Homework 5

Marc Hughes

Table of contents

Link to the Github repository

! Due: Wed, Apr 19, 2023 @ 11:59pm

Please read the instructions carefully before submitting your assignment.

1. This assignment requires you to only upload a `PDF` file on Canvas
2. Don't collapse any code cells before submitting.
3. Remember to make sure all your code output is rendered properly before uploading your submission.

Please add your name to the author information in the frontmatter before submitting your assignment

In this assignment, we will explore decision trees, support vector machines and neural networks for classification and regression. The assignment is designed to test your ability to fit and analyze these models with different configurations and compare their performance.

We will need the following packages:

```
packages <- c(
  "dplyr",
  "readr",
  "tidyr",
```

```
  "purrr",
  "broom",
  "magrittr",
  "corrplot",
  "caret",
  "rpart",
  "rpart.plot",
  "e1071",
  "torch",
  "luz"
)

# renv::install(packages)
sapply(packages, require, character.only=T)
```

Question 1

> 60 points
>
> Prediction of Median House prices

1.1 (2.5 points)

The `data` folder contains the `housing.csv` dataset which contains housing prices in California from the 1990 California census. The objective is to predict the median house price for California districts based on various features.

Read the data file as a tibble in R. Preprocess the data such that:

1. the variables are of the right data type, e.g., categorical variables are encoded as factors
2. all column names to lower case for consistency
3. Any observations with missing values are dropped

```
path <- "data/housing.csv"

# reading in the dataset
df <- read_csv(path)
```

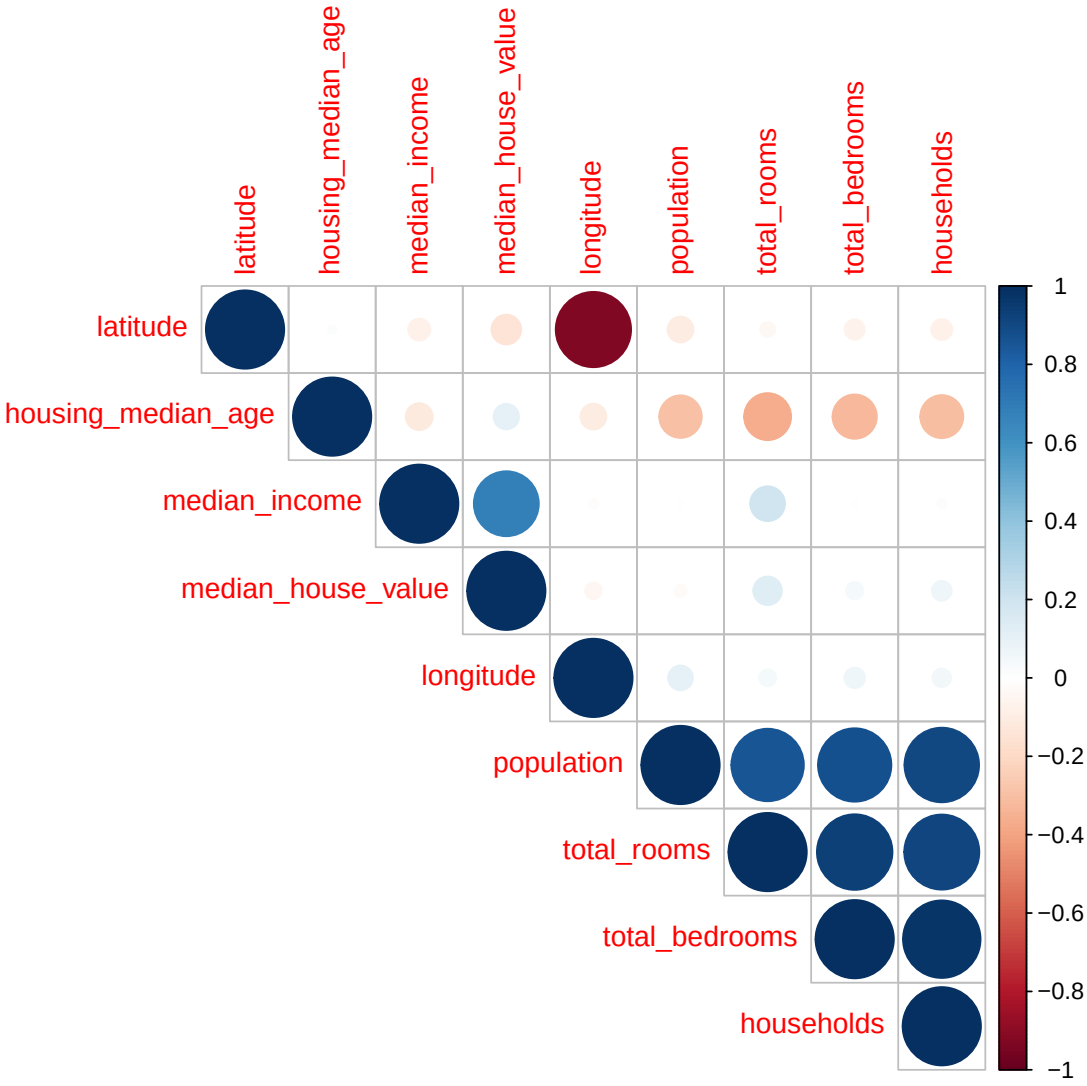
```
Rows: 20640 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): ocean_proximity
dbl (9): longitude, latitude, housing_median_age, total_rooms, total_bedroom...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# dropping all NA values
df <- df %>%
  drop_na()
```

1.2 (2.5 points)

Visualize the correlation matrix of all numeric columns in `df` using `corrplot()`

```r
df %>%
  # keeping only the numeric columns
  keep(is.numeric) %>%
  cor() %>%
  corrplot(type = "upper", order = "hclust")
```



1.3 (5 points)

Split the data `df` into `df_train` and `df_split` using `test_ind` in the code below:

```
set.seed(42)
test_ind <- sample(
```

```
  1:nrow(df),
  floor( nrow(df)/10 ),
  replace=FALSE
)

df_train <- df[-test_ind, ]
df_test  <- df[test_ind, ]
```

1.4 (5 points)

Fit a linear regression model to predict the `median_house_value` :

- `latitude`
- `longitude`
- `housing_median_age`
- `total_rooms`
- `total_bedrooms`
- `population`
- `median_income`
- `ocean_proximity`

Interpret the coefficients and summarize your results.

```
# fitting the model and deselecting the 'households' variable
lm_fit <- lm(median_house_value ~ ., df_train %>% select(-households))
summary(lm_fit)
```

```
Call:
lm(formula = median_house_value ~ ., data = df_train %>% select(-households))

Residuals:
    Min      1Q  Median      3Q     Max
-559024  -42322  -10389   28743  710215

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             -2.273e+06  9.138e+04 -24.873  < 2e-16 ***
longitude               -2.681e+04  1.060e+03 -25.305  < 2e-16 ***
latitude                -2.539e+04  1.047e+03 -24.244  < 2e-16 ***
```

```
housing_median_age         1.074e+03  4.616e+01  23.261  < 2e-16 ***
total_rooms               -6.159e+00  8.431e-01  -7.306 2.87e-13 ***
total_bedrooms             1.353e+02  4.254e+00  31.804  < 2e-16 ***
population                -3.413e+01  9.838e-01 -34.694  < 2e-16 ***
median_income              3.936e+04  3.573e+02 110.154  < 2e-16 ***
ocean_proximityINLAND     -4.018e+04  1.836e+03 -21.891  < 2e-16 ***
ocean_proximityISLAND      1.324e+05  3.442e+04   3.847  0.00012 ***
ocean_proximityNEAR BAY   -2.522e+03  2.022e+03  -1.247  0.21226
ocean_proximityNEAR OCEAN  4.349e+03  1.658e+03   2.622  0.00875 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 68780 on 18378 degrees of freedom
Multiple R-squared:  0.643, Adjusted R-squared:  0.6428
F-statistic:  3009 on 11 and 18378 DF,  p-value: < 2.2e-16
```

Most of the p-values of the covariates are statistically significant with the ocean_proximity specifically 'NEAR OCEAN' being less significant than the other covariates, and the 'NEAR BAY' not being significant at all. As a result, we reject the null hypothesis and accept the alternative hypothesis. Because the 'ocean_proximity' variable is categorical, the variable is split into four subgroups each representing a different category type.

1.5 (5 points)

Complete the `rmse` function for computing the Root Mean-Squared Error between the true `y` and the predicted `yhat`, and use it to compute the RMSE for the regression model on `df_test`

```r
rmse <- function(y, yhat) {
  sqrt(mean((y - yhat)^2))
}

# predicting using the model
lm_predictions <- predict(lm_fit, newdata = df_test, type = "response")

# calling the 'rmse()' function using the response variable and the predictions
lm_rmse <- rmse(df_test$median_house_value, lm_predictions)
lm_rmse
```

```
[1] 68339.82
```

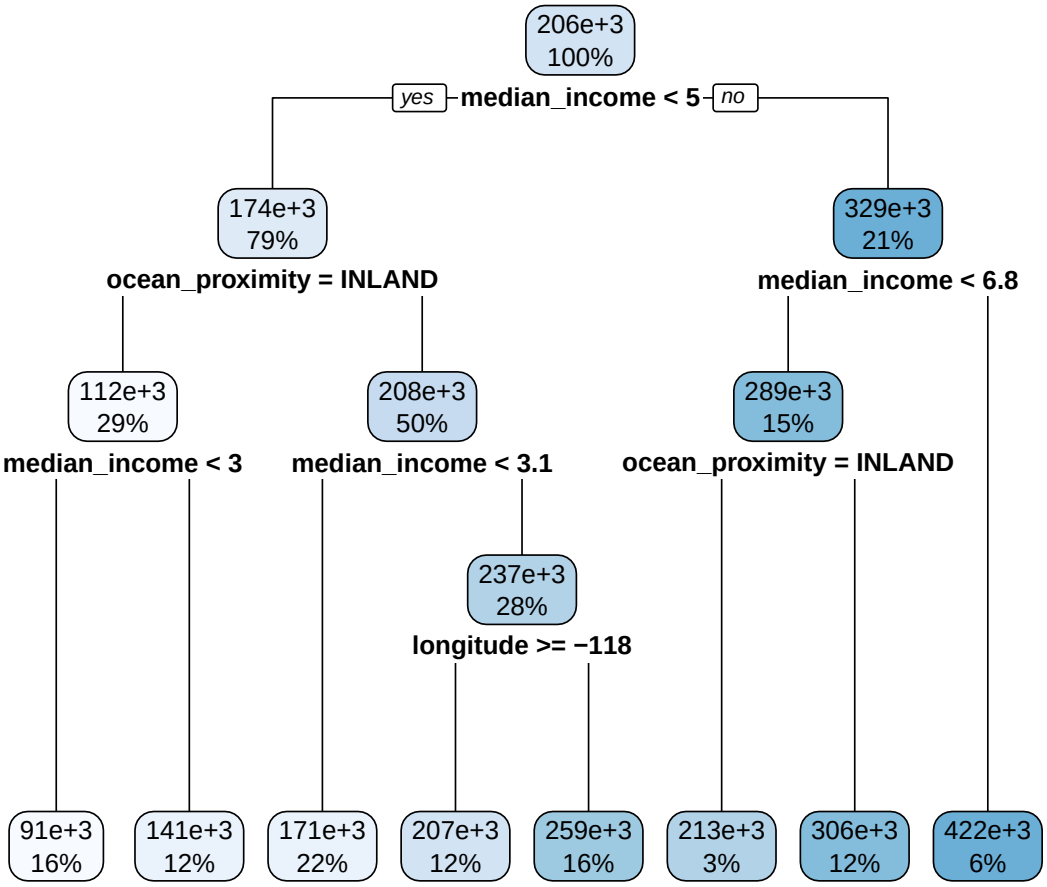
1.6 (5 points)

Fit a decision tree model to predict the `median_house_value` using the same predictors as in 1.4. Use the `rpart()` function.

```
# fitting the decision tree and deselecting the 'households' variable
rpart_fit <- rpart(median_house_value ~ ., df_train %>% select(-households))
# predicting the response values on the test data
rpart_predictions <- predict(rpart_fit, newdata = df_test)
```

Visualize the decision tree using the `rpart.plot()` function.

```
# plotting the decision tree
rpart.plot(rpart_fit)
```



Report the root mean squared error on the test set.

```
rpart_predictions <- predict(rpart_fit, newdata = df_test)

# using the 'rmse()' function to find the root mean squared error
# y = 'df_test$median_house_value' and yhat = 'rpart_predictions'
rpart_rmse <- rmse(df_test$median_house_value, rpart_predictions)
```

```
rpart_rmse
```

```
[1] 75876.87
```

1.7 (5 points)

Fit a support vector machine model to predict the `median_house_value` using the same predictors as in 1.4. Use the `svm()` function and use any kernel of your choice. Report the root mean squared error on the test set.

```
# fitting the svm, deselecting the 'households' variable, and selecting the kernel
svm_fit <- svm(median_house_value ~ ., df_train %>% select(-households),
               kernel = "radial")
# making predictions
svm_predictions <- predict(svm_fit, newdata = df_test)

# calculating the root mean squared error
svm_rmse <- rmse(df_test$median_house_value, svm_predictions)
svm_rmse
```

```
[1] 56678.84
```

1.8 (25 points)

Initialize a neural network model architecture:

```
# creating the neural network
NNet <- nn_module(
    initialize = function(p, q1, q2, q3){
      # initializing the architecture
      self$hidden1 <- nn_linear(p, q1)
      self$hidden2 <- nn_linear(q1, q2)
      self$hidden3 <- nn_linear(q2, q3)
      self$output <- nn_linear(q3, 1)
      self$activation <- nn_relu()
    },
```

```
    forward = function(x){
      # creating the forward function
      x %>%
        self$hidden1() %>%
        self$activation() %>%
        self$hidden2() %>%
        self$activation() %>%
        self$hidden3() %>%
        self$activation() %>%
        self$output()
    }
)
```

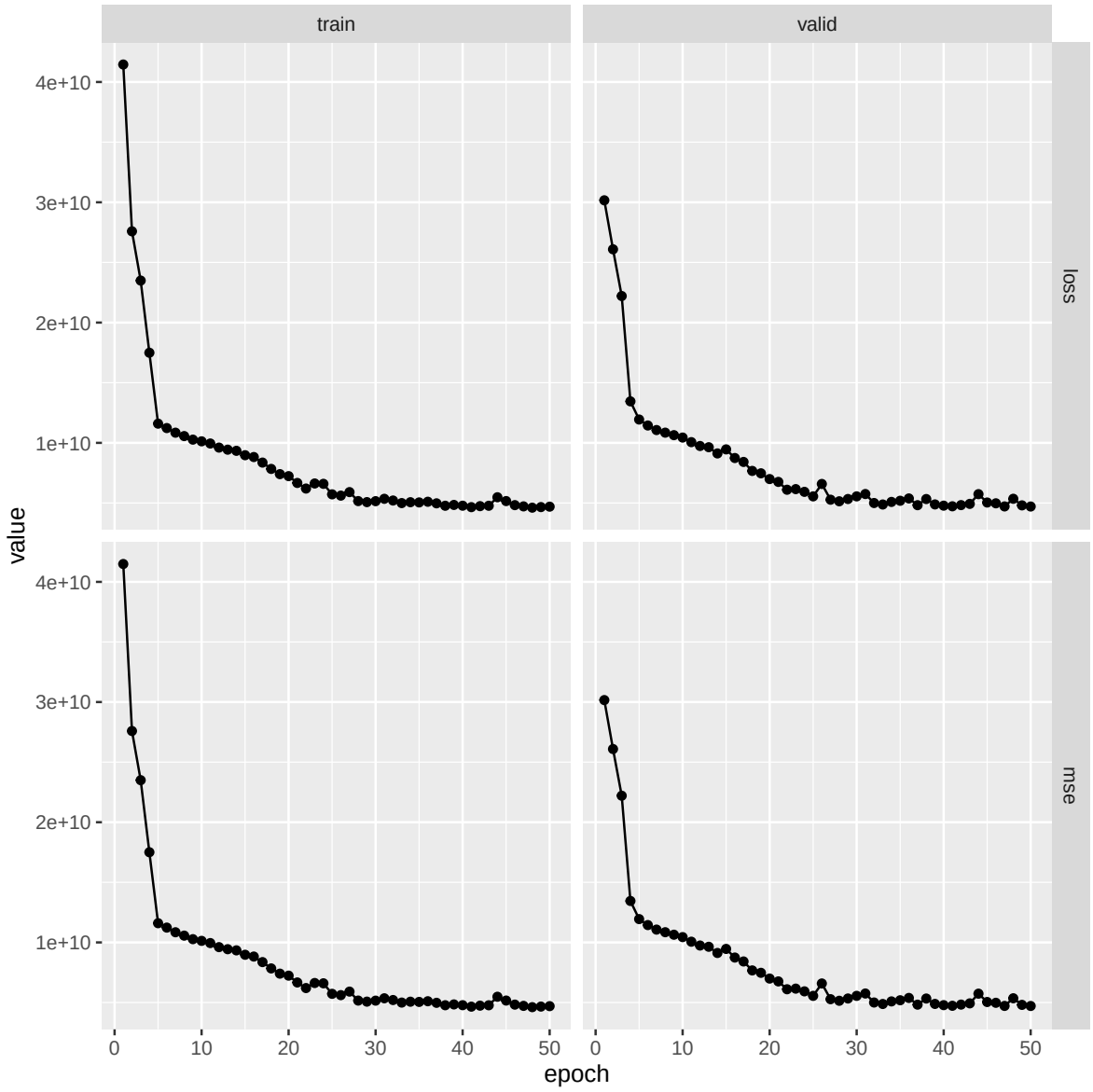
Fit a neural network model to predict the `median_house_value` using the same predictors as in 1.4. Use the `model.matrix` function to create the covariate matrix and `luz` package for fitting the network with $32, 16, 8$ nodes in each of the three hidden layers.

```
# creating the covariate matrix and deselecting the 'households' variable
# doing '0 + .,' to un-include the response variable
M <- model.matrix(median_house_value ~ 0 + ., data = df_train %>% select(-households))

# fitting the neural network model
nnet_fit <- NNet %>%
  setup(
    loss = nn_mse_loss(),
    optimizer = optim_adam,
    metrics = list(
      luz_metric_mse()
    )
  ) %>%
  set_hparams(
    p=ncol(M), q1 = 32, q2 = 16, q3 = 8
  ) %>%
  set_opt_hparams(
    lr = 0.05
  ) %>%
  fit(
    data = list(
      model.matrix(median_house_value ~ 0 + ., data = df_train %>% select(-households)),
      df_train %>% select(median_house_value) %>% as.matrix
    ),
    valid_data = list(
```

```
      model.matrix(median_house_value ~ 0 + ., data = df_test %>% select(-households)),
      df_test %>% select(median_house_value) %>% as.matrix
    ),
    epochs = 50,
    dataloader_options = list(batch_size = 1024, shuffle = TRUE),
    verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting

  )
```

Plot the results of the training and validation loss and accuracy.

```
# plotting the results
plot(nnet_fit)
```



Report the root mean squared error on the test set.

```
# making predictions and converting into an array
nnet_predictions <- predict(nnet_fit, model.matrix(median_house_value ~ 0 + ., df_test %>%

# calculating the RMSE
nnet_rmse <- rmse(df_test$median_house_value, nnet_predictions)
nnet_rmse
```

```
[1] 68621.42
```

> **Warning**
>
> Remember to use the `as_array()` function to convert the predictions to a vector of numbers before computing the RMSE with `rmse()`

1.9 (5 points)

Summarize your results in a table comparing the RMSE for the different models. Which model performed best? Why do you think that is?

```
# creating the data frame of all the different models' rmse
rmse_summary <- data.frame(model_type = c("lm", "rpart", "svm", "nnet"),
                           rmse_value = (c(lm_rmse, rpart_rmse, svm_rmse, nnet_rmse)))
rmse_summary
```

```
  model_type rmse_value
1         lm   68339.82
2      rpart   75876.87
3        svm   56678.84
4       nnet   68621.42
```

Question 2

> **50 points**
>
> Spam email classification

The `data` folder contains the `spam.csv` dataset. This dataset contains features extracted from a collection of spam and non-spam emails. The objective is to classify the emails as spam or non-spam.

2.1 (2.5 points)

Read the data file as a tibble in R. Preprocess the data such that:

1. the variables are of the right data type, e.g., categorical variables are encoded as factors
2. all column names to lower case for consistency
3. Any observations with missing values are dropped

```
# reading the csv
df <- read_csv("data/spambase.csv")
```

```
Rows: 4601 Columns: 58
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (58): word_freq_1, word_freq_2, word_freq_3, word_freq_4, word_freq_5, w...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# converting 'spam' to a factor and dropping NA values
df <- df %>%
  mutate(spam = as.factor(spam)) %>%
  drop_na()
```

2.2 (2.5 points)

Split the data `df` into `df_train` and `df_split` using `test_ind` in the code below:

```
set.seed(42)
test_ind <- sample(
  1:nrow(df),
  floor( nrow(df)/10 ),
  replace=FALSE
)

# splitting the data
df_train <- df[-test_ind, ]
df_test  <- df[test_ind, ]
```

Complete the `overview` function which returns a data frame with the following columns: `accuracy`, `error`, `false positive rate`, `true positive rate`, between the true `true_class` and the predicted `pred_class` for any classification model.

```
overview <- function(pred_class, true_class) {
  true_positives <- sum(pred_class == 1 & true_class == 1)
  true_negatives <- sum(pred_class == 0 & true_class == 0)
  false_positives <- sum(pred_class == 1 & true_class == 0)
  false_negatives <- sum(pred_class == 0 & true_class == 1)
  true_positive_rate <- true_positives / (true_positives + false_negatives)
  false_positive_rate <- false_positives / (false_positives + true_negatives)
  accuracy <- mean(pred_class == true_class)
  error <- 1-accuracy
  return(
    data.frame(
      accuracy = accuracy,
      error = error,
      true_positive_rate = true_positive_rate,
      false_positive_rate = false_positive_rate
    )
  )
}
```

2.3 (5 points)

Fit a logistic regression model to predict the `spam` variable using the remaining predictors. Report the prediction accuracy on the test set.

```
# fitting the logistic reg model using glm()
glm_fit <- glm(spam ~ ., df_train, family = binomial())

# making predictions
glm_predictions <- predict(glm_fit, newdata = df_test)

# converting the probabilities to a binary value
glm_classes <- ifelse(glm_predictions > 0.5, 1, 0)

# using the overview function to find the accuracy
glm_overview <- overview(glm_classes, df_test$spam)
```

```
print(paste("The prediction accuracy on the test set is", glm_overview$accuracy))
```

```
[1] "The prediction accuracy on the test set is 0.910869565217391"
```

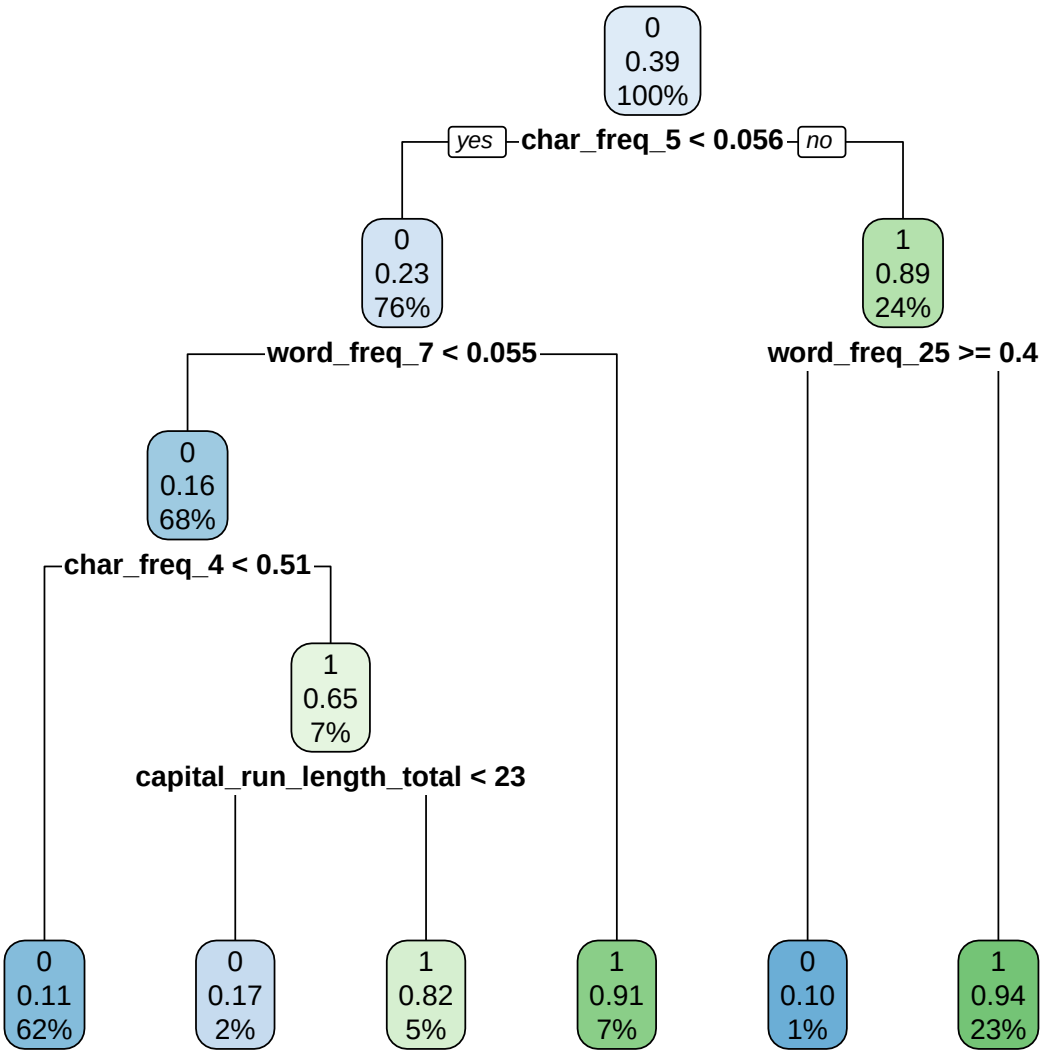
2.4 (5 points)

Fit a decision tree model to predict the `spam` variable using the remaining predictors. Use the `rpart()` function and set the `method` argument to `"class"`.

```
# fitting the decision tree
rpart_fit2 <- rpart(spam ~ ., df_train, method = "class")

# making the porbability predictions
rpart_probabilities <- predict(rpart_fit2, newdata = df_test)

# converting the probabilities to binary vales
rpart_classes <- ifelse(rpart_probabilities > 0.5, 1, 0)
```

Visualize the decision tree using the `rpart.plot()` function.

```
# plotting the decision tree
rpart.plot(rpart_fit2)
```



Report the prediction accuracy on the test set.

```
# converting the probabilities to binary classes
rpart_classes <- ifelse(rpart_probabilities > 0.5, 1, 0)

# using the overview functions to find the accuracy
rpart_overview <- overview(rpart_classes, df_test$spam)
```

```r
print(paste("The prediction accuracy on the test set is", rpart_overview$accuracy))
```

```
[1] "The prediction accuracy on the test set is 0.5"
```

2.5 (5 points)

Fit a support vector machine model to predict the `spam` variable using the remaining predictors. Use the `svm()` function and use any kernel of your choice. Remember to set the `type` argument to `"C-classification"` **if you haven't** already converted `spam` to be of type `factor`.

```r
# fitting the svm model
svm_fit <- svm(spam ~ ., df_train, kernel = "radial", type = "C-classification")
```

Report the prediction accuracy on the test set.

```r
# making the predictions
svm_classes <- predict(svm_fit, newdata = df_test)

# using the overview function to find the accuracy
svm_overview <- overview(svm_classes, df_test$spam)

print(paste("The prediction accuracy on the test set is", svm_overview$accuracy))
```

```
[1] "The prediction accuracy on the test set is 0.923913043478261"
```

2.6 (25 points)

Using the same neural network architecture as in 1.9, fit a neural network model to predict the `spam` variable using the remaining predictors.

> ⚠ Classification vs. Regression
>
> Note that the neural network in **Q 1.9** was a regression model. You will need to modify the neural network architecture to be a classification model by changing the output layer to have a single node with a sigmoid activation function.

```
# creating the neural network
NNet <- nn_module(
    initialize = function(p, q1, q2, q3){
      # initializing the architecture
      self$hidden1 <- nn_linear(p, q1)
      self$hidden2 <- nn_linear(q1, q2)
      self$hidden3 <- nn_linear(q2, q3)
      self$output <- nn_linear(q3, 1)
      self$activation <- nn_relu()
      self$sigmoid <- nn_sigmoid()
    },
    forward = function(x){
      # creating the forward function
      x %>%
        self$hidden1() %>%
        self$activation() %>%
        self$hidden2() %>%
        self$activation() %>%
        self$hidden3() %>%
        self$activation() %>%
        self$output() %>%
        self$sigmoid()
    }
)
```

Use the `model.matrix` function to create the covariate matrix and `luz` package for fitting the network with $32, 16, 8$ nodes in each of the three hidden layers.

```
# creating the covariate matrix
M <- model.matrix(spam ~ 0 + ., df_train)

# will have to use binary cross entropy loss
nnet_fit <- NNet %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(
    p=ncol(M), q1 = 32, q2 = 16, q3 = 8
  ) %>%
  set_opt_hparams(
    lr = 0.002
```

```
  ) %>%
  fit(
    data = list(
      model.matrix(spam ~ 0 + ., data = df_train),
      (df_train[["spam"]] %>% as.numeric() - 1) %>% as.matrix
    ),
    valid_data = list(
      model.matrix(spam ~ 0 + ., data = df_test),
      (df_test[["spam"]] %>% as.numeric() - 1) %>% as.matrix
    ),
    epochs = 100,
    dataloader_options = list(batch_size = 1024, shuffle = TRUE),
    verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting

  )
```

```
# making predictions
nnet_pred <- predict(nnet_fit, model.matrix(spam ~ 0 + ., df_test))

nnet_classes <- ifelse(nnet_pred > 0.5, 1, 0)

# finding the test accuracy
nnet_overview <- overview(nnet_classes, df_test$spam)

print(paste("The accuracy on the test set is", nnet_overview$accuracy))
```

```
[1] "The accuracy on the test set is 0.908695652173913"
```

2.7 (5 points)

Summarize your results in a table comparing the accuracy metrics for the different models.

```
# creating a data frame to display a summary of the accuracy
accuracy_summary <- data.frame(model_type = c("glm", "rpart", "svm", "nnet"),
                               accuracy_value = c(glm_overview$accuracy, rpart_overview$ac

accuracy_summary
```

```
  model_type accuracy_value
1        glm      0.9108696
2      rpart      0.5000000
3        svm      0.9239130
4       nnet      0.9086957
```

If you were to choose a model to classify spam emails, which model would you choose? Think about the context of the problem and the cost of false positives and false negatives.

Question 3

60 points

Three spirals classification

To better illustrate the power of depth in neural networks, we will use a toy dataset called the "Three Spirals" data. This dataset consists of two intertwined spirals, making it challenging for shallow models to classify the data accurately.

This is a multi-class classification problem

The dataset can be generated using the provided R code below:

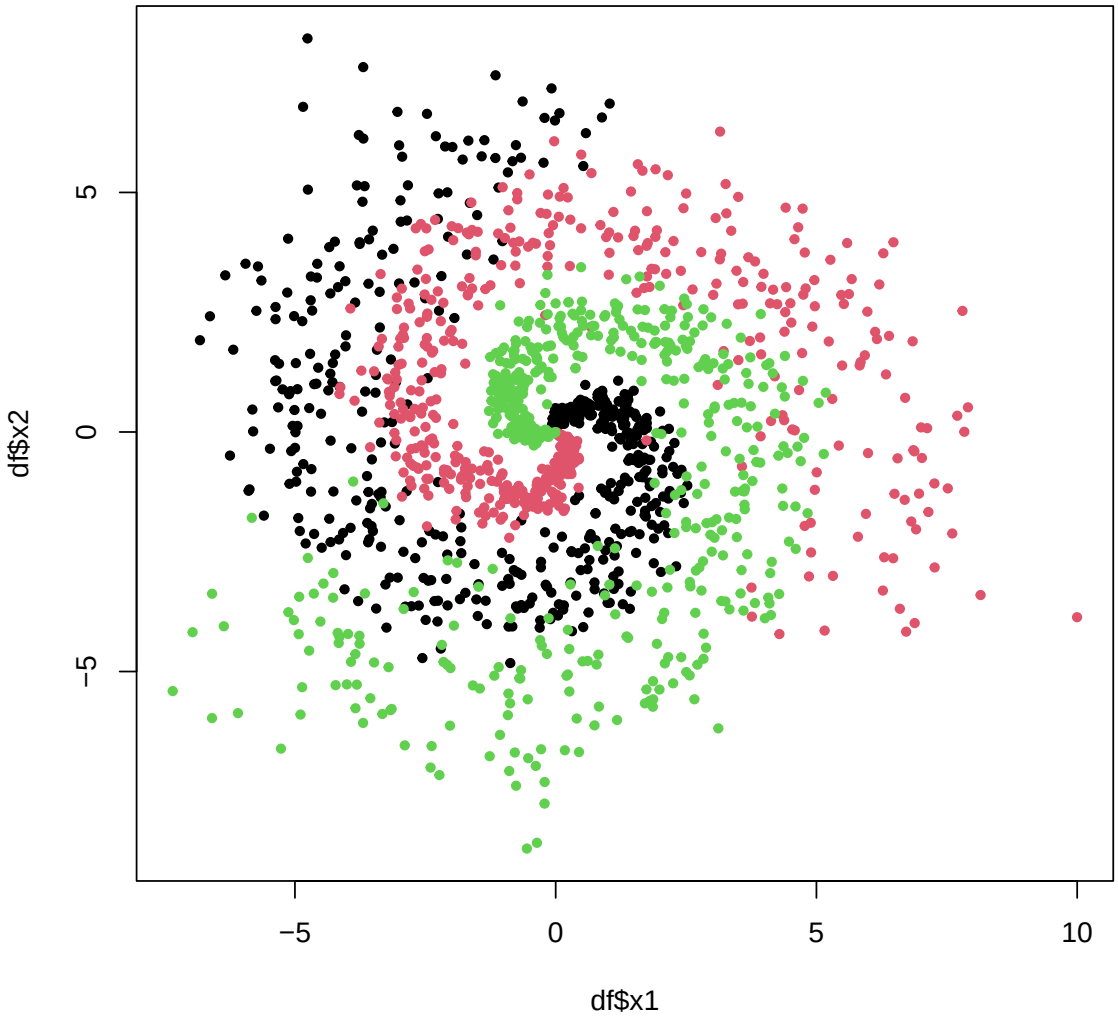
```r
generate_three_spirals <- function(){
  set.seed(42)
  n <- 500
  noise <- 0.2
  t <- (1:n) / n * 2 * pi
  x1 <- c(
      t * (sin(t) + rnorm(n, 0, noise)),
      t * (sin(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (sin(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  x2 <- c(
      t * (cos(t) + rnorm(n, 0, noise)),
      t * (cos(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (cos(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
```

```
  y <- as.factor(
    c(
      rep(0, n),
      rep(1, n),
      rep(2, n)
    )
  )
  return(tibble(x1=x1, x2=x2, y=y))
}
```

3.1 (5 points)

Generate the three spirals dataset using the code above. Plot x_1 vs x_2 and use the `y` variable to color the points.

```
df <- generate_three_spirals()

plot(
  df$x1, df$x2,
  col = df$y,
  pch = 20
)
```



Define a grid of 100 points from -10 to 10 in both x_1 and x_2 using the `expand.grid()`. Save it as a tibble called `df_test`.

```
# defining the grid
grid <- expand.grid(x1 = seq(-10, 10, length.out = 100),
                    x2 = seq(-10, 10, length.out = 100))
# saving the grid as a tibble
```

```r
df_test <- as_tibble(grid)
```

3.2 (10 points)

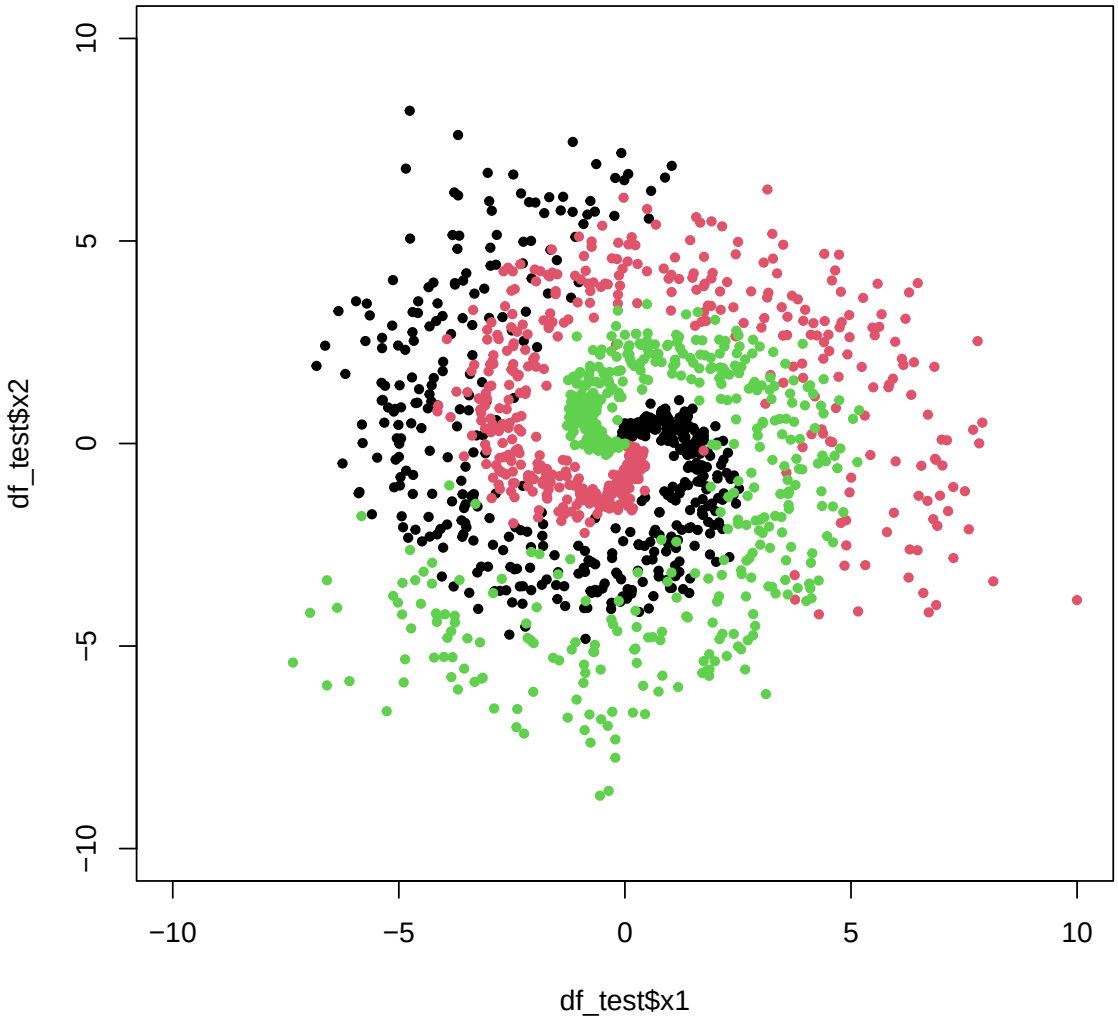
Fit a classification tree model to predict the `y` variable using the `x1` and `x2` predictors, and plot the decision boundary.

```r
# fitting the decision tree with response and specified predictors
rpart_fit <- rpart(y ~ ., df)
# making predictions on the testing data
rpart_classes <- predict(rpart_fit, newdata = df_test)
```

Plot the decision boundary using the following function:

```r
plot_decision_boundary <- function(predictions){
  plot(
    df_test$x1, df_test$x2,
    col = predictions,
    pch = 0
  )
  points(
    df$x1, df$x2,
    col = df$y,
    pch = 20
  )
}
```

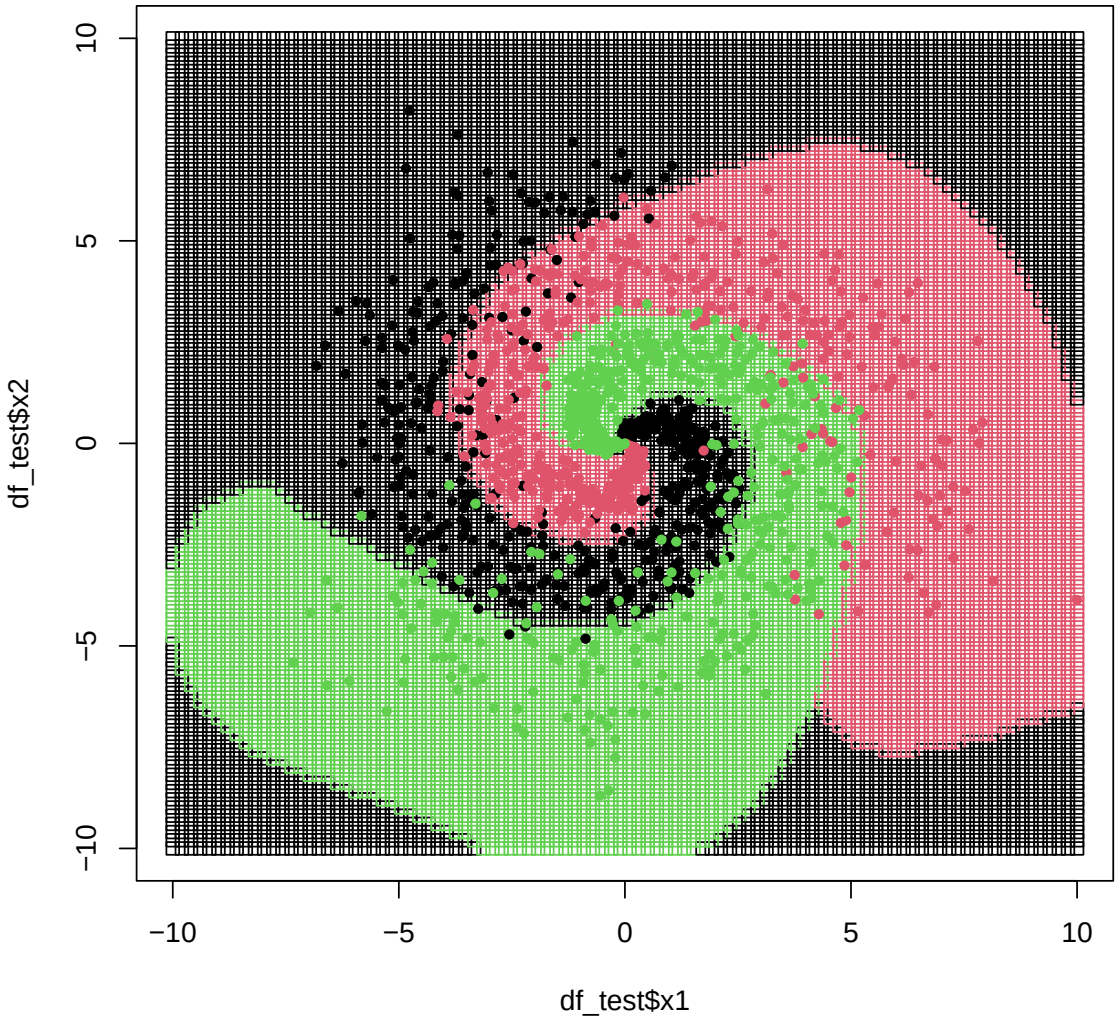
```r
plot_decision_boundary(rpart_classes)
```



3.3 (10 points)

Fit a support vector machine model to predict the `y` variable using the `x1` and `x2` predictors. Use the `svm()` function and use any kernel of your choice. Remember to set the `type` argument to `"C-classification"` **if you haven't** converted `y` to be of type `factor`.

```r
# fitting the model and predicting the data
svm_fit <- svm(y ~ ., df, kernel = "radial", type = "C-classification")
svm_classes <- predict(svm_fit, newdata = df_test)
plot_decision_boundary(svm_classes)
```



Instructions

For the next questions, you will need to fit a series of neural networks. In all cases, you can:

- set the number of units in each hidden layer to 10
- set the output dimension `o` to 3 (remember this is multinomial classification)
- use the appropriate loss function for the problem (**not `nn_bce_loss`**)
- set the number of epochs to 50
- fit the model using the `luz` package

You can use any optimizer of your choice, but you **will need to tune the learning rate for each problem**.

3.4 (10 points)

Fit a neural network with **1 hidden layer** to predict the `y` variable using the `x1` and `x2` predictors.

```
NN1 <- nn_module(
  initialize = function(p, q1, o){
    self$hidden1 <- nn_linear(p, q1)
    self$output <- nn_linear(q1, o)
    self$activation <- nn_relu()
  },
  forward = function(x){
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$output()
  }
)

fit_1 <- NN1 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(
    p=ncol(df_test), q1 = 10, o = 3
  ) %>%
  set_opt_hparams(
    lr = 0.02
```

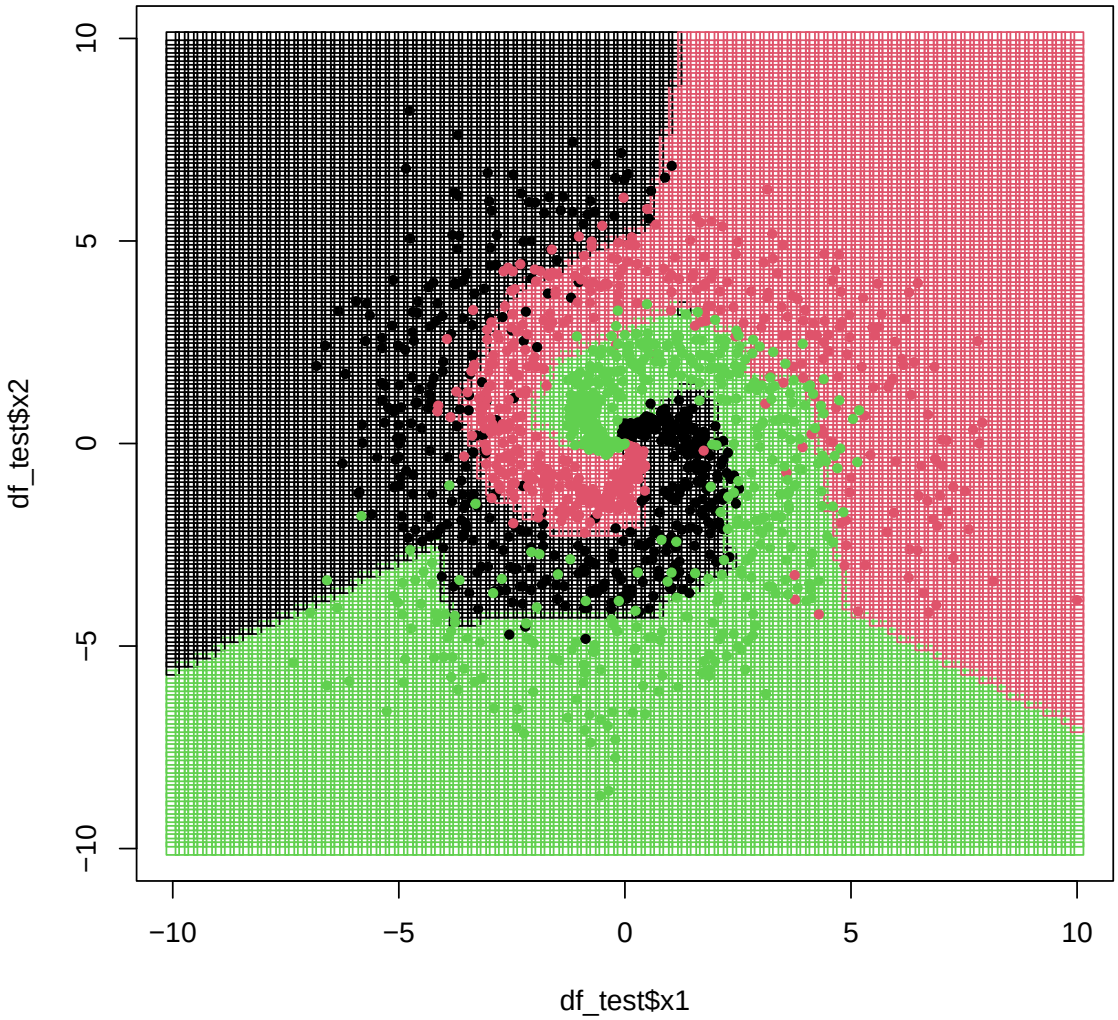
```
  ) %>%
  fit(
    data = list(
      df %>% select(x1, x2) %>% as.matrix,
      df$y %>% as.integer
    ),
    epochs = 50,
    dataloader_options = list(batch_size = 100, shuffle = TRUE),
    verbose = FALSE
  )
```

In order to generate the class predictions, you will need to use the `predict()` function as follows

```
test_matrix <- df_test %>% select(x1, x2) %>% as.matrix

fit_1_predictions <- predict(fit_1, test_matrix) %>%
  torch_argmax(2) %>%
  as.integer()
```

Plot the results using the `plot_decision_boundary()` function.

```
plot_decision_boundary(fit_1_predictions)
```



3.5 (10 points)

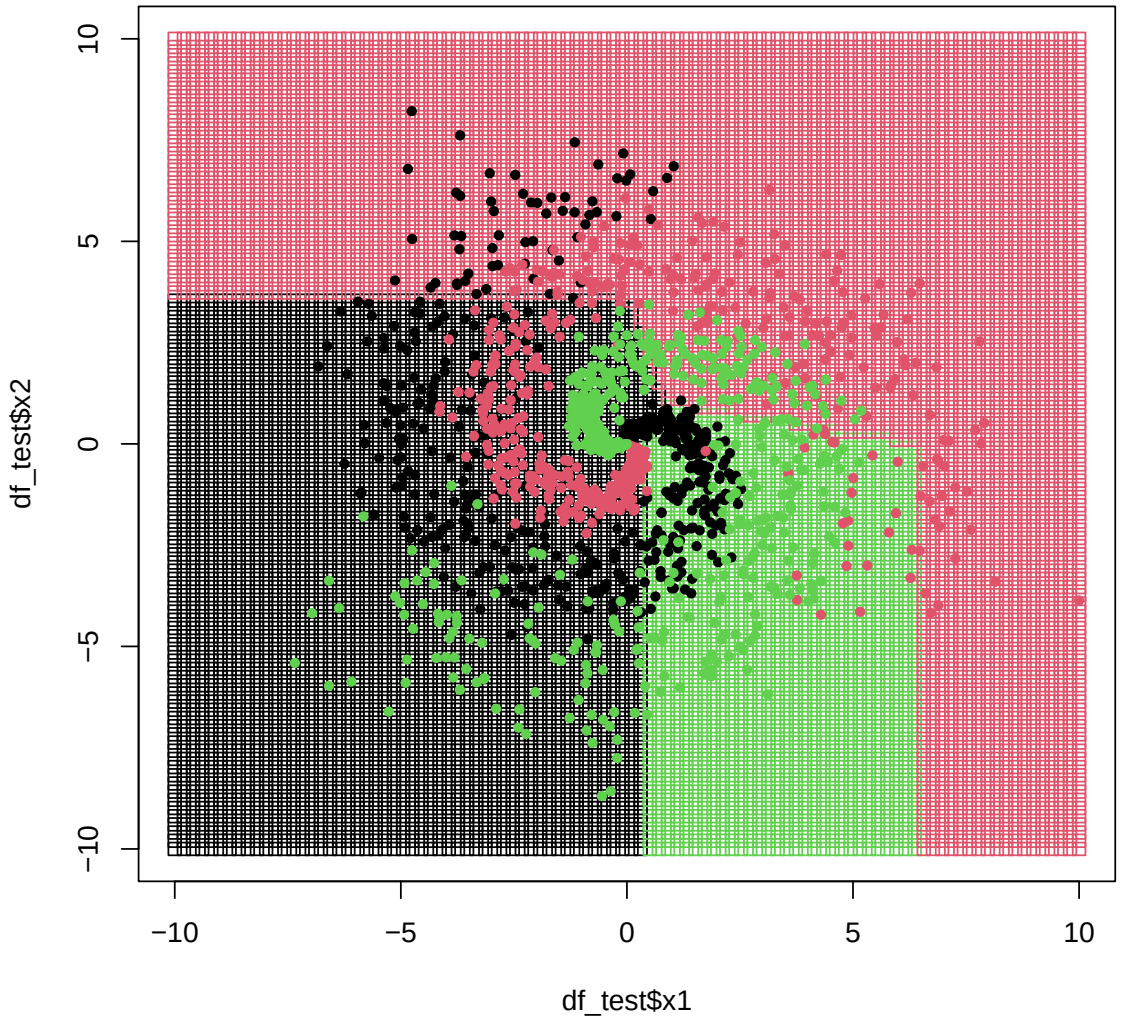
Fit a neural network with **0 hidden layers** to predict the `y` variable using the `x1` and `x2` predictors.

```
NN0 <- nn_module(
  initialize = function(p, o){
    self$output <- nn_linear(p, o)
    self$activation <- nn_relu()
  },
  forward = function(x){
    x %>%
    self$activation() %>%
    self$output()
  }
)

fit_0 <- NN0 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(
    p=ncol(df_test), o = 3
  ) %>%
  set_opt_hparams(
    lr = 0.002
  ) %>%
  fit(
    data = list(
      df %>% select(x1, x2) %>% as.matrix,
      df$y %>% as.integer
    ),,
    epochs = 50,
    dataloader_options = list(batch_size = 100, shuffle = TRUE),
    verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting
  )
```

Plot the results using the `plot_decision_boundary()` function.

```
test_matrix <- df_test %>% select(x1, x2) %>% as.matrix

fit_0_predictions <- predict(fit_0, test_matrix) %>%
  torch_argmax(2) %>%
  as.integer()
```

```
plot_decision_boundary(fit_0_predictions)
```



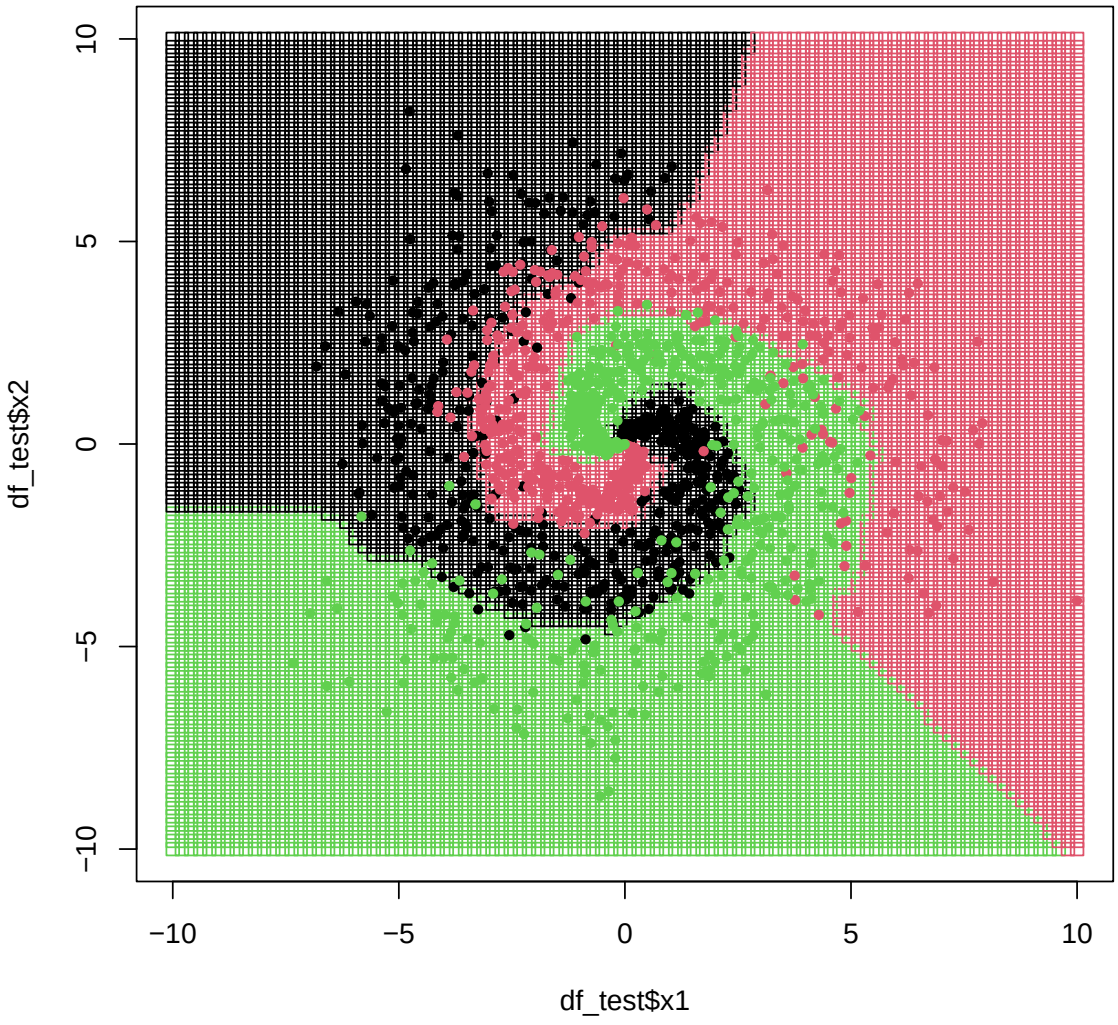
3.6 (10 points)

Fit a neural network with **3 hidden layers** to predict the `y` variable using the `x1` and `x2` predictors.

```
NN3 <- nn_module(
  initialize = function(p, q1, q2, o){
    self$hidden1 <- nn_linear(p, q1)
    self$hidden2 <- nn_linear(q1, q2)
    self$output <- nn_linear(q2, o)
    self$activation <- nn_relu()
  },
  forward = function(x){
    x %>%
    self$hidden1() %>%
    self$activation() %>%
    self$hidden2() %>%
    self$activation() %>%
    self$output()
  }
)

fit_2 <- NN3 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(
    p=ncol(df_test), q1=10, q2=10, o=3
  ) %>%
  set_opt_hparams(
    lr = 0.02
  ) %>%
  fit(
    data = list(
      df %>% select(x1, x2) %>% as.matrix,
      df$y %>% as.integer
    ),
    epochs = 50,
    dataloader_options = list(batch_size = 100, shuffle = TRUE),
    verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting
  )
```

Plot the results using the `plot_decision_boundary()` function.

```
test_matrix <- df_test %>% select(x1, x2) %>% as.matrix

fit_2_predictions <- predict(fit_2, test_matrix) %>%
  torch_argmax(2) %>%
  as.integer()

plot_decision_boundary(fit_2_predictions)
```



3.7 (5 points)

What are the differences between the models? How do the decision boundaries change as the number of hidden layers increases?

The models are practically identical other than the number of hidden layers present and of course the chosen learning rates for each. Ultimately, the number of hidden layers in the models drastically changes the loss gained. I found that the model with 3 hidden layers performed much better than the model with 0 hidden layers and slightly better than the model with 1 hidden layer. As the number of hidden layers increases, the decision boundaries become more and more accurate. In other words, there are less incorrectly colored data points in the decision boundaries.

Session Information

Print your R session information using the following command

```
sessionInfo()
```

```
R version 4.2.1 (2022-06-23 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 22000)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
 [1] luz_0.3.1        torch_0.9.0      e1071_1.7-13     rpart.plot_3.1.1
 [5] rpart_4.1.19     caret_6.0-94     lattice_0.20-45  ggplot2_3.4.1
 [9] corrplot_0.92    magrittr_2.0.3   broom_1.0.4      purrr_1.0.1
[13] tidyr_1.3.0      readr_2.1.4      dplyr_1.1.0

loaded via a namespace (and not attached):
 [1] nlme_3.1-162         fs_1.6.1             lubridate_1.9.2
 [4] bit64_4.0.5          progress_1.2.2       tools_4.2.1
 [7] backports_1.4.1      utf8_1.2.3           R6_2.5.1
[10] colorspace_2.1-0     nnet_7.3-18          withr_2.5.0
[13] tidyselect_1.2.0     prettyunits_1.1.1    processx_3.8.0
[16] bit_4.0.5            compiler_4.2.1       cli_3.6.0
[19] labeling_0.4.2       scales_1.2.1         callr_3.7.3
[22] proxy_0.4-27         stringr_1.5.0        digest_0.6.31
[25] rmarkdown_2.21       coro_1.0.3           pkgconfig_2.0.3
[28] htmltools_0.5.5      parallelly_1.35.0    fastmap_1.1.1
[31] rlang_1.0.6          farver_2.1.1         generics_0.1.3
```

```
[34] jsonlite_1.8.4       vroom_1.6.1          ModelMetrics_1.2.2.2
[37] Matrix_1.5-3         Rcpp_1.0.10          munsell_0.5.0
[40] fansi_1.0.4          lifecycle_1.0.3      stringi_1.7.12
[43] pROC_1.18.0          yaml_2.3.7           MASS_7.3-58.3
[46] plyr_1.8.8           recipes_1.0.5        grid_4.2.1
[49] parallel_4.2.1       listenv_0.9.0        crayon_1.5.2
[52] splines_4.2.1        hms_1.1.2            zeallot_0.1.0
[55] knitr_1.42           ps_1.7.2             pillar_1.8.1
[58] future.apply_1.10.0  reshape2_1.4.4       codetools_0.2-19
[61] stats4_4.2.1         glue_1.6.2           evaluate_0.20
[64] data.table_1.14.8    renv_0.16.0-53       vctrs_0.5.2
[67] tzdb_0.3.0           foreach_1.5.2        gtable_0.3.1
[70] future_1.32.0        xfun_0.38            gower_1.0.1
[73] prodlim_2019.11.13   class_7.3-21         survival_3.5-5
[76] timeDate_4022.108    tibble_3.1.8         iterators_1.0.14
[79] hardhat_1.2.0        lava_1.7.2.1         timechange_0.2.0
[82] globals_0.16.2       ellipsis_0.3.2       ipred_0.9-14
```