Homework 5

Jacob Adams

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:

```
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

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

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.
```

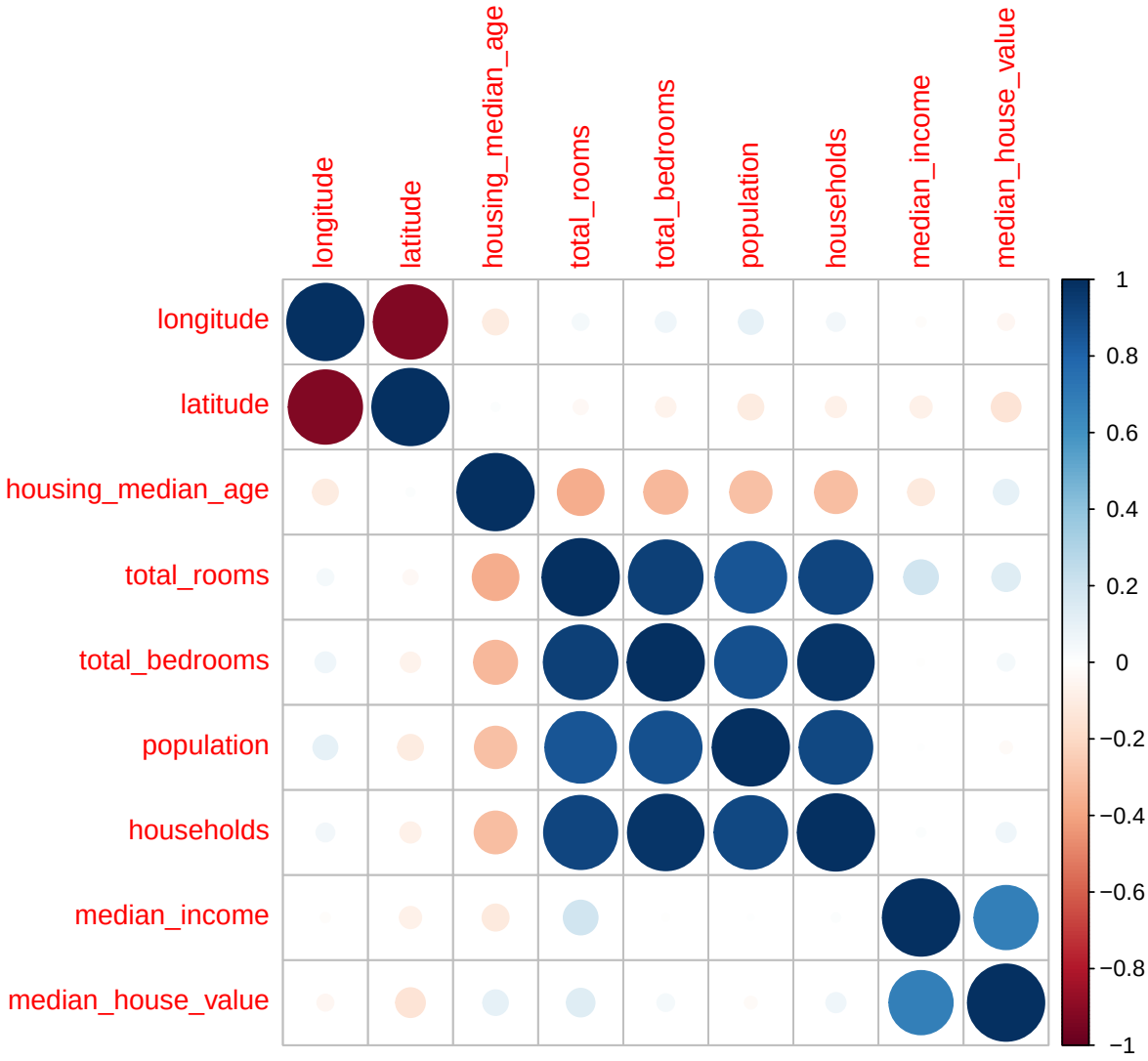
```r
df <- df %>%
  mutate(across(where(is.character), as.factor)) %>%
  rename_all(tolower) %>%
  drop_na()
```

1.2 (2.5 points)

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

```r
numeric_data <- df %>% select_if(is.numeric)
correlation_matrix <- cor(numeric_data)

corrplot(correlation_matrix)
```



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_test  <- df[-test_ind,]
df_train <- 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.

```
lm_fit <- lm(median_house_value ~ latitude + longitude + housing_median_age + total_rooms + t
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
-193918  -44327  -10232   30789  344646

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -3.203e+06  2.901e+05 -11.041  < 2e-16 ***
```

```
latitude                  -3.517e+04  3.345e+03 -10.515  < 2e-16 ***
longitude                 -3.746e+04  3.376e+03 -11.096  < 2e-16 ***
housing_median_age         1.144e+03  1.426e+02   8.027 1.67e-15 ***
total_rooms               -8.063e+00  2.314e+00  -3.484 0.000504 ***
total_bedrooms             1.568e+02  1.151e+01  13.622  < 2e-16 ***
population                -3.821e+01  3.271e+00 -11.682  < 2e-16 ***
median_income              3.981e+04  1.028e+03  38.709  < 2e-16 ***
ocean_proximityINLAND     -3.001e+04  5.671e+03  -5.292 1.34e-07 ***
ocean_proximityISLAND      2.053e+05  6.837e+04   3.002 0.002712 **
ocean_proximityNEAR BAY   -1.759e+04  5.968e+03  -2.947 0.003243 **
ocean_proximityNEAR OCEAN  1.064e+03  4.901e+03   0.217 0.828219
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 68200 on 2031 degrees of freedom
Multiple R-squared:  0.6711,    Adjusted R-squared:  0.6693
F-statistic: 376.7 on 11 and 2031 DF,  p-value: < 2.2e-16
```

Interpretation of Coefficients: Latitude: When all covariates are held constant, there is a 2.539e+04 decrease in median house value for every unit increase in latitude. Longitude: When all covariates are held constant, there is a 2.681e+04 decrease in median house value for every unit increase in longitude. Median House Age: When all covariates are held constant, there is a 1.074e+03 increase in median house value for every unit increase in median house age. Total Rooms: When all covariates are held constant, there is a 6.159e+00 decrease in median house value for every unit increase in total rooms. Total Bedrooms: When all covariates are held constant, there is a 1.353e+02 increase in median house value for every unit increase in total bedrooms. Population: When all covariates are held constant, there is a 3.413e+01 decrease in median house value for every unit increase in population. Median Income: When all covariates are held constant, there is a 3.936e+04 increase in median house value for every unit increase in population. Ocean proximity: When all covariates are held constant, there is a -4.018e+04, 1.324e+05, -2.522e+03, and 4.349e+03 change in median house value when a resident lives inland, on an island, near bay, and near ocean respectively. It should be noted living near a bay is the only non-significant predictor compared to the rest of the covariates.

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)
rmse_vals <- rmse(df_test$median_house_value, lm_predictions)
```

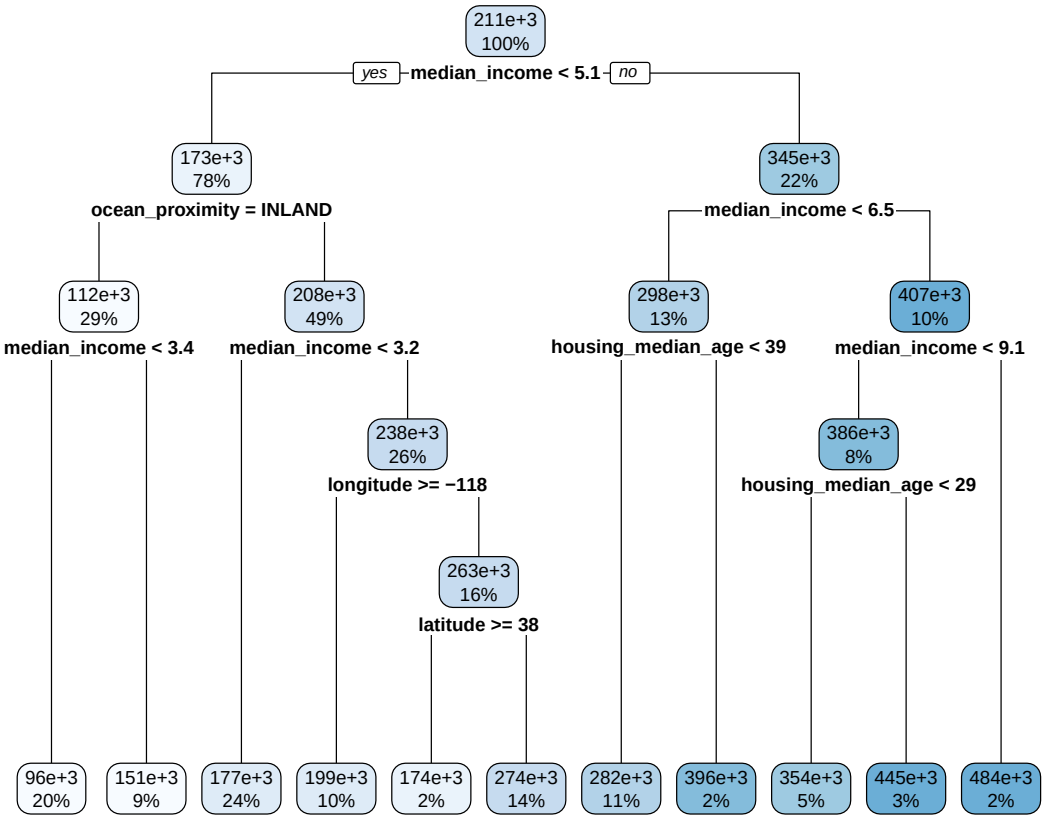
1.6 (5 points)

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

```r
rpart_fit <- rpart(median_house_value ~ latitude + longitude + housing_median_age + total_roo
rpart_predictions <- predict(rpart_fit, newdata = df_test)
```

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

```r
library(rpart.plot)
rpart.plot(rpart_fit, box.palette = "Blues")
```



Report the root mean squared error on the test set.

```
rpart_predictions_rmse <- rmse(df_test$median_house_value, rpart_predictions)
print(rpart_predictions_rmse)
```

```
[1] 74346.43
```

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.

```
svm_fit <- svm(median_house_value ~ latitude + longitude + housing_median_age + total_rooms +
svm_predictions <- predict(svm_fit, newdata = df_test)
```

```
svm_rmse <- rmse(svm_predictions, df_test$median_house_value)
svm_rmse
```

```
[1] 61064.47
```

1.8 (25 points)

Initialize a neural network model architecture:

```
NNet <- nn_module(
    initialize = function(p, q1, q2, q3){
      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){
      x %>%
        self$hidden1() %>% self$activation() %>%
        self$hidden2() %>% self$activation() %>%
        self$hidden3() %>% self$activation() %>%
        self$output() %>% self$sigmoid()
    }
)
```

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.

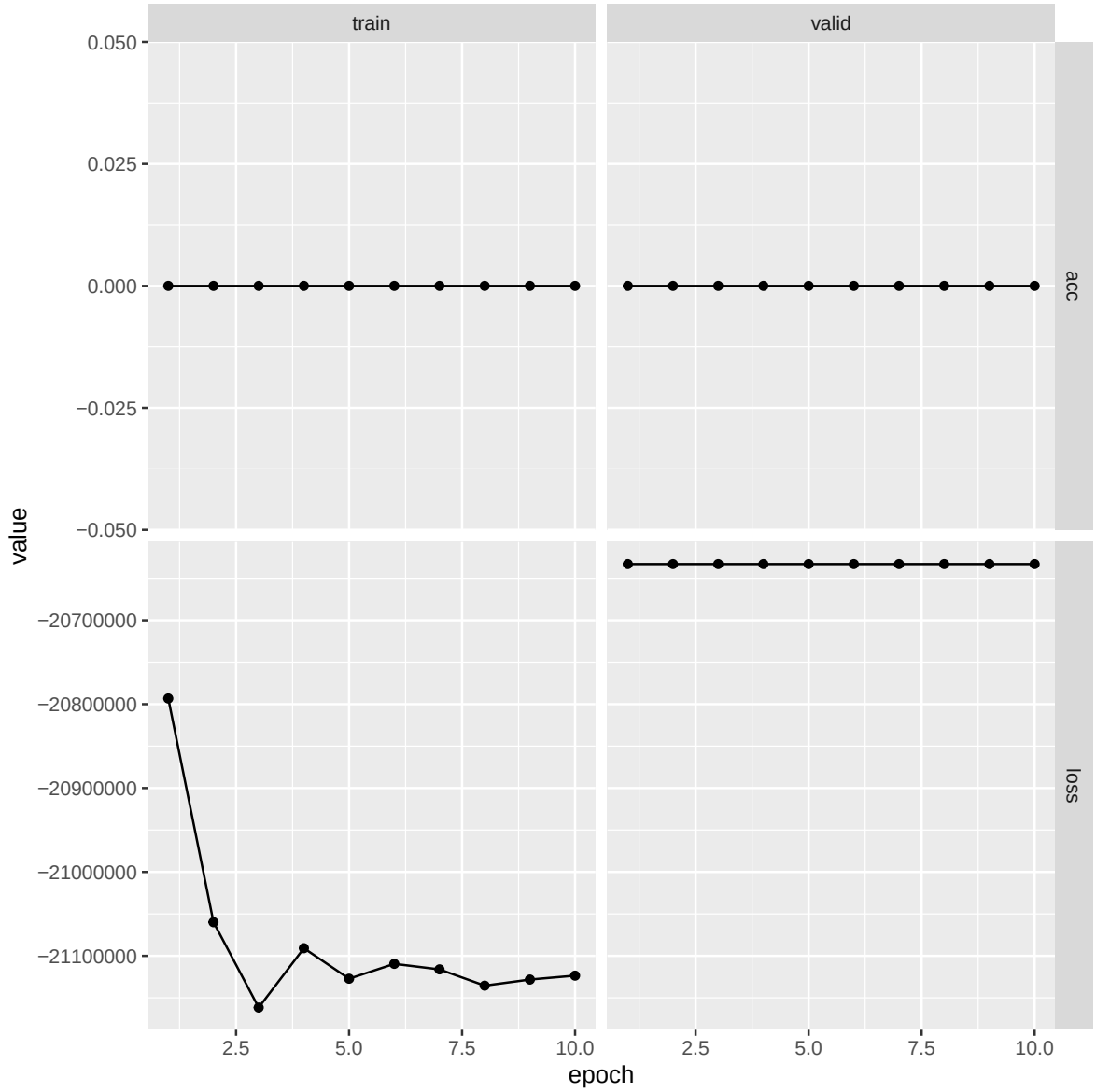
```r
covariate_matrix <- model.matrix(median_house_value ~ 0 + ., data =  df_train)

nnet_fit <- NNet %>%
  setup(loss = nn_bce_loss(), opt = optim_adam, metrics = list(luz_metric_accuracy())) %>%
  set_hparams(p = 13, q1 = 32, q2 = 16, q3 = 8) %>%
  set_opt_hparams(lr = 0.02) %>%
  fit(data = list(covariate_matrix, df_train %>% select(median_house_value) %>% as.matrix),
      valid_data = list(model.matrix(median_house_value ~ 0 + ., data =  df_test), df_test %>
      epochs  = 10,
      verbose = TRUE)
```

```
Epoch 1/10
Train metrics: Loss: -20793198 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 2/10
Train metrics: Loss: -21059852 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 3/10
Train metrics: Loss: -21161658 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 4/10
Train metrics: Loss: -21090882 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 5/10
Train metrics: Loss: -21127252 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 6/10
Train metrics: Loss: -21109514 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 7/10
Train metrics: Loss: -21116116 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 8/10
Train metrics: Loss: -21135516 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 9/10
Train metrics: Loss: -21128248 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
Epoch 10/10
Train metrics: Loss: -21123570 - Acc: 0
Valid metrics: Loss: -20633008 - Acc: 0
```

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

```
plot(nnet_fit)
```



Report the root mean squared error on the test set.

```
nnet_predictions <- predict(nnet_fit, model.matrix(median_house_value ~ 0 + ., data = df_test
```

```r
nnet_rmse <- rmse(nnet_predictions, df_test$median_house_value)
nnet_rmse
```

```
[1] 236302.8
```

> **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
rmse_df <- data.frame(Model = c("Linear Regression", "Decision Tree", "SVM" , "Neural Networ

rmse_df
```

```
              Model      RMSE
1 Linear Regression  69082.77
2     Decision Tree  74346.43
3               SVM  61064.47
4    Neural Network 236302.83
```

The SVM performed the best. This is likely because this is not a big data set, and we have not fine tuned the hyper parameters for the neural network.

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(readr)
df2 <- 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.
```

```
df2 <- df2 %>%
  mutate(across(where(is.character), as.factor)) %>%
  rename_all(tolower) %>%
  na.omit()
```

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 <- df2[-test_ind,]
df_test2  <- df2[test_ind,]
df_train2 <- na.omit(df_train2)
df_test2 <- na.omit(df_test2)
```

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(predicted, expected) {
      confusion_matrix <- table(predicted,expected)
    accuracy <- (confusion_matrix[1] + confusion_matrix[4]) / length(expected) * 100 #
    error <- 100 - accuracy # Insert your code here
    total_false_positives <- confusion_matrix[3]
    total_true_positives <- confusion_matrix[4]
    total_false_negatives <- confusion_matrix[2]
    total_true_negatives <- confusion_matrix[1]
    false_positive_rate <- total_false_positives / (total_false_positives + total_true_negati
    false_negative_rate <- total_false_negatives / (total_false_negatives + total_true_positi

    return(
    data.frame(
    accuracy = accuracy,
    error=error,
    false_positive_rate = false_positive_rate,
    false_negative_rate = false_negative_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.

```
glm_fit <- glm(spam ~., data = df_train2)
spam_predictions <- predict(glm_fit, newdata = df_test2, type = "response")
predicted_factors <- ifelse(spam_predictions > 0.5, 1, 0)
glm_classes <- overview(predicted_factors, df_test2$spam)
glm_classes
```

```
  accuracy   error false_positive_rate false_negative_rate
1  89.1258 10.8742           0.1360759          0.05228758
```

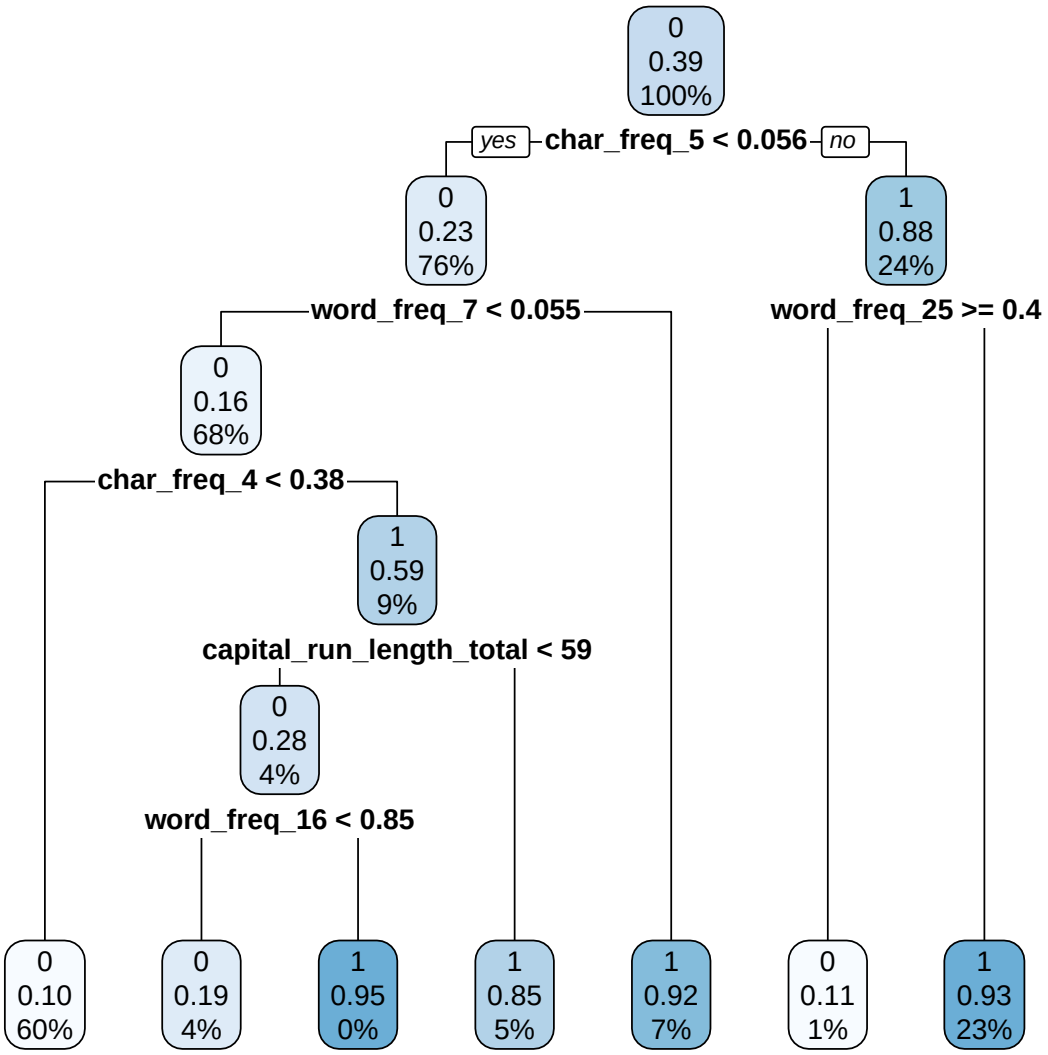
2.4 (5 points)

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 <- rpart(spam ~. , data = df_train2, method = "class")
rpart_predictions2 <- predict(rpart_classes, newdata = df_test2, type = "class")
```

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

```
rpart.plot(rpart_classes, box.palette = "Blues")
```



Report the prediction accuracy on the test set.

```
rpart_classes <- overview(rpart_predictions2, df_test2$spam)
rpart_classes
```

```
  accuracy    error false_positive_rate false_negative_rate
1 90.40512 9.594883           0.1066667          0.07692308
```

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`.

```
svm_fit2 <- svm(spam ~. , data = df_train2, type = "C-classification")
```

Report the prediction accuracy on the test set.

```
svm_predictions2 <- predict(svm_fit2, newdata = df_test2, type  = "class")
svm_classes <- overview(svm_predictions2, df_test2$spam)
svm_classes
```

```
  accuracy    error false_positive_rate false_negative_rate
1 92.96375 7.036247          0.07241379          0.06703911
```

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.

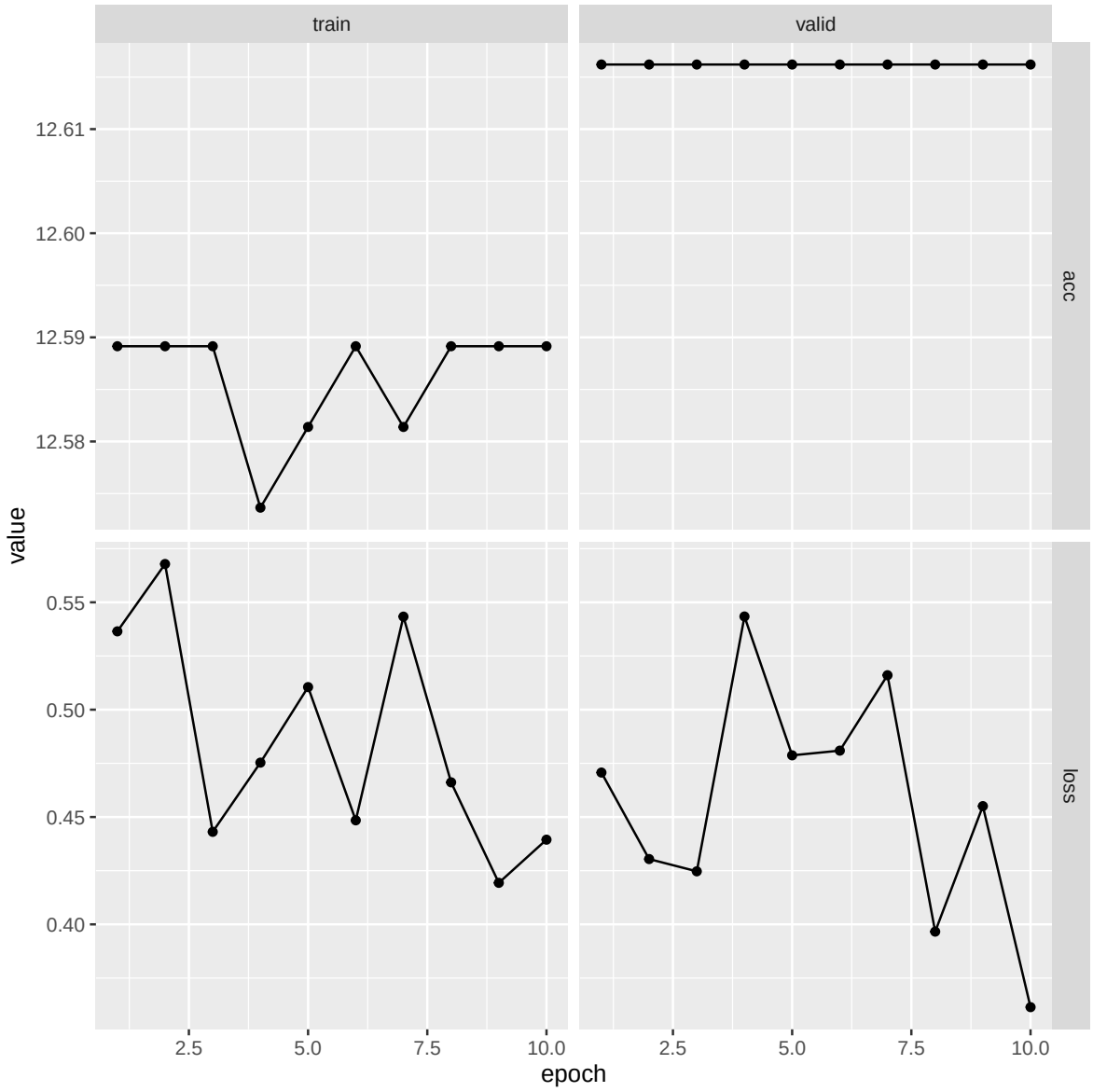
Use the `model.matrix` function to create the covariate matrix and `luz` package for fitting the network with $32, 16, 8$ nodes in each of the three hidden layers.

```
covariate_matrix2 <- model.matrix(spam ~ 0 + ., data =  df_train2)

nnet_fit2 <- NNet %>%
  setup(loss = nn_bce_loss(), opt = optim_adam , metrics = list(luz_metric_accuracy())) %>%
  set_hparams(p = ncol(covariate_matrix2), q1 = 32, q2 = 16, q3 = 8) %>%
  set_opt_hparams(lr = 0.02) %>%
  fit(data = list(covariate_matrix2, df_train2 %>% select(spam) %>% as.matrix),
      valid_data = list(model.matrix(spam ~ 0 + ., data =  df_test2), df_test2 %>% select(spa
      epochs  = 10,verbose = TRUE)
```

```
Epoch 1/10
Train metrics: Loss: 0.5365 - Acc: 12.5891
Valid metrics: Loss: 0.4707 - Acc: 12.6162
Epoch 2/10
Train metrics: Loss: 0.5678 - Acc: 12.5891
Valid metrics: Loss: 0.4304 - Acc: 12.6162
Epoch 3/10
Train metrics: Loss: 0.4431 - Acc: 12.5891
Valid metrics: Loss: 0.4247 - Acc: 12.6162
Epoch 4/10
Train metrics: Loss: 0.4753 - Acc: 12.5736
Valid metrics: Loss: 0.5434 - Acc: 12.6162
Epoch 5/10
Train metrics: Loss: 0.5105 - Acc: 12.5814
Valid metrics: Loss: 0.4787 - Acc: 12.6162
Epoch 6/10
Train metrics: Loss: 0.4485 - Acc: 12.5891
Valid metrics: Loss: 0.4809 - Acc: 12.6162
Epoch 7/10
Train metrics: Loss: 0.5433 - Acc: 12.5814
Valid metrics: Loss: 0.5161 - Acc: 12.6162
Epoch 8/10
Train metrics: Loss: 0.4661 - Acc: 12.5891
Valid metrics: Loss: 0.3966 - Acc: 12.6162
Epoch 9/10
Train metrics: Loss: 0.4193 - Acc: 12.5891
Valid metrics: Loss: 0.4551 - Acc: 12.6162
Epoch 10/10
Train metrics: Loss: 0.4395 - Acc: 12.5891
Valid metrics: Loss: 0.3614 - Acc: 12.6162
```

```
plot(nnet_fit2)
```



2.7 (5 points)

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

```
nnet_predictions <- predict(nnet_fit2, model.matrix(spam ~ 0 + ., data = df_test2)) %>% as.ar
nn_classes2 <- overview(nnet_predictions, df_test2$spam)
```

```r
summary <- data.frame(Model = c("Logistic regression" , "Decision tree", "SVM", "Neural Netw
summary
```

```
                Model   Accurage
1 Logistic regression 89.1257996
2       Decision tree 90.4051173
3                 SVM 92.9637527
4      Neural Network  0.4264392
```

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.

The neural network had the lowest loss value, thus I would think to pick the neural network; however, the svm model had a much higher accuracy on the test data compared to the nnet. Thus, the svm would be more likely to be used if the test data is not biased.

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)),
```

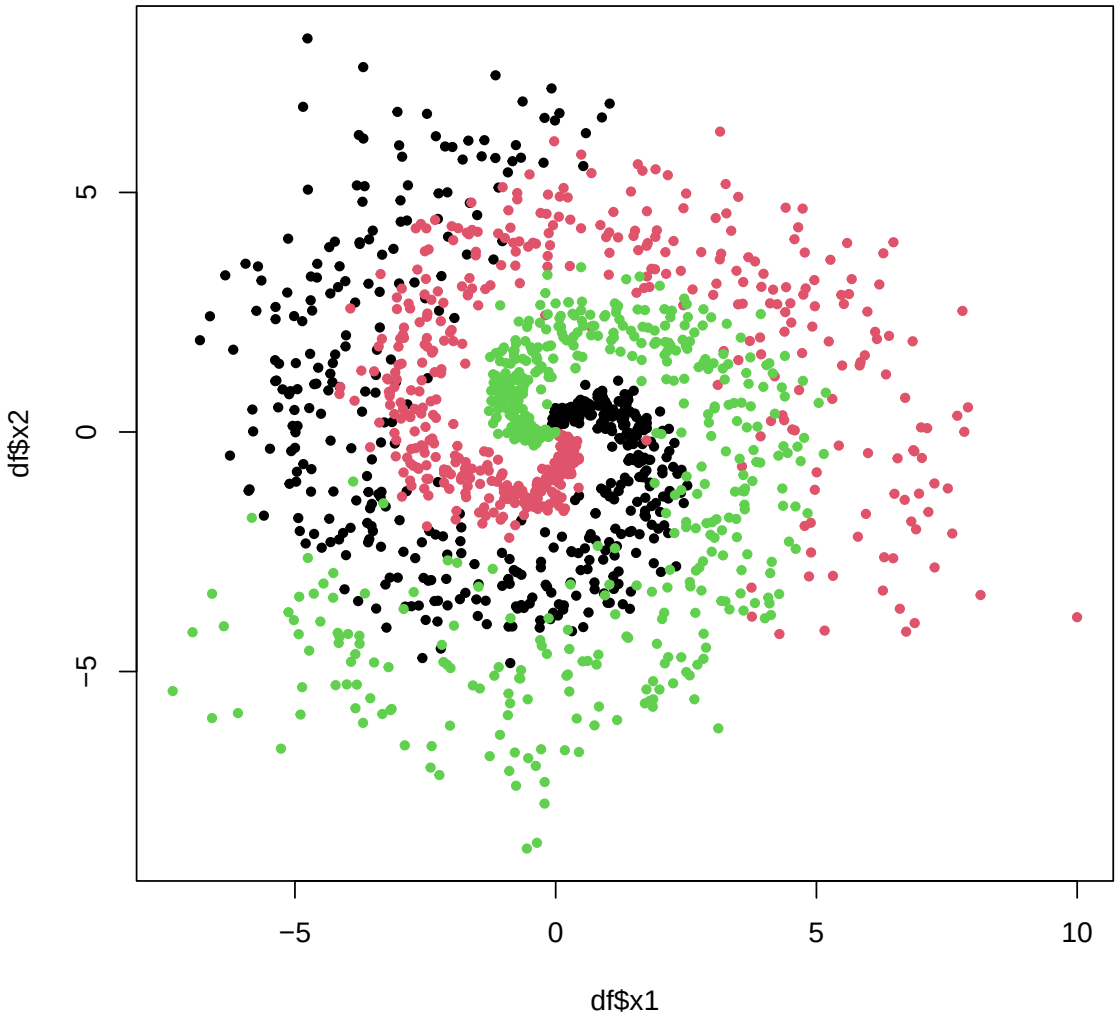
```
      t * (sin(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  x2 <- c(
      t * (cos(t) + rnorm(n, 0, noise)),
      t * (cos(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (cos(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  y <- as.factor(
    c(
      rep(0, n),
      rep(1, n),
      rep(2, n)
   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

```
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`.

```
x_values <- seq(from = -10, to = 10, length.out = 100)
grid <- expand.grid(x1 = x_values, x2 = x_values) # Insert your code here

n <- nrow(grid)
y <- as.factor(c(rep(0, n), rep(1, n), rep(2, n)))
```

```r
# Combine x1, x2, and y into a tibble
df_test2 <- cbind(grid, y)
df_test2 <- as_tibble(df_test2)
```

3.2 (10 points)

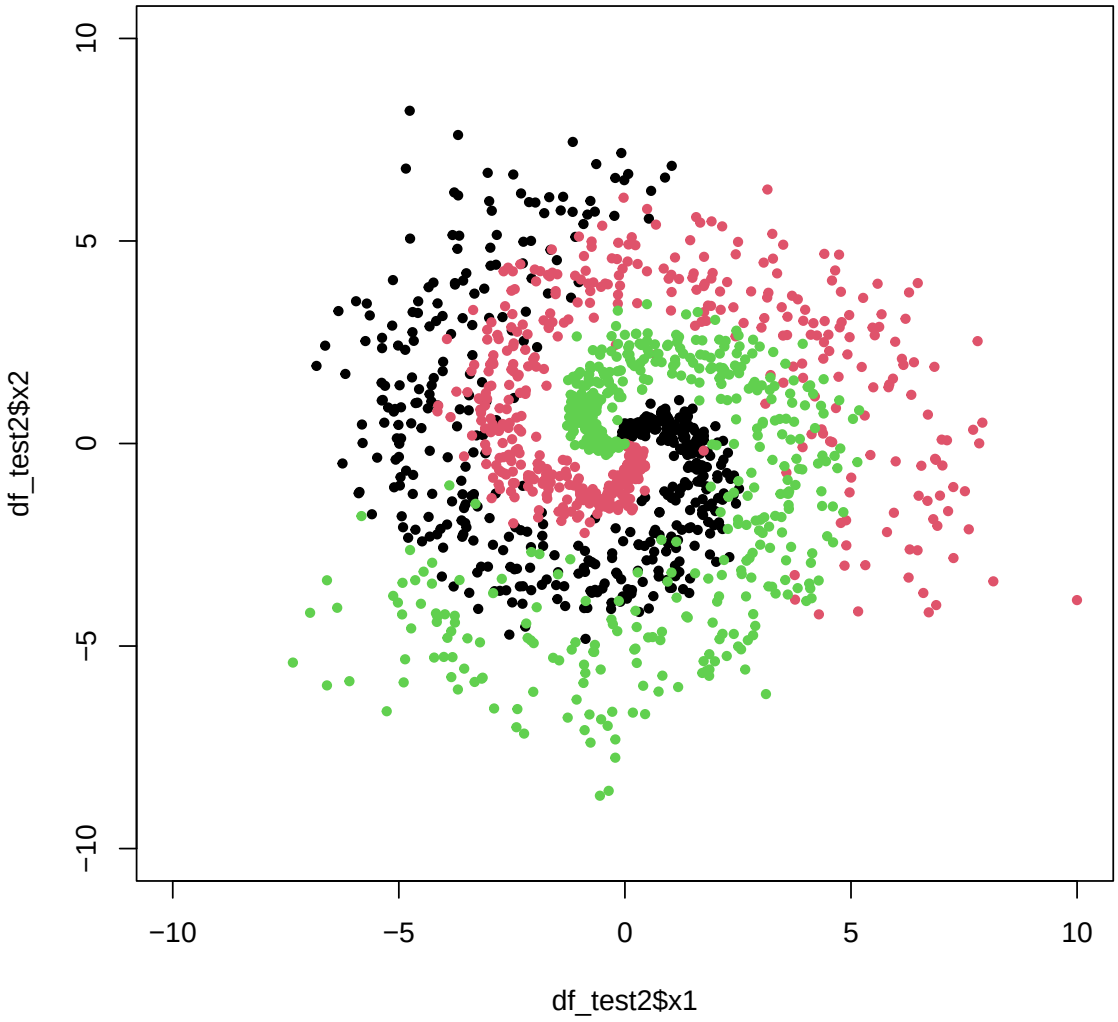
Fit a classification tree model to predict the `y` variable using the `x1` and `x2` predictors, and plot the decision boundary.

```r
rpart_fit <- rpart(y ~ x1 + x2, data = df, method = "class") # Insert your code here
rpart_classes <- predict(rpart_fit, df_test2)
```

Plot the decision boundary using the following function:

```r
plot_decision_boundary <- function(predictions){
  plot(
    df_test2$x1, df_test2$x2,
    col = predictions,
    pch = 0
  )
  points(
    df$x1, df$x2,
    col = df$y,
    pch = 20
  )
}
```

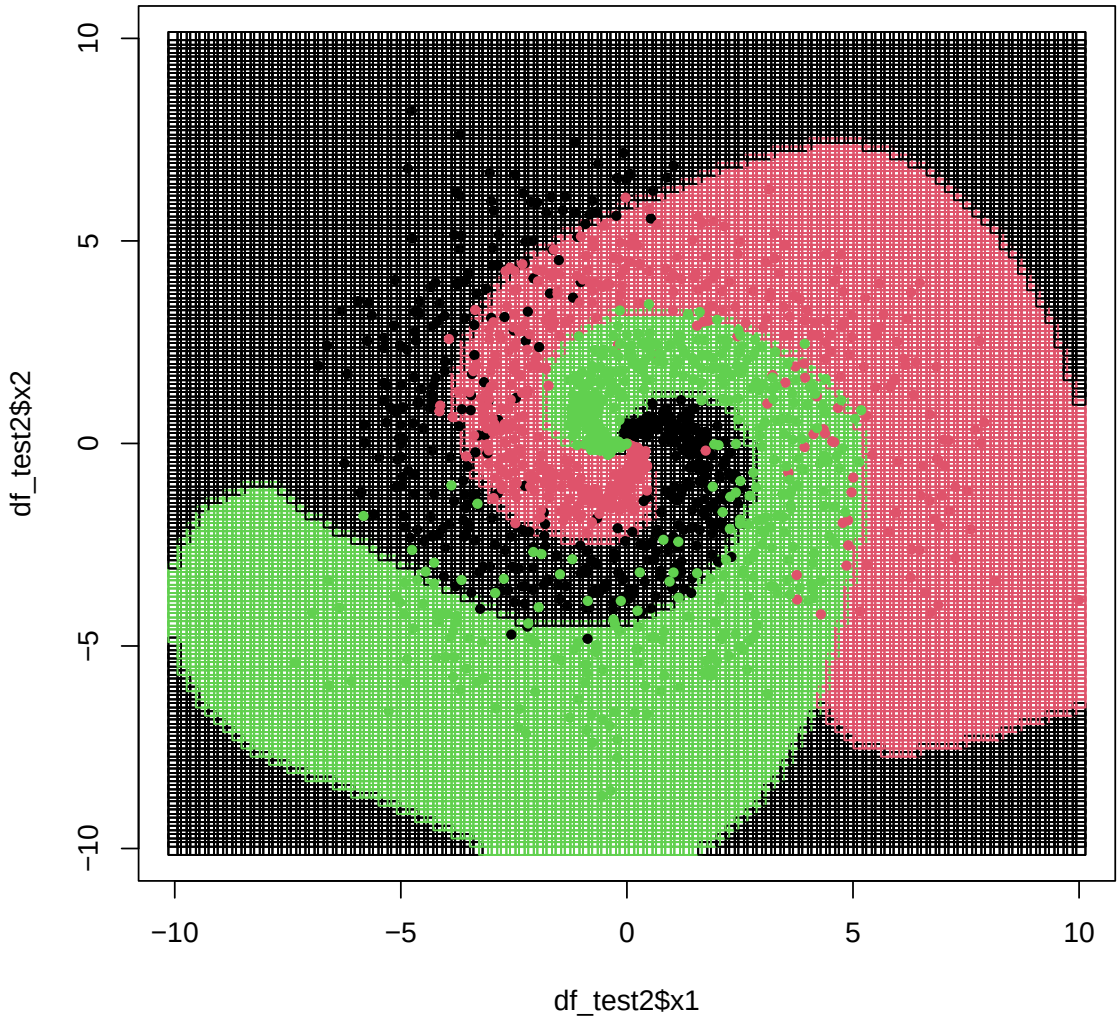
```r
plot_decision_boundary(rpart_classes)
```



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
svm_fit <- svm(y ~ x1 + x2, data = df, type = "C-classification")
svm_classes <- predict(svm_fit, newdata = df_test2)
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()
  }
)


df_test2 <- df_test2 %>% mutate(y = as.numeric(y))

fit_1 <- NN1 %>%
  setup(
    loss = nn_cross_entropy_loss(), opt = optim_adam, metric = luz_metric_accuracy()
  ) %>%
  set_hparams(
    p = ncol(df), q1 = 10, o = 3
```

```
  ) %>%
  set_opt_hparams(
    lr = 0.02
  ) %>%
  fit(
    data = list(
      df %>% select(x1,x2) %>% as.matrix, df %>% select(y)
    ),
    valid = list(
            df_test2 %>% select(x1,x2) %>% as.matrix, df_test2 %>% select(y) ),
    epochs = 50,
    verbose = TRUE
  )
```

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

```
test_matrix <- df_test2 %>% select(x1, x2) %>% as.matrix

fit_1_predictions <- predict(fit_1, test_matrix) %>%
  as.array()

plot_decision_boundary(overview(fit_1_predicitons, df_test2$y))
```

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

```
      self$hidden1() %>%
      self$activation() %>%
      self$output() %>%
      self$sigmoid()
  }
)

fit_0 <- NN0 %>%
  setup(loss = nn_cross_entropy_loss(), opt = optim_adam, metric = luz_metric_accuracy()) %>%
  set_hparams( p = ncol(df), o = 3) %>%
  set_opt_hparams(lr = 0.02) %>%
  fit(data = list(
     df %>% select(x1,x2) %>% as.matrix, df %>% select(y)
    ),
    valid = list(
            df_test2 %>% select(x1,x2) %>% as.matrix, df_test2 %>% select(y)
            ),
    epochs = 50,
    verbose = TRUE)
```

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

```
test_matrix <- df_test2 %>% select(x1, x2) %>% as.matrix

fit_0_predictions <- predict(fit_0, test_matrix) %>%
  as.array()

plot_decision_boundary(overview(fit_0_predicitons, df_test2$y))
```

3.6 (10 points)

Fit a neural network with **2 hidden layers** to predict the `y` variable using the `x1` and `x2` predictors.

```
NN2 <- 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$activation <- nn_relu()
    self$sigmoid <- nn_sigmoid()
  },
  forward = function(x){
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$hidden2() %>%
      self$activation() %>%
      self$output() %>%
      self$sigmoid()
  }
)

fit_2 <- NN3 %>%
  setup(loss = nn_cross_entropy_loss(), opt = optim_adam, metric = luz_metric_accuracy()) %>%
  set_hparams(p = ncol(df), q1 = 10, q2 = 10, o = 3) %>%
  set_opt_params(lr = 0.02) %>%
  fit(data = list(
      model.matrix(y ~ 0 + ., data = df), df %>% select(y) %>% as.matrix
    ),
    valid = list(
            model.matrix(y ~ 0 + ., data = df_test2), df_test2 %>% select(y) %>% as.matrix
            ),
    epochs = 50,
    verbose = TRUE)
```

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

```
fit_2_predictions <- predict(fit_2, test_matrix) %>%
  as.array()

plot_decision_boundary(overview(fit_2_predicitons, df_test2$y))
```

3.7 (5 points)

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

—

Session Information

Print your R session information using the following command

```
sessionInfo()
```

```
R version 4.3.1 (2023-06-16 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 datasets  utils     methods   base

other attached packages:
 [1] luz_0.4.0        torch_0.12.0     e1071_1.7-14     rpart.plot_3.1.2
 [5] rpart_4.1.23     caret_6.0-94     lattice_0.21-8   ggplot2_3.5.0
 [9] corrplot_0.92    magrittr_2.0.3   broom_1.0.5      purrr_1.0.2
[13] tidyr_1.3.1      readr_2.1.5      dplyr_1.1.4      tibble_3.2.1

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.1     timeDate_4032.109    farver_2.1.1
 [4] fastmap_1.1.1        pROC_1.18.5          digest_0.6.35
 [7] timechange_0.3.0     lifecycle_1.0.4      ellipsis_0.3.2
[10] processx_3.8.4       survival_3.5-5       compiler_4.3.1
[13] progress_1.2.3       rlang_1.1.3          tools_4.3.1
[16] utf8_1.2.4           yaml_2.3.8           data.table_1.15.2
[19] knitr_1.45           labeling_0.4.3       prettyunits_1.2.0
```

```
[22] bit_4.0.5            plyr_1.8.9           withr_3.0.0
[25] nnet_7.3-19          grid_4.3.1           stats4_4.3.1
[28] fansi_1.0.6          colorspace_2.1-0     future_1.33.2
[31] globals_0.16.3       scales_1.3.0         iterators_1.0.14
[34] MASS_7.3-60          zeallot_0.1.0        cli_3.6.2
[37] crayon_1.5.2         rmarkdown_2.26       generics_0.1.3
[40] future.apply_1.11.2  reshape2_1.4.4       tzdb_0.4.0
[43] proxy_0.4-27         stringr_1.5.1        splines_4.3.1
[46] parallel_4.3.1       coro_1.0.4           vctrs_0.6.5
[49] hardhat_1.3.1        Matrix_1.5-4.1       jsonlite_1.8.8
[52] callr_3.7.6          hms_1.1.3            bit64_4.0.5
[55] listenv_0.9.1        foreach_1.5.2        gower_1.0.1
[58] recipes_1.0.10       glue_1.7.0           parallelly_1.37.1
[61] ps_1.7.6             codetools_0.2-19     lubridate_1.9.3
[64] stringi_1.8.3        gtable_0.3.4         munsell_0.5.0
[67] pillar_1.9.0         htmltools_0.5.8      ipred_0.9-14
[70] lava_1.8.0           R6_2.5.1             vroom_1.6.5
[73] evaluate_0.23        backports_1.4.1      renv_1.0.3
[76] class_7.3-22         Rcpp_1.0.12          nlme_3.1-162
[79] prodlim_2023.08.28   xfun_0.43            fs_1.6.3
[82] pkgconfig_2.0.3      ModelMetrics_1.2.2.2
```