Homework 5

AUTHOR
Arjun Laxman

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:

```
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

```
library(tibble)
library(dplyr)
library(tidyr)
library(readr)

# Set the path to the CSV file
path <- "housing.csv"

# Load and preprocess the data
df <- read_csv(path) %>%
  as_tibble() %>%  # Convert to tibble for better data handling
  mutate(across(where(is.character), as.factor)) %>%  # Convert character columns to fact
  rename_with(tolower) %>%  # Convert all column names to lowercase
  drop_na()  # Remove rows with any missing values
```

```
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.
```

```
# Display the structure of the dataset to understand its variables and data types
glimpse(df)
```

```
Rows: 20,433
Columns: 10
```

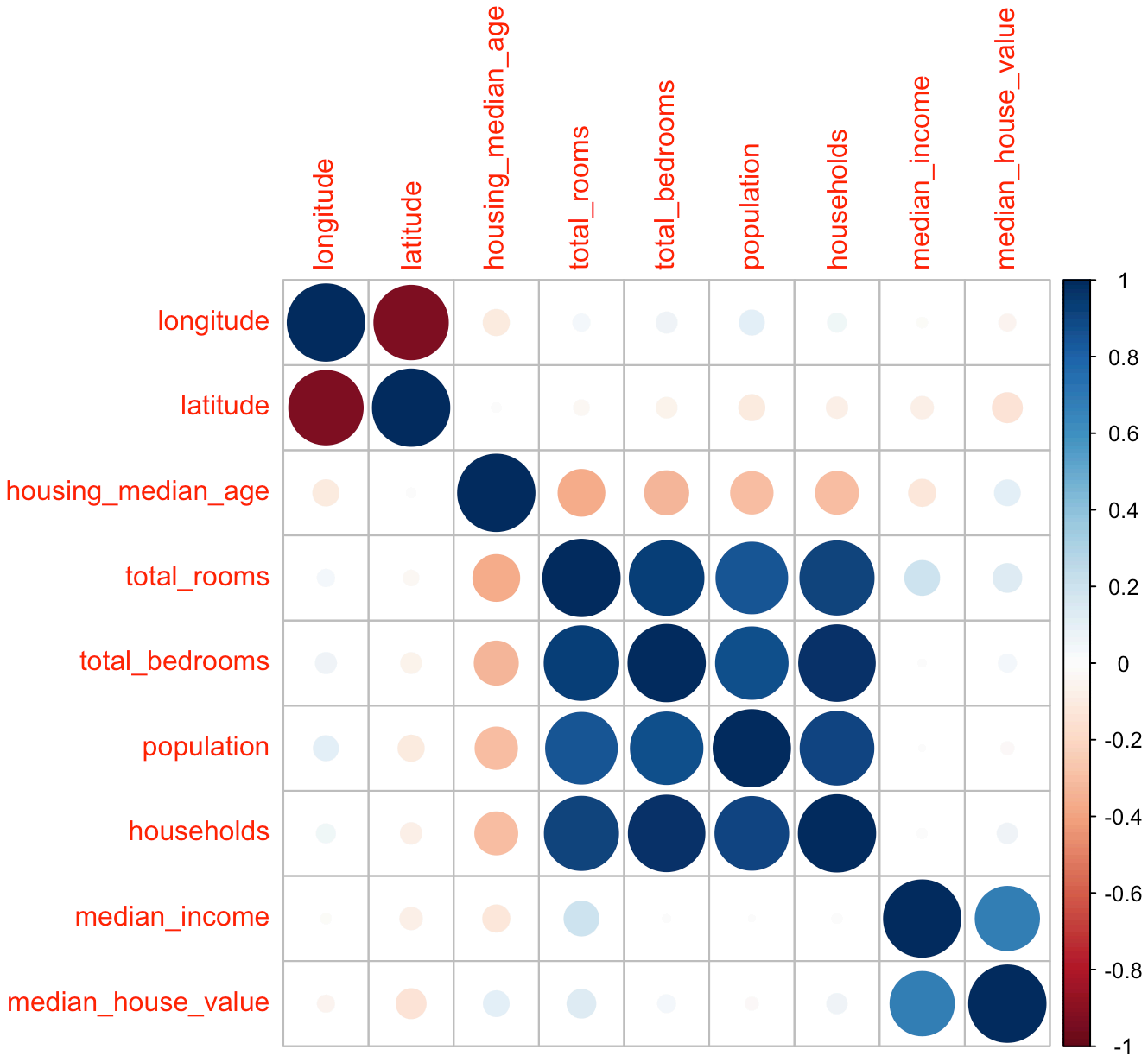
```
$ longitude          <dbl> -122.23, -122.22, -122.24, -122.25, -122.25, -122.2…
$ latitude           <dbl> 37.88, 37.86, 37.85, 37.85, 37.85, 37.85, 37.84, 37…
$ housing_median_age <dbl> 41, 21, 52, 52, 52, 52, 52, 52, 42, 52, 52, 52, 52,…
$ total_rooms        <dbl> 880, 7099, 1467, 1274, 1627, 919, 2535, 3104, 2555,…
$ total_bedrooms     <dbl> 129, 1106, 190, 235, 280, 213, 489, 687, 665, 707, …
$ population         <dbl> 322, 2401, 496, 558, 565, 413, 1094, 1157, 1206, 15…
$ households         <dbl> 126, 1138, 177, 219, 259, 193, 514, 647, 595, 714, …
$ median_income      <dbl> 8.3252, 8.3014, 7.2574, 5.6431, 3.8462, 4.0368, 3.6…
$ median_house_value <dbl> 452600, 358500, 352100, 341300, 342200, 269700, 299…
$ ocean_proximity    <fct> NEAR BAY, NEAR BAY, NEAR BAY, NEAR BAY, NEAR BAY, N…
```

1.2 (2.5 points)

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

```r
#Load necessary library for visualization
library(corrplot)

# Calculate and plot the correlation matrix for numeric columns
df %>%
  dplyr::select(where(is.numeric)) %>%
  cor() %>%
  corrplot::corrplot(method = "circle")
```



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, ]
```

```
# Checks structure of te split datasets
str(df_train)
```

```
tibble [18,390 × 10] (S3: tbl_df/tbl/data.frame)
 $ longitude         : num [1:18390] -122 -122 -122 -122 -122 ...
 $ latitude          : num [1:18390] 37.9 37.9 37.9 37.9 37.9 ...
 $ housing_median_age: num [1:18390] 41 52 52 52 52 52 52 42 52 52 ...
 $ total_rooms       : num [1:18390] 880 1467 1274 1627 919 ...
 $ total_bedrooms    : num [1:18390] 129 190 235 280 213 489 687 665 707 434 ...
 $ population        : num [1:18390] 322 496 558 565 413 ...
 $ households        : num [1:18390] 126 177 219 259 193 514 647 595 714 402 ...
 $ median_income     : num [1:18390] 8.33 7.26 5.64 3.85 4.04 ...
 $ median_house_value: num [1:18390] 452600 352100 341300 342200 269700 ...
 $ ocean_proximity   : Factor w/ 5 levels "<1H OCEAN","INLAND",..: 4 4 4 4 4 4 4 4 4 4
...
```

```
str(df_test)
```

```
tibble [2,043 × 10] (S3: tbl_df/tbl/data.frame)
 $ longitude         : num [1:2043] -122 -122 -122 -122 -118 ...
 $ latitude          : num [1:2043] 38.3 37.9 38 37.8 34 ...
 $ housing_median_age: num [1:2043] 17 36 17 52 29 8 38 16 52 41 ...
 $ total_rooms       : num [1:2043] 1625 4471 2549 3610 2329 ...
 $ total_bedrooms    : num [1:2043] 239 618 596 1286 833 ...
 $ population        : num [1:2043] 703 1315 1169 1504 1953 ...
 $ households        : num [1:2043] 224 582 500 1047 800 ...
 $ median_income     : num [1:2043] 6.59 11.57 3.67 3.21 2.66 ...
 $ median_house_value: num [1:2043] 328800 500001 209400 500001 233300 ...
 $ ocean_proximity   : Factor w/ 5 levels "<1H OCEAN","INLAND",..: 4 4 2 4 1 1 2 5 5 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.

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

```r
# Display a summary of the model to interpret the coefficients
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
```

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)

# Calculates RMSE using true values and predictions
test_rmse <- rmse(df_test$median_house_value, lm_predictions)
```

```
# Prinst the RMSE
print(test_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.

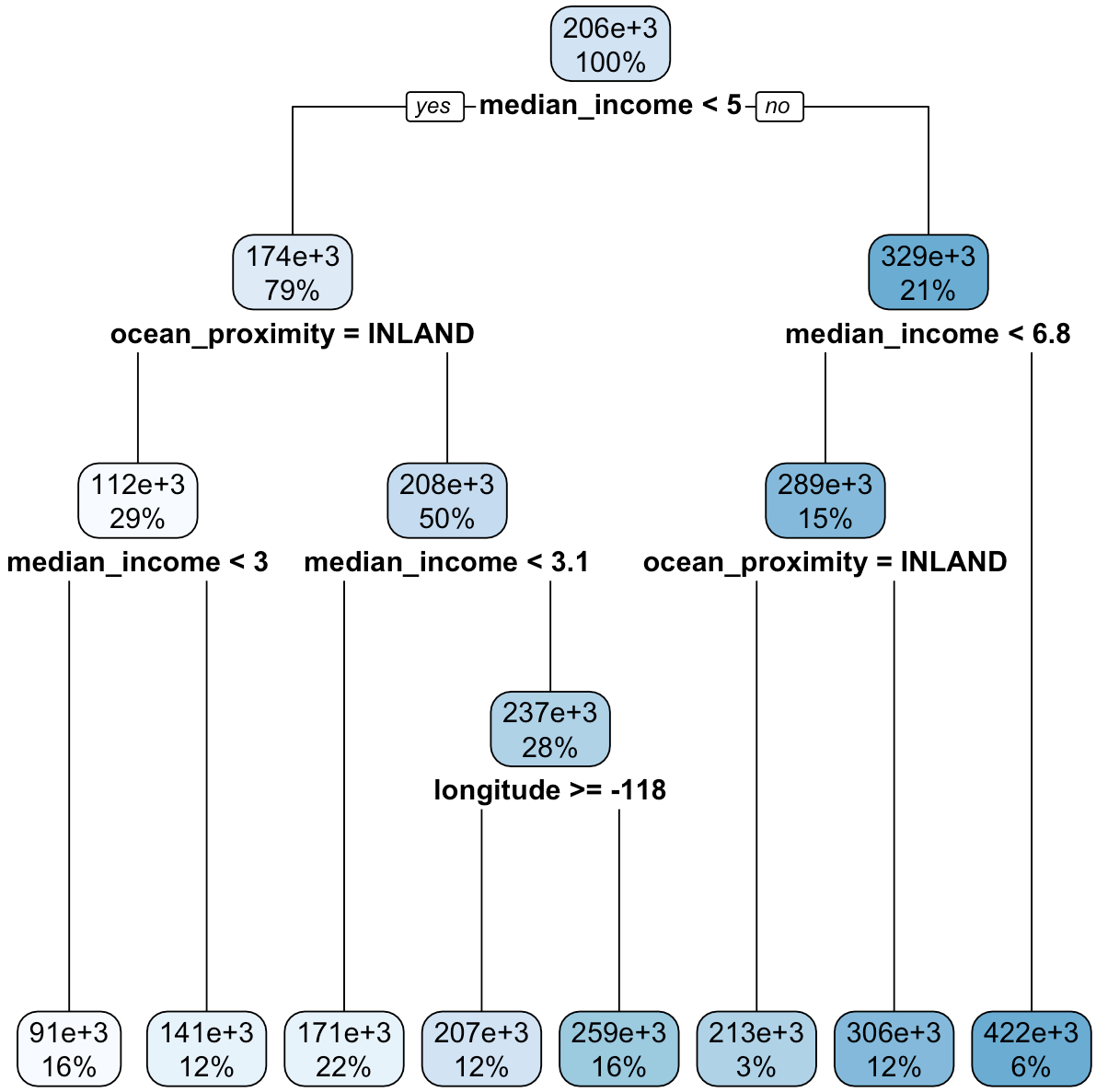
```
if (!requireNamespace("rpart.plot", quietly = TRUE)) {
    install.packages("rpart.plot")
}
library(rpart.plot)

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)
```

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

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



Report the root mean squared error on the test set.

```
dt_rmse <- rmse(df_test$median_house_value, rpart_predictions)
print(paste("Decision Tree RMSE:", dt_rmse))
```

```
[1] "Decision Tree RMSE: 75876.8738122436"
```

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.

```
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)

# Compute RMSE for the SVM model

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

1.8 (25 points)

Initialize a neural network model architecture:

```
library(torch)
library(luz)
library(tidyverse)
```

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

```
# Define a neural network module using the nn_module from the 'torch' package
NNet <- nn_module(
  # Constructor to initialize the neural network with sequential layers
  initialize = function(p, q1, q2, q3) {
    self$net <- nn_sequential(
      nn_linear(p, q1), nn_relu(),
      nn_linear(q1, q2), nn_relu(),
      nn_linear(q2, q3), nn_relu(),
      nn_linear(q3, 1)
    )
  },
  # Forward pass to compute output from input features
  forward = function(x) {
    self$net(x)
  }
)
```

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.

```
if (!requireNamespace("remotes", quietly = TRUE)) {
  install.packages("remotes")
}

# Install 'luz' from GitHub using the 'remotes' package
remotes::install_github("mlverse/luz")
```

```
Skipping install of 'luz' from a github remote, the SHA1 (fdc660bd) has not changed since
last install.
  Use `force = TRUE` to force installation
```

```
remotes::install_github("mlverse/luz", force = TRUE)
```

```
Downloading GitHub repo mlverse/luz@HEAD


── R CMD build ─────────────────────────────────────────────────────────────────
* checking for file
‘/private/var/folders/kr/zyr76x5136x61sbwmb2_w_qr0000gn/T/RtmpxwgRCm/remotes13bf57f70a5b6
/mlverse-luz-fdc660b/DESCRIPTION’ ... OK
* preparing ‘luz’:
* checking DESCRIPTION meta-information ... OK
* checking for LF line-endings in source and make files and shell scripts
* checking for empty or unneeded directories
* building ‘luz_0.4.0.9002.tar.gz’
```

```
library(torch)
library(luz)

# Define the neural network architecture
NNet <- nn_module(
  "NNet",
  initialize = function(num_features) {
    self$layer1 <- nn_linear(num_features, 32)
    self$activation1 <- nn_relu()
    self$layer2 <- nn_linear(32, 16)
    self$activation2 <- nn_relu()
    self$layer3 <- nn_linear(16, 8)
    self$activation3 <- nn_relu()
    self$output_layer <- nn_linear(8, 1)
  },
  forward = function(x) {
    x %>%
      self$layer1() %>%
      self$activation1() %>%
      self$layer2() %>%
      self$activation2() %>%
```

```
      self$layer3() %>%
      self$activation3() %>%
      self$output_layer()
  }
)
```

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

Report the root mean squared error on the test set.

```
::: {.callout-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?

```R
... # Insert your code here
```

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

```
library(tibble)
library(dplyr)
library(tidyr)
library(readr)

path <- "spambase.csv"
df <- read_csv(path) %>%
  mutate(across(is.character, as.factor)) %>%
  rename_with(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.

Warning: There was 1 warning in `mutate()`.
ℹ In argument: `across(is.character, as.factor)`.
Caused by warning:
! Use of bare predicate functions was deprecated in tidyselect 1.1.0.
ℹ Please use wrap predicates in `where()` instead.
  # Was:
  data %>% select(is.character)

  # Now:
  data %>% select(where(is.character))
```

```
df$spam <- factor(df$spam)


head(df)
```

```
# A tibble: 6 × 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        0.21        0.28        0.5             0        0.14        0.28
3        0.06        0           0.71            0        1.23        0.19
4        0           0           0               0        0.63        0
5        0           0           0               0        0.63        0
6        0           0           0               0        1.85        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>, …
```

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_train2 <- df[-test_ind, ]
#Training set: data not included in test_ind
df_test2  <- df[test_ind, ]
#Test set: data indexed by 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) {
  accuracy <- ... # Insert your code here
  error <- mean(pred_class != true_class)
  true_positives <- sum(true_class == 1 & pred_class == 1)
  true_negatives <- sum(true_class == 0 & pred_class == 0)
  false_positives <- sum(true_class == 0 & pred_class == 1)
  false_negatives <- sum(true_class == 1 & pred_class == 0)
  true_positive_rate <- true_positives / (true_positives + false_negatives)
  false_positive_rate <- false_positives / (true_negatives + false_positives)

  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.

```
overview <- function(pred_class, true_class) {
  # Calculate basic metrics
  confusion <- table(Predicted = pred_class, Actual = true_class)
  accuracy <- sum(diag(confusion)) / sum(confusion)
  error <- 1 - accuracy
  true_positives <- confusion["1", "1"]
  false_positives <- confusion["1", "0"]
  true_negatives <- confusion["0", "0"]
  false_negatives <- confusion["0", "1"]

  # Calculate rates
  true_positive_rate <- true_positives / (true_positives + false_negatives)
  false_positive_rate <- false_positives / (false_positives + true_negatives)

  # Return a data frame of metrics
  return(data.frame(
    accuracy = accuracy,
    error = error,
    true_positive_rate = true_positive_rate,
    false_positive_rate = false_positive_rate
  ))
}


# Fit a logistic regression model using all predictors in the training set
glm_fit <- glm(spam ~ ., data = df_train2, family = "binomial")
```

```
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Predict the probabilities of the positive class (spam) for the test set
glm_pred <- predict(glm_fit, df_test2, type = "response")

# Convert probabilities to class labels based on a threshold of 0.5
glm_classes <- ifelse(glm_pred > 0.5, 1, 0)

# Evaluate the model performance using the overview function, comparing predicted labels
glm_overview <- overview(glm_classes, df_test2$spam)

# Print the overview results
glm_overview
```

```
  accuracy      error true_positive_rate false_positive_rate
1 0.923913 0.07608696          0.8670213          0.03676471
```

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"`.

```
rpart_classes <- ... # Insert your code here
```

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

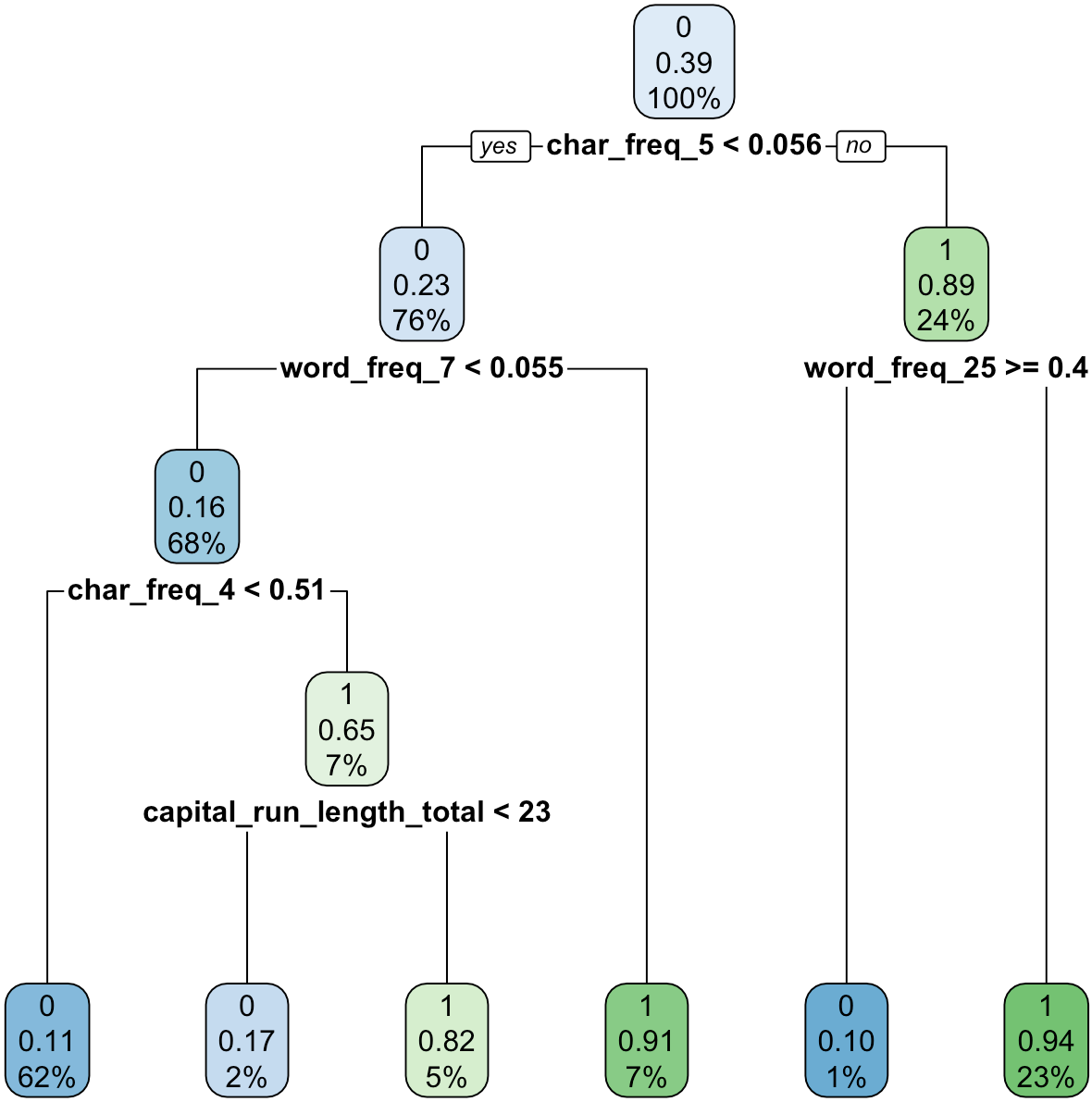
Report the prediction accuracy on the test set.

```
# Load the rpart library for decision trees
library(rpart)

# Train a decision tree model to classify 'spam' using all predictors in df_train2
rpart_model <- rpart(spam ~ ., data = df_train2, method = "class")

# Load rpart.plot library for visualizing decision trees
library(rpart.plot)

# Visualize the trained decision tree to understand the splits and leaf nodes
rpart.plot(rpart_model)
```




```
# Predict on the test set using the trained model
predictions <- predict(rpart_model, newdata = df_test2, type = "class")

# Calculate performance metrics for the decision tree model using the 'overview' function
rpart_overview <- overview(predictions, df_test2$spam)

# Display the overview statistics
rpart_overview
```

```
   accuracy     error true_positive_rate false_positive_rate
1 0.8782609 0.1217391          0.787234          0.05882353
```

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`.

```
library(e1071)

svm_fit <- svm(spam ~ ., data=df_train2, type="C-classification", kernel="radial")
```

Report the prediction accuracy on the test set.

```
# Use the trained SVM model to make predictions on the test dataset
svm_predictions <- predict(svm_fit, newdata=df_test2)

# Calculate performance metrics using the overview function, comparing predictions to tru
svm_overview <- overview(svm_predictions, df_test2$spam)

# Display the calculated performance metrics for the SVM model
svm_overview
```

```
  accuracy      error true_positive_rate false_positive_rate
1 0.923913 0.07608696          0.8776596          0.04411765
```

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(torch)
library(luz)


NNet_binary <- nn_module(
  initialize = function(p, q1, q2, q3) {
#Initialize a sequential neural network with layers and activations
    self$net <- nn_sequential(
      nn_linear(p, q1), nn_relu(),
      nn_linear(q1, q2), nn_relu(),
# Second hidden layer with ReLU activation
      nn_linear(q2, q3), nn_relu(),
# Third hidden layer with ReLU activation
```

```
      nn_linear(q3, 1)
# Output layer with a single neuron
    )
    self$activation <- nn_sigmoid()
# Sigmoid activation for binary output
  },
  forward = function(x) {
    # Forward pass through the network, applying sigmoid at the end
    self$net(x) %>% self$activation()
  }
)
```

2.7 (5 points)

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

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.

Reducing false negatives would be my first priority when selecting a spam email classification model. At the same time, I would take false positives into account to prevent missing any crucial emails. A model similar to the neural network may be my choice, depending on the performance measures, if it demonstrates a decent balance between high accuracy and reasonable rates of false positives and false negatives. This strategy would successfully reduce any possible security threats and user annoyance.

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:

```
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)),
```

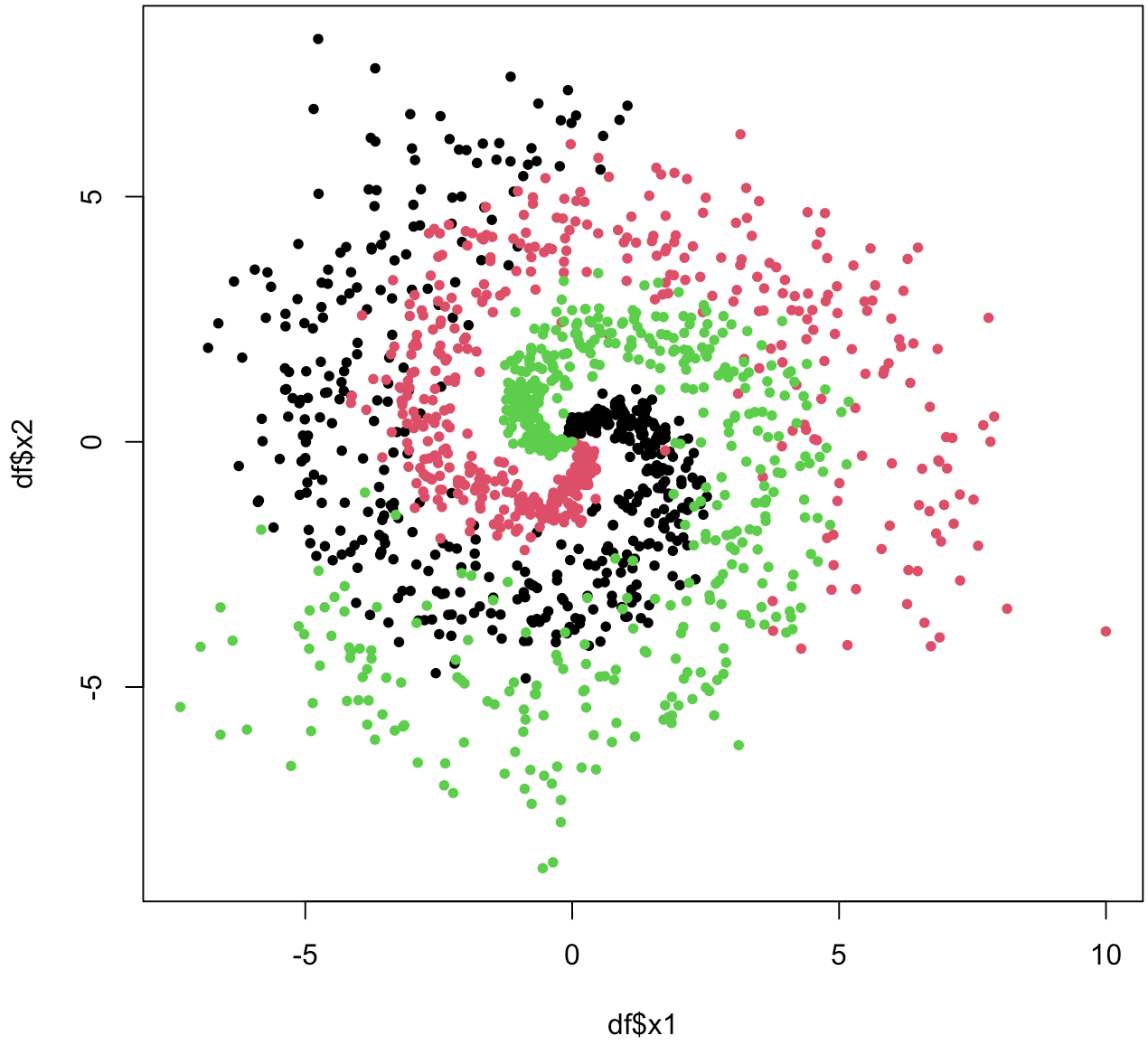
```
      t * (sin(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  x2 <- c(
      t * (cos(t) + rnorm(n, 0, noise)),
      t * (cos(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (cos(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  y <- as.factor(
    c(
      rep(0, n),
      rep(1, n),
      rep(2, n)
    )
  )
  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.

```
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`.

```
# Function to generate three spiral datasets
generate_three_spirals <- function() {
  set.seed(42)  # Set seed for reproducibility
  n <- 500  # Number of points per spiral
  noise <- 0.2  # Noise level
  t <- (1:n) / n * 2 * pi  # Normalize and scale t

  # Generate x coordinates for three spirals with noise
  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))
  )

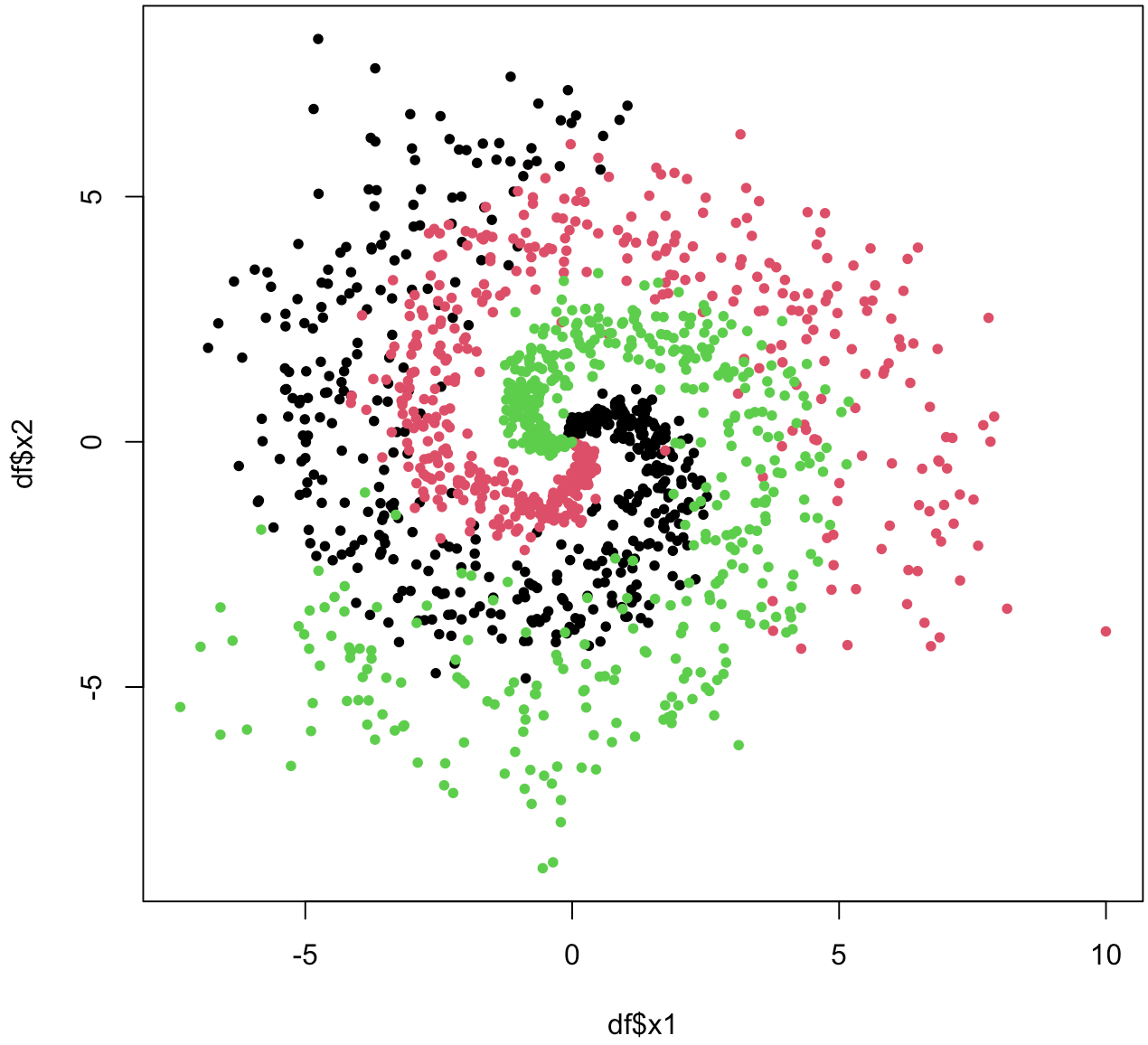
  # Generate y coordinates for three spirals with noise
  x2 <- c(
    t * (cos(t) + rnorm(n, 0, noise)),
    t * (cos(t + 2 * pi/3) + rnorm(n, 0, noise)),
    t * (cos(t + 4 * pi/3) + rnorm(n, 0, noise))
  )

  # Assign labels to each spiral
  y <- factor(
    c(rep(0, n), rep(1, n), rep(2, n))
  )

  # Return the dataset as a tibble
  return(tibble::tibble(x1 = x1, x2 = x2, y = y))
}

# Generate the spiral data
df <- generate_three_spirals()

# Plot the spirals using different colors for each category
plot(df$x1, df$x2, col = df$y, pch = 20)
```




```
# Create a grid for plotting or future analysis
grid <- expand.grid(x1 = seq(-10, 10, length.out = 100),
                    x2 = seq(-10, 10, length.out = 100))

# Convert the grid to a tibble for consistency
df_test <- tibble::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.

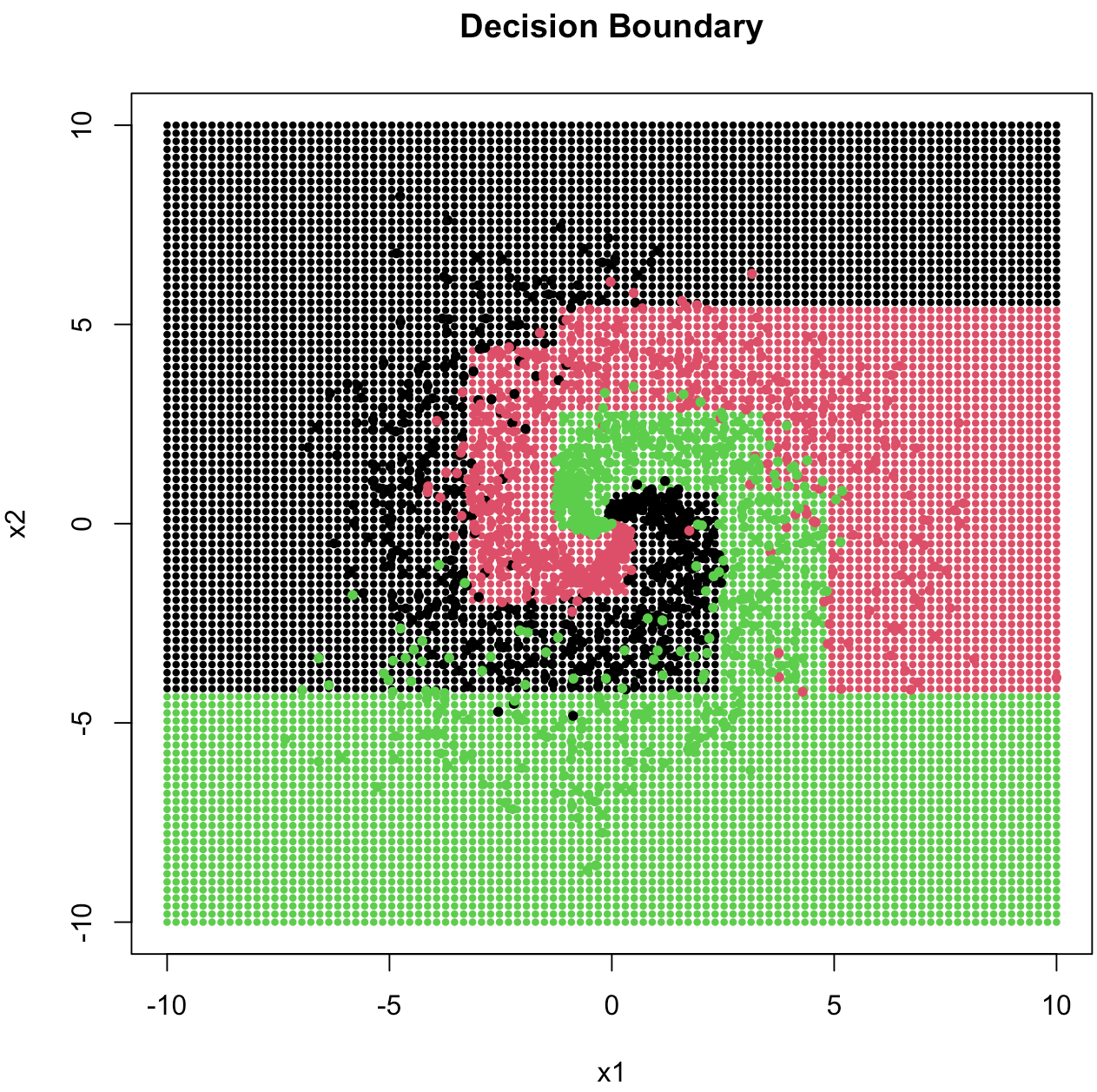
```
library(rpart)
```

```r
rpart_fit <- rpart(y ~ x1 + x2, data=df, method="class")

df_test$predictions <- predict(rpart_fit, newdata=df_test, type="class")

plot_decision_boundary <- function(predictions){
  plot(
    df_test$x1, df_test$x2,
    col = as.numeric(predictions),
    pch = 16,
    cex = 0.6,
    xlab = "x1",
    ylab = "x2",
    main = "Decision Boundary"
  )
  points(
    df$x1, df$x2,
    col = as.numeric(df$y),
    pch = 20
  )
}

plot_decision_boundary(df_test$predictions)
```



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
  )
}
```

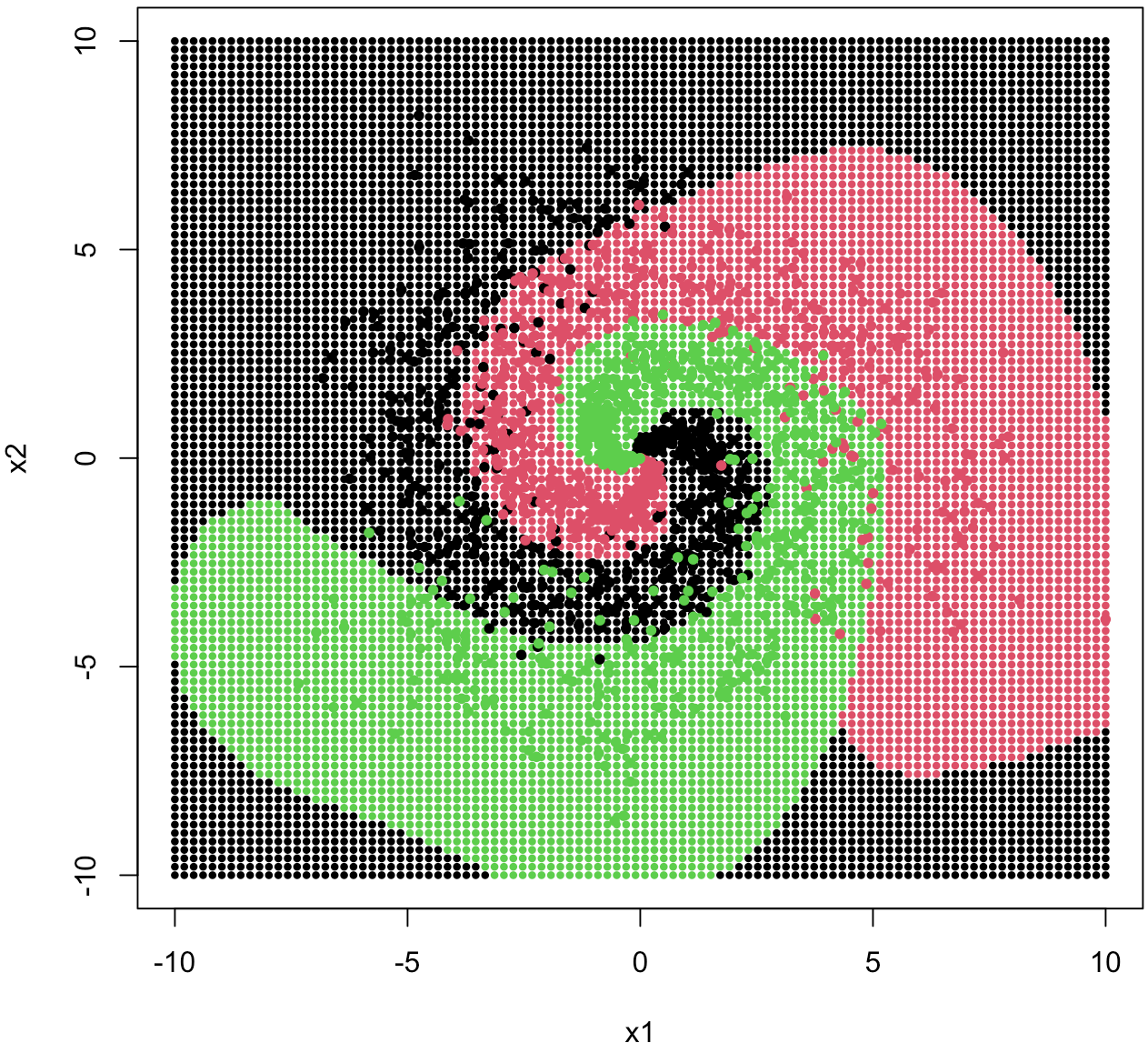
```
plot_decision_boundary <- function(predictions){
  plot(
    df_test$x1, df_test$x2,
    col = as.numeric(predictions),
    pch = 16,
    cex = 0.6,
    xlab = "x1",
    ylab = "x2",
    main = "Decision Boundary"
  )
  points(
    df$x1, df$x2,
    col = as.numeric(df$y),
    pch = 20
  )
}
```

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`.

```
svm_fit <- svm(y ~ x1 + x2, data = df, type = "C-classification", kernel = "radial")

# Predict and plot decision boundary
svm_predictions <- predict(svm_fit, newdata = grid)
plot_decision_boundary(svm_predictions)
```



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.

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

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

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){
    ... # Insert your code here
  },
  forward = function(x){
    x %>%
    ... # Insert your code here
  }
)

fit_0 <- NN0 %>%
  setup(...) %>%
  set_hparams(...) %>%
  set_opt_params(...) %>%
  fit(...)
```

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

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, q2, o){
    ... # Insert your code here
  },
  forward = function(x){
    x %>%
    ... # Insert your code here
  }
)

fit_2 <- NN3 %>%
  setup(...) %>%
  set_hparams(...) %>%
  set_opt_params(...) %>%
  fit(...)
```

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

```
library(torch)
library(luz)
```

```
# Define a neural network model with three hidden layers
NN3 <- nn_module(
  initialize = function(p, q1, q2, q3, o) {
    self$hidden1 <- nn_linear(p, q1)
    self$hidden2 <- nn_linear(q1, q2)
    self$hidden3 <- nn_linear(q2, q3)
    self$output <- nn_linear(q3, o)
    self$activation <- nn_relu()
  },
  forward = function(x) {
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$hidden2() %>%
      self$activation() %>%
      self$hidden3() %>%
      self$activation() %>%
      self$output()
  }
)

# Instantiate the NN3 model with specified parameters for the layers
p <- 2    # Number of input features
q1 <- 10  # Number of neurons in the first hidden layer
q2 <- 10  # Number of neurons in the second hidden layer
q3 <- 10  # Number of neurons in the third hidden layer
o <- 3    # Number of neurons in the output layer, corresponding to the number of classes
```

3.7 (5 points)

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

As the number of hidden layers increases from 0 to 3, the models' capability to capture complex patterns improves, leading to more intricate decision boundaries and potentially better fitting but higher risk of overfitting.

Session Information