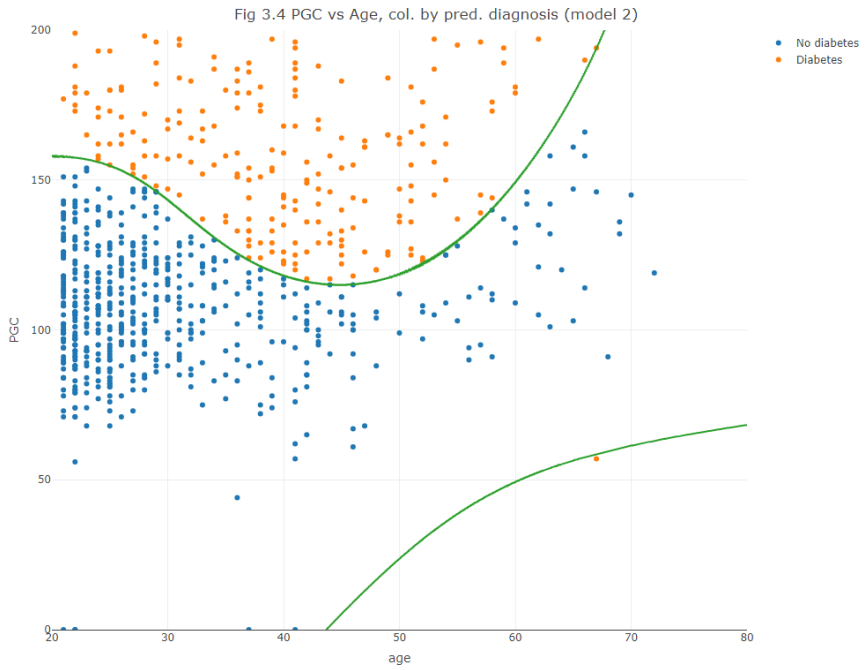
We now perform a basis function expansion by computing new features and including them in a new logistic regression model. For this new model, a classification threshold of $r = 0.5$ will be used. The new features are

$$\left\{ \begin{array}{c} z_1 = x_1^4 \\ z_2 = x_1^3 x_2 \\ z_3 = x_1^2 x_2^2 \\ z_4 = x_1 x_2^3 \\ z_5 = x_2^4 \end{array} \right\}.$$

These new features complicate the classification boundary, but the setup of the problem of finding the classification boundary is still the same. For our purposes it is solved numerically. The classification boundary consists of all points that satisfy the following equation.

$$\theta_0 + \theta_1 x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_1^4 + \theta_4 \cdot x_1^3 x_2 + \theta_5 \cdot x_1^2 x_2^2 + \theta_6 \cdot x_1 x_2^3 + \theta_7 \cdot x_2^4 = 0$$

In Figure 3.4 we plot again the PGC against Age, and color datapoints based on predicted diagnosis by the new model. The classification boundary is included.



In Figure 3.4 we see a similar behaviour to the first model(s), but now the inclusion of the new features have allowed the model to take into account interplay between the PGC and Age features. Visually it looks promising, as the new decision boundary captures the large group of young people without diabetes, as well as the fact that diabetes is very common among individuals with high PGC values. It also captures the fact that for Ages from about 30 onwards, the presence of diabetes increases. The missclassification rate of 0.246 is only slightly lower than that of the first model (with $r = 0.5$). This indicates that potential benefits of including other features in the model should be investigated.

On the other hand the boundary highlights a region with high Age (~50-80) and low PGC values (< 60) as indicative of diabetes. In fact the only datapoint in this region is incorrectly diagnosed, which we attribute as a side effect of the new transformed features, and thus we recommend ignoring the lower curve of the decision boundary.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(caret)
library(kknn)
library(ggplot2)

# Read data
digit_data <- read.csv("optdigits.csv", header=FALSE)

# Partition data (according to Oleg)
n = dim(digit_data)[1]

set.seed(12345)
id = sample(1:n, floor(n*0.5))
digits_train = digit_data[id,]
id1 = setdiff(1:n, id)
```

```r
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
digits_valid = digit_data[id2,]
id3 = setdiff(id1,id2)
digits_test = digit_data[id3,]
#Fitting the classifier
digits_kknn_test <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = 30, kernel = "rectangular")
kknn_fit_test <- fitted(digits_kknn_test)
conf_mat_test <- table(obs = digits_valid$V65, pred = kknn_fit_test)
conf_mat_test

digits_kknn_train <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_train, k = 30, kernel = "rectangular")
kknn_fit_train <- fitted(digits_kknn_train)
conf_mat_train <- table(obs = digits_train$V65, pred = kknn_fit_train)
conf_mat_train

#Misclassification error
get_mean_misc_err <- function(conf_mat){
  #Given a confusion matrix, return the misclassification error
  (sum(conf_mat)-sum(diag(conf_mat)))/sum(conf_mat)
}
get_indiv_misc_err <- function(conf_mat){
  #Given a confusion matrix, return the misclassification error for each digit
  err = c()
  for(i in 1:10){
    #correct this
    err[i] <- (sum(conf_mat[i,]-conf_mat[i,i])/sum(conf_mat[i,]))
  }
  return(Misc_error = err)
}
print("Misclassification error for test data :")
get_mean_misc_err(conf_mat_test)
get_indiv_misc_err(conf_mat_test)
print("Misclassification error for train data :")
get_mean_misc_err(conf_mat_train)
get_indiv_misc_err(conf_mat_train)
#Find all fitted values classified as 8 (CL = 8), find the max and min probabilities in
 ↪  prob

#Fitted values
fitted_eight <- predict(digits_kknn_train, data = digits_train)
#True values
actual_eight <- digits_train$V65
#Probabilities
probs <- digits_kknn_train[["prob"]]
#Probabilities for 8
probs_eight <- probs[,9]

my_df <- data.frame("actual" = actual_eight, "fitted" = fitted_eight , "prob" =
 ↪  probs_eight)
```

```r
#filter my_df to only keep the 8s correctly fitted
my_df <- my_df[my_df$actual == 8,]
my_df <- my_df[my_df$fitted == 8,]

#Find the max and min probabilities
maxp <- 0
minp <- 1
for(i in 1:length(probs_eight)){
  if(factor(digits_kknn_train$fitted.values[i]) == 8){
    #https://www.tutorialspoint.com/how-to-extract-the-factor-levels-from-factor-column-in-an-r-data-
    maxp <- max(probs_eight[i], maxp)
    minp <- min(probs_eight[i], minp)
  }
}
# Find all the index where p=maxp and p= minp :
high_idx = c()
low_idx = c()
for(i in 1:length(probs_eight)){
  if(probs_eight[i] == maxp){
    high_idx <- append(high_idx, i)
  }
  if(probs_eight[i] == minp){
    low_idx <- append(low_idx, i)
  }
}
high_idx
# 129  195  211  233  292  294  515  601  650  679  684  693  726  729  752  763  768
 ↪  779 855  864  899  929 1006 1092 1134 1216 1227 1261 1295 1318 1355 1380 1387 1397
 ↪  1419 1472 1533 1607 1646 1686
low_idx
# 141  258  469  560  629  881 1274 1716
visualise_dig <- function(idx, data){
  #Given an index (idx) and a dataframe (data), visualize the digit at data[idx]
  raw_dig <- data[idx,][-65]
  mat = matrix(as.numeric(raw_dig), nrow = 8)
  heatmap(apply(t(mat),2,rev), Colv=NA, Rowv=NA, col=paste("gray",99:1,sep=""))
}
#High probabilities
for(i in 1:2){
  visualise_dig(high_idx[i], digits_train)
}
#Low probabilities
for(i in 1:3){
  visualise_dig(low_idx[i], digits_train)
}
k_values <- c()
misc_errs_test <- c()
misc_errs_train <- c()

for(i in c(1:30)){
  #Fitting the classifier
  #Test:
  digits_kknn_test_i <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = i, kernel = "rectangular")
```

```
  #Train:
  digits_knn_train_i <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_train, k = i, kernel = "rectangular")

  #kbetter <- train.kknn(formula = as.factor(V65) ~ . , data = digits_train, ks=i, kernel
  ↪  = "rectangular")
  #Predicting the values
  fitted_values_valid <- predict(digits_knn_test_i, data = digits_valid)
  fitted_values_train <- predict(digits_knn_train_i, data = digits_train)
  #Confusion matrix
  conf_mat_test <- table(obs = digits_valid$V65, pred = fitted_values_valid)
  conf_mat_train <- table(obs = digits_train$V65, pred = fitted_values_train)
  #Misclassification error
  misc_err_test <- get_mean_misc_err(conf_mat_test)
  misc_err_train <- get_mean_misc_err(conf_mat_train)

  #Append to the vectors
  k_values <- append(k_values, i)
  misc_errs_test <- append(misc_errs_test, misc_err_test)
  misc_errs_train <- append(misc_errs_train, misc_err_train)
}
df_ks <- data.frame("k" = k_values, "misc_err_test" = misc_errs_test, "misc_err_train" =
 ↪  misc_errs_train)
#Plotting misc_errs_test and misc_errs_train as a function of k
ggplot(df_ks, aes(x = k, y = misc_err_test)) + geom_line() + geom_point() +
 ↪  geom_line(aes(y = misc_err_train), color = "red") + geom_point(aes(y =
 ↪  misc_err_train), color = "red") + ggtitle("Misc error as a function of k") +
 ↪  xlab("k") + ylab("Misc error")
#Estimation of the model for k=3

#Fitting the classifier
digits_knn_test_3 <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_test, k = 3, kernel = "rectangular")
digits_knn_train_3 <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_train, k = 3, kernel = "rectangular")
digits_knn_valid_3 <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = 3, kernel = "rectangular")

#Predicting the values
fitted_values_test_3 <- predict(digits_knn_test_3, data = digits_test)
fitted_values_train_3 <- predict(digits_knn_train_3, data = digits_train)
fitted_values_valid_3 <- predict(digits_knn_valid_3, data = digits_valid)
#Validation error
conf_mat_test_3 <- table(obs = digits_test$V65, pred = fitted_values_test_3)
conf_mat_train_3 <- table(obs = digits_train$V65, pred = fitted_values_train_3)
conf_mat_valid_3 <- table(obs = digits_valid$V65, pred = fitted_values_valid_3)
#Misclassification error
misc_err_test_3 <- get_mean_misc_err(conf_mat_test_3)
misc_err_train_3 <- get_mean_misc_err(conf_mat_train_3)
misc_err_valid_3 <- get_mean_misc_err(conf_mat_valid_3)

print(paste("Test error for k=3: ", misc_err_test_3))
print(paste("Train error for k=3: ", misc_err_train_3))
```

```r
print(paste("Valid error for k=3: ", misc_err_valid_3))

get_cross_entropy <- function(prob, true_vals){
  ce_sum <- 0
  for(i in 1:length(true_vals)){ #for each prediction
    truth <- true_vals[i] #Cm
    ce_sum <- ce_sum + log(prob[i,truth+1] + 1e-15)
  }
  return (-1*ce_sum)
}
k_values <- c()
x_ent_errs <- c()
for(i in 1:30){
  #Fitting the classifier
  cat(i, " ")
  #Valid:
  digits_kknn_ce <- kknn(formula = as.factor(V65) ~ . , train = digits_train, test =
 ↪  digits_valid, k = i, kernel = "rectangular")

  #Cross entropy Error:
  ce_err <- get_cross_entropy(digits_kknn_ce[["prob"]], digits_valid$V65)
  #Append to the vectors
  k_values <- append(k_values, i)
  x_ent_errs <- append(x_ent_errs, ce_err)
}
df_CE <- data.frame("k" = k_values, "CE_err" = x_ent_errs)
#Plotting misc_errs_test and misc_errs_train as a function of k
ggplot(df_CE, aes(x = k, y = CE_err)) + geom_line() + geom_point() + ggtitle("Cross
 ↪  Entropy error as a function of k") + xlab("k") + ylab("CE error")
# Read in data
df_park <- read.csv("parkinsons.csv")

# Split training/test and scale
set.seed(12345)
train_ind <- sample(1:nrow(df_park), floor(nrow(df_park) * 0.6))
df_train <- df_park[train_ind,]
df_test <- df_park[-train_ind,]

scaler <- preProcess(df_train)
df_train_scaled <- predict(scaler, df_train)
df_test_scaled <- predict(scaler, df_test)

# Train model
mod <- lm(motor_UPDRS ~ . + 0 - subject. - age - sex - test_time - total_UPDRS,
 ↪  data=df_train_scaled) #
 ↪  https://stats.stackexchange.com/questions/143155/doing-multiple-regression-without-intercept-in-r-w

# Predict values
pred_train <- data.frame(pred=predict(mod, df_train_scaled),
 ↪  act=df_train_scaled$motor_UPDRS)
pred_test <- data.frame(pred=predict(mod, df_test_scaled),
 ↪  act=df_test_scaled$motor_UPDRS)
```

```r
# Compute MSE
# https://www.statology.org/how-to-calculate-mse-in-r/
mse_train <- mean((pred_train$act - pred_train$pred)^2)
mse_test <- mean((pred_test$act - pred_test$pred)^2)

cat("MSE training data: ", mse_train,
    "\nMSE test data: ", mse_test,
    "\n\n",
    sep="")
summary(mod)

loglikelihood <- function(data, formula, theta, sigma) {
  # Get variable names and y and x matrices
  y_var <- all.vars(formula)[1]
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }

  y <- data[, y_var]
  x <- data[, x_var]

  # Get n observations
  n <- nrow(data)

  # Compute log likelihood
  log_likelihood <- -n * log(sqrt(2 * pi * sigma^2)) - 1 / (2 * sigma^2) * sum((y -
 ↪  t(theta) * x)^2)

  return(log_likelihood)
}

ridge <- function(v_theta_sigma, data, formula, lambda) {
  theta <- v_theta_sigma[1:length(v_theta_sigma) - 1]
  sigma <- v_theta_sigma[length(v_theta_sigma)]

  penalty <- lambda * sum(theta^2) #
 ↪  https://stackoverflow.com/questions/10933945/how-to-calculate-the-euclidean-norm-of-a-vector-in-r
  min_loglikelihood <- -loglikelihood(data, formula, theta, sigma)
  penalized_min_loglikelihood <- min_loglikelihood + penalty

  return(penalized_min_loglikelihood)
}

ridge_opt <- function(data, formula, lambda) {
  # #
   ↪  https://stackoverflow.com/questions/24623488/how-do-i-use-a-function-with-parameters-in-optim-in-
  #
   ↪  https://stackoverflow.com/questions/59517244/r-optim-can-i-pass-a-list-to-parameter-par
  y_var <- all.vars(formula)[1]
```

```
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }

  opt <- optim(c(rep(0, length(x_var)), 1), ridge, data=data, formula=formula,
 ↪  lambda=lambda, method="BFGS")
  opt_theta <- opt$par[1:length(opt$par) - 1]
  opt_sigma <- opt$par[length(opt$par)]

  return(list(theta=opt_theta, sigma=opt_sigma))
}

df <- function(data, formula, lambda) {
  # Get x matrix
  y_var <- all.vars(formula)[1]
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }

  x <- as.matrix(data[, x_var])

  # Compute trace of hat matrix
  sum(diag((x %*% solve(t(x) %*% x + lambda * diag(ncol(x))) %*% t(x)))) #
   ↪  https://online.stat.psu.edu/stat508/lesson/5/5.1
}

formula <- motor_UPDRS ~ . - subject. - age - sex - test_time - total_UPDRS

# Get opt theta and sigma for different lambdas
l_param1 <- ridge_opt(df_train_scaled, formula, lambda=1)
l_param100 <- ridge_opt(df_train_scaled, formula, lambda=100)
l_param1000 <- ridge_opt(df_train_scaled, formula, lambda=1000)

# Function for computing MSE
mse <- function(data, formula, theta) {
  # Get variable names and y and x matrices
  y_var <- all.vars(formula)[1]
  x_var <- all.vars(formula)[2:length(all.vars(formula))]

  if ("." %in% x_var) {
    x_var_sub <- x_var[x_var != "."]
    x_var <- colnames(data)[colnames(data) != y_var]
    x_var <- x_var[!(x_var %in% x_var_sub)]
  }
```

```
  y <- data[, y_var]
  x <- as.matrix(data[, x_var])

  y_hat <- x %*% theta

  # Compute MSE
  mean((y_hat - y)^2)
}

# Get y hat on training and test for different lambdas
mse_tr_1 <- mse(df_train_scaled, formula, l_param1$theta)
mse_te_1 <- mse(df_test_scaled, formula, l_param1$theta)

mse_tr_100 <- mse(df_train_scaled, formula, l_param100$theta)
mse_te_100 <- mse(df_test_scaled, formula, l_param100$theta)

mse_tr_1000 <- mse(df_train_scaled, formula, l_param1000$theta)
mse_te_1000 <- mse(df_test_scaled, formula, l_param1000$theta)

# Compute df for different models
df_tr_1 <- df(df_train_scaled, formula, 1)
df_te_1 <- df(df_test_scaled, formula, 1)

df_tr_100 <- df(df_train_scaled, formula, 100)
df_te_100 <- df(df_test_scaled, formula, 100)

df_tr_1000 <- df(df_train_scaled, formula, 1000)
df_te_1000 <- df(df_test_scaled, formula, 1000)

# Report output
cat("lambda = 1",
    "\nMSE train: ", mse_tr_1,
    "\nMSE test: ", mse_te_1,
    "\ndf train: ", df_tr_1,
    "\ndf test: ", df_te_1,
    "\n\n",
    "lambda = 100",
    "\nMSE train: ", mse_tr_100,
    "\nMSE test: ", mse_te_100,
    "\ndf train: ", df_tr_100,
    "\ndf test: ", df_te_100,
    "\n\n",
    "lambda = 1000",
    "\nMSE train: ", mse_tr_1000,
    "\nMSE test: ", mse_te_1000,
    "\ndf train: ", df_tr_1000,
    "\ndf test: ", df_te_1000,
    sep=""
)

# Load packages
library(plotly)
library(magrittr) # pipe operator
```

```r
library(webshot2) # htmlwidgets

# Read data
diab_data <- read.csv("pima-indians-diabetes.csv")
colnames(diab_data) <- c("n_pregnant", "PGC", "blood_pressure", "skindfold_thickness",
 ↪  "serum_insulin", "bmi", "diab_pedigree", "age", "diabetes")

# Define the logit function
logit <- function(z){
  exp(z) / (1 + exp(z))
}
p_31 <- plot_ly(type="scatter", mode="markers",
        data = diab_data,
        x = ~age,
        y = ~PGC,
        colors = c("#1f77b4", "#ff7f0e"),
        color = ~factor(x = diab_data$diabetes, labels = c("No diabetes", "Diabetes")),
        legendgrouptitle = list(text = "Diagnosis")) %>%
  layout(title = "Fig 3.1. PGC vs Age, colored by true diagnosis")

# Create webshot for the pdf report
htmlwidgets::saveWidget(widget=p_31, file="p_31.html")
webshot(url="p_31.html", file="p_31.png")
# Train logistic regression model
model1 <- glm(formula = diabetes ~PGC + age,
              family = binomial(link = "logit"),
              data = diab_data)

#summary(model1)

# Calculate predictions for classification thresholds
# r = 0.2, 0.5, 0.8
diab_data <- diab_data %>%
  mutate(predicted_02 = as.integer(fitted(model1) >= 0.2)) %>%
  mutate(predicted_05 = as.integer(fitted(model1) >= 0.5)) %>%
  mutate(predicted_08 = as.integer(fitted(model1) >= 0.8))

# Extract misclassification error
mis_class_error <- 1- (diab_data$diabetes == diab_data$predicted_05) %>% sum() /
 ↪  nrow(diab_data)

#cat("Misclassification error:", mis_class_error, "\n")
r_05_curve <- function(x){
  -model1$coefficients["(Intercept)"]/model1$coefficients["PGC"] -
   ↪  (model1$coefficients["age"]*x)/model1$coefficients["PGC"]
}

# Create scatterplot of predicted diabetes diagnosis
p_32 <- plot_ly(type="scatter", mode="markers",
        data = diab_data,
        x = ~age,
        y = ~PGC,
        colors = c("#1f77b4", "#ff7f0e"),
```

```
        color = ~factor(x = diab_data$predicted_05, labels = c("No diabetes",
         ↪  "Diabetes")),
        legendgrouptitle = list(text = "Diagnosis")) %>%
  layout(title = "Fig 3.2. PGC vs Age, colored by predicted diagnosis (r = 0.5)") %>%

  # Add curve
  add_trace(inherit = F, type="scatter", mode="lines",
            x = c(20,100),
            y = c(r_05_curve(20), r_05_curve(100)),
            name = "Classification boundary")


# Create webshot
htmlwidgets::saveWidget(widget=p_32, file="p_32.html")
webshot(url="p_32.html", file="p_32.png")
# Plotly named colours
# https://community.plotly.com/t/plotly-colours-list/11730/3


# r = 0.2
p_r02 <- plot_ly(type="scatter", mode="markers",
        data = diab_data,
        x = ~age,
        y = ~PGC,
        colors = c("#1f77b4", "#ff7f0e"),
        color = ~factor(x = diab_data$predicted_02, labels = c("No diabetes",
         ↪  "Diabetes")),
        legendgrouptitle = list(text = "Diagnosis")) %>%
  layout(title = "Fig 3.? PGC vs Age, colored by predicted diagnosis (r=0.2)")
# add curve


# r = 0.8
p_r08 <- plot_ly(type="scatter", mode="markers",
        data = diab_data,
        x = ~age,
        y = ~PGC,
        colors = c("#1f77b4", "#ff7f0e"),
        color = ~factor(x = diab_data$predicted_08, labels = c("No diabetes",
         ↪  "Diabetes")),
        showlegend = FALSE) %>%
  layout(title = "Fig 3.? PGC vs Age, colored by predicted diagnosis (r=0.8)")
# add curve


a<-subplot(p_r02, p_r08) %>%
  layout(title = "Fig 3.3. PGC vs Age, colored by predicted diagnosis (r = 0.2 and r =
   ↪  0.8)")

htmlwidgets::saveWidget(widget=a, file="a.html")
webshot(url="a.html", file="a.png")
# Create new variables z_1, ... z_5
diab_data <- diab_data %>%
```

```
  mutate(z1 = PGC^4 * age^0,
         z2 = PGC^3 * age^1,
         z3 = PGC^2 * age^2,
         z4 = PGC^1 * age^3,
         z5 = PGC^0 * age^4)

# New model
model2 <- glm(formula = diabetes ~ PGC + age + z1 + z2 + z3 + z4 + z5,
    family = binomial(link = "logit"),
    data = diab_data)

#summary(model2)

diab_data_newmodel <- diab_data %>%
  mutate(predicted_z_05 = as.integer(fitted(model2) >= 0.5))

# Extract misclassification error
mis_class_error_model2 <- 1- (diab_data_newmodel$diabetes ==
 ↪  diab_data_newmodel$predicted_z_05) %>% sum() / nrow(diab_data_newmodel)
# Find classification boundary points numerically for model 2
class_bound <- data.frame(age=numeric(),
                          pgc = numeric(),
                          group=numeric())

age_values <- seq(20, 80, 0.1)
pgc_values <- seq(0, 200, 0.1)
for (age in age_values){
  for (pgc in pgc_values){
    # evaluate theta^T %*% (1, x_1, ..., x_2^4)
    # It will be zero at the class boundary
    value <- sum(model2$coefficients*c(1,
                                       pgc,
                                       age,
                                       pgc^4 * age^0,
                                       pgc^3 * age^1,
                                       pgc^2 * age^2,
                                       pgc^1 * age^3,
                                       pgc^0 * age^4))
    if (abs(value) < 0.01){

      # Split the boundary in two separate groups
      # This was determined visually
      if (pgc < 100){group <- 1}
      else {group <- 2}

      # Save this point as a boundary point
      class_bound[nrow(class_bound)+1,] <- c(age, pgc, group)
    }
  }
}
p_model2 <- plot_ly(type="scatter", mode="markers",
                    data = diab_data_newmodel,
                    x = ~age,
```

```
                    y = ~PGC,
                    colors = c("#1f77b4", "#ff7f0e"),
                    color = ~factor(x = diab_data_newmodel$predicted_z_05,
                                    labels = c("No diabetes", "Diabetes"))) %>%

  # Add classifaction boundary trace
  add_trace(inherit = F, type="scatter", mode="lines",
            data = class_bound,
            x = ~age,
            y = ~pgc,
            line = list(color='#2ca02c'),
            split = ~group,
            showlegend=F) %>%

  # Configure layout
  layout(title = "Fig 3.4 PGC vs Age, col. by pred. diagnosis (model 2)",
         xaxis = list(range = c(20, 80)),
         yaxis = list(range = c(0, 200)))

#p_model2

# Create webshots for the pdf report
htmlwidgets::saveWidget(widget=p_model2, file="p_model2.html")
webshot(url="p_model2.html", file="p_model2.png")
```