Laboration report in Machine Learning

Computer lab 1 block 1

732A99

Simge Cinar
Duc Tran
William Wiik

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

16 november 2023

Contents

1 Assignment 1. Handwritten digit recognition with K-nearest neighbors.

The data in this task is from the file optdigits.csv. Data consists of 3822 handwritten digits from 0 to 9 and are stored as images of size 8x8.

1.1 Question 1.1

Question: 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.

Answer: The code used is presented as follows:

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

# Renaming the response variable and changing it to a factor variable
data <- rename(data, y=X0.26)
data$y <- as.factor(data$y)

# Partitioning training data (50%)
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]

# Partitioning validation data (25%)
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=data[id2,]

# Partitioning test data (25%)
id3=setdiff(id1,id2)
test=data[id3,]
```

1.2 Question 1.2

Question: 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

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

Answer: The confusion matrix for the model trained on training data with k=30 and evaluated on training data is presented in table 1.

```
# kknn on training data and evaluation on training data
model_kknn_train <- kknn(formula=y~.,
                         train=train,
```

```
                         test=train,
                         kernel="rectangular",
                         k=30)
conf_mat_train <- table(train$y, model_kknn_train$fitted.values)
acc_train <- sum(diag(conf_mat_train)) / sum(conf_mat_train)
miss_train <- 1-acc_train

# kknn on training data and evaluation on test data
model_kknn_test <- kknn(formula=y~.,
                         train=train,
                         test=test,
                         kernel="rectangular",
                         k=30)
conf_mat_test <- table(test$y, model_kknn_test$fitted.values)
acc_test <- sum(diag(conf_mat_test)) / sum(conf_mat_test)
miss_test <- 1-acc_test

# Rows are true values, columns are model prediction
kable(conf_mat_train, caption = "Confusion matrix for training data, model
      predictions by columns and true value by rows.")
```

Table 1: Confusion matrix for training data, model predictions by columns and true value by rows.

	0	1	2	3	4	5	6	7	8	9
0	177	0	0	0	1	0	0	0	0	0
1	0	174	9	0	0	0	1	0	1	3
2	0	0	170	0	0	0	0	1	2	0
3	0	0	0	197	0	2	0	1	0	0
4	0	1	0	0	166	0	2	6	2	2
5	0	0	0	0	0	183	1	2	0	11
6	0	0	0	0	0	0	200	0	0	0
7	0	1	0	1	0	1	0	192	0	0
8	0	10	0	1	0	0	2	0	190	2
9	0	3	0	4	2	0	0	2	4	181

```
miss_train
```

```
## [1] 0.04238619
```

The misclassification error on training data from table 1 is around 4.24%. The two number with the highest number of wrong predictions are 8 and 9, with 15 wrong predictions each. The model also struggles to predict number 1 and 5 where both had 14 wrong predictions. Examining the most common misclassification, the number 5 was predicted as 9 a total of eleven times and the number 8 was predicted as 1 a total of ten times.

The easiest numbers to predict are 0 and 6. For 0 the model correctly predicted 177 out of 178 numbers and did not predict the number 0 for any other number. For the number 6, the model correctly predicted all 200 numbers, but it did however predict six other numbers as 6 incorrectly.

The confusion matrix for the model trained on training data with k=30 and evaluated on test data is presented

in table 2.

```
kable(conf_mat_test, caption = "Confusion matrix for test data, model
      predictions by columns and true value by rows.")
```

Table 2: Confusion matrix for test data, model predictions by columns and true value by rows.

	0	1	2	3	4	5	6	7	8	9
0	97	0	0	0	0	0	1	0	0	0
1	0	91	3	0	0	0	0	0	0	3
2	0	0	93	1	0	0	0	0	1	0
3	0	0	0	95	0	0	0	2	1	0
4	1	0	0	0	89	0	1	5	1	3
5	0	1	0	1	0	79	1	0	0	5
6	0	0	0	0	0	0	94	0	0	0
7	0	2	0	0	0	1	0	91	1	0
8	0	3	0	1	0	0	1	0	86	0
9	0	0	0	4	0	0	0	2	1	94

```
miss_test
```

```
## [1] 0.04916318
```

The misclassification error on test data from table 2 is around 4.92%. The two number with the highest number of wrong predictions are 4 and 5. The number 4 had eleven wrong predictions and the number 5 had eight wrong predictions.

The easiest numbers to predict are 0 and 6. For 0 the model correctly predicted 97 out of 98 number and the misclassification was as number 6.
For the number 6, the model correctly predicted all 94 numbers, but it did however predict four times for other numbers as 6 incorrectly.

The overall prediction quality from the model is good, especially for the numbers 0 and 6.

1.3 Question 1.3

Question: 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). 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.

Answer: The code used to find the 2 digits that are hardest to classify were found with the code as follows

```
y <- train$y
fit_y <- model_kknn_train$fitted.values
# probabilities given from number 0 to 9, index 9 = number 8.
prob_8 <- model_kknn_train$prob[, 9]

# Data frame consisting of true value of y, model prediction and the models
# probability that the number is 8.
```

```
data_8 <- data.frame(y = y, fit_y = fit_y, prob = prob_8)
data_8$observation_id <- rownames(data_8)

# Only observations with the label 8 is kept.
data_8 <- data_8[data_8$y == "8", ]
head(arrange(data_8, prob), 2)
```

```
##   y fit_y      prob observation_id
## 1 8     6 0.1000000           1624
## 2 8     1 0.1666667           1663
```

From the output, observation 1624 and 1663 were hardest to classify as 8 from the model. The three observations that were easiest to identify as 8 were found with the code as follows

```
tail(arrange(data_8, prob), 3)
```

```
##     y fit_y prob observation_id
## 203 8     8    1           1810
## 204 8     8    1           1811
## 205 8     8    1           1864
```

From the output observation 1810, 1811, and 1864 were three observations that were easiest to identify as 8 with a probability from the model as 100% (in total there were 49 observations that had 100% probability).

A function that reshapes each observation to a 8x8 cases and then visualizing the result in a heatmap was done with the code as follows

```
# Change colour palette to black and white
colfunc <- colorRampPalette(c("white", "black"))

plot_8 <- function(index){
  title <- paste0("Obs: ", index)
  # Reshapes the observations to a 8x8
  plot <- as.matrix(train[index, -65]) # Remove response variable
  plot <- matrix(plot, nrow=8, byrow=TRUE)
  heatmap(plot, col=colfunc(16), Colv=NA, Rowv=NA, main=title, margins=c(2,2))
}
```

The heatmaps for observations 1624 and 1663 are presented in figure 1.

```
plot_8(1624)
plot_8(1663)
```

Figure 1: Heatmap for two observations that were hard to classify: 1624 and 1663.

In figure 1, it is hard to visually recognize what number the observations 1624 and 1663 are.

The heatmaps for observations 1810, 1811, and 1864 are presented in figure 2.

```
plot_8(1810)
plot_8(1811)
plot_8(1864)
```

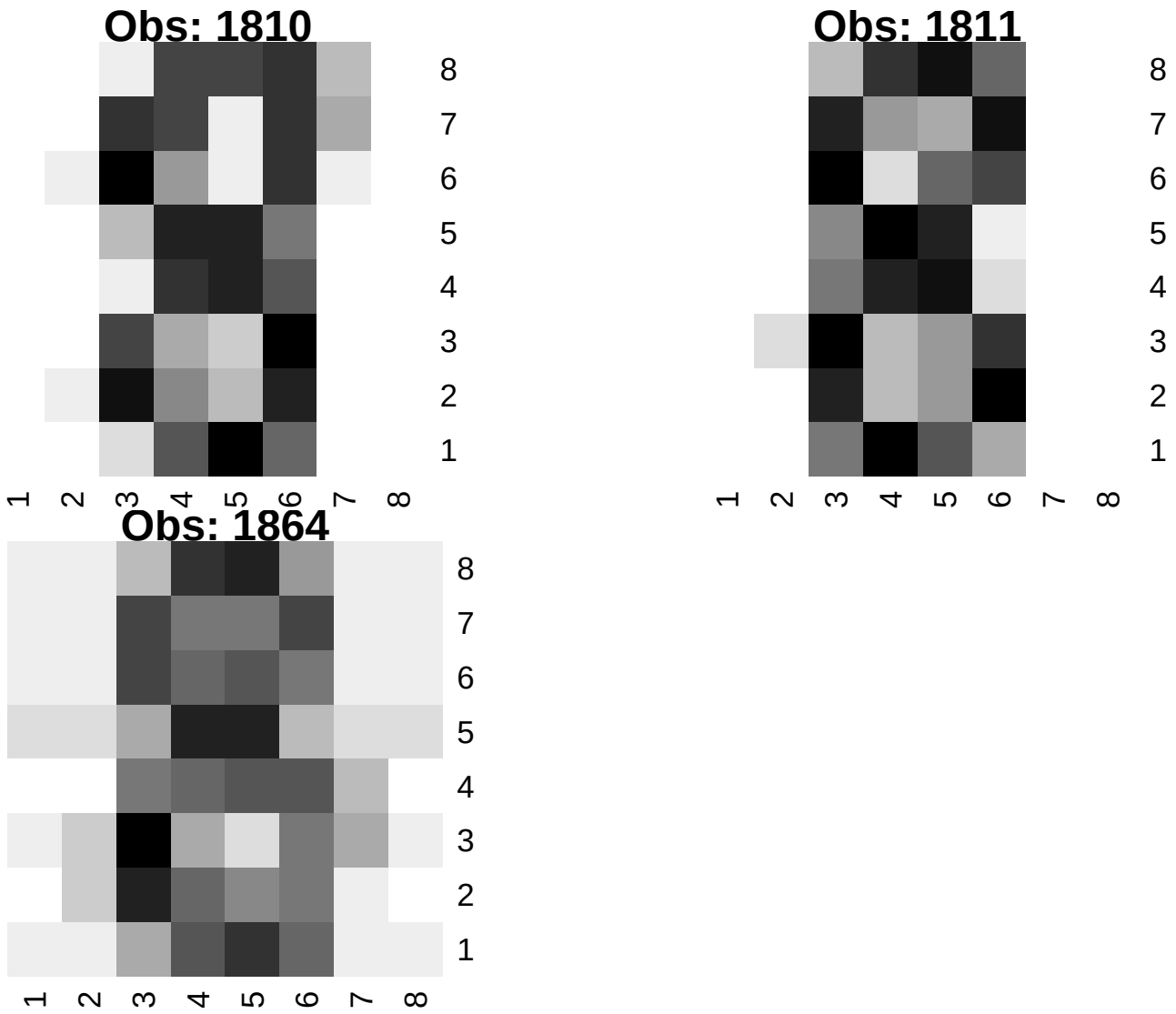

Figure 2: Heatmap for three observations that were easy to classify: 1810, 1811, and 1864.

In figure 2, is is easy to visually recognize that observations 1810, 1811, and 1864 are of the number 8.

1.4 Question 1.4

Question: Fit a K-nearest neighbor classifiers to the training data for different values of $K = 1, 2, ..., 30$ and plot the dependence of the training and validation misclassification errors on the value of K (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 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.

Answer: The code to create K-nearest neighbor classifiers for different K and the misclassification error on training and validation is as follows

```
fit_kknn <- function(k){
  model_kknn_train <- kknn(formula=y~., train=train, test=train, kernel="rectangular", k=k)
  # Confusion matrix for train data
  conf_mat_train <- table(model_kknn_train$fitted.values, train$y)
  acc_train <- sum(diag(conf_mat_train)) / sum(conf_mat_train)
  # Missclassification for training data
  miss_train <- 1-acc_train

  model_kknn_valid <- kknn(formula=y~., train=train, test=valid, kernel="rectangular", k=k)
  # Confusion matrix for validation data
  conf_mat_valid <- table(model_kknn_valid$fitted.values, valid$y)
  acc_valid <- sum(diag(conf_mat_valid)) / sum(conf_mat_valid)
  # Missclassification for validation data
  miss_valid <- 1-acc_valid

  result <- c(miss_train, miss_valid)
  return(result)
}

# Missclassification for k=1,...,30 for training and validation data
result <- data.frame(train = 0, valid = 0)
for(i in 1:30){
  model <- fit_kknn(i)
  result[i,1] <- model[1]
  result[i,2] <- model[2]
}
result$index <- 1:30
```

The plot showing the dependence of misclassification error for training and validation data is presented in figure 3.

```
ggplot(result, aes(x=index)) +
  geom_line(aes(y=train, colour="train")) +
  geom_point(aes(y=train, colour="train")) +
  geom_line(aes(y=valid, colour="valid")) +
  geom_point(aes(y=valid, colour="valid")) +
  scale_color_manual(name = "Data",
                     values = c("train" = "steelblue", "valid" = "indianred")) +
  scale_x_continuous(breaks = c(seq(from=0, to=30, by=5))) +
  scale_y_continuous(limits = c(0, 0.06)) +
  theme_bw() +
  labs(x = "k",
       y = "Missclassification rate")
```

In figure 3, the model complexity is highest when k = 1 and decreases with larger k. With k = 1, the prediction

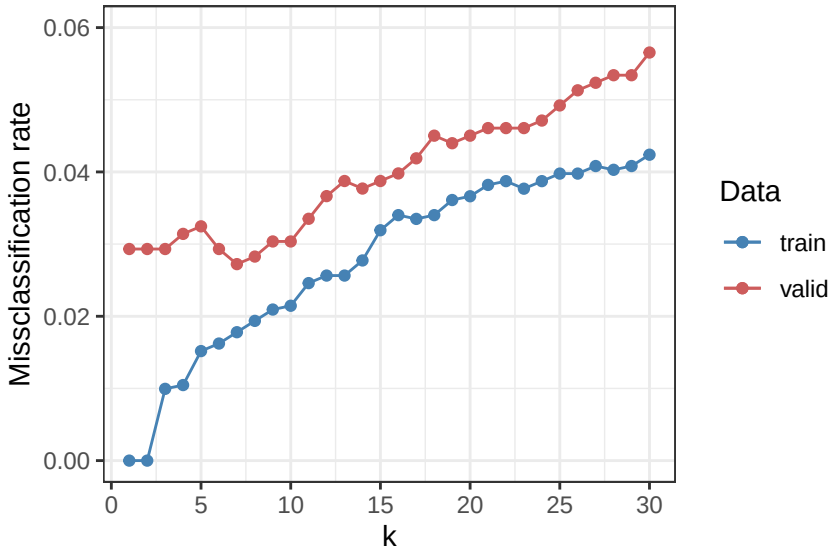

Figure 3: Misclassification errors for training and validation data for k=1,...,30 on model trained on training data.

in the model is by the observation in training data closest to the value we want to predict and when k is equal to the number of observations, the model will always predict the same value (except situations with ties). The training error increases when the model complexity decreases, this is because the model will be less overfitted on the training data with larger k and after some k, the model will be underfitted. Validation error also generally increased when the model complexity decreased, however at some k, the model will be between overfitted and underfitted which will give lowest validation error. This happened at k=7, which is considered to be the optimal k.

```
which(result$valid == min(result$valid))
```

```
## [1] 7
```

With k=7, the error for the test data is calculated. The errors for training, validation, and test data are compared in table 3.

```
model_test_7 <- kknn(formula = y~., train = train, test = test, kernel = "rectangular", k=7)

conf_mat_test <- table(model_test_7$fitted.values, test$y)
acc_test <- sum(diag(conf_mat_test)) / sum(conf_mat_test)
miss_test <- 1-acc_test

table_data <- cbind(training=result[7, 1], validation=result[7, 2], test=miss_test)
kable(table_data, digits=3, caption="Misclassification error k=7 for different data.")
```

Table 3: Misclassification error k=7 for different data.

training	validation	test
0.018	0.027	0.039

In table 3, the error for training is the lowest, followed by validation, and then test. Since the error for training is decently lower the model is a bit overfitted on training data. The difference between validation and test can be interpreted that for k=7 the error is smallest for the validation data, but it might not be the best since there is a bias when picking k from validation data.

1.5 Question 1.5

Question: Fit K-nearest neighbor classifiers to the training data for different values of $K = 1, 2, ..., 30$, 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. What is the optimal 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?

Answer: The code used to compute cross-entropy for validation data

```
cross_entropy <- function(k){
  model_kknn_valid <-
    kknn(formula = y~.,
         train = train,
         test = valid,
         kernel = "rectangular",
         k=k)

  y <- as.integer(valid$y)

  prob <- c()
  for(i in 1:length(y)){
    prob[i] <- model_kknn_valid$prob[i, y[i]]
  }

  value <- -sum(log(prob + 1e-15))
  return(value)
}

result <- c()
for(i in 1:30){
  model <- cross_entropy(i)
  result[i] <- model
}
```

The cross-entropy for different k is presented in figure 4.

```
plot_data <- data.frame(index=1:30, result)
ggplot(plot_data, aes(x=index, y=result)) +
  geom_point(color="forestgreen") +
  geom_line(color="forestgreen") +
  scale_x_continuous(breaks = c(seq(from=0, to=30, by=5))) +
  theme_bw() +
  labs(x="K",
       y="Cross-entropy")
```

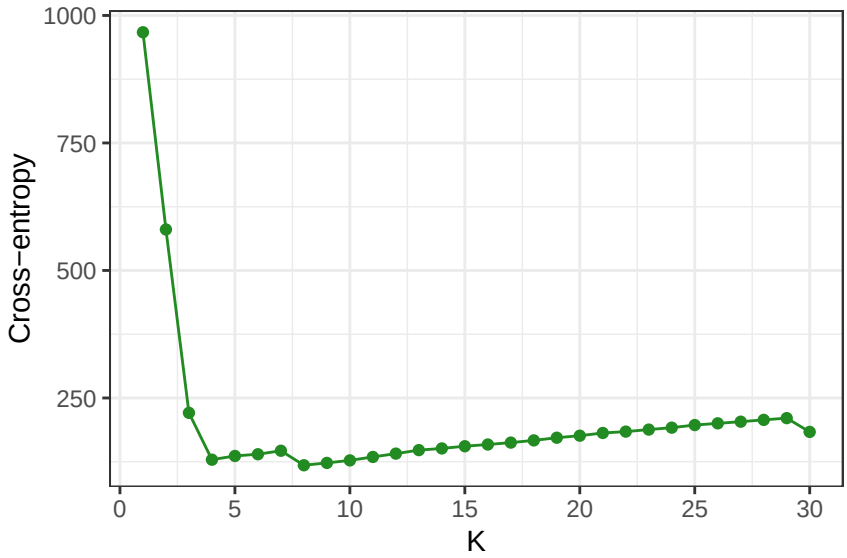

Figure 4: Cross-entropy error for validation data for different values of k for kknn models.

```
which(min(result) == result)
```

```
## [1] 8
```

From figure 4, the model with k=8 has the lowest value for cross-entropy and is considered to be the best k.

Cross-entropy for M classes $C_1, ..., C_M$ is defined as:

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

where the aim is to minimize the function.

When the probability of an observation Y_i for the true class C_m is low from the model, cross-entropy will give a larger penalty compared to a higher probability. For example if the model had probability 10% for the true value of observation Y_i cross-entropy would increase with $-log(0.1) \approx 2.3$. If the model instead had the probability 49%, cross-entropy would increase with $-log(0.49) \approx 0.7$. The reason why cross-entropy might be a better choice of error function is: misclassification error will only give us information of how often the model is wrong, cross-entropy will give a measure of how certain the model is on the true values.

2 Assignment 2: Linear Regression and Ridge Regression

The file "parkinson.csv" is composed of a range of biomedical voice measurements. Data consists of 5875 rows and 17 columns. The purpose is to predict Parkinson's disease score (motor_UPDRS) under the some following voice characteristics:

- Jitter(%), Jitter(Abs), Jitter:RAP, Jitter:PPQ5, Jitter.DDP - Several measures of variation in fundamental frequency
- Shimmer, Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5, Shimmer:APQ11, Shimmer:DDA - Several measures of variation in amplitude
- NHR, HNR- Two measures of ratio of noise to tonal components in the voice
- RPDE - A nonlinear dynamical complexity measure
- DFA - Signal fractal scaling exponent
- PPE - A nonlinear measure of fundamental frequency variation

```
df2_init <- read.csv("parkinsons.csv")
df2 <- df2_init[, c("motor_UPDRS", "Jitter...", "Jitter.Abs.", "Jitter.RAP", "Jitter.PPQ5",
                    "Jitter.DDP", "Shimmer", "Shimmer.dB.", "Shimmer.APQ3", "Shimmer.APQ5",
                    "Shimmer.APQ11", "Shimmer.DDA", "NHR", "HNR", "RPDE", "DFA", "PPE")]
```

2.1 Question 2.1)

Question: 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.

Answer: The code is as follows:

```
library(caret)
# Split the data into training and test
n <- dim(df2)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.6))
train_data <- df2[id,]
test_data <- df2[-id,]

# Scale the data
scaler <- preProcess(train_data)
trainS <- predict(scaler, train_data)
testS <- predict(scaler, test_data)
```

2.2 Question 2.2)

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

Answer: The summary of the linear regression model can be seen below. The variables "Jitter.Abs.", "Shimmer" "Shimmer.APQ5", "Shimmer.APQ11", "NHR", "HNR", "DFA" and "PPE" contribute significantly to the model at 5% significance level.

```
# Linear regression model
lm_model <- lm(motor_UPDRS ~ . -1, data = trainS)
summary(lm_model)
```

```
##
## Call:
## lm(formula = motor_UPDRS ~ . - 1, data = trainS)
##
## 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
```

Train and test data is estimated with the given model and MSE is calculated. The corresponding values can be seen in the table below.

```
# Predictions on the train & test data
predictor_cols <- setdiff(names(train_data), "motor_UPDRS")
trainS_x <- trainS[, predictor_cols]
testS_x <- testS[, predictor_cols]

predS_train <- predict(lm_model, newdata = trainS_x)
predS_test <- predict(lm_model, newdata = testS_x)
```

```
mse_train <- mean((trainS$motor_UPDRS - predS_train)^2)
mse_test <- mean((testS$motor_UPDRS - predS_test)^2)

cat("Mean Squared Error (MSE) on the training data:", mse_train, "\n")
```

```
## Mean Squared Error (MSE) on the training data: 0.8785431
```

```
cat("Mean Squared Error (MSE) on the test data:", mse_test, "\n")
```

```
## Mean Squared Error (MSE) on the test data: 0.9354477
```

Training data	Test data
0.8785431	0.9354477

Table 4: MSE on training and test data

2.3 Question 2.3)

Question: Implement 4 following functions by using basic R commands only (no external packages):

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

```
Loglikelihood <- function(theta, std){
  n <- nrow(trainS_x)
  prediction <- as.matrix(trainS_x) %*% as.matrix(theta)
  actual <- trainS$motor_UPDRS
  res <- actual-prediction
  likelihood <- (-(n/2) * log(2*pi*std^2) - (1/(2*std^2)) * sum(res^2))
  return(likelihood)
}
```

b) *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 loglikelihood.
Answer:

```
Ridge <- function(theta, std, lambda){
  likelihood_ridge <- -Loglikelihood(theta, std) + lambda*sum(theta^2)
  return(likelihood_ridge)
}
```

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

```
#optim() function minimizes
RidgeOpt <- function(lambda){ # optimize theta and std with given lambda
  # Define a new function to optimize
  my_fnc <- function(parameters){
    theta <- parameters[1:(length(parameters)-1)]
```

```
    std <- parameters[length(parameters)]
    return(Ridge(theta, std, lambda))
  }
  initial_values <- c(rep(0, ncol(trainS_x)), 1) # give 0 to theta and 1 to sigma to initialize

  optimal_values <- optim(par = initial_values, fn = my_fnc, method = "BFGS")$par
  optimal_theta <- optimal_values[1:length(predictor_cols)]
  optimal_std <- optimal_values[length(optimal_values)] #the last one is sigma
  result_list <- list(lambda = lambda, theta = optimal_theta, std = optimal_std)
  return(result_list)
}
```

d) *DF* function that for a given scalar λ computes the degrees of freedom of the Ridge model based on the training data.
Answer: Formula for the degree of freedom in ridge regression is as follows:

$$df(\lambda) = trace(X(X^T X + \lambda I)^{-1} X^T)$$

```
DF <- function(lambda){
  X <- as.matrix(trainS_x)
  dof <- sum(diag((X %*% (solve(t(X) %*% X + lambda*diag(ncol(trainS_x)))) %*% t(X))))
  return(dof)
}
```

2.4 Question 2.4)

Question: 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 the 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.

Answer:

```
lambda <- 1
result_ridge_1 <- RidgeOpt(lambda)

optimal_theta_1 <- result_ridge_1$theta
optimal_std_1 <- result_ridge_1$std
optimal_lambda_1 <- result_ridge_1$lambda

ridge_pred_train_1 <- as.matrix(trainS_x) %*% as.matrix(optimal_theta_1)
ridge_pred_test_1 <- as.matrix(testS_x) %*% as.matrix(optimal_theta_1)

mse_ridge_train_1 <- mean((trainS$motor_UPDRS - ridge_pred_train_1)^2)
mse_ridge_test_1 <- mean((testS$motor_UPDRS - ridge_pred_test_1)^2)

cat("Lambda:", lambda, "\n")
```

```
## Lambda: 1
```

```r
cat("Mean Squared Error (MSE) ridge regression on training data:", mse_ridge_train_1, "\n")
```

```
## Mean Squared Error (MSE) ridge regression on training data: 0.8786271
```

```r
cat("Mean Squared Error (MSE) ridge regression on test data:", mse_ridge_test_1, "\n")
```

```
## Mean Squared Error (MSE) ridge regression on test data: 0.9349969
```

```r
cat("Degree of freedom:", DF(lambda), "\n")
```

```
## Degree of freedom: 13.86074
```

```r
lambda <- 100
result_ridge_100 <- RidgeOpt(lambda)

optimal_theta_100 <- result_ridge_100$theta
optimal_std_100 <- result_ridge_100$std
optimal_lambda_100 <- result_ridge_100$lambda

ridge_pred_train_100 <- as.matrix(trainS_x) %*% as.matrix(optimal_theta_100)
ridge_pred_test_100 <- as.matrix(testS_x) %*% as.matrix(optimal_theta_100)

mse_ridge_train_100 <- mean((trainS$motor_UPDRS - ridge_pred_train_100)^2)
mse_ridge_test_100 <- mean((testS$motor_UPDRS - ridge_pred_test_100)^2)

cat("Lambda:", lambda, "\n")
```

```
## Lambda: 100
```

```r
cat("Mean Squared Error (MSE) ridge regression on training data:", mse_ridge_train_100, "\n")
```

```
## Mean Squared Error (MSE) ridge regression on training data: 0.8844104
```

```r
cat("Mean Squared Error (MSE) ridge regression on test data:", mse_ridge_test_100, "\n")
```

```
## Mean Squared Error (MSE) ridge regression on test data: 0.9323316
```

```r
cat("Degree of freedom:", DF(lambda), "\n")
```

```
## Degree of freedom: 9.924887
```

```r
lambda <- 1000
result_ridge_1000 <- RidgeOpt(lambda)

optimal_theta_1000 <- result_ridge_1000$theta
optimal_std_1000 <- result_ridge_1000$std
optimal_lambda_1000 <- result_ridge_1000$lambda

ridge_pred_train_1000 <- as.matrix(trainS_x) %*% as.matrix(optimal_theta_1000)
ridge_pred_test_1000 <- as.matrix(testS_x) %*% as.matrix(optimal_theta_1000)

mse_ridge_train_1000 <- mean((trainS$motor_UPDRS - ridge_pred_train_1000)^2)
mse_ridge_test_1000 <- mean((testS$motor_UPDRS - ridge_pred_test_1000)^2)
```

```
cat("Lambda:", lambda, "\n")
```

```
## Lambda: 1000
```

```
cat("Mean Squared Error (MSE) ridge regression on training data:", mse_ridge_train_1000, "\n")
```

```
## Mean Squared Error (MSE) ridge regression on training data: 0.9211216
```

```
cat("Mean Squared Error (MSE) ridge regression on test data:", mse_ridge_test_1000, "\n")
```

```
## Mean Squared Error (MSE) ridge regression on test data: 0.9539482
```

```
cat("Degree of freedom:", DF(lambda), "\n")
```

```
## Degree of freedom: 5.643925
```

Lambda	MSE training data	MSE test data	Degrees of freedom
1	0.8786271	0.9349969	13.86074
100	0.8844104	0.9323316	9.924887
1000	0.9211216	0.9539482	5.643925

Table 5: Ridge Regression Comparison

The comparison result can be seen in the table above. The penalty parameter that is most appropriate is $\theta = 100$, because it get the lowest MSE on test data. The degrees of freedom gets lower when λ gets higher and ....

3 Assignment 3. Logistic regression and basis function expansion

The data contains information about the onset of diabetes within 5 years in Pima Indians given medical details. The variables are:

- Number of times pregnant
- Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
- Diastolic blood pressure (mm Hg).
- Triceps skinfold thickness (mm).
- 2-Hour serum insulin (mu U/ml).
- Body mass index (weight in kg/(height in m)^2).
- Diabetes pedigree function.
- Age (years).
- Diabetes (0=no or 1=yes).

```
# Reading data
diabetes_df <- read.csv("pima-indians-diabetes.csv", header=FALSE)

colnames(diabetes_df) <- c("times_pregnant", "plasma_glucose_conc",
                           "diastolic_blood_pressure", "triceps_skinfold_thickness",
                           "serum_insulin","body_mass_index", "diabetes_pedigree",
                           "age", "diabetes")

diabetes_df$diabetes <- ifelse(diabetes_df$diabetes == 0, "no", "yes")
diabetes_df$diabetes <- as.factor(diabetes_df$diabetes)
```

3.1 Question 3.1

Question: Make a scatterplot showing a Plasma glucose concentration on Age where observations are colored by Diabetes levels.

```
library(ggplot2)

ggplot(diabetes_df, aes(x = plasma_glucose_conc, y = age, color = diabetes)) +
  geom_point() +
  theme_bw() +
  theme(axis.title.y = element_text(angle = 0,vjust = 0.5)) +
  labs(colour = "Diabetes",
       x = "Plasma glucose concentration",
       y = "Age")
```

Question: Do you think that Diabetes is easy to classify by a standard logistic regression model that uses these two variables as features? Motivate your answer.

Answer Since the decision boundary from a logistic model is a linear line, we believe that diabetes is not easy to classify by a standard logistic regression model that uses age and plasma glucose concentration as variables. They do not seem to separate the diabetes classes well.

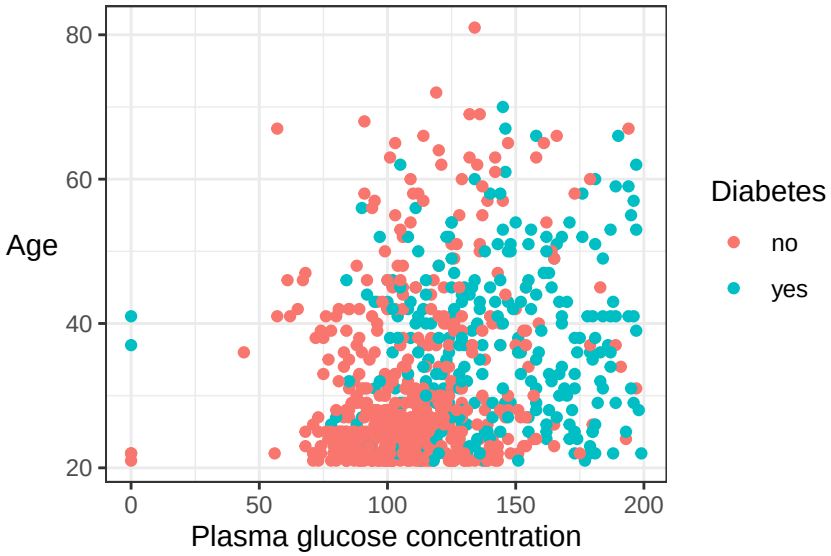

Figure 5: Scatterplot showing a Plasma glucose concentration on Age where observations are colored by Diabetes levels.

3.2 Question 3.2

Question:

Train a logistic regression model with y = Diabetes as target x_1 = Plasma glucose concentration and x_2 = Age as features and make a prediction for all observations by using $r = 0.5$ as the classification threshold. Report the probabilistic equation of the estimated model and compute also the training misclassification error.

The probabilistic equation:

$$p(y = 1) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 \cdot \text{Plasma glucose} + \theta_2 \cdot \text{Age})}}$$

$$p(y = 1) = \frac{1}{1 + e^{-(-5.91 + 0.04 \cdot \text{Plasma glucose} + 0.02 \cdot \text{Age})}}$$

$$\hat{y} = 1 \text{ if } p(y = 1) > 0.5$$

```
model <- glm(diabetes ~ plasma_glucose_conc + age, data = diabetes_df,
             family =  "binomial")

pred <- predict(model, newdata = diabetes_df, type = "response")

# Using 0.5 as the classification threshold
pred <- ifelse(pred > 0.5, "yes", "no")

# cunfusion matrix to calculate the misclassification error
confusion <- table(diabetes_df$diabetes, pred)
misclass_rate <- (confusion[1,2] + confusion[2,1]) / sum(confusion)
```

Table 6: Misclassification error

Misclassification error
0.263

The misclassification error is 0.263.

Question:

Make a scatter plot of the same kind as in step 1 but showing the predicted values of Diabetes as a color instead.

```
diabetes_df_pred <- diabetes_df
diabetes_df_pred$pred <- pred

ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
  geom_point() +
  theme_bw() +
  theme(axis.title.y = element_text(angle = 0,vjust = 0.5)) +
  labs(colour = "Predicted values of diabetes",
       x = "Plasma glucose concentration",
       y = "Age")
```

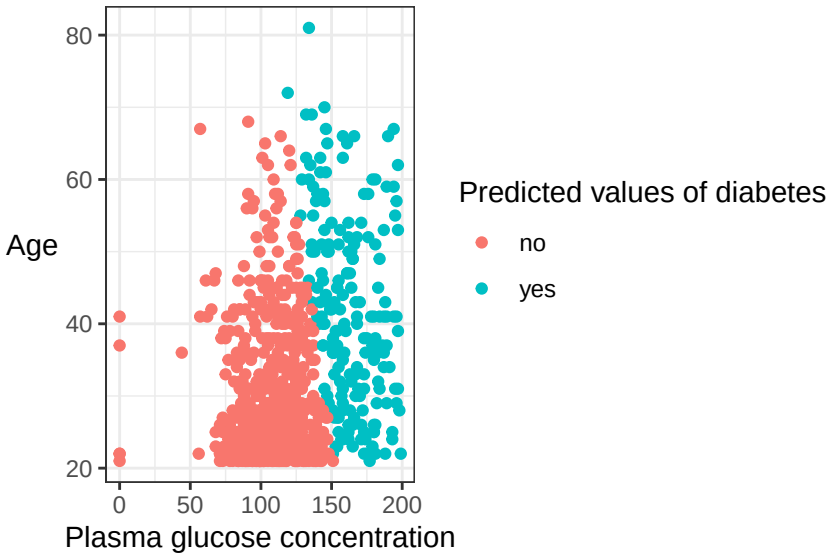

Figure 6: Scatterplot showing a Plasma glucose concentration on Age where observations are colored by predicted values of Diabetes.

Question:

Comment on the quality of the classification by using these results.

Answer

The quality of the classification is not best, because approximately 26% of the observations in our training data are incorrectly classified by the logistic regression model. If we compare this figure 6 to the figure 5 in step 3.1

we can see that the predicted values from figure 6 do not entirely align with the true values from figure 5.

3.3 Question 3.3

Question:

Use the model estimated in step 2 to a) report the equation of the decision boundary between the two classes b) add a curve showing this boundary to the scatter plot in step 2.

The decision boundary equation:

The decision boundary occurs when the predicted probability is equal to the threshold r.

$$\frac{1}{1+e^{-(\theta_0+\theta_1 \cdot \text{Plasma glucose}+\theta_2 \cdot \text{Age})}} = 0.5 \Rightarrow \theta_0 + \theta_1 \cdot \text{Plasma glucose} + \theta_2 \cdot \text{Age} = 0$$

$$-5.91 + 0.04 \cdot \text{Plasma glucose} + 0.02 \cdot \text{Age} = 0$$

```
ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
  geom_point() +
  theme_bw() +
 stat_function(fun = ({function(x) (-coef(model)[1] - coef(model)[2]*x)/ coef(model)[3] }),
               size=1.5, color = "black") +
  ylim(20,90) +
  labs(colour = "Predicted values of diabetes",
       x = "Plasma glucose concentration",
       y = "Age")
```

```
## Warning: Removed 76 row(s) containing missing values (geom_path).
```

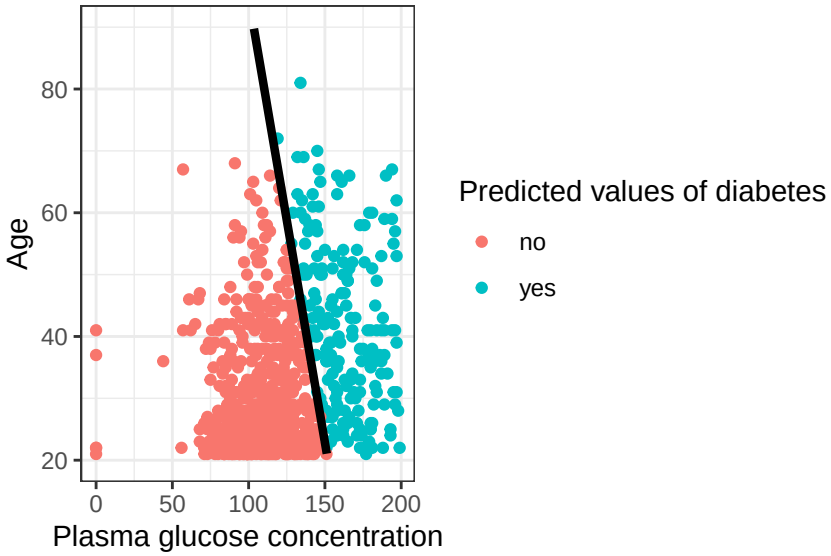

Figure 7: Scatterplot showing a Plasma glucose concentration on Age where observations are colored by predicted values of Diabetes with decision boundary.

Question:

Comment whether the decision boundary seems to catch the data distribution well.

Answer

The decision boundary is not able to capture the data distribution well. It is notable that the boundary between diabetes classes does not show a linear pattern in figure 5, which means that a linear decision boundary is never going to catch the data distribution very well.

3.4 Question 3.4

Question:

Make same kind of plots as in step 2 but use thresholds $r = 0.2$ and $r = 0.8$.

```
library(ggpubr)

# Using 0.2 as the classification threshold
pred <- predict(model, newdata = diabetes_df, type = "response")
pred <- ifelse(pred > 0.2, "yes", "no")
diabetes_df_pred$pred <- pred


p1 <- ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
      geom_point() +
      theme_bw() +
      labs(colour = "Predicted values of diabetes",
           x = "Plasma glucose concentration",
           y = "Age") +
      ggtitle("r = 0.2")


# Using 0.8 as the classification threshold
pred <- predict(model, newdata = diabetes_df, type = "response")
pred <- ifelse(pred > 0.8, "yes", "no")
diabetes_df_pred$pred <- pred


p2 <- ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
      geom_point() +
      theme_bw() +
      labs(colour = "Predicted values of diabetes",
           x = "Plasma glucose concentration",
           y = "Age") +
  ggtitle("r = 0.8")


ggarrange(p1, p2,  ncol = 1, nrow = 2)
```

Question:

By using these plots, comment on what happens with the prediction when r value changes.

Answer

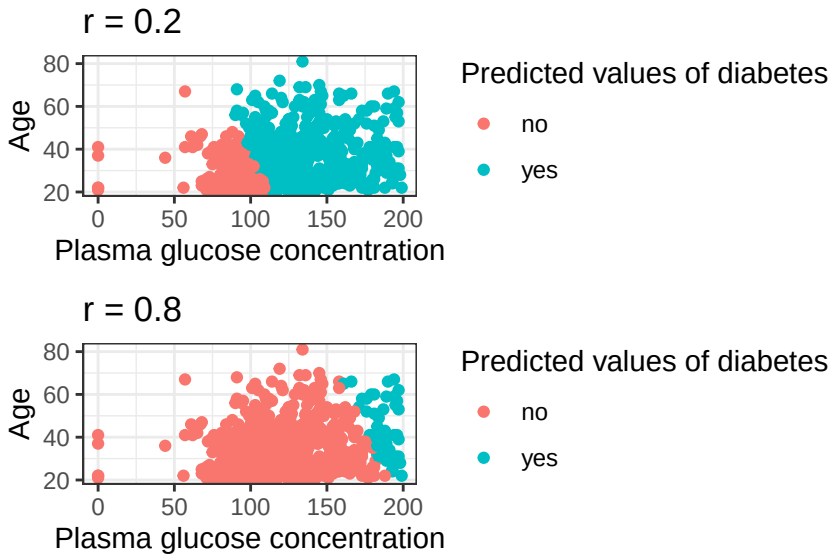

Figure 8: Scatterplot showing a Plasma glucose concentration on Age where observations are colored by predicted values of Diabetes for different thresholds.

As r increases, the model predict more observations as "no" for diabetes. Opposite, as r decreases, the model predict more observations as "yes" for diabetes.

3.5 Question 3.5

Question:

Perform a basis function expansion trick by computing new features $z_1 = x_1^4$, $z_2 = x_1^3 x^2$, $z_3 = x_1^2 x_2^2$, $z_4 = x_1 x_2^3$, $z_5 = x_2^4$, adding them to the data set and then computing a logistic regression model with y as target and $x_1, x_2, z_1, ..., z_5$ as features. Create a scatterplot of the same kind as in step 2 for this model and compute the training misclassification rate.

```
# new features
diabetes_df$z1  <- diabetes_df$times_pregnant^4
diabetes_df$z2  <- diabetes_df$times_pregnant^3 * diabetes_df$plasma_glucose_conc^2
diabetes_df$z3  <- diabetes_df$times_pregnant^2 * diabetes_df$plasma_glucose_conc^2
diabetes_df$z4  <- diabetes_df$times_pregnant * diabetes_df$plasma_glucose_conc^3
diabetes_df$z5  <- diabetes_df$plasma_glucose_conc^4


model <- glm(diabetes ~ plasma_glucose_conc + age + z1 + z2 + z3 + z4 + z5, data = diabetes_df,
             family =  "binomial")

pred <- predict(model, newdata = diabetes_df, type = "response")

# Using 0.5 as the classification threshold
pred <- ifelse(pred > 0.5, "yes", "no")

# cunfusion matrix to calculate the misclassification error
```

```r
confusion <- table(diabetes_df$diabetes, pred)
misclass_rate <- (confusion[1,2] + confusion[2,1]) / sum(confusion)


diabetes_df_pred <- diabetes_df
diabetes_df_pred$pred <- pred

ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
  geom_point() +
  theme_bw() +
  theme(axis.title.y = element_text(angle = 0,vjust = 0.5)) +
  labs(colour = "Predicted values of diabetes",
       x = "Plasma glucose concentration",
       y = "Age")
```

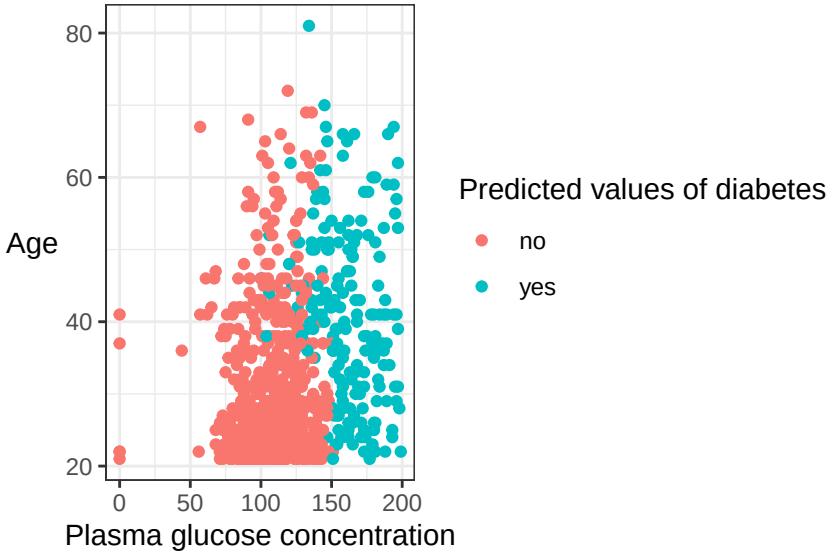

Figure 9: Scatterplot showing a Plasma glucose concentration on Age where observations are colored by predicted values of Diabetes with the new features

Table 7: Misclassification error

Misclassification error
0.2487

The misclassification error is 0.25.

Question:

What can you say about the quality of this model compared to the previous logistic regression model? How have the basis expansion trick affected the shape of the decision boundary and the prediction accuracy?

Answer

The misclassification rate is one percentage point lower than the model in 3.2, indicating a slight improvement. However, this model is more complex and the parameters are more harder to interpret than the model in 3.2, which could argue that the model in 3.2 is better.

The decision boundary's shape is not linear, but the data distribution of the predicted values remains very similar as in figure 5.

4 Statement of Contribution

We worked on the assignment individually for the computer labs (to be more efficient when asking questions), Duc on task 1, Sigme on task 2, and William on task 3. We later solved all assignment individually and compared and discussed our solutions before dividing the task of writing the laboration report.

4.1 Question 1

Text written by Duc.

4.2 Question 2

Text written by Sigme.

4.3 Question 3

Text written by William.

5 Appendix

The code used in this laboration report are summarised in the code as follows:

```
library(ggplot2)
library(kknn)
library(dplyr)
library(knitr)
library(caret)
library(psych)
knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 4.5,
  fig.height = 3)
# Read in data
data <- read.csv("optdigits.csv")

# Renaming the response variable and changing it to a factor variable
data <- rename(data, y=X0.26)
data$y <- as.factor(data$y)

# Partitioning training data (50%)
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]

# Partitioning validation data (25%)
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=data[id2,]
```

```r
# Partitioning test data (25%)
id3=setdiff(id1,id2)
test=data[id3,]
# kknn on training data and evaluation on training data
model_kknn_train <- kknn(formula=y~.,
                         train=train,
                         test=train,
                         kernel="rectangular",
                         k=30)
conf_mat_train <- table(train$y, model_kknn_train$fitted.values)
acc_train <- sum(diag(conf_mat_train)) / sum(conf_mat_train)
miss_train <- 1-acc_train

# kknn on training data and evaluation on test data
model_kknn_test <- kknn(formula=y~.,
                        train=train,
                        test=test,
                        kernel="rectangular",
                        k=30)
conf_mat_test <- table(test$y, model_kknn_test$fitted.values)
acc_test <- sum(diag(conf_mat_test)) / sum(conf_mat_test)
miss_test <- 1-acc_test

# Rows are true values, columns are model prediction
kable(conf_mat_train, caption = "Confusion matrix for training data, model
      predictions by columns and true value by rows.")
miss_train
kable(conf_mat_test, caption = "Confusion matrix for test data, model
      predictions by columns and true value by rows.")
miss_test
y <- train$y
fit_y <- model_kknn_train$fitted.values
# probabilities given from number 0 to 9, index 9 = number 8.
prob_8 <- model_kknn_train$prob[, 9]

# Data frame consisting of true value of y, model prediction and the models
# probability that the number is 8.
data_8 <- data.frame(y = y, fit_y = fit_y, prob = prob_8)
data_8$observation_id <- rownames(data_8)

# Only observations with the label 8 is kept.
data_8 <- data_8[data_8$y == "8", ]
head(arrange(data_8, prob), 2)
tail(arrange(data_8, prob), 3)
# Change colour palette to black and white
colfunc <- colorRampPalette(c("white", "black"))

plot_8 <- function(index){
```

```
  title <- paste0("Obs: ", index)
  # Reshapes the observations to a 8x8
  plot <- as.matrix(train[index, -65]) # Remove response variable
  plot <- matrix(plot, nrow=8, byrow=TRUE)
  heatmap(plot, col=colfunc(16), Colv=NA, Rowv=NA, main=title, margins=c(2,2))
}
plot_8(1624)
plot_8(1663)
plot_8(1810)
plot_8(1811)
plot_8(1864)
fit_kknn <- function(k){
  model_kknn_train <- kknn(formula=y~., train=train, test=train, kernel="rectangular", k=k)
  # Confusion matrix for train data
  conf_mat_train <- table(model_kknn_train$fitted.values, train$y)
  acc_train <- sum(diag(conf_mat_train)) / sum(conf_mat_train)
  # Missclassification for training data
  miss_train <- 1-acc_train

  model_kknn_valid <- kknn(formula=y~., train=train, test=valid, kernel="rectangular", k=k)
  # Confusion matrix for validation data
  conf_mat_valid <- table(model_kknn_valid$fitted.values, valid$y)
  acc_valid <- sum(diag(conf_mat_valid)) / sum(conf_mat_valid)
  # Missclassification for validation data
  miss_valid <- 1-acc_valid

  result <- c(miss_train, miss_valid)
  return(result)
}

# Missclassification for k=1,...,30 for training and validation data
result <- data.frame(train = 0, valid = 0)
for(i in 1:30){
  model <- fit_kknn(i)
  result[i,1] <- model[1]
  result[i,2] <- model[2]
}
result$index <- 1:30
ggplot(result, aes(x=index)) +
  geom_line(aes(y=train, colour="train")) +
  geom_point(aes(y=train, colour="train")) +
  geom_line(aes(y=valid, colour="valid")) +
  geom_point(aes(y=valid, colour="valid")) +
  scale_color_manual(name = "Data",
                     values = c("train" = "steelblue", "valid" = "indianred")) +
  scale_x_continuous(breaks = c(seq(from=0, to=30, by=5))) +
  scale_y_continuous(limits = c(0, 0.06)) +
  theme_bw() +
  labs(x = "k",
```

```
       y = "Missclassification rate")
which(result$valid == min(result$valid))
model_test_7 <- kknn(formula = y~., train = train, test = test, kernel = "rectangular", k=7)

conf_mat_test <- table(model_test_7$fitted.values, test$y)
acc_test <- sum(diag(conf_mat_test)) / sum(conf_mat_test)
miss_test <- 1-acc_test

table_data <- cbind(training=result[7, 1], validation=result[7, 2], test=miss_test)
kable(table_data, digits=3, caption="Misclassification error k=7 for different data.")
cross_entropy <- function(k){
  model_kknn_valid <-
    kknn(formula = y~.,
         train = train,
         test = valid,
         kernel = "rectangular",
         k=k)

  y <- as.integer(valid$y)

  prob <- c()
  for(i in 1:length(y)){
    prob[i] <- model_kknn_valid$prob[i, y[i]]
  }

  value <- -sum(log(prob + 1e-15))
  return(value)
}

result <- c()
for(i in 1:30){
  model <- cross_entropy(i)
  result[i] <- model
}
plot_data <- data.frame(index=1:30, result)
ggplot(plot_data, aes(x=index, y=result)) +
  geom_point(color="forestgreen") +
  geom_line(color="forestgreen") +
  scale_x_continuous(breaks = c(seq(from=0, to=30, by=5))) +
  theme_bw() +
  labs(x="K",
       y="Cross-entropy")
which(min(result) == result)
df2_init <- read.csv("parkinsons.csv")
df2 <- df2_init[, c("motor_UPDRS", "Jitter...", "Jitter.Abs.", "Jitter.RAP", "Jitter.PPQ5",
                    "Jitter.DDP", "Shimmer", "Shimmer.dB.", "Shimmer.APQ3", "Shimmer.APQ5",
                    "Shimmer.APQ11", "Shimmer.DDA", "NHR", "HNR", "RPDE", "DFA", "PPE")]
library(caret)
# Split the data into training and test
```

```r
n <- dim(df2)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.6))
train_data <- df2[id,]
test_data <- df2[-id,]

# Scale the data
scaler <- preProcess(train_data)
trainS <- predict(scaler, train_data)
testS <- predict(scaler, test_data)
# Linear regression model
lm_model <- lm(motor_UPDRS ~ . -1, data = trainS)
summary(lm_model)
# Predictions on the train & test data
predictor_cols <- setdiff(names(train_data), "motor_UPDRS")
trainS_x <- trainS[, predictor_cols]
testS_x <- testS[, predictor_cols]

predS_train <- predict(lm_model, newdata = trainS_x)
predS_test <- predict(lm_model, newdata = testS_x)

mse_train <- mean((trainS$motor_UPDRS - predS_train)^2)
mse_test <- mean((testS$motor_UPDRS - predS_test)^2)

cat("Mean Squared Error (MSE) on the training data:", mse_train, "\n")
cat("Mean Squared Error (MSE) on the test data:", mse_test, "\n")
Loglikelihood <- function(theta, std){
  n <- nrow(trainS_x)
  prediction <- as.matrix(trainS_x) %*% as.matrix(theta)
  actual <- trainS$motor_UPDRS
  res <- actual-prediction
  likelihood <- (-(n/2) * log(2*pi*std^2) - (1/(2*std^2)) * sum(res^2))
  return(likelihood)
}
Ridge <- function(theta, std, lambda){
  likelihood_ridge <- -Loglikelihood(theta, std) + lambda*sum(theta^2)
  return(likelihood_ridge)
}
#optim() function minimizes
RidgeOpt <- function(lambda){ # optimize theta and std with given lambda
  # Define a new function to optimize
  my_fnc <- function(parameters){
    theta <- parameters[1:(length(parameters)-1)]
    std <- parameters[length(parameters)]
    return(Ridge(theta, std, lambda))
  }
  initial_values <- c(rep(0, ncol(trainS_x)), 1) # give 0 to theta and 1 to sigma to initialize

  optimal_values <- optim(par = initial_values, fn = my_fnc, method = "BFGS")$par
```

```
  optimal_theta <- optimal_values[1:length(predictor_cols)]
  optimal_std <- optimal_values[length(optimal_values)] #the last one is sigma
  result_list <- list(lambda = lambda, theta = optimal_theta, std = optimal_std)
  return(result_list)
}
DF <- function(lambda){
  X <- as.matrix(trainS_x)
  dof <- sum(diag((X %*% (solve(t(X) %*% X + lambda*diag(ncol(trainS_x)))) %*% t(X))))
  return(dof)
}
lambda <- 1
result_ridge_1 <- RidgeOpt(lambda)

optimal_theta_1 <- result_ridge_1$theta
optimal_std_1 <- result_ridge_1$std
optimal_lambda_1 <- result_ridge_1$lambda

ridge_pred_train_1 <- as.matrix(trainS_x) %*% as.matrix(optimal_theta_1)
ridge_pred_test_1 <- as.matrix(testS_x) %*% as.matrix(optimal_theta_1)

mse_ridge_train_1 <- mean((trainS$motor_UPDRS - ridge_pred_train_1)^2)
mse_ridge_test_1 <- mean((testS$motor_UPDRS - ridge_pred_test_1)^2)

cat("Lambda:", lambda, "\n")
cat("Mean Squared Error (MSE) ridge regression on training data:", mse_ridge_train_1, "\n")
cat("Mean Squared Error (MSE) ridge regression on test data:", mse_ridge_test_1, "\n")
cat("Degree of freedom:", DF(lambda), "\n")
lambda <- 100
result_ridge_100 <- RidgeOpt(lambda)

optimal_theta_100 <- result_ridge_100$theta
optimal_std_100 <- result_ridge_100$std
optimal_lambda_100 <- result_ridge_100$lambda

ridge_pred_train_100 <- as.matrix(trainS_x) %*% as.matrix(optimal_theta_100)
ridge_pred_test_100 <- as.matrix(testS_x) %*% as.matrix(optimal_theta_100)

mse_ridge_train_100 <- mean((trainS$motor_UPDRS - ridge_pred_train_100)^2)
mse_ridge_test_100 <- mean((testS$motor_UPDRS - ridge_pred_test_100)^2)

cat("Lambda:", lambda, "\n")
cat("Mean Squared Error (MSE) ridge regression on training data:", mse_ridge_train_100, "\n")
cat("Mean Squared Error (MSE) ridge regression on test data:", mse_ridge_test_100, "\n")
cat("Degree of freedom:", DF(lambda), "\n")
lambda <- 1000
result_ridge_1000 <- RidgeOpt(lambda)

optimal_theta_1000 <- result_ridge_1000$theta
optimal_std_1000 <- result_ridge_1000$std
```

```r
optimal_lambda_1000 <- result_ridge_1000$lambda

ridge_pred_train_1000 <- as.matrix(trainS_x) %*% as.matrix(optimal_theta_1000)
ridge_pred_test_1000 <- as.matrix(testS_x) %*% as.matrix(optimal_theta_1000)

mse_ridge_train_1000 <- mean((trainS$motor_UPDRS - ridge_pred_train_1000)^2)
mse_ridge_test_1000 <- mean((testS$motor_UPDRS - ridge_pred_test_1000)^2)

cat("Lambda:", lambda, "\n")
cat("Mean Squared Error (MSE) ridge regression on training data:", mse_ridge_train_1000, "\n")
cat("Mean Squared Error (MSE) ridge regression on test data:", mse_ridge_test_1000, "\n")
cat("Degree of freedom:", DF(lambda), "\n")

# Reading data
diabetes_df <- read.csv("pima-indians-diabetes.csv", header=FALSE)

colnames(diabetes_df) <- c("times_pregnant", "plasma_glucose_conc",
                           "diastolic_blood_pressure", "triceps_skinfold_thickness",
                           "serum_insulin","body_mass_index", "diabetes_pedigree",
                           "age", "diabetes")

diabetes_df$diabetes <- ifelse(diabetes_df$diabetes == 0, "no", "yes")
diabetes_df$diabetes <- as.factor(diabetes_df$diabetes)




library(ggplot2)

ggplot(diabetes_df, aes(x = plasma_glucose_conc, y = age, color = diabetes)) +
  geom_point() +
  theme_bw() +
  theme(axis.title.y = element_text(angle = 0,vjust = 0.5)) +
  labs(colour = "Diabetes",
       x = "Plasma glucose concentration",
       y = "Age")


model <- glm(diabetes ~ plasma_glucose_conc + age, data = diabetes_df,
             family =  "binomial")

pred <- predict(model, newdata = diabetes_df, type = "response")

# Using 0.5 as the classification threshold
pred <- ifelse(pred > 0.5, "yes", "no")

# cunfusion matrix to calculate the misclassification error
confusion <- table(diabetes_df$diabetes, pred)
misclass_rate <- (confusion[1,2] + confusion[2,1]) / sum(confusion)
```

```
knitr::kable(as.data.frame(round(misclass_rate,4)), col.names = "Misclassification error",
             caption = "Misclassification error")


diabetes_df_pred <- diabetes_df
diabetes_df_pred$pred <- pred

ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
  geom_point() +
  theme_bw() +
  theme(axis.title.y = element_text(angle = 0,vjust = 0.5)) +
  labs(colour = "Predicted values of diabetes",
       x = "Plasma glucose concentration",
       y = "Age")




ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
  geom_point() +
  theme_bw() +
 stat_function(fun = ({function(x) (-coef(model)[1] - coef(model)[2]*x)/ coef(model)[3] }),
               size=1.5, color = "black") +
  ylim(20,90) +
  labs(colour = "Predicted values of diabetes",
       x = "Plasma glucose concentration",
       y = "Age")



library(ggpubr)

# Using 0.2 as the classification threshold
pred <- predict(model, newdata = diabetes_df, type = "response")
pred <- ifelse(pred > 0.2, "yes", "no")
diabetes_df_pred$pred <- pred


p1 <- ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
      geom_point() +
      theme_bw() +
      labs(colour = "Predicted values of diabetes",
           x = "Plasma glucose concentration",
           y = "Age") +
      ggtitle("r = 0.2")
```

```r
# Using 0.8 as the classification threshold
pred <- predict(model, newdata = diabetes_df, type = "response")
pred <- ifelse(pred > 0.8, "yes", "no")
diabetes_df_pred$pred <- pred


p2 <- ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
      geom_point() +
      theme_bw() +
      labs(colour = "Predicted values of diabetes",
           x = "Plasma glucose concentration",
           y = "Age") +
  ggtitle("r = 0.8")


ggarrange(p1, p2,  ncol = 1, nrow = 2)


# new features
diabetes_df$z1  <- diabetes_df$times_pregnant^4
diabetes_df$z2  <- diabetes_df$times_pregnant^3 * diabetes_df$plasma_glucose_conc^2
diabetes_df$z3  <- diabetes_df$times_pregnant^2 * diabetes_df$plasma_glucose_conc^2
diabetes_df$z4  <- diabetes_df$times_pregnant * diabetes_df$plasma_glucose_conc^3
diabetes_df$z5  <- diabetes_df$plasma_glucose_conc^4


model <- glm(diabetes ~ plasma_glucose_conc + age + z1 + z2 + z3 + z4 + z5, data = diabetes_df,
             family =  "binomial")

pred <- predict(model, newdata = diabetes_df, type = "response")

# Using 0.5 as the classification threshold
pred <- ifelse(pred > 0.5, "yes", "no")

# cunfusion matrix to calculate the misclassification error
confusion <- table(diabetes_df$diabetes, pred)
misclass_rate <- (confusion[1,2] + confusion[2,1]) / sum(confusion)


diabetes_df_pred <- diabetes_df
diabetes_df_pred$pred <- pred

ggplot(diabetes_df_pred, aes(x = plasma_glucose_conc, y = age, color = pred)) +
  geom_point() +
  theme_bw() +
  theme(axis.title.y = element_text(angle = 0,vjust = 0.5)) +
  labs(colour = "Predicted values of diabetes",
```

```
       x = "Plasma glucose concentration",
       y = "Age")



knitr::kable(as.data.frame(round(misclass_rate,4)), col.names = "Misclassification error",
             caption = "Misclassification error")
```