Homework 5

Kate Miller

Table of contents

❗ Important

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:

```r
packages <- c(
  "tibble",
  "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

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

df <- read_csv(path) %>%
  mutate_if(is.character, as.factor) %>%
  rename_all(tolower) %>%
  drop_na()
```

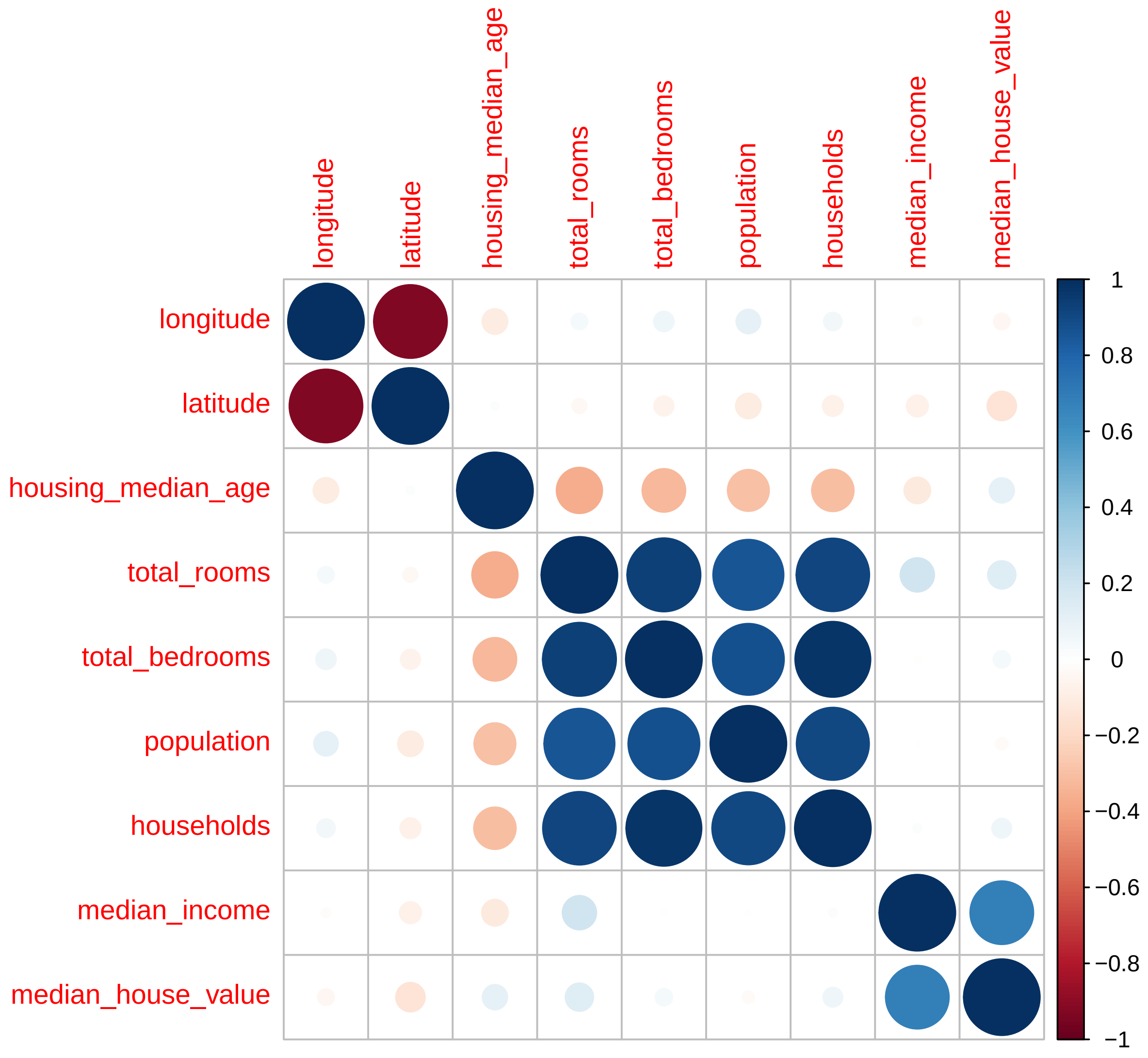
```
Rows: 20640 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): ocean_proximity
dbl (9): longitude, latitude, housing_median_age, total_rooms, total_bedroom...

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.
```

1.2 (2.5 points)

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

```r
corrplot(cor(df %>% select_if(is.numeric)), method = "circle")
```



```
# I could not determine how to do it with the beginning code being df %>%, so I chose to d
```

1.3 (5 points)

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

```r
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, ]

glimpse(df_train)
```

```
Rows: 18,390
Columns: 10
$ longitude          <dbl> -122.23, -122.24, -122.25, -122.25, -122.25, -122.2~
$ latitude           <dbl> 37.88, 37.85, 37.85, 37.85, 37.85, 37.84, 37.84, 37~
$ housing_median_age <dbl> 41, 52, 52, 52, 52, 52, 52, 42, 52, 52, 52, 52, 52,~
$ total_rooms        <dbl> 880, 1467, 1274, 1627, 919, 2535, 3104, 2555, 3549,~
$ total_bedrooms     <dbl> 129, 190, 235, 280, 213, 489, 687, 665, 707, 434, 7~
$ population         <dbl> 322, 496, 558, 565, 413, 1094, 1157, 1206, 1551, 91~
$ households         <dbl> 126, 177, 219, 259, 193, 514, 647, 595, 714, 402, 7~
$ median_income      <dbl> 8.3252, 7.2574, 5.6431, 3.8462, 4.0368, 3.6591, 3.1~
$ median_house_value <dbl> 452600, 352100, 341300, 342200, 269700, 299200, 241~
$ ocean_proximity    <fct> NEAR BAY, NEAR BAY, NEAR BAY, NEAR BAY, NEAR BAY, N~
```

```r
glimpse(df_test)
```

```
Rows: 2,043
Columns: 10
$ longitude          <dbl> -122.15, -122.47, -121.63, -122.41, -118.46, -117.6~
$ latitude           <dbl> 38.29, 37.87, 38.03, 37.79, 34.02, 33.59, 34.16, 32~
$ housing_median_age <dbl> 17, 36, 17, 52, 29, 8, 38, 16, 52, 41, 27, 23, 34, ~
$ total_rooms        <dbl> 1625, 4471, 2549, 3610, 2329, 2649, 2458, 1675, 249~
$ total_bedrooms     <dbl> 239, 618, 596, 1286, 833, 340, 488, 354, 458, 129, ~
$ population         <dbl> 703, 1315, 1169, 1504, 1953, 1238, 1135, 604, 1081,~
$ households         <dbl> 224, 582, 500, 1047, 800, 354, 453, 332, 471, 125, ~
$ median_income      <dbl> 6.5891, 11.5706, 3.6694, 3.2059, 2.6639, 8.0409, 2.~
$ median_house_value <dbl> 328800, 500001, 209400, 500001, 233300, 337900, 991~
$ ocean_proximity    <fct> NEAR BAY, NEAR BAY, INLAND, NEAR BAY, <1H OCEAN, <1~
```

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.

```r
lm_fit <- lm(median_house_value ~ latitude + longitude + housing_median_age +
               total_rooms + total_bedrooms + population + median_income +
               ocean_proximity, data = df_train)
summary(lm_fit)
```

```

Call:
lm(formula = median_house_value ~ latitude + longitude + housing_median_age +
    total_rooms + total_bedrooms + population + median_income +
    ocean_proximity, data = df_train)

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 ***
latitude                  -2.539e+04  1.047e+03 -24.244  < 2e-16 ***
longitude                 -2.681e+04  1.060e+03 -25.305  < 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
```

A one-unit increase in latitude is associated with a decrease in median household value of about 25,390 dollars. A one-unit increase in longitude corresponds with an approximate decrease of 26,810 dollars in median household value. A one-year increase in house median age is associated with an increase of 1,074 dollars in median household value. For every one-unit increase in total rooms, there is an estimated decrease of approximately 6 dollars in median house value. For every additional bedroom, there is an approximate increase of 135 dollars in median house value. For every additional unit of population, there is a decrease in median house value by about 34 dollars.

In summary, a higher income may mean a higher median household value. If there are more people in a given location, the value of the house decreases. In addition, location of the house plays a role in its value. If a house is inland from an ocean, it will likely decrease in value, whereas if it is an island, it will increase significantly in value of about 132,000 dollars. Lastly, additional bedrooms will increase the median household value.

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))
}

lm_predictions <- predict(lm_fit, newdata = df_test)

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.

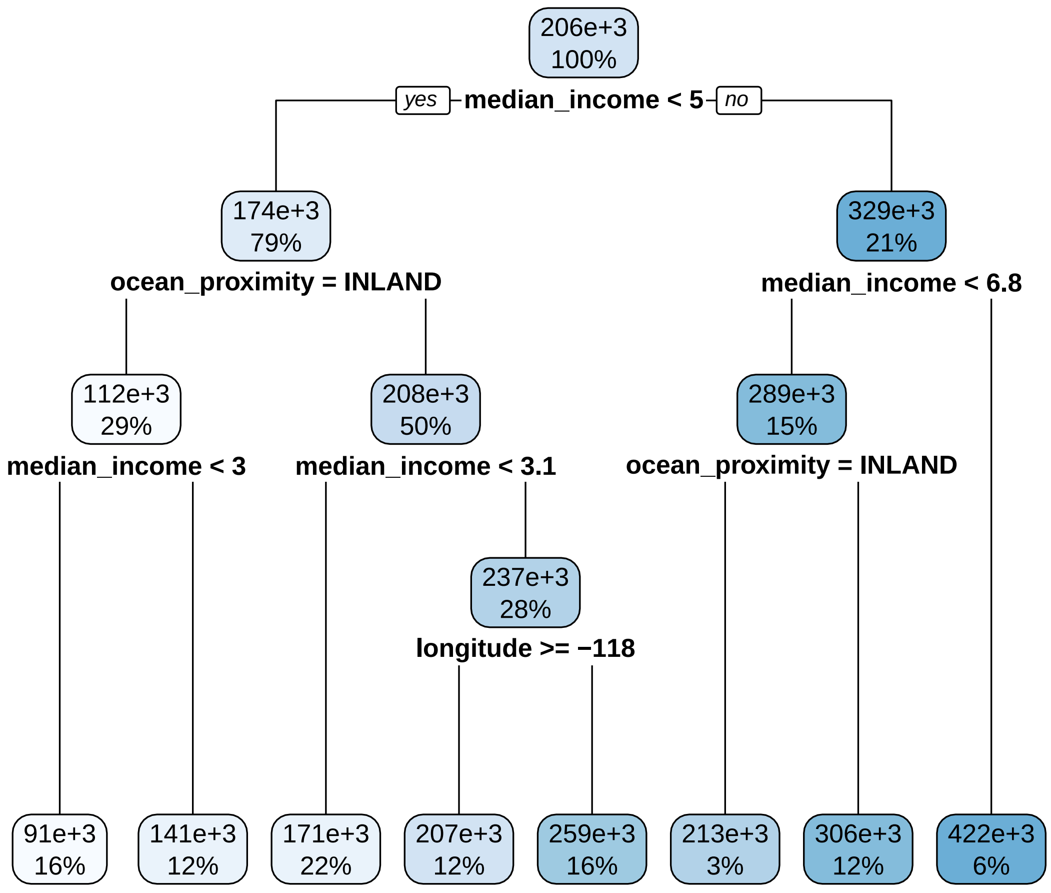
```
rpart_fit <- rpart(median_house_value ~ latitude + longitude + housing_median_age +
                      total_rooms + total_bedrooms + population + median_income +
                      ocean_proximity, data = df_train)

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

```
Named num [1:2043] 305561 422430 141012 258810 171464 ...
- attr(*, "names")= chr [1:2043] "1" "2" "3" "4" ...
```

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

```
library(rpart)
library(rpart.plot)
rpart.plot(rpart_fit)
```



Report the root mean squared error on the test set.

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

```r
library("e1071")
svm_fit <- svm(median_house_value ~ latitude + longitude + housing_median_age +
                  total_rooms + total_bedrooms + population + median_income +
                  ocean_proximity, data = df_train, kernel = "radial")
svm_predictions <- predict(svm_fit, newdata = df_test)
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:

```r
#library(torch)
#NNet <- nn_module(
   # initialize = function(p, q1, q2, q3){
   #   self$input_to_hidden1 <- nn_linear(p, q1)
   #   self$hidden1_to_hidden2 <- nn_linear(q1, q2)
  #    self$hidden2_to_output <- nn_linear(q2, 1)
  #    self$activation <- nn_relu()
  #    self$sigmoid <- nn_sigmoid()

   # },
   # forward = function(x, weights, biases){
  #    x %>%
   #   self$input_to_hidden1() %>%
   #   self$activation() %>%
    #  self$hidden1_to_hidden2() %>%
   #   self$activation() %>%
    #  self$hidden2_to_output()
    #}
#)
#NNet
```

I could not get my neural network to run, so above is the hypothetical code I would be using.

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.

```
#X_train <- model.matrix(~ latitude + longitude + housing_median_age +
  #                       total_rooms + total_bedrooms + population + #median_income + oce

#y_train <- df_train$median_house_value
#library(luz)
#nnet_fit <- NNet %>%
#  setup(
#    loss = "mean_squared_error",
#    optimizer = "adam"
#  ) %>%
#  set_hparams(
#    p = ncol(X_train),
#    q1 = 32,
#    q2 = 16,
#    q3 = 8
#  ) %>%
#  set_opt_hparams(
#    lr = 0.001
#  ) %>%
#  fit(
#    X_train,
#    y_train,
#    dataloader_options = list(batch_size = 64),
#    verbose = FALSE
#  )
```

My code for the above problem was not running, but I commented out my code so that you can see what I would hypothetically be doing.

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

```
#history <- nnet_fit$history

#plot(history$loss, type = "l", col = "blue", xlab = "Epoch", ylab = "Loss", main = "Train
#lines(history$val_loss, type = "l", col = "red")
#legend("topright", legend = c("Training Loss", "Validation Loss"), col = c("blue", "red")

#plot(history$accuracy, type = "l", col = "blue", xlab = "Epoch", ylab = "Accuracy", main
#lines(history$val_accuracy, type = "l", col = "red")
#legend("topright", legend = c("Training Accuracy", "Validation Accuracy"), col = c("blue"
```

Given no error with the neural network code chunk, this code chunk would have ran to create the plot.

Report the root mean squared error on the test set.

Again, if my neural network code chunk would have ran, this code chunk would have also ran for me to output the rmse with no error.

```
#nnet_predictions <- predict(nnet_fit, X_test) %>% as_array()

#nnet_rmse <- rmse(y_test, nnet_predictions)
#nnet_rmse
```

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?

```
#lm_rmse <- rmse(df_test$median_house_value, lm_predictions)
#rpart_rmse <- rmse(df_test$median_house_value, rpart_predictions)
#svm_rmse <- rmse(df_test$median_house_value, svm_predictions)
#nnet_rmse <- rmse(df_test$median_house_value, nnet_predictions)

#results <- data.frame(Model = c("Linear Regression", "Decision Tree", "Support Vector Mac
                    #  RMSE = c(lm_rmse, rpart_rmse, svm_rmse, nnet_rmse))
#library(knitr)

#kable(results)
```

Again, if the neural network code chunk would have ran, this would have ran since I would have had the nnet_rmse predictions.

Since I do not have the values of the root mean square error, I do not know which model will perform the best. However, the model that performs the best will be the one with the lowest mean square error.

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

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.0     v stringr   1.5.1
v lubridate 1.9.3
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x magrittr::extract()   masks tidyr::extract()
x dplyr::filter()       masks stats::filter()
x dplyr::lag()          masks stats::lag()
x caret::lift()         masks purrr::lift()
x magrittr::set_names() masks purrr::set_names()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

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

df <- read_csv(path) %>%
  mutate_if(is.character, as.factor) %>%
  rename_all(tolower) %>%
  drop_na()
```

```
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.
```

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
)

df_train <- df[-test_ind, ]
df_test  <- df[test_ind, ]
head(df_train)
```

```
# A tibble: 6 x 58
  word_freq_1 word_freq_2 word_freq_3 word_freq_4 word_freq_5 word_freq_6
        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
1        0           0.64        0.64           0        0.32        0
2        0.06        0           0.71           0        1.23        0.19
3        0           0           0              0        0.63        0
4        0           0           0              0        0.63        0
5        0           0           0              0        1.85        0
6        0           0           0              0        1.92        0
```

```
# i 52 more variables: word_freq_7 <dbl>, word_freq_8 <dbl>, word_freq_9 <dbl>,
#   word_freq_10 <dbl>, word_freq_11 <dbl>, word_freq_12 <dbl>,
#   word_freq_13 <dbl>, word_freq_14 <dbl>, word_freq_15 <dbl>,
#   word_freq_16 <dbl>, word_freq_17 <dbl>, word_freq_18 <dbl>,
#   word_freq_19 <dbl>, word_freq_20 <dbl>, word_freq_21 <dbl>,
#   word_freq_22 <dbl>, word_freq_23 <dbl>, word_freq_24 <dbl>,
#   word_freq_25 <dbl>, word_freq_26 <dbl>, word_freq_27 <dbl>, ...
```

```r
head(df_test)
```

```
# A tibble: 6 x 58
  word_freq_1 word_freq_2 word_freq_3 word_freq_4 word_freq_5 word_freq_6
        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
1           0           0        0              0        0              0
2           0           0        0              0        0              0
3           0           0        0              0        0.77           0
4           0           0        0.4            0        0              0
5           0           0        0.72           0        0.72           0
6           0           0        0              0        0              0
# i 52 more variables: word_freq_7 <dbl>, word_freq_8 <dbl>, word_freq_9 <dbl>,
#   word_freq_10 <dbl>, word_freq_11 <dbl>, word_freq_12 <dbl>,
#   word_freq_13 <dbl>, word_freq_14 <dbl>, word_freq_15 <dbl>,
#   word_freq_16 <dbl>, word_freq_17 <dbl>, word_freq_18 <dbl>,
#   word_freq_19 <dbl>, word_freq_20 <dbl>, word_freq_21 <dbl>,
#   word_freq_22 <dbl>, word_freq_23 <dbl>, word_freq_24 <dbl>,
#   word_freq_25 <dbl>, word_freq_26 <dbl>, word_freq_27 <dbl>, ...
```

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.

```r
overview <- function(pred_class, true_class) {
  accuracy <- mean(pred_class == true_class)
  error <- 1 - accuracy
  confusion <- table(pred_class, true_class)
  true_positives <- confusion["spam", "spam"]
  true_negatives <- confusion["nonspam", "nonspam"]
  false_positives <- confusion["spam", "nonspam"]
  false_negatives <- confusion["nonspam", "spam"]
  true_positive_rate <-  true_positives / (true_positives + false_negatives)
  false_positive_rate <- false_positives / (false_positives + true_negatives)
  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.

```r
glm_fit <- glm(spam ~ ., data = df_train, family = binomial)
glm_classes <- predict(glm_fit, newdata = df_test, type = "response") > 0.5
prediction_accuracy <- mean(glm_classes == df_test$spam)
prediction_accuracy
```

```
[1] 0.923913
```

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"`.

```r
library(rpart)
rpart_fit <- rpart(spam ~ ., data = df_train, method = "class")
rpart_classes <- predict(rpart_fit, newdata = df_test, type = "class")
rpart_fit
```

```
n= 4141

node), split, n, loss, yval, (yprob)
      * denotes terminal node
```

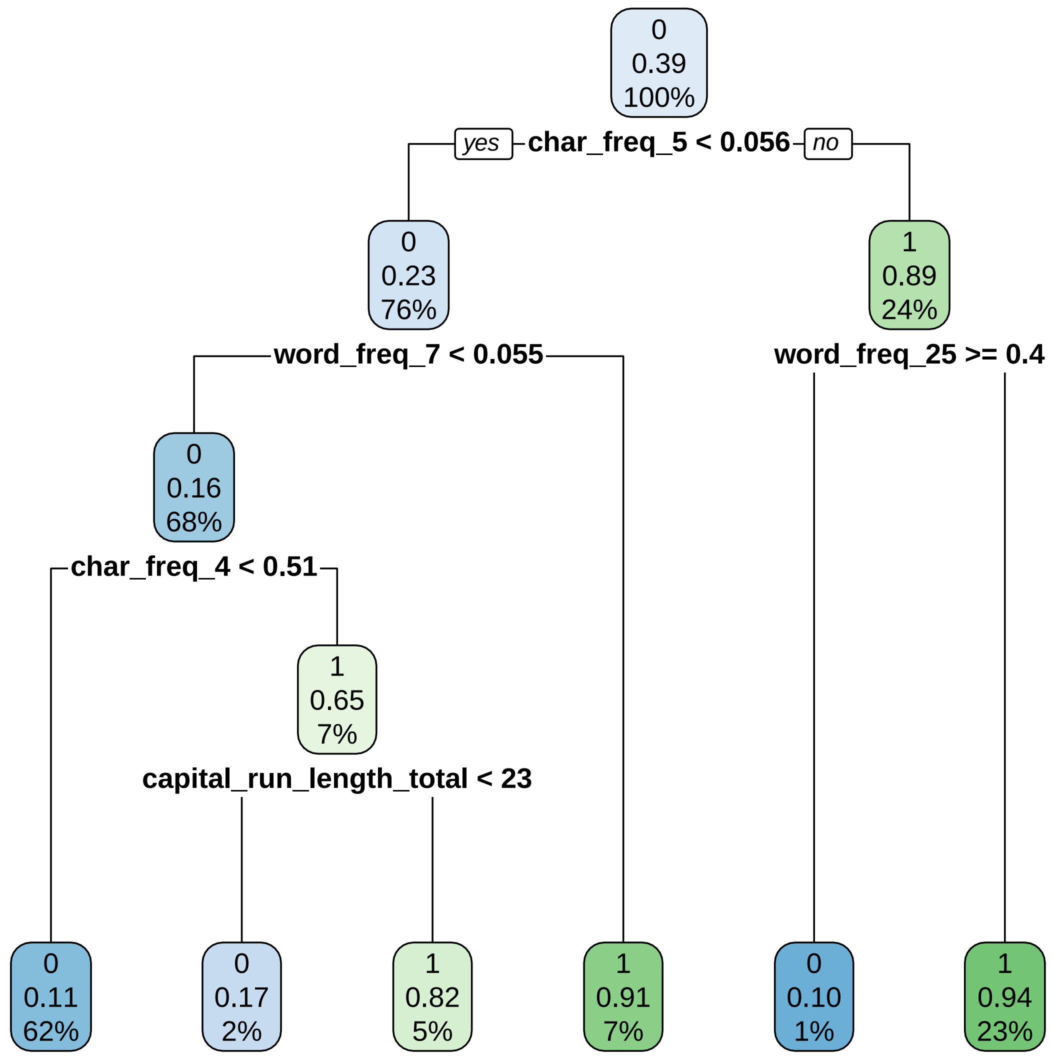
```
1) root 4141 1625 0 (0.60758271 0.39241729)
  2) char_freq_5< 0.0555 3134  730 0 (0.76707084 0.23292916)
    4) word_freq_7< 0.055 2832  454 0 (0.83968927 0.16031073)
      8) char_freq_4< 0.5085 2549  271 0 (0.89368380 0.10631620) *
      9) char_freq_4>=0.5085 283  100 1 (0.35335689 0.64664311)
       18) capital_run_length_total< 22.5 76   13 0 (0.82894737 0.17105263) *
       19) capital_run_length_total>=22.5 207   37 1 (0.17874396 0.82125604) *
    5) word_freq_7>=0.055 302   26 1 (0.08609272 0.91390728) *
  3) char_freq_5>=0.0555 1007  112 1 (0.11122145 0.88877855)
    6) word_freq_25>=0.4 58    6 0 (0.89655172 0.10344828) *
    7) word_freq_25< 0.4 949   60 1 (0.06322445 0.93677555) *
```

```r
glimpse(rpart_classes)
```

```
Factor w/ 2 levels "0","1": 1 1 1 2 2 1 1 1 2 1 ...
- attr(*, "names")= chr [1:460] "1" "2" "3" "4" ...
```

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

```r
library(rpart)
library(rpart.plot)
rpart.plot(rpart_fit)
```



Report the prediction accuracy on the test set.

```
rpart_classes <- predict(rpart_fit, newdata = df_test, type = "class")

prediction_accuracy <- mean(rpart_classes == df_test$spam)
prediction_accuracy
```

```
[1] 0.8782609
```

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`.

```
df_train$spam <- as.factor(df_train$spam)
df_test$spam <- as.factor(df_test$spam)
library(e1071)
svm_fit <- svm(spam ~ ., data = df_train, kernel = "radial", type = "C-classification")
svm_fit
```

```
Call:
svm(formula = spam ~ ., data = df_train, kernel = "radial", type = "C-classification")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  1157
```

Report the prediction accuracy on the test set.

```
svm_classes <- predict(svm_fit, newdata = df_test)

prediction_accuracy <- mean(svm_classes == df_test$spam)
prediction_accuracy
```

```
[1] 0.923913
```

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.

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.

```
#library("luz")
#X_train <- model.matrix(~ . - spam, data = df_train)
#y_train <- as.numeric(df_train$spam) - 1
#nnet_fit <- NNet %>%
  #setup(
   #loss = "mean_squared_error",
   #optimizer = "adam"
 # ) %>%
 # set_hparams(
 #   p = ncol(X_train),
 #   q1 = 32,
 #   q2 = 16,
 #   q3 = 8
#  ) %>%
#  set_opt_hparams(
 #   lr = 0.001
 # ) %>%
 # fit(
 #   X_train,
#    y_train,
#    dataloader_options = list(batch_size = 64),
#    verbose = TRUE # Change to TRUE while tuning. But, set to FALSE before submitting

#)
```

Once again, I was unable to render my neural network properly, resulting in no output.

2.7 (5 points)

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

```
#library(knitr)
#results <- c(
 # logistic_regression = mean(glm_classes == df_test$spam),
 # decision_tree = mean(rpart_classes == df_test$spam),
 # support_vector_machine = mean(svm_classes == df_test$spam),
 # neural_network = nnet_fit$metrics$val_accuracy
#)

#results <- data.frame(Model = names(results), Accuracy = results)
#kable(results)
```

I attempted to create a table comparing the accuracy metrics, but since my nnet_fit was not working, I could not get the actual code output.

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.

Since I cannot necessarily see the output for each of the models, I will go into the hypotheticals again. Whichever model that had the least percentage of false negatives and false positives or the overall highest percentage of true positive and true negatives would likely be the best model. This is because we would not want spam emails to be marked as real, and we would not want real emails to be marked as spam.

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:

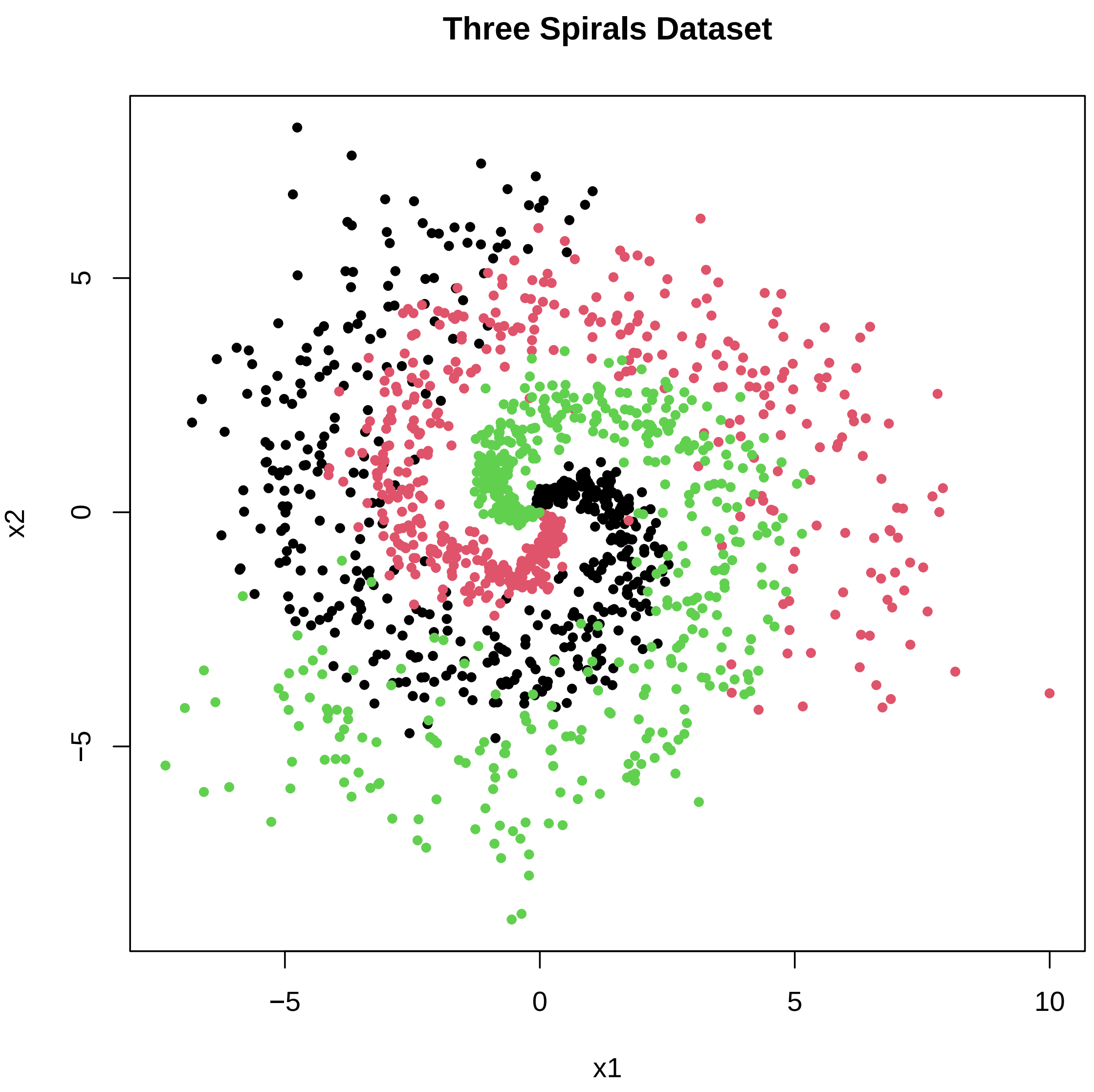
```r
generate_three_spirals <- function(){
  set.seed(42)
  n <- 500
  noise <- 0.2
  t <- (1:n) / n * 2 * pi
  x1 <- c(
      t * (sin(t) + rnorm(n, 0, noise)),
      t * (sin(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (sin(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  x2 <- c(
      t * (cos(t) +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

```r
df <- generate_three_spirals()

plot(
  df$x1, df$x2,
  col = df$y,
  pch = 20,
  xlab = "x1",
  ylab = "x2",
  main = "Three Spirals Dataset"
)
```



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`.

```r
grid <- expand.grid(
  x1 = seq(-10, 10, length.out = 100),
  x2 = seq(-10, 10, length.out = 100))
df_test <- as_tibble(grid)
```

```
df_test
```

```
# A tibble: 10,000 x 2
       x1    x2
    <dbl> <dbl>
 1 -10      -10
 2  -9.80   -10
 3  -9.60   -10
 4  -9.39   -10
 5  -9.19   -10
 6  -8.99   -10
 7  -8.79   -10
 8  -8.59   -10
 9  -8.38   -10
10  -8.18   -10
# i 9,990 more rows
```

3.2 (10 points)

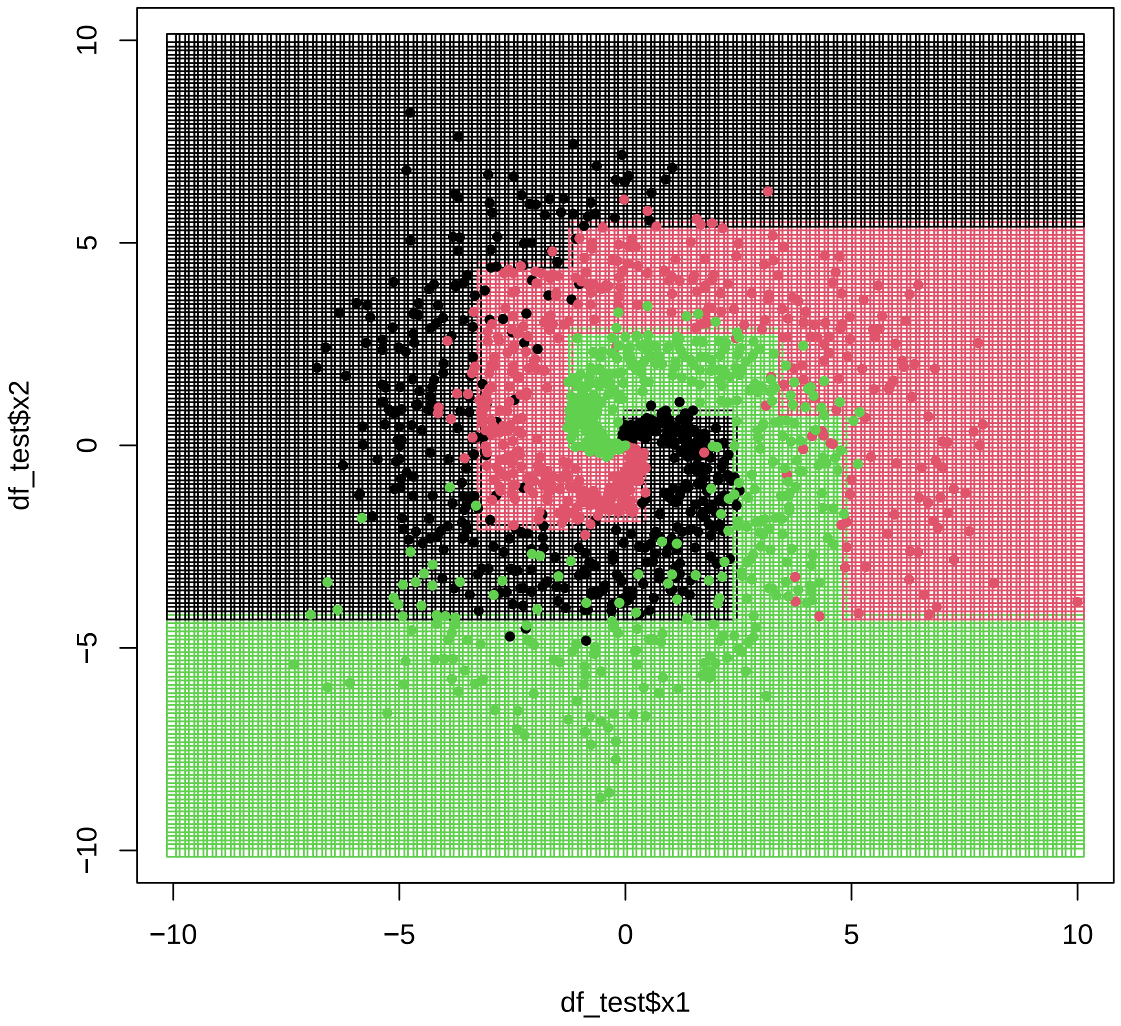
Fit a classification tree model to predict the `y` variable using the `x1` and `x2` predictors, and plot the decision boundary.

```
rpart_fit <- rpart(y ~ x1 + x2, data = df, method = "class")
rpart_classes <- predict(rpart_fit, newdata = df_test, type = "class")
```

Plot the decision boundary using the following function:

```
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
  )
}
```

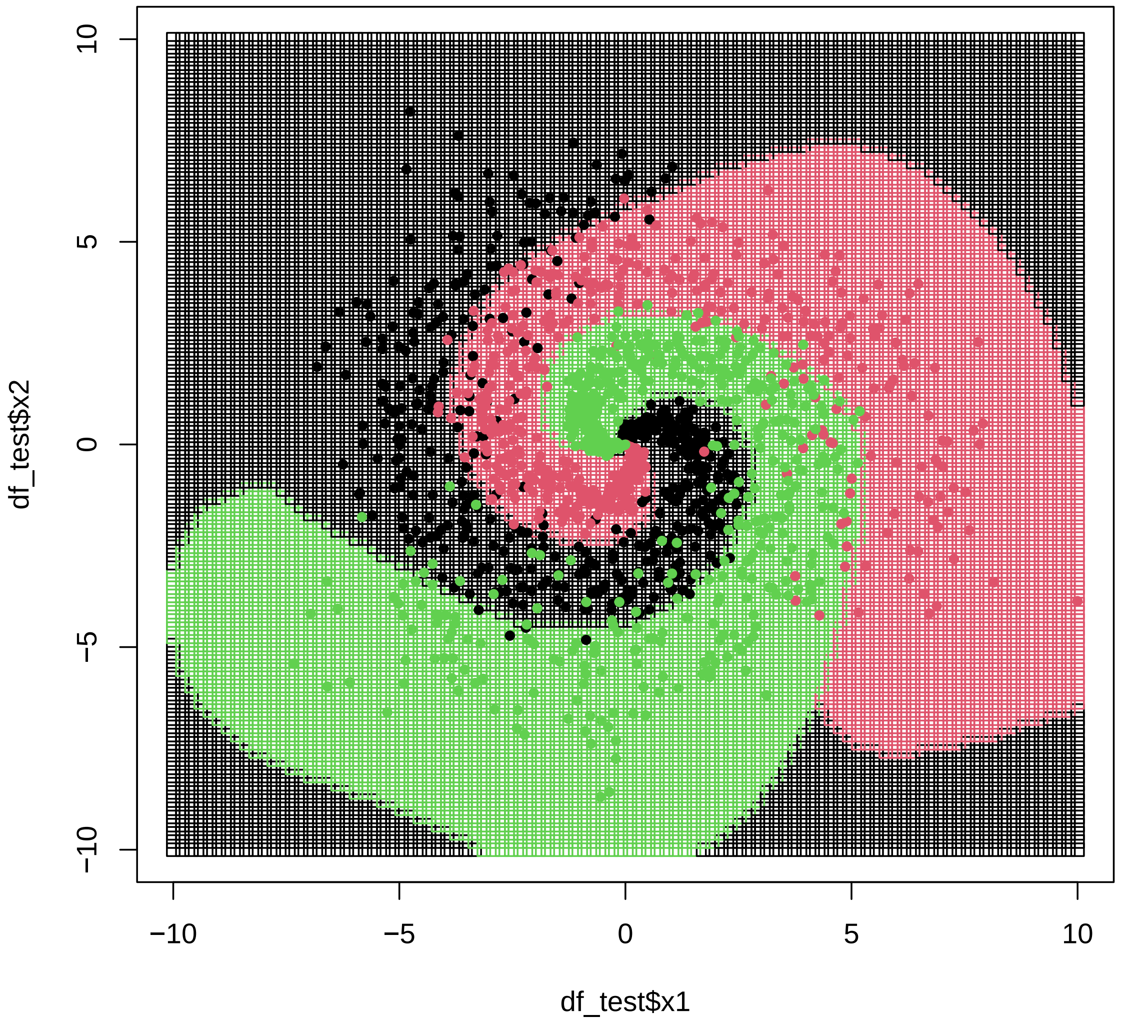
```r
plot_decision_boundary(rpart_classes)
```



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.

```
df$y <- as.factor(df$y)
svm_fit <- svm(y ~ x1 + x2, data = 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.

```
#library(tidyverse)
#p <- 1
#q1 <- 20
#q2 <- 100
#NN1 <- nn_module(
 # initialize = function(p, q1, o){
 #   self$input_to_hidden1 <- nn_linear(p, q1)
 #   self$hidden1_to_hidden2 <- nn_linear(q1, q2)
  #   self$hidden2_to_output <- nn_linear(q2, 1)
  #   self$activation <- nn_relu()
 #   self$output_activation <- nn_sigmoid()
 # },
 # forward = function(x){
 #   x %>%
  #    self$input_to_hidden1() %>%
  #    self$activation() %>%
  #    self$hidden1_to_hidden2() %>%
  #    self$activation() %>%
  #    self$hidden2_to_output() %>%
  #    self$output_activation()
 # }
#)
```

```
#fit_1 <- NN1 %>%
#  setup(
 #   loss = nn_cross_entropy_loss(),
 #   optimizer = "adam"
 # ) %>%
 # set_hparams(
  #  p = p,
  #  q1 = q1
#  ) %>%
 # set_opt_hparams(
#    learning_rate = 0.01
 # ) %>%
 # fit(
#    data = list(
#      X = df %>% select(x1, x2) %>% as.matrix,
 #     y = df$y %>% as.integer,
 #     dataloader_options = list(batch_size = nrow(df), shuffle = TRUE),
#      verbose = FALSE
#  ))
```

Once again, I attempted to make the proper neural network, but I got the same error that I usually get, which is "Error in ctx_check_optimizers(): Expected a torch optimizer but got an object with class '{class(i)[1]}'" I don't know how to fix that error unfortunately.

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

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

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

Once again, if the neural network had been done correctly, this code chunk would have ran.

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

```
#plot_decision_boundary(fit_1_predictions)
```

I would have been able to plot the predictions if the neural network were up and running.

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$output <- nn_linear(o, activation = "softmax")
#  },
#  forward = function(x){
#    x %>%
#    self$output()
 # }
#)

#fit_0 <- NN0 %>%
 # setup(input_size = 2,
  #  output_size = 3) %>%
 # set_hparams(lr = 0.01,
 #   epochs = 50) %>%
 # set_opt_hparams(...) %>%
 # fit(data = list(
 #     df %>% select(x1, x2) %>% as.matrix,
  #    df$y %>% as.integer),
 #   optimizer = "adam",
 #   dataloader_options = list(batch_size = nrow(df)),
  #  verbose = FALSE
 # )
```

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

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

#plot_decision_boundary(fit_0_predictions)
```

If the neural network had been implemented correctly, this chunk would have run.

3.6 (10 points)

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

```
#NN2 <- nn_module(
#  initialize = function(p, q1, o){
 #   self$hidden1 <- nn_linear(p, q1, activation = "relu")
#    self$output <- nn_linear(q1, o, activation = "softmax")
#    self$activation <- nn_relu()
#  },
 # forward = function(x){
  #  x %>%
 #     self$hidden1() %>%
 #     self$hidden2() %>%
  #    self$hidden3() %>%
 #     self$output()
#  }
#)

#fit_2 <- NN3 %>%
#  setup(input_size = 2,
#    hidden_sizes = c(10, 10, 10),
  #  output_size = 3) %>%
 # set_hparams(lr = 0.01) %>%
 # set_opt_params(optimizer = "adam") %>%
 # fit(data = list(
 #     df %>% select(x1, x2) %>% as.matrix,
 #     df$y %>% as.integer),
 #   optimizer = "adam",
#    dataloader_options = list(batch_size = nrow(df)),
#    verbose = FALSE
#  )
```

I had troubles with the neural network implementation again, so my apologies.

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

```
#fit_3_predictions <- predict(fit_3, test_matrix) %>%
 # argmax(2) %>%
 # as.integer()

#plot_decision_boundary(fit_3_predictions)
```

The above code chunk would have ran if I had had my neural network up and running.

3.7 (5 points)

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

Each model is its own unique way of serving a similar purpose.

Linear regression assumes a linear relationship between the two analyzed variables. The decision boundary is a straight line. Decision trees work based on the threshold of different features. The decision boundary is made of axis-aligned lines. Support vector machines (SVM) finds proper separation of the features and uses dimensions to make a linear decision boundary. Neural networks can be used for nonlinear relationships, and the decision boundary is based on the number of layers in the neural network.

Increasing the number of layers makes more complex and flexible decision boundaries, which overall improves the model's capabilities.

Session Information

Print your R session information using the following command

```
sessionInfo()
```

```
R version 4.3.2 (2023-10-31 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 11 x64 (build 22631)

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

time zone: America/New_York
tzcode source: internal

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

other attached packages:
 [1] lubridate_1.9.3  forcats_1.0.0    stringr_1.5.1    tidyverse_2.0.0
 [5] luz_0.4.0        torch_0.12.0     e1071_1.7-14     rpart.plot_3.1.2
 [9] rpart_4.1.23     caret_6.0-94     lattice_0.22-5   ggplot2_3.4.4
[13] corrplot_0.92    magrittr_2.0.3   broom_1.0.5      purrr_1.0.2
[17] tidyr_1.3.0      readr_2.1.4      dplyr_1.1.4      tibble_3.2.1

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.0     timeDate_4032.109    fastmap_1.1.1
 [4] pROC_1.18.5          digest_0.6.33        timechange_0.2.0
 [7] lifecycle_1.0.4      survival_3.5-7       processx_3.8.2
[10] compiler_4.3.2       progress_1.2.2       rlang_1.1.2
[13] tools_4.3.2          utf8_1.2.4           yaml_2.3.7
[16] data.table_1.14.8    knitr_1.45           prettyunits_1.2.0
```

```
[19] bit_4.0.5            plyr_1.8.8           withr_2.5.2
[22] nnet_7.3-19          grid_4.3.2           stats4_4.3.2
[25] fansi_1.0.5          colorspace_2.1-0     future_1.33.1
[28] globals_0.16.2       scales_1.2.1         iterators_1.0.14
[31] MASS_7.3-60          zeallot_0.1.0        cli_3.6.1
[34] crayon_1.5.2         rmarkdown_2.25       generics_0.1.3
[37] rstudioapi_0.15.0    future.apply_1.11.1  reshape2_1.4.4
[40] tzdb_0.4.0           proxy_0.4-27         splines_4.3.2
[43] parallel_4.3.2       coro_1.0.4           vctrs_0.6.4
[46] hardhat_1.3.1        Matrix_1.6-3         jsonlite_1.8.7
[49] callr_3.7.3          hms_1.1.3            bit64_4.0.5
[52] listenv_0.9.1        foreach_1.5.2        gower_1.0.1
[55] recipes_1.0.9        glue_1.6.2           parallelly_1.36.0
[58] ps_1.7.5             codetools_0.2-19     stringi_1.8.2
[61] gtable_0.3.4         munsell_0.5.0        pillar_1.9.0
[64] htmltools_0.5.7      ipred_0.9-14         lava_1.7.3
[67] R6_2.5.1             evaluate_0.23        backports_1.4.1
[70] class_7.3-22         Rcpp_1.0.11          nlme_3.1-163
[73] prodlim_2023.08.28   xfun_0.41            fs_1.6.3
[76] pkgconfig_2.0.3      ModelMetrics_1.2.2.2
```