Homework 5

Jiaxi li

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_with(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.
```

```r
df$ocean_proximity <- as.factor(df$ocean_proximity)
head(df) # Insert your code here
```

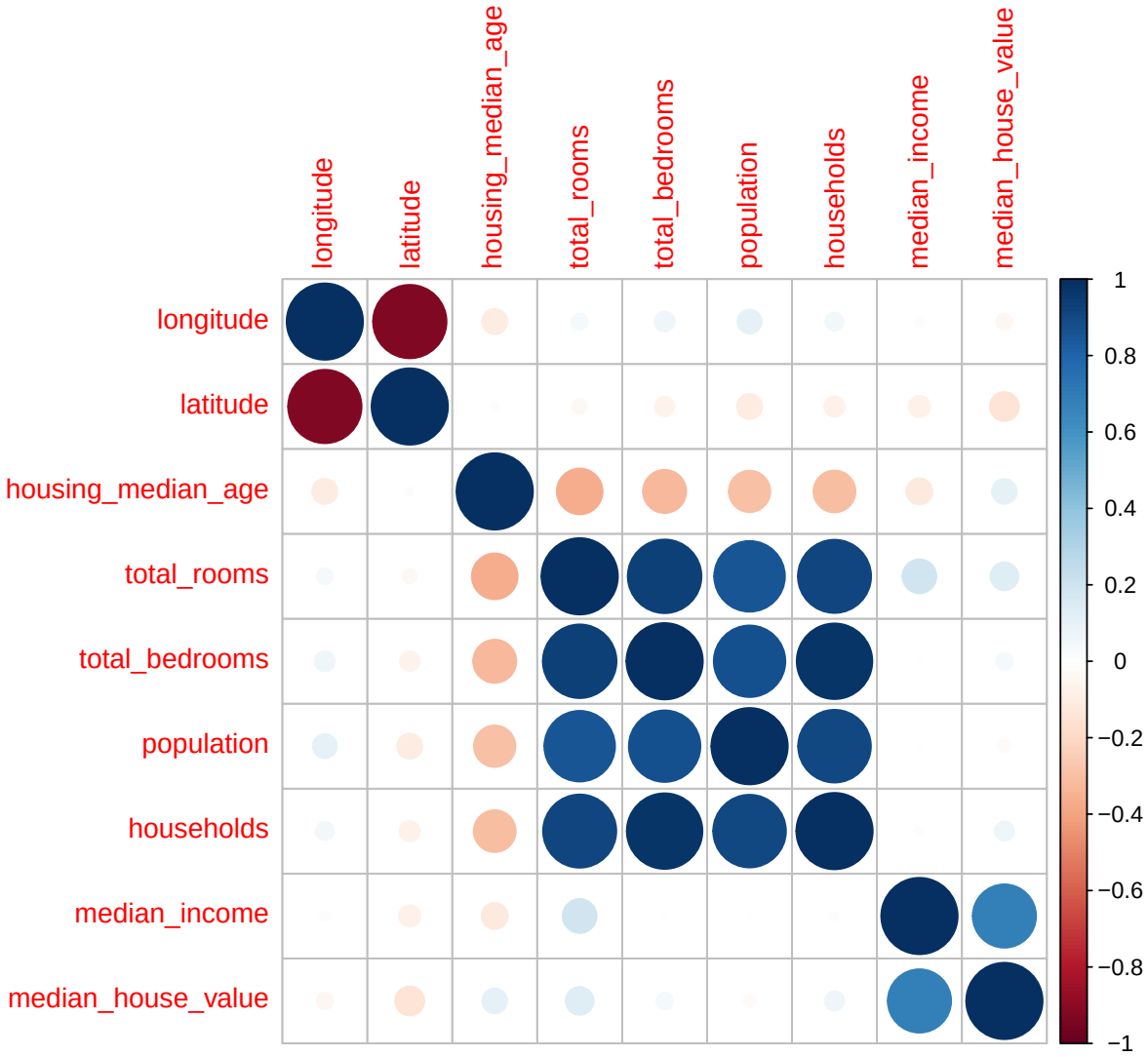
```
# A tibble: 6 x 10
  longitude latitude housing_median_age total_rooms total_bedrooms population
      <dbl>    <dbl>              <dbl>       <dbl>          <dbl>      <dbl>
1     -122.     37.9                 41         880            129        322
2     -122.     37.9                 21        7099           1106       2401
3     -122.     37.8                 52        1467            190        496
4     -122.     37.8                 52        1274            235        558
5     -122.     37.8                 52        1627            280        565
6     -122.     37.8                 52         919            213        413
# i 4 more variables: households <dbl>, median_income <dbl>,
#   median_house_value <dbl>, ocean_proximity <fct>
```

1.2 (2.5 points)

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

```r
numeric_columns <- df %>% select_if(is.numeric)
cor_matrix <- cor(numeric_columns)
```

```r
corrplot(cor_matrix, method = "circle")
```



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, ] # Insert your code here
df_test <- df[test_ind, ] # Insert your code here
```

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
df_train$ocean_proximity <- as.factor(df_train$ocean_proximity)
```

```r
lm_fit <- lm(median_house_value ~ . - households, data = df_train) # Insert your code here
summary(lm_fit) # Insert your code here
```

```
Call:
lm(formula = median_house_value ~ . - households, 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 ***
longitude                 -2.681e+04  1.060e+03 -25.305  < 2e-16 ***
latitude                  -2.539e+04  1.047e+03 -24.244  < 2e-16 ***
housing_median_age         1.074e+03  4.616e+01  23.261  < 2e-16 ***
total_rooms               -6.159e+00  8.431e-01  -7.306 2.87e-13 ***
total_bedrooms             1.353e+02  4.254e+00  31.804  < 2e-16 ***
population                -3.413e+01  9.838e-01 -34.694  < 2e-16 ***
median_income              3.936e+04  3.573e+02 110.154  < 2e-16 ***
ocean_proximityINLAND     -4.018e+04  1.836e+03 -21.891  < 2e-16 ***
ocean_proximityISLAND      1.324e+05  3.442e+04   3.847  0.00012 ***
ocean_proximityNEAR BAY   -2.522e+03  2.022e+03  -1.247  0.21226
ocean_proximityNEAR OCEAN  4.349e+03  1.658e+03   2.622  0.00875 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

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) # Insert your code here
rmse_lm <- rmse(df_test$median_house_value, lm_predictions)
print(rmse_lm)
```

```
[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.

```r
rpart_fit <- rpart(median_house_value ~ . - households, data = df_train) # Insert your cod
summary(rpart_fit) # Insert your code here
```

```
Call:
rpart(formula = median_house_value ~ . - households, data = df_train)
  n= 18390

          CP nsplit rel error    xerror        xstd
1 0.30492555      0 1.0000000 1.0000592 0.011312769
2 0.12731459      1 0.6950744 0.6979744 0.008826537
3 0.05953042      2 0.5677599 0.5701815 0.007578389
4 0.04048333      3 0.5082294 0.5107395 0.007457761
5 0.01381152      4 0.4677461 0.4720499 0.007116065
6 0.01380725      5 0.4539346 0.4570311 0.007005307
7 0.01315679      6 0.4401273 0.4489693 0.006943697
8 0.01000000      7 0.4269706 0.4277617 0.006761393

Variable importance
     median_income    ocean_proximity           latitude          longitude
                64                 22                 7                  3
housing_median_age        total_rooms         population
                 2                  1                  1

Node number 1: 18390 observations,    complexity param=0.3049256
  mean=206394.2, MSE=1.324089e+10
  left son=2 (14519 obs) right son=3 (3871 obs)
  Primary splits:
      median_income   < 5.03515  to the left,  improve=0.30492560, (0 missing)
      ocean_proximity splits as  RLRRR,        improve=0.23381360, (0 missing)
      latitude        < 37.935   to the right, improve=0.06666687, (0 missing)
      longitude       < -121.865 to the right, improve=0.03779756, (0 missing)
      total_rooms     < 2015.5   to the left,  improve=0.02725354, (0 missing)
  Surrogate splits:
      total_rooms < 13799    to the left,  agree=0.791, adj=0.005, (0 split)

Node number 2: 14519 observations,    complexity param=0.1273146
  mean=173584.7, MSE=8.384653e+09
  left son=4 (5255 obs) right son=5 (9264 obs)
  Primary splits:
      ocean_proximity    splits as  RLRRR,        improve=0.25465680, (0 missing)
      median_income      < 3.12885  to the left,  improve=0.16224220, (0 missing)
      latitude           < 37.925   to the right, improve=0.06160970, (0 missing)
```

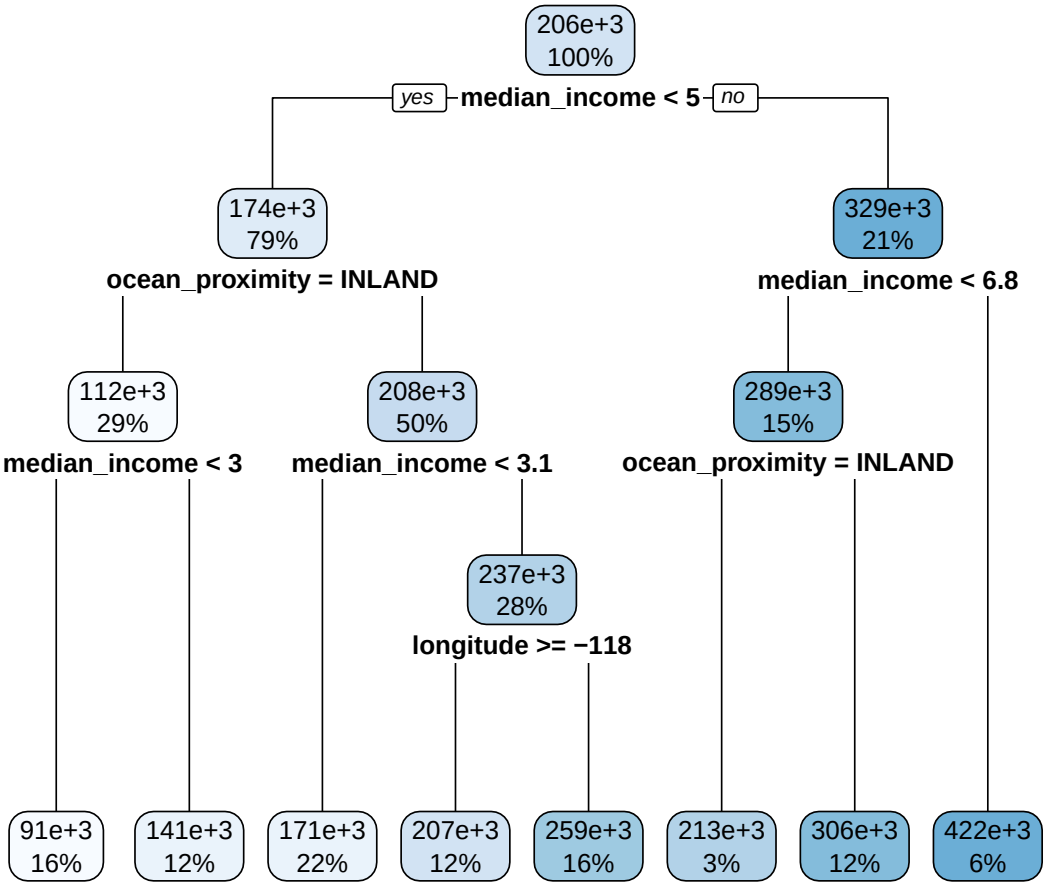
```
      housing_median_age < 51.5     to the left,  improve=0.03829309, (0 missing)
      longitude          < -121.865 to the right, improve=0.03606499, (0 missing)
  Surrogate splits:
      latitude           < 37.955   to the right, agree=0.727, adj=0.246, (0 split)
      housing_median_age < 15.5     to the left,  agree=0.664, adj=0.071, (0 split)
      longitude          < -116.905 to the right, agree=0.659, adj=0.057, (0 split)
      population         < 343.5    to the left,  agree=0.649, adj=0.029, (0 split)
      median_income      < 1.79275  to the left,  agree=0.644, adj=0.016, (0 split)

Node number 3: 3871 observations,    complexity param=0.05953042
  mean=329452.8, MSE=1.22743e+10
  left son=6 (2701 obs) right son=7 (1170 obs)
  Primary splits:
      median_income      < 6.81955  to the left,  improve=0.30508300, (0 missing)
      ocean_proximity    splits as  RL-RR,        improve=0.12921480, (0 missing)
      housing_median_age < 27.5     to the left,  improve=0.08620557, (0 missing)
      latitude           < 37.965   to the right, improve=0.06037955, (0 missing)
      longitude          < -118.035 to the right, improve=0.03922245, (0 missing)
  Surrogate splits:
      total_bedrooms < 35       to the right, agree=0.702, adj=0.015, (0 split)
      population     < 64       to the right, agree=0.702, adj=0.014, (0 split)
      total_rooms    < 208      to the right, agree=0.702, adj=0.013, (0 split)

Node number 4: 5255 observations,    complexity param=0.01315679
  mean=112232.1, MSE=2.870763e+09
  left son=8 (3027 obs) right son=9 (2228 obs)
  Primary splits:
      median_income      < 3.03555  to the left,  improve=0.21236300, (0 missing)
      latitude           < 34.825   to the right, improve=0.04093847, (0 missing)
      longitude          < -118.295 to the left,  improve=0.03080534, (0 missing)
      total_rooms        < 2296     to the left,  improve=0.02651350, (0 missing)
      housing_median_age < 17.5     to the right, improve=0.02167716, (0 missing)
  Surrogate splits:
      housing_median_age < 14.5     to the right, agree=0.631, adj=0.129, (0 split)
      total_rooms        < 3012.5   to the left,  agree=0.623, adj=0.110, (0 split)
      population         < 2761.5   to the left,  agree=0.597, adj=0.049, (0 split)
      total_bedrooms     < 970      to the left,  agree=0.593, adj=0.041, (0 split)
      latitude           < 34.705   to the right, agree=0.589, adj=0.030, (0 split)

Node number 5: 9264 observations,    complexity param=0.04048333
  mean=208387, MSE=8.166e+09
  left son=10 (4061 obs) right son=11 (5203 obs)
  Primary splits:
```

```
      median_income      < 3.12285  to the left,  improve=0.13030680, (0 missing)
      longitude          < -118.305 to the right, improve=0.07512986, (0 missing)
      housing_median_age < 51.5     to the left,  improve=0.03855985, (0 missing)
      latitude           < 33.995   to the left,  improve=0.03798139, (0 missing)
      total_rooms        < 2032     to the left,  improve=0.02982308, (0 missing)
  Surrogate splits:
      total_rooms        < 1280.5   to the left,  agree=0.600, adj=0.088, (0 split)
      latitude           < 32.775   to the left,  agree=0.580, adj=0.041, (0 split)
      longitude          < -123.09  to the left,  agree=0.572, adj=0.024, (0 split)
      housing_median_age < 37.5     to the right, agree=0.572, adj=0.024, (0 split)
      total_bedrooms     < 125.5    to the left,  agree=0.567, adj=0.012, (0 split)

Node number 6: 2701 observations,    complexity param=0.01381152
  mean=289177.6, MSE=8.861805e+09
  left son=12 (479 obs) right son=13 (2222 obs)
  Primary splits:
      ocean_proximity    splits as  RL-RR,        improve=0.14050550, (0 missing)
      housing_median_age < 36.5     to the left,  improve=0.10106890, (0 missing)
      median_income      < 5.78465  to the left,  improve=0.07031948, (0 missing)
      latitude           < 37.955   to the right, improve=0.05584316, (0 missing)
      longitude          < -122.105 to the right, improve=0.04390701, (0 missing)
  Surrogate splits:
      latitude           < 38.335   to the right, agree=0.857, adj=0.192, (0 split)
      housing_median_age < 2.5      to the left,  agree=0.827, adj=0.023, (0 split)
      longitude          < -116.62  to the right, agree=0.826, adj=0.019, (0 split)

Node number 7: 1170 observations
  mean=422430.1, MSE=7.762749e+09

Node number 8: 3027 observations
  mean=91049.03, MSE=1.619675e+09

Node number 9: 2228 observations
  mean=141011.9, MSE=3.132597e+09

Node number 10: 4061 observations
  mean=171463.8, MSE=6.028421e+09

Node number 11: 5203 observations,    complexity param=0.01380725
  mean=237205.9, MSE=7.939788e+09
  left son=22 (2182 obs) right son=23 (3021 obs)
  Primary splits:
      longitude          < -118.275 to the right, improve=0.08138477, (0 missing)
```

```
      housing_median_age < 47.5     to the left,  improve=0.06867000, (0 missing)
      latitude           < 33.985   to the left,  improve=0.04674031, (0 missing)
      median_income      < 4.13035  to the left,  improve=0.02487397, (0 missing)
      total_bedrooms     < 418.5    to the left,  improve=0.02369109, (0 missing)
  Surrogate splits:
      latitude           < 34.145   to the left,  agree=0.884, adj=0.723, (0 split)
      ocean_proximity    splits as  L-RRR,        agree=0.597, adj=0.040, (0 split)
      housing_median_age < 7.5      to the left,  agree=0.591, adj=0.024, (0 split)
      population         < 3681.5   to the right, agree=0.582, adj=0.003, (0 split)
      total_rooms        < 11924    to the right, agree=0.582, adj=0.003, (0 split)

Node number 12: 479 observations
  mean=213177.9, MSE=6.03936e+09

Node number 13: 2222 observations
  mean=305561, MSE=7.956696e+09

Node number 22: 2182 observations
  mean=207295.4, MSE=5.051886e+09

Node number 23: 3021 observations
  mean=258809.6, MSE=8.912757e+09
```

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

```
rpart.plot(rpart_fit) # Insert your code here
```



Report the root mean squared error on the test set.

```
rpart_predictions <- predict(rpart_fit, newdata = df_test) # Insert your code here
rmse_dt <- rmse(df_test$median_house_value, rpart_predictions)
print(rmse_dt)
```

```
[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.

```
svm_fit <- svm(median_house_value ~ . - households, data = df_train) # Insert your code he
svm_predictions <- predict(svm_fit, newdata = df_test) # Insert your code here
svm_rmse <- rmse(df_test$median_house_value, svm_predictions)
print(svm_rmse)
```

```
[1] 56678.84
```

1.8 (25 points)

Initialize a neural network model architecture:

```
nn <- 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) # Single output node for binary classification
    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() # Sigmoid activation function for binary classifica
  }
)
```

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.

```
test_ind <- sample(1:nrow(df),23,replace = FALSE)
```

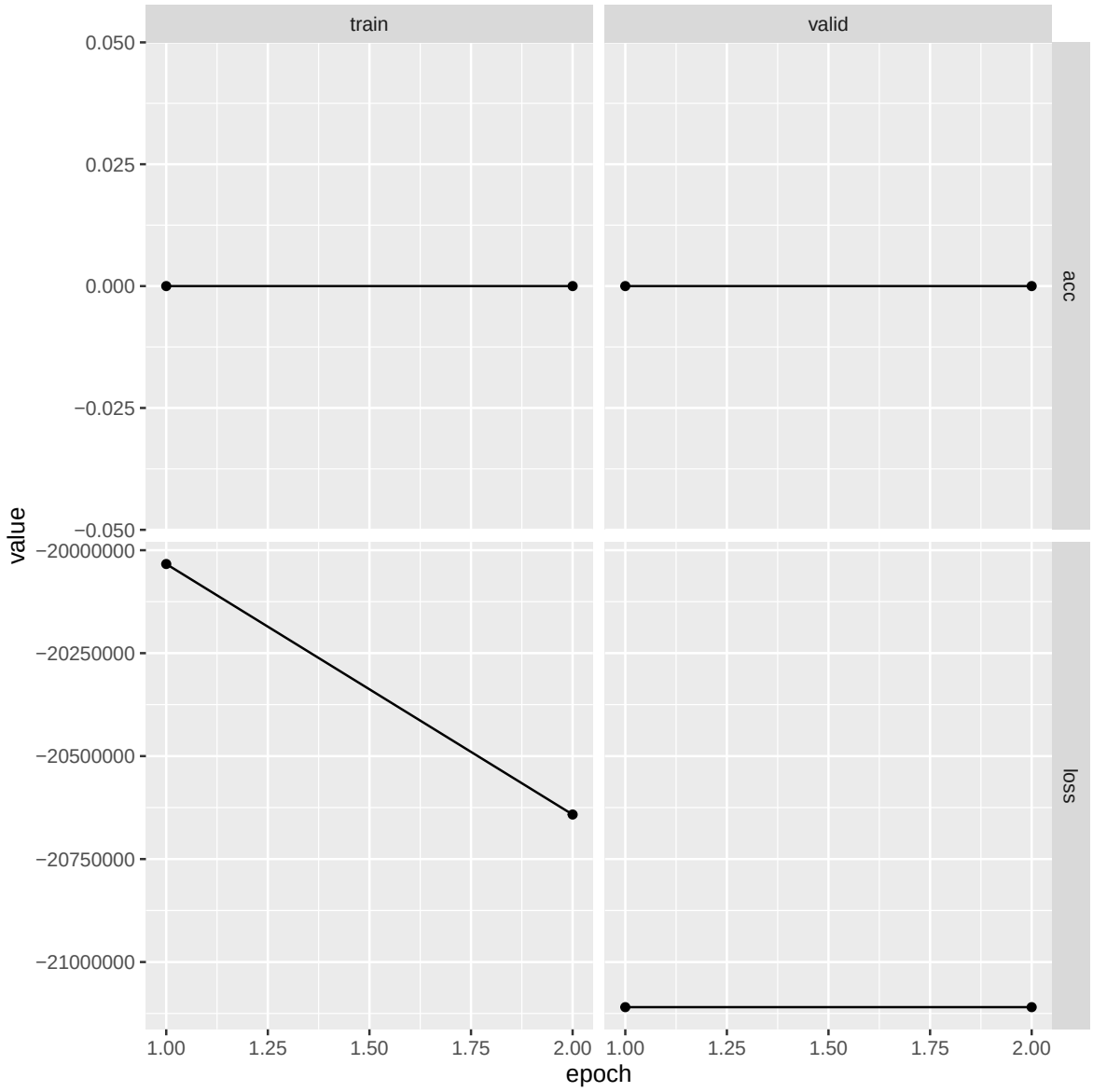
```
M <- model.matrix(median_house_value ~ 0 + . , data = df_train)
```

```
fit_nn <- nn %>%
  setup(loss = nn_bce_loss(),
        optimizer = optim_adam,
        metrics = list(luz_metric_accuracy())) %>%
  set_hparams(p = ncol(M),q1 = 32, q2 = 16, q3 = 8) %>%
  set_opt_hparams(lr= 0.001) %>%
  fit(data=list(
      model.matrix(median_house_value ~ 0 + .,data = df_train),df_train %>% select(median_
      valid_data = list(
      model.matrix(median_house_value ~ 0 + .,data = df_test),df_test %>% select(median_ho
      epochs = 2,
      verbose = TRUE
      )
```

```
Epoch 1/2
Train metrics: Loss: -20033504 - Acc: 0
Valid metrics: Loss: -21109578 - Acc: 0
Epoch 2/2
Train metrics: Loss: -20641816 - Acc: 0
Valid metrics: Loss: -21109578 - Acc: 0
```

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

```
plot(fit_nn) # Insert your code here
```



Report the root mean squared error on the test set.

```
test_result <- model.matrix(median_house_value ~ 0 + . , data = df_test)
```

```
nnet_predictions <- predict(fit_nn, test_result)
```

```r
nnet_predictions <- as.array(nnet_predictions)

# Assuming `df_test$median_house_value` is your true values for the test set
rmse <- sqrt(mean((df_test$median_house_value - nnet_predictions)^2))
print(rmse)
```

```
[1] 242118.3
```

> **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
model_comparisons <- data.frame(
  Model = c("Linear Regression", "Decision Tree", "SVM","nn_module"),
  RMSE = c(rmse_lm, rmse_dt, svm_rmse,rmse)
)

# Print the table
model_comparisons
```

```
             Model      RMSE
1 Linear Regression  68339.82
2     Decision Tree  75876.87
3               SVM  56678.84
4         nn_module 242118.32
```

```r
knitr::kable(model_comparisons, caption = "Model Comparison based on RMSE")
```

Table 1: Model Comparison based on RMSE

Model	RMSE
Linear Regression	68339.82
Decision Tree	75876.87
SVM	56678.84
nn_module	242118.32

svm was the best so far

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

```
spam_data <- read_csv("data/spambase.csv", show_col_types = FALSE) %>%
  mutate_if(is.character, as.factor) %>%
  rename_all(tolower) %>%
  drop_na()
```

2.2 (2.5 points)

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

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

spam_train <- spam_data[-test_ind, ]
spam_test <- spam_data[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 <- mean(pred_class == true_class)
  error <- mean(pred_class != true_class)
  true_positives <- sum(pred_class == 1 & true_class == 1)
  true_negatives <- sum(pred_class == 0 & true_class == 0)
  false_positives <- sum(pred_class == 1 & true_class == 0)
  false_negatives <- sum(pred_class == 0 & true_class == 1)
  true_positive_rate <- true_positives / (true_positives + false_negatives)
  false_positive_rate <- false_positives / (false_positives + true_negatives)
  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_model <- glm(spam ~ ., family = binomial, data = spam_train)
```

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

```r
predictions <- predict(glm_model, newdata = spam_test, type = "response")
predicted_classes <- ifelse(predictions > 0.5, 1, 0)
```

```r
accuracy1 <- mean(predicted_classes == spam_test)
print(accuracy1)
```

```
[1] 0.5123313
```

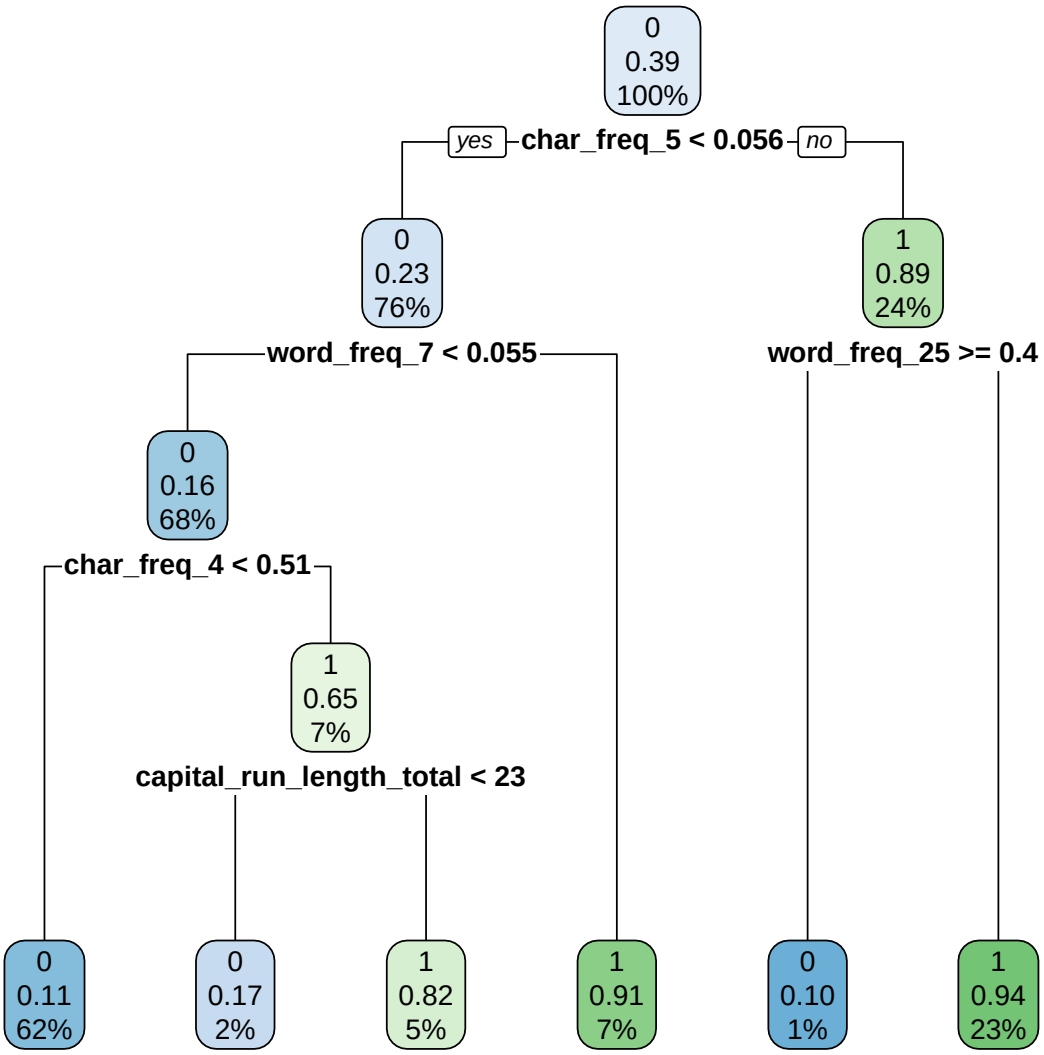
2.4 (5 points)

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
rpart_classes <- rpart(spam ~ ., data = spam_train, method = "class") # Insert your code h
```

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

```r
rpart.plot(rpart_classes) # Insert your code here
```



Report the prediction accuracy on the test set.

```
rpart_predictions <- predict(rpart_classes, newdata = spam_test, type = "class")
rpart_predictions_num <- as.numeric(levels(rpart_predictions))[rpart_predictions]
rpart_accuracy <- mean(rpart_predictions_num == spam_test) # Insert your code here
print(rpart_accuracy)
```

```
[1] 0.5263118
```

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

Report the prediction accuracy on the test set.

```
svm_classes <- predict(svm_fit, newdata = spam_test) # Insert your code here
svm_accuracy <- mean(svm_classes == spam_test$spam)

# Print the accuracy
print(svm_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.

```
M2 <- model.matrix(spam ~ 0 + . , data = spam_train)
```

```
fit_nn2 <- nn %>%
  setup(loss = nn_bce_loss(),
        optimizer = optim_adam,
```

```
      metrics = list(luz_metric_accuracy())) %>%
  set_hparams(p = ncol(M2),q1 = 32, q2 = 16, q3 = 8) %>%
  set_opt_hparams(lr= 0.001) %>%
  fit(data=list(
      model.matrix(spam ~ 0 + .,data = spam_train),spam_train %>% select(spam) %>% as.matr
      valid_data = list(
      model.matrix(spam ~ 0 + .,data = spam_test),spam_test %>% select(spam) %>% as.matrix
      epochs = 100,
      verbose = TRUE
      )
```

```
Epoch 1/100
Train metrics: Loss: 0.7532 - Acc: 12.5659
Valid metrics: Loss: 0.6081 - Acc: 12.8609
Epoch 2/100
Train metrics: Loss: 0.4521 - Acc: 12.5581
Valid metrics: Loss: 0.5522 - Acc: 12.8609
Epoch 3/100
Train metrics: Loss: 0.3537 - Acc: 12.5426
Valid metrics: Loss: 0.3075 - Acc: 12.8609
Epoch 4/100
Train metrics: Loss: 0.3727 - Acc: 12.5814
Valid metrics: Loss: 0.6486 - Acc: 12.8609
Epoch 5/100
Train metrics: Loss: 0.3093 - Acc: 12.5581
Valid metrics: Loss: 0.451 - Acc: 12.8609
Epoch 6/100
Train metrics: Loss: 0.2851 - Acc: 12.5581
Valid metrics: Loss: 0.4946 - Acc: 12.8609
Epoch 7/100
Train metrics: Loss: 0.232 - Acc: 12.5504
Valid metrics: Loss: 0.4382 - Acc: 12.8609
Epoch 8/100
Train metrics: Loss: 0.2709 - Acc: 12.5504
Valid metrics: Loss: 0.2461 - Acc: 12.8609
Epoch 9/100
Train metrics: Loss: 0.3106 - Acc: 12.5736
Valid metrics: Loss: 0.4305 - Acc: 12.8609
Epoch 10/100
Train metrics: Loss: 0.2446 - Acc: 12.5271
Valid metrics: Loss: 0.4824 - Acc: 12.8609
Epoch 11/100
```

```
Train metrics: Loss: 0.2616 - Acc: 12.5581
Valid metrics: Loss: 0.2398 - Acc: 12.8609
Epoch 12/100
Train metrics: Loss: 0.1983 - Acc: 12.5659
Valid metrics: Loss: 0.4451 - Acc: 12.8609
Epoch 13/100
Train metrics: Loss: 0.184 - Acc: 12.5659
Valid metrics: Loss: 0.4156 - Acc: 12.8609
Epoch 14/100
Train metrics: Loss: 0.2089 - Acc: 12.5659
Valid metrics: Loss: 0.2306 - Acc: 12.8609
Epoch 15/100
Train metrics: Loss: 0.2202 - Acc: 12.5504
Valid metrics: Loss: 0.4522 - Acc: 12.8609
Epoch 16/100
Train metrics: Loss: 0.1782 - Acc: 12.5426
Valid metrics: Loss: 0.4237 - Acc: 12.8609
Epoch 17/100
Train metrics: Loss: 0.2366 - Acc: 12.5814
Valid metrics: Loss: 0.4312 - Acc: 12.8609
Epoch 18/100
Train metrics: Loss: 0.1756 - Acc: 12.5504
Valid metrics: Loss: 0.4034 - Acc: 12.8609
Epoch 19/100
Train metrics: Loss: 0.2083 - Acc: 12.5426
Valid metrics: Loss: 0.276 - Acc: 12.8609
Epoch 20/100
Train metrics: Loss: 0.202 - Acc: 12.5581
Valid metrics: Loss: 0.4093 - Acc: 12.8609
Epoch 21/100
Train metrics: Loss: 0.2151 - Acc: 12.5504
Valid metrics: Loss: 0.224 - Acc: 12.8609
Epoch 22/100
Train metrics: Loss: 0.1671 - Acc: 12.5426
Valid metrics: Loss: 0.2155 - Acc: 12.8609
Epoch 23/100
Train metrics: Loss: 0.1904 - Acc: 12.5504
Valid metrics: Loss: 0.4119 - Acc: 12.8609
Epoch 24/100
Train metrics: Loss: 0.1747 - Acc: 12.5581
Valid metrics: Loss: 0.2215 - Acc: 12.8609
Epoch 25/100
Train metrics: Loss: 0.2382 - Acc: 12.5581
```

```
Valid metrics: Loss: 0.4598 - Acc: 12.8609
Epoch 26/100
Train metrics: Loss: 0.2182 - Acc: 12.5581
Valid metrics: Loss: 0.4114 - Acc: 12.8609
Epoch 27/100
Train metrics: Loss: 0.1738 - Acc: 12.5736
Valid metrics: Loss: 0.4961 - Acc: 12.8609
Epoch 28/100
Train metrics: Loss: 0.1847 - Acc: 12.5426
Valid metrics: Loss: 0.4022 - Acc: 12.8609
Epoch 29/100
Train metrics: Loss: 0.1566 - Acc: 12.5581
Valid metrics: Loss: 0.3983 - Acc: 12.8609
Epoch 30/100
Train metrics: Loss: 0.185 - Acc: 12.5581
Valid metrics: Loss: 0.4091 - Acc: 12.8609
Epoch 31/100
Train metrics: Loss: 0.1724 - Acc: 12.5426
Valid metrics: Loss: 0.4295 - Acc: 12.8609
Epoch 32/100
Train metrics: Loss: 0.1956 - Acc: 12.5814
Valid metrics: Loss: 0.4236 - Acc: 12.8609
Epoch 33/100
Train metrics: Loss: 0.1572 - Acc: 12.5659
Valid metrics: Loss: 0.406 - Acc: 12.8609
Epoch 34/100
Train metrics: Loss: 0.1624 - Acc: 12.5504
Valid metrics: Loss: 0.4036 - Acc: 12.8609
Epoch 35/100
Train metrics: Loss: 0.1515 - Acc: 12.5736
Valid metrics: Loss: 0.403 - Acc: 12.8609
Epoch 36/100
Train metrics: Loss: 0.1632 - Acc: 12.5659
Valid metrics: Loss: 0.5226 - Acc: 12.8609
Epoch 37/100
Train metrics: Loss: 0.1603 - Acc: 12.5581
Valid metrics: Loss: 0.3986 - Acc: 12.8609
Epoch 38/100
Train metrics: Loss: 0.1484 - Acc: 12.5426
Valid metrics: Loss: 0.4024 - Acc: 12.8609
Epoch 39/100
Train metrics: Loss: 0.1554 - Acc: 12.5504
Valid metrics: Loss: 0.4059 - Acc: 12.8609
```

```
Epoch 40/100
Train metrics: Loss: 0.1816 - Acc: 12.5426
Valid metrics: Loss: 0.4087 - Acc: 12.8609
Epoch 41/100
Train metrics: Loss: 0.1615 - Acc: 12.5426
Valid metrics: Loss: 0.472 - Acc: 12.8609
Epoch 42/100
Train metrics: Loss: 0.1519 - Acc: 12.5581
Valid metrics: Loss: 0.4347 - Acc: 12.8609
Epoch 43/100
Train metrics: Loss: 0.139 - Acc: 12.5736
Valid metrics: Loss: 0.4263 - Acc: 12.8609
Epoch 44/100
Train metrics: Loss: 0.1916 - Acc: 12.5659
Valid metrics: Loss: 0.2802 - Acc: 12.8609
Epoch 45/100
Train metrics: Loss: 0.1503 - Acc: 12.5736
Valid metrics: Loss: 0.4141 - Acc: 12.8609
Epoch 46/100
Train metrics: Loss: 0.1914 - Acc: 12.5581
Valid metrics: Loss: 0.4424 - Acc: 12.8609
Epoch 47/100
Train metrics: Loss: 0.1502 - Acc: 12.5426
Valid metrics: Loss: 0.4071 - Acc: 12.8609
Epoch 48/100
Train metrics: Loss: 0.1848 - Acc: 12.5426
Valid metrics: Loss: 0.4209 - Acc: 12.8609
Epoch 49/100
Train metrics: Loss: 0.1622 - Acc: 12.5891
Valid metrics: Loss: 0.4166 - Acc: 12.8609
Epoch 50/100
Train metrics: Loss: 0.1524 - Acc: 12.5581
Valid metrics: Loss: 0.4095 - Acc: 12.8609
Epoch 51/100
Train metrics: Loss: 0.1403 - Acc: 12.5659
Valid metrics: Loss: 0.4071 - Acc: 12.8609
Epoch 52/100
Train metrics: Loss: 0.1498 - Acc: 12.5736
Valid metrics: Loss: 0.4025 - Acc: 12.8609
Epoch 53/100
Train metrics: Loss: 0.1388 - Acc: 12.5581
Valid metrics: Loss: 0.4025 - Acc: 12.8609
Epoch 54/100
```

```
Train metrics: Loss: 0.1691 - Acc: 12.5581
Valid metrics: Loss: 0.2318 - Acc: 12.8609
Epoch 55/100
Train metrics: Loss: 0.1311 - Acc: 12.5659
Valid metrics: Loss: 0.3971 - Acc: 12.8609
Epoch 56/100
Train metrics: Loss: 0.1346 - Acc: 12.5736
Valid metrics: Loss: 0.397 - Acc: 12.8609
Epoch 57/100
Train metrics: Loss: 0.1406 - Acc: 12.5659
Valid metrics: Loss: 0.2118 - Acc: 12.8609
Epoch 58/100
Train metrics: Loss: 0.1251 - Acc: 12.5504
Valid metrics: Loss: 0.2247 - Acc: 12.8609
Epoch 59/100
Train metrics: Loss: 0.1355 - Acc: 12.5659
Valid metrics: Loss: 0.3976 - Acc: 12.8609
Epoch 60/100
Train metrics: Loss: 0.1427 - Acc: 12.5581
Valid metrics: Loss: 0.2122 - Acc: 12.8609
Epoch 61/100
Train metrics: Loss: 0.1258 - Acc: 12.5581
Valid metrics: Loss: 0.2142 - Acc: 12.8609
Epoch 62/100
Train metrics: Loss: 0.1292 - Acc: 12.5504
Valid metrics: Loss: 0.4086 - Acc: 12.8609
Epoch 63/100
Train metrics: Loss: 0.1214 - Acc: 12.5581
Valid metrics: Loss: 0.2201 - Acc: 12.8609
Epoch 64/100
Train metrics: Loss: 0.1267 - Acc: 12.5736
Valid metrics: Loss: 0.2206 - Acc: 12.8609
Epoch 65/100
Train metrics: Loss: 0.1274 - Acc: 12.5504
Valid metrics: Loss: 0.5663 - Acc: 12.8609
Epoch 66/100
Train metrics: Loss: 0.1293 - Acc: 12.5504
Valid metrics: Loss: 0.2279 - Acc: 12.8609
Epoch 67/100
Train metrics: Loss: 0.1204 - Acc: 12.5581
Valid metrics: Loss: 0.2345 - Acc: 12.8609
Epoch 68/100
Train metrics: Loss: 0.1235 - Acc: 12.5581
```

```
Valid metrics: Loss: 0.2234 - Acc: 12.8609
Epoch 69/100
Train metrics: Loss: 0.1288 - Acc: 12.5581
Valid metrics: Loss: 0.2455 - Acc: 12.8609
Epoch 70/100
Train metrics: Loss: 0.1218 - Acc: 12.5736
Valid metrics: Loss: 0.2335 - Acc: 12.8609
Epoch 71/100
Train metrics: Loss: 0.1247 - Acc: 12.5736
Valid metrics: Loss: 0.419 - Acc: 12.8609
Epoch 72/100
Train metrics: Loss: 0.119 - Acc: 12.5659
Valid metrics: Loss: 0.246 - Acc: 12.8609
Epoch 73/100
Train metrics: Loss: 0.1714 - Acc: 12.5349
Valid metrics: Loss: 0.2336 - Acc: 12.8609
Epoch 74/100
Train metrics: Loss: 0.1235 - Acc: 12.5814
Valid metrics: Loss: 0.4178 - Acc: 12.8609
Epoch 75/100
Train metrics: Loss: 0.1204 - Acc: 12.5736
Valid metrics: Loss: 0.3476 - Acc: 12.8609
Epoch 76/100
Train metrics: Loss: 0.1586 - Acc: 12.5504
Valid metrics: Loss: 0.4054 - Acc: 12.8609
Epoch 77/100
Train metrics: Loss: 0.1419 - Acc: 12.5581
Valid metrics: Loss: 0.4254 - Acc: 12.8609
Epoch 78/100
Train metrics: Loss: 0.1172 - Acc: 12.5659
Valid metrics: Loss: 0.4131 - Acc: 12.8609
Epoch 79/100
Train metrics: Loss: 0.1179 - Acc: 12.5659
Valid metrics: Loss: 0.4218 - Acc: 12.8609
Epoch 80/100
Train metrics: Loss: 0.1119 - Acc: 12.5349
Valid metrics: Loss: 0.4189 - Acc: 12.8609
Epoch 81/100
Train metrics: Loss: 0.1079 - Acc: 12.5659
Valid metrics: Loss: 0.4195 - Acc: 12.8609
Epoch 82/100
Train metrics: Loss: 0.1209 - Acc: 12.5659
Valid metrics: Loss: 0.2373 - Acc: 12.8609
```

```
Epoch 83/100
Train metrics: Loss: 0.1122 - Acc: 12.5504
Valid metrics: Loss: 0.407 - Acc: 12.8609
Epoch 84/100
Train metrics: Loss: 0.1365 - Acc: 12.5504
Valid metrics: Loss: 0.4046 - Acc: 12.8609
Epoch 85/100
Train metrics: Loss: 0.1346 - Acc: 12.5504
Valid metrics: Loss: 0.2421 - Acc: 12.8609
Epoch 86/100
Train metrics: Loss: 0.1061 - Acc: 12.5736
Valid metrics: Loss: 0.2282 - Acc: 12.8609
Epoch 87/100
Train metrics: Loss: 0.1167 - Acc: 12.5349
Valid metrics: Loss: 0.2423 - Acc: 12.8609
Epoch 88/100
Train metrics: Loss: 0.1038 - Acc: 12.5504
Valid metrics: Loss: 0.2251 - Acc: 12.8609
Epoch 89/100
Train metrics: Loss: 0.1072 - Acc: 12.5349
Valid metrics: Loss: 0.2406 - Acc: 12.8609
Epoch 90/100
Train metrics: Loss: 0.0999 - Acc: 12.5581
Valid metrics: Loss: 0.23 - Acc: 12.8609
Epoch 91/100
Train metrics: Loss: 0.1066 - Acc: 12.5426
Valid metrics: Loss: 0.2506 - Acc: 12.8609
Epoch 92/100
Train metrics: Loss: 0.1215 - Acc: 12.5504
Valid metrics: Loss: 0.421 - Acc: 12.8609
Epoch 93/100
Train metrics: Loss: 0.1053 - Acc: 12.5504
Valid metrics: Loss: 0.2723 - Acc: 12.8609
Epoch 94/100
Train metrics: Loss: 0.097 - Acc: 12.5426
Valid metrics: Loss: 0.2511 - Acc: 12.8609
Epoch 95/100
Train metrics: Loss: 0.1205 - Acc: 12.5349
Valid metrics: Loss: 0.2265 - Acc: 12.8609
Epoch 96/100
Train metrics: Loss: 0.0977 - Acc: 12.5736
Valid metrics: Loss: 0.238 - Acc: 12.8609
Epoch 97/100
```

```
Train metrics: Loss: 0.1005 - Acc: 12.5349
Valid metrics: Loss: 0.2579 - Acc: 12.8609
Epoch 98/100
Train metrics: Loss: 0.0949 - Acc: 12.5814
Valid metrics: Loss: 0.2371 - Acc: 12.8609
Epoch 99/100
Train metrics: Loss: 0.0958 - Acc: 12.5426
Valid metrics: Loss: 0.2775 - Acc: 12.8609
Epoch 100/100
Train metrics: Loss: 0.0933 - Acc: 12.5504
Valid metrics: Loss: 0.2335 - Acc: 12.8609
```

```r
spam_test_result <- model.matrix(spam ~ 0 + . , data = spam_test)
```

```r
nnet2_predictions <- predict(fit_nn2, spam_test_result)
```

```r
nnet_predictions <- as.array(nnet2_predictions)

# Assuming `df_test$median_house_value` is your true values for the test set
nn_accuracy <- mean(nnet_predictions == spam_test$spam)
print(nn_accuracy)
```

```
[1] 0.08043478
```

2.7 (5 points)

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

```r
model_comparisons2 <- data.frame(
  Model = c("logistic regression model", "decision tree model", "SVM","nn"),
  RMSE = c(accuracy1, rpart_accuracy, svm_accuracy,nn_accuracy)
)

# Print the table
print(model_comparisons2)
```

```
                    Model       RMSE
```

```
1 logistic regression model 0.51233133
2       decision tree model 0.52631184
3                       SVM 0.92391304
4                        nn 0.08043478
```

```
knitr::kable(model_comparisons2, caption = "Model Comparison based on RMSE")
```

Table 2: Model Comparison based on RMSE

Model	RMSE
logistic regression model	0.5123313
decision tree model	0.5263118
SVM	0.9239130
nn	0.0804348

svm was the highest accuracy

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

Question 3

60 points

Three spirals classification

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

This is a multi-class classification problem

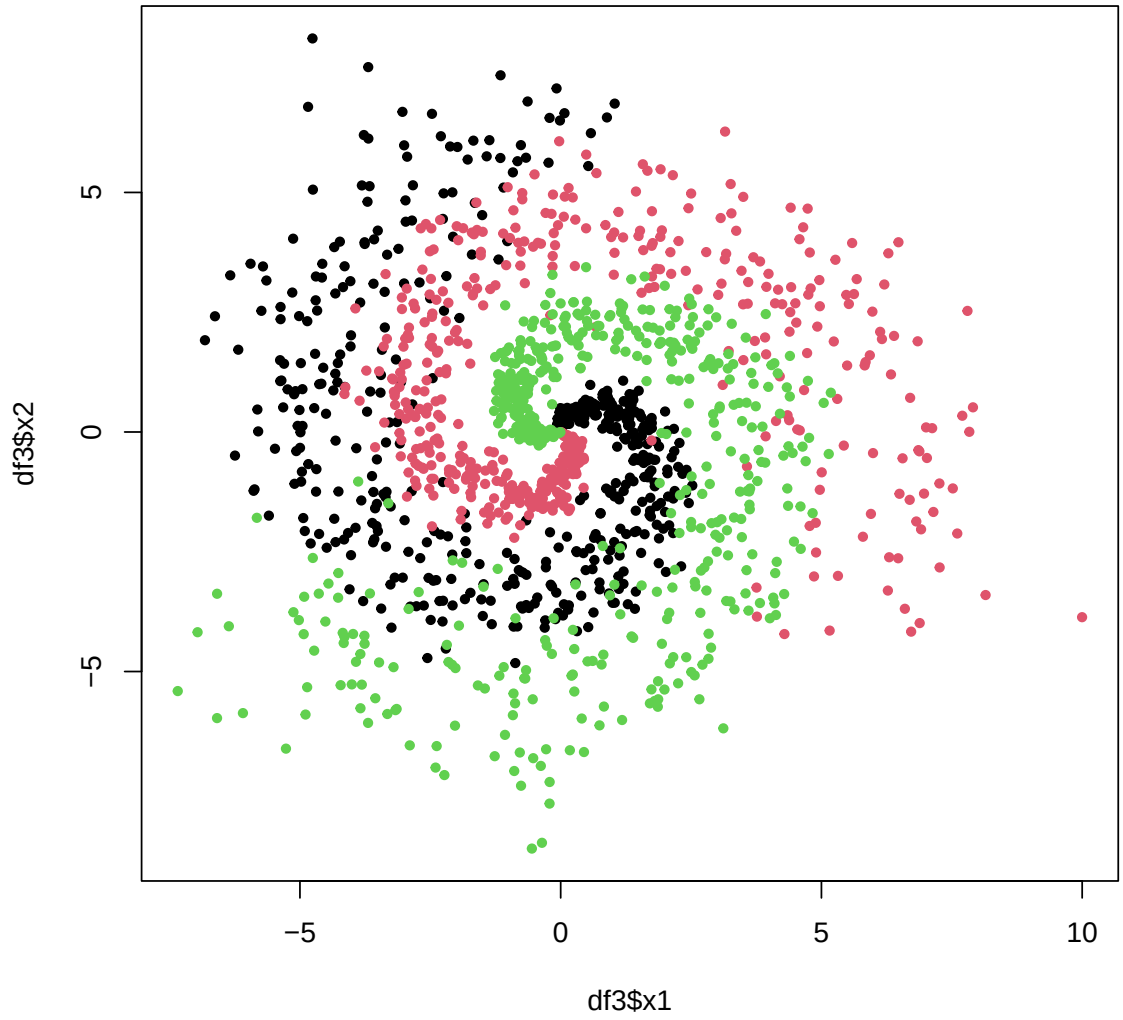
The dataset can be generated using the provided R code below:

```r
generate_three_spirals <- function(){
  set.seed(42)
  n <- 500
  noise <- 0.2
  t <- (1:n) / n * 2 * pi
  x1 <- c(
      t * (sin(t) + rnorm(n, 0, noise)),
      t * (sin(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (sin(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  x2 <- c(
      t * (cos(t) + rnorm(n, 0, noise)),
      t * (cos(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (cos(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  y <- as.factor(
    c(
      rep(0, n),
      rep(1, n),
      rep(2, n)
    )
  )
  return(tibble::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
df3 <- generate_three_spirals()

plot(
  df3$x1, df3$x2,
  col = df3$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`.

```
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
grid <- expand.grid(x1 = x1, x2 = x2)
df_test3 <- as_tibble(grid)
```

3.2 (10 points)

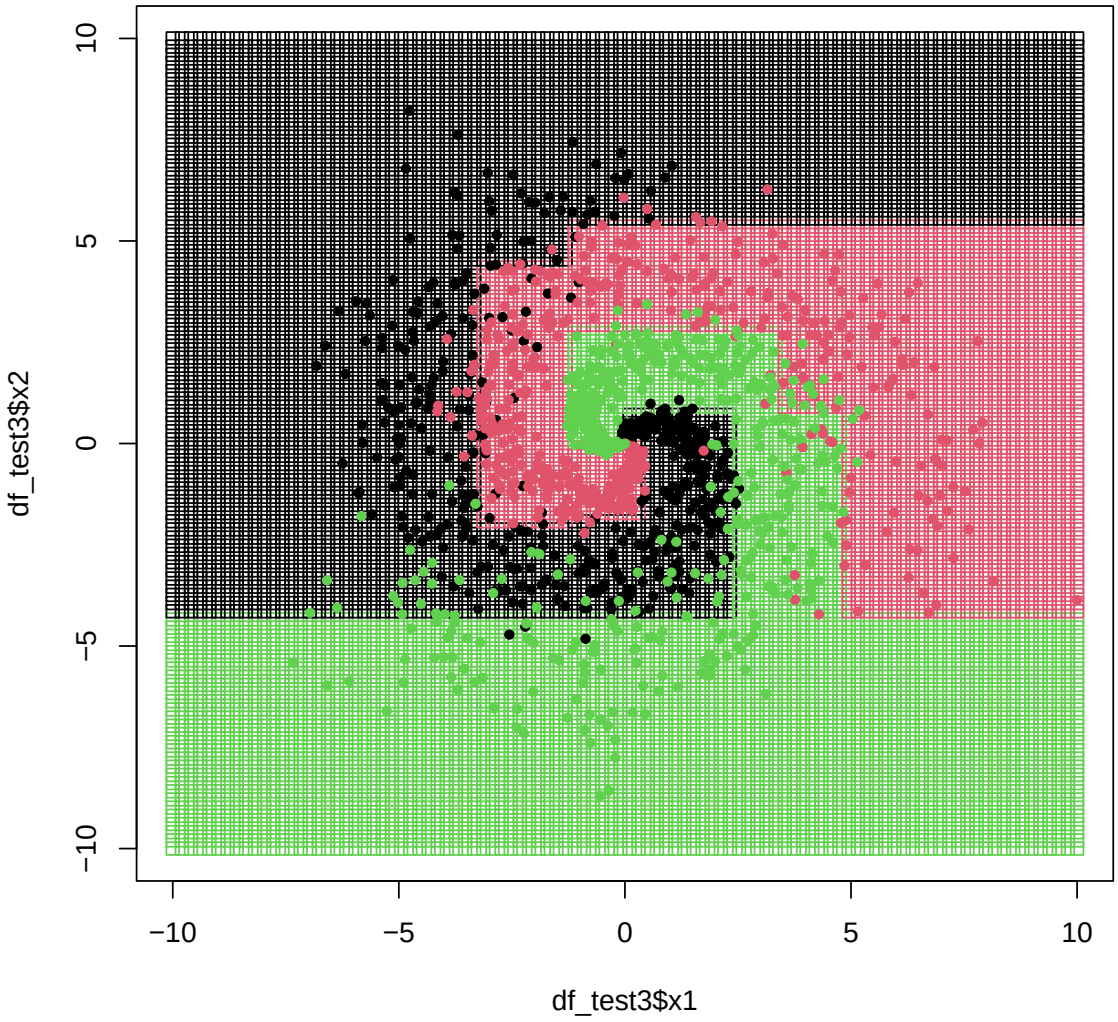
Fit a classification tree model to predict the `y` variable using the `x1` and `x2` predictors, and plot the decision boundary.

```
rpart_fit <- rpart(y ~ x1 + x2, data = df3, method = "class")
rpart_classes <- predict(rpart_fit, newdata = df_test3, type = "class")
```

Plot the decision boundary using the following function:

```
plot_decision_boundary <- function(predictions){
  plot(
    df_test3$x1, df_test3$x2,
    col = predictions,
    pch = 0
  )
  points(
    df3$x1, df3$x2,
    col = df3$y,
    pch = 20
  )
}
```

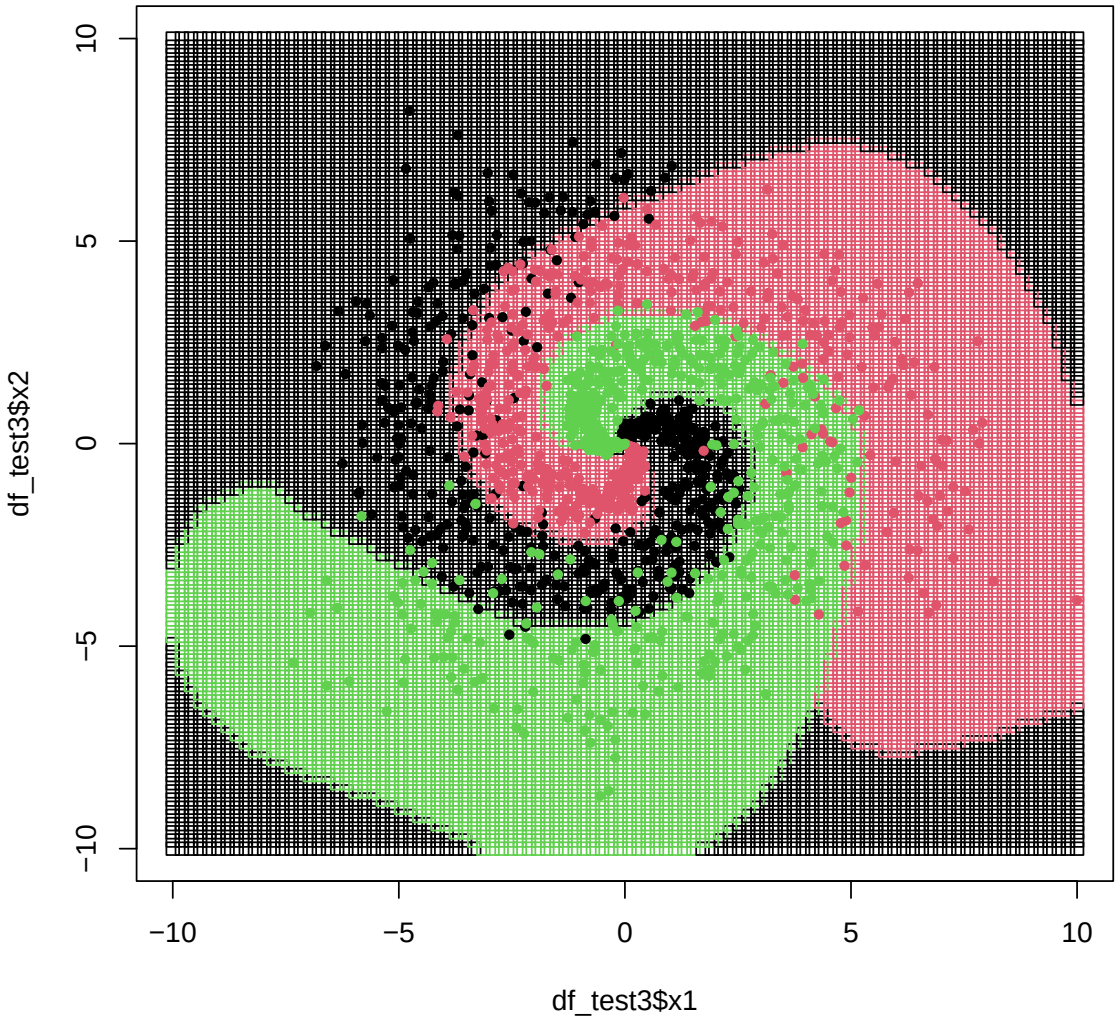
```
plot_decision_boundary(rpart_classes)
```



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
df3$y <- as.factor(df3$y)
svm_fit <- svm(y ~ x1 + x2, data = df3, type = "C-classification", kernel = "radial") # In
svm_classes <- predict(svm_fit, newdata = df_test3) # Insert your code here
plot_decision_boundary(svm_classes)
```



Instructions

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

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

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

3.4 (10 points)

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

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



fit_1 <- NN1 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam,
    metrics = list(luz_metric_accuracy())
  ) %>%
```

```
    set_hparams(
      p = 2,  # Number of input features (x1 and x2)
      q1 = 10,  # Number of hidden units
      o = 3  # Number of output classes
    ) %>%
    set_opt_hparams(lr= 0.005) %>%
    fit(
      data = list(
        x = df3 %>% select(x1, x2) %>% as.matrix(),
        y = df3$y %>% as.integer()
      ),
      epochs = 50,
      verbose = TRUE
    )
```

```
Epoch 1/50
Train metrics: Loss: 1.0672 - Acc: 0.3906
Epoch 2/50
Train metrics: Loss: 0.9795 - Acc: 0.3641
Epoch 3/50
Train metrics: Loss: 0.9367 - Acc: 0.4076
Epoch 4/50
Train metrics: Loss: 0.8973 - Acc: 0.4735
Epoch 5/50
Train metrics: Loss: 0.8558 - Acc: 0.5543
Epoch 6/50
Train metrics: Loss: 0.8221 - Acc: 0.5951
Epoch 7/50
Train metrics: Loss: 0.791 - Acc: 0.5965
Epoch 8/50
Train metrics: Loss: 0.7616 - Acc: 0.6488
Epoch 9/50
Train metrics: Loss: 0.7345 - Acc: 0.6311
Epoch 10/50
Train metrics: Loss: 0.7089 - Acc: 0.6556
Epoch 11/50
Train metrics: Loss: 0.6833 - Acc: 0.6732
Epoch 12/50
Train metrics: Loss: 0.6606 - Acc: 0.6936
Epoch 13/50
Train metrics: Loss: 0.6459 - Acc: 0.6902
Epoch 14/50
```

```
Train metrics: Loss: 0.6309 - Acc: 0.7113
Epoch 15/50
Train metrics: Loss: 0.6154 - Acc: 0.7079
Epoch 16/50
Train metrics: Loss: 0.6014 - Acc: 0.7167
Epoch 17/50
Train metrics: Loss: 0.5916 - Acc: 0.7181
Epoch 18/50
Train metrics: Loss: 0.5838 - Acc: 0.7133
Epoch 19/50
Train metrics: Loss: 0.5738 - Acc: 0.7385
Epoch 20/50
Train metrics: Loss: 0.5658 - Acc: 0.7262
Epoch 21/50
Train metrics: Loss: 0.5569 - Acc: 0.7405
Epoch 22/50
Train metrics: Loss: 0.5532 - Acc: 0.7371
Epoch 23/50
Train metrics: Loss: 0.546 - Acc: 0.7534
Epoch 24/50
Train metrics: Loss: 0.5395 - Acc: 0.7473
Epoch 25/50
Train metrics: Loss: 0.5387 - Acc: 0.7357
Epoch 26/50
Train metrics: Loss: 0.5336 - Acc: 0.7541
Epoch 27/50
Train metrics: Loss: 0.5309 - Acc: 0.7534
Epoch 28/50
Train metrics: Loss: 0.5256 - Acc: 0.7554
Epoch 29/50
Train metrics: Loss: 0.5241 - Acc: 0.75
Epoch 30/50
Train metrics: Loss: 0.5141 - Acc: 0.7561
Epoch 31/50
Train metrics: Loss: 0.5156 - Acc: 0.7493
Epoch 32/50
Train metrics: Loss: 0.5161 - Acc: 0.7527
Epoch 33/50
Train metrics: Loss: 0.5103 - Acc: 0.7473
Epoch 34/50
Train metrics: Loss: 0.5094 - Acc: 0.7534
Epoch 35/50
Train metrics: Loss: 0.5079 - Acc: 0.7615
```

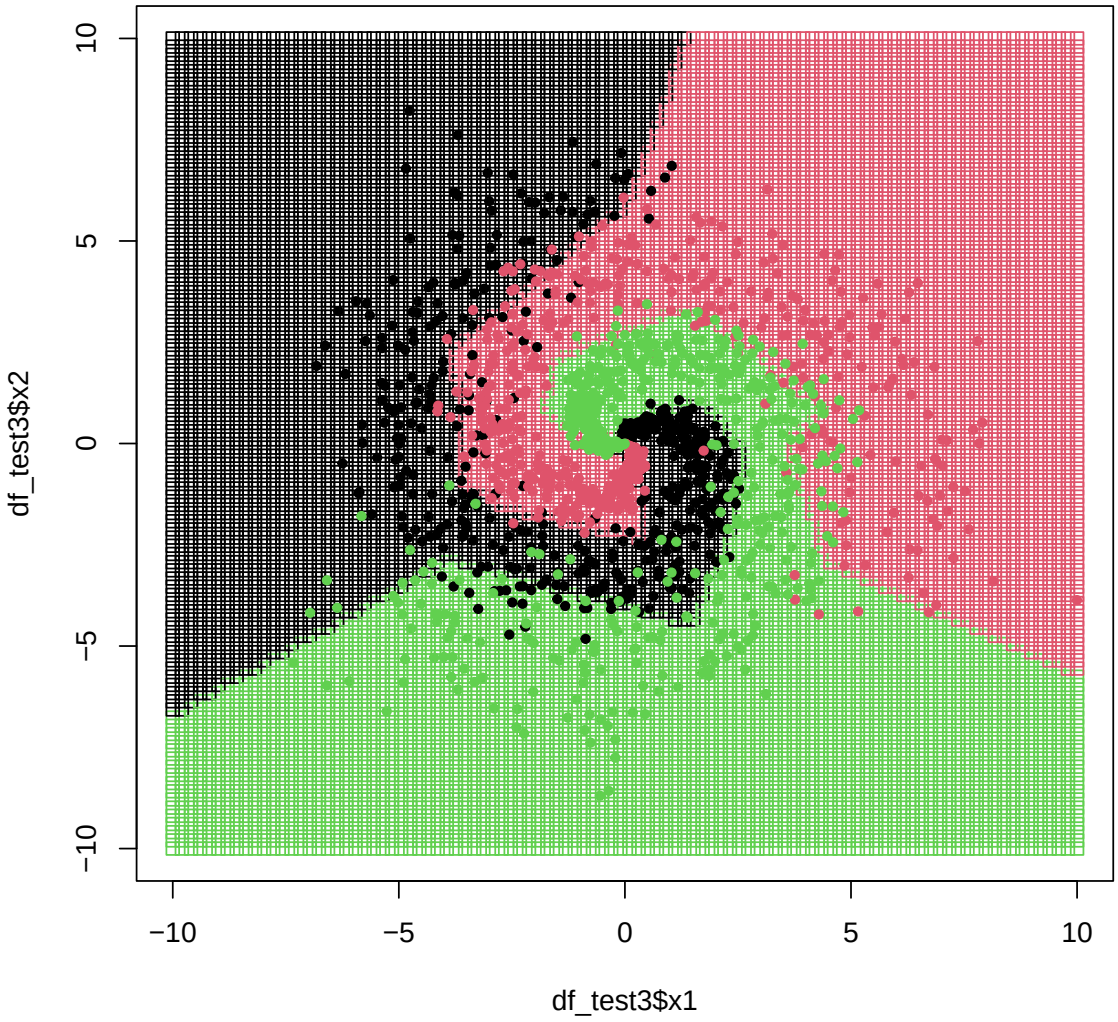
```
Epoch 36/50
Train metrics: Loss: 0.5016 - Acc: 0.7697
Epoch 37/50
Train metrics: Loss: 0.4982 - Acc: 0.7731
Epoch 38/50
Train metrics: Loss: 0.5033 - Acc: 0.7609
Epoch 39/50
Train metrics: Loss: 0.5062 - Acc: 0.7609
Epoch 40/50
Train metrics: Loss: 0.4949 - Acc: 0.7704
Epoch 41/50
Train metrics: Loss: 0.5013 - Acc: 0.7697
Epoch 42/50
Train metrics: Loss: 0.4894 - Acc: 0.7602
Epoch 43/50
Train metrics: Loss: 0.4979 - Acc: 0.7595
Epoch 44/50
Train metrics: Loss: 0.4908 - Acc: 0.7636
Epoch 45/50
Train metrics: Loss: 0.4882 - Acc: 0.7806
Epoch 46/50
Train metrics: Loss: 0.4919 - Acc: 0.7656
Epoch 47/50
Train metrics: Loss: 0.487 - Acc: 0.7785
Epoch 48/50
Train metrics: Loss: 0.4851 - Acc: 0.7595
Epoch 49/50
Train metrics: Loss: 0.4838 - Acc: 0.769
Epoch 50/50
Train metrics: Loss: 0.4882 - Acc: 0.769
```

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

```
test_matrix <- df_test3 %>% select(x1, x2) %>% as.matrix

fit_1_predictions <- max.col(predict(fit_1, test_matrix))
```

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

```
plot_decision_boundary(fit_1_predictions)
```



3.5 (10 points)

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

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

fit_0 <- NN0 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam,
    metrics = list(luz_metric_accuracy())
  ) %>%
  set_hparams(
    p = 2,
    o = 3
  ) %>%
  set_opt_hparams(lr= 0.005) %>%
  fit(
    data = list(
      x = df3 %>% select(x1, x2) %>% as.matrix(),
      y = df3$y %>% as.integer()
    ),
    epochs = 50,
    verbose = TRUE
  )
```

```
Epoch 1/50
Train metrics: Loss: 1.233 - Acc: 0.2996
Epoch 2/50
Train metrics: Loss: 1.1301 - Acc: 0.2908
Epoch 3/50
Train metrics: Loss: 1.0813 - Acc: 0.3166
Epoch 4/50
Train metrics: Loss: 1.065 - Acc: 0.3519
Epoch 5/50
Train metrics: Loss: 1.0578 - Acc: 0.3702
```

```
Epoch 6/50
Train metrics: Loss: 1.0553 - Acc: 0.3743
Epoch 7/50
Train metrics: Loss: 1.054 - Acc: 0.3757
Epoch 8/50
Train metrics: Loss: 1.0538 - Acc: 0.3838
Epoch 9/50
Train metrics: Loss: 1.0532 - Acc: 0.3784
Epoch 10/50
Train metrics: Loss: 1.0521 - Acc: 0.3947
Epoch 11/50
Train metrics: Loss: 1.0536 - Acc: 0.3818
Epoch 12/50
Train metrics: Loss: 1.0529 - Acc: 0.3601
Epoch 13/50
Train metrics: Loss: 1.055 - Acc: 0.3791
Epoch 14/50
Train metrics: Loss: 1.053 - Acc: 0.3764
Epoch 15/50
Train metrics: Loss: 1.0519 - Acc: 0.3811
Epoch 16/50
Train metrics: Loss: 1.0508 - Acc: 0.3879
Epoch 17/50
Train metrics: Loss: 1.0531 - Acc: 0.3723
Epoch 18/50
Train metrics: Loss: 1.0532 - Acc: 0.3716
Epoch 19/50
Train metrics: Loss: 1.0538 - Acc: 0.3689
Epoch 20/50
Train metrics: Loss: 1.0525 - Acc: 0.3764
Epoch 21/50
Train metrics: Loss: 1.0533 - Acc: 0.3689
Epoch 22/50
Train metrics: Loss: 1.0537 - Acc: 0.373
Epoch 23/50
Train metrics: Loss: 1.054 - Acc: 0.3784
Epoch 24/50
Train metrics: Loss: 1.0534 - Acc: 0.3662
Epoch 25/50
Train metrics: Loss: 1.0539 - Acc: 0.3832
Epoch 26/50
Train metrics: Loss: 1.052 - Acc: 0.3702
Epoch 27/50
```

```
Train metrics: Loss: 1.0533 - Acc: 0.377
Epoch 28/50
Train metrics: Loss: 1.0523 - Acc: 0.3655
Epoch 29/50
Train metrics: Loss: 1.0527 - Acc: 0.3852
Epoch 30/50
Train metrics: Loss: 1.0529 - Acc: 0.3798
Epoch 31/50
Train metrics: Loss: 1.0539 - Acc: 0.3757
Epoch 32/50
Train metrics: Loss: 1.0523 - Acc: 0.3696
Epoch 33/50
Train metrics: Loss: 1.0552 - Acc: 0.3655
Epoch 34/50
Train metrics: Loss: 1.0516 - Acc: 0.3872
Epoch 35/50
Train metrics: Loss: 1.0545 - Acc: 0.3832
Epoch 36/50
Train metrics: Loss: 1.0539 - Acc: 0.3635
Epoch 37/50
Train metrics: Loss: 1.0539 - Acc: 0.3716
Epoch 38/50
Train metrics: Loss: 1.0542 - Acc: 0.3709
Epoch 39/50
Train metrics: Loss: 1.0525 - Acc: 0.3743
Epoch 40/50
Train metrics: Loss: 1.0513 - Acc: 0.3743
Epoch 41/50
Train metrics: Loss: 1.0528 - Acc: 0.3764
Epoch 42/50
Train metrics: Loss: 1.0531 - Acc: 0.3798
Epoch 43/50
Train metrics: Loss: 1.0531 - Acc: 0.377
Epoch 44/50
Train metrics: Loss: 1.0542 - Acc: 0.3777
Epoch 45/50
Train metrics: Loss: 1.0541 - Acc: 0.3736
Epoch 46/50
Train metrics: Loss: 1.0522 - Acc: 0.3736
Epoch 47/50
Train metrics: Loss: 1.0524 - Acc: 0.3757
Epoch 48/50
Train metrics: Loss: 1.0531 - Acc: 0.3709
```

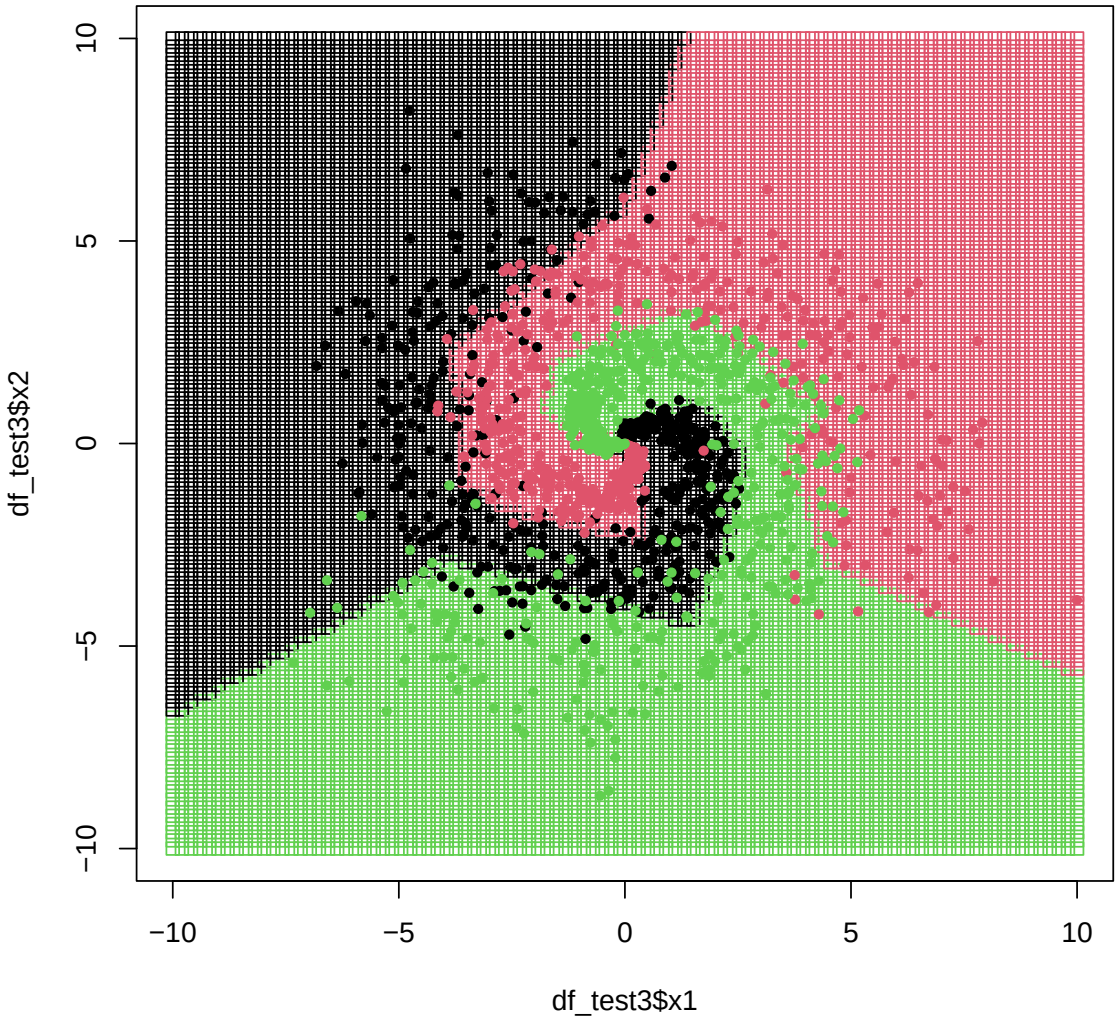
```
Epoch 49/50
Train metrics: Loss: 1.0517 - Acc: 0.3716
Epoch 50/50
Train metrics: Loss: 1.0538 - Acc: 0.3859
```

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

```
fit_0_predictions <- max.col(predict(fit_0, test_matrix))
```

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

```
plot_decision_boundary(fit_1_predictions)
```



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, 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) # Single output node for binary classification
    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() # Sigmoid activation function for binary classifica
  }
)

fit_2 <- NN2 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam,
    metrics = list(luz_metric_accuracy())
  ) %>%
  set_hparams(
    p = 2,
    q1 = 10,
    q2 = 10,
    q3 = 10,
    o = 3
  ) %>%
  set_opt_hparams(lr= 0.005) %>%
  fit(
    data = list(
      x = df3 %>% select(x1, x2) %>% as.matrix(),
      y = df3$y %>% as.integer()
    ),
    epochs = 50,
    verbose = TRUE
  )
```

```
Epoch 1/50
```

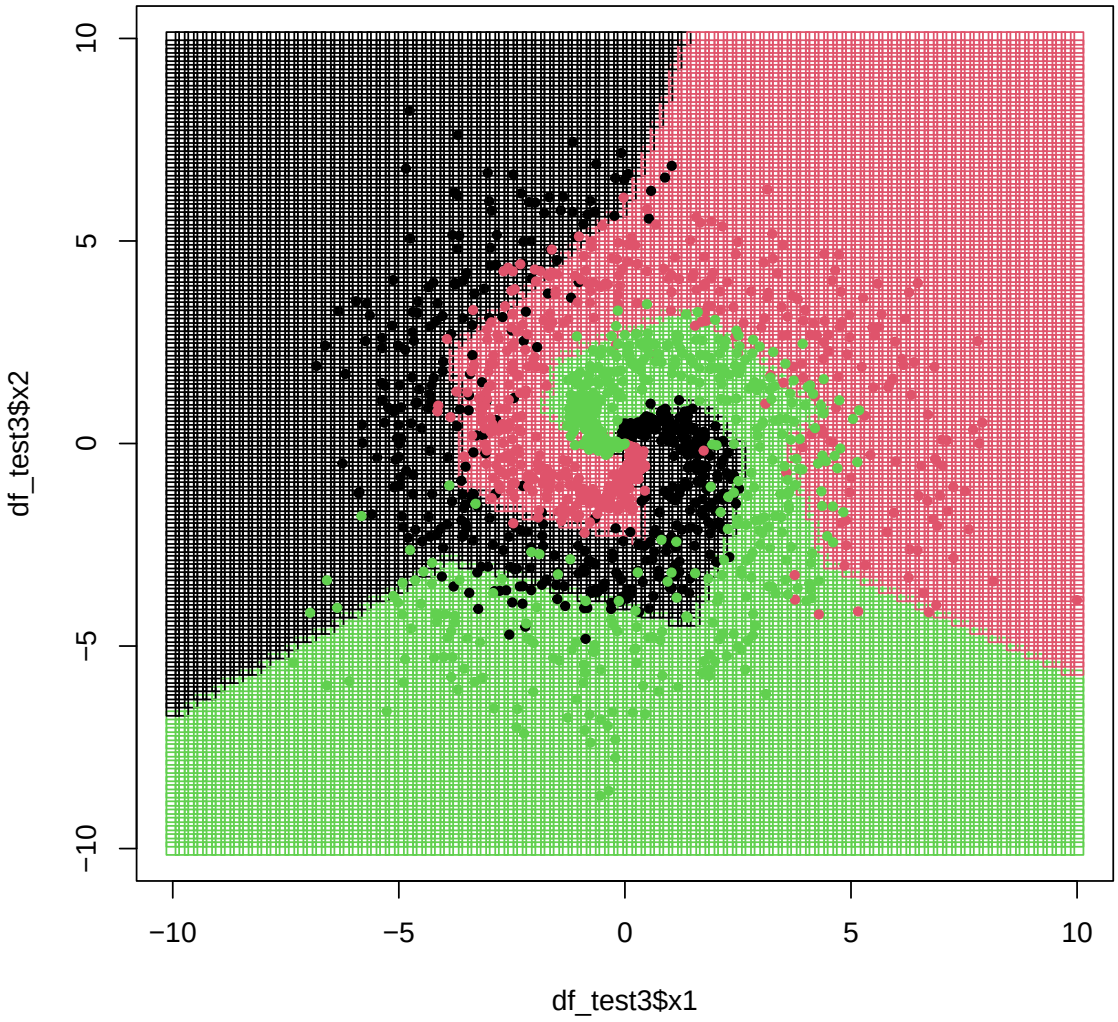
```
Train metrics: Loss: 1.0833 - Acc: 0.3899
Epoch 2/50
Train metrics: Loss: 1.0396 - Acc: 0.3995
Epoch 3/50
Train metrics: Loss: 0.9916 - Acc: 0.4769
Epoch 4/50
Train metrics: Loss: 0.9463 - Acc: 0.5312
Epoch 5/50
Train metrics: Loss: 0.8878 - Acc: 0.6603
Epoch 6/50
Train metrics: Loss: 0.8492 - Acc: 0.6821
Epoch 7/50
Train metrics: Loss: 0.8041 - Acc: 0.7086
Epoch 8/50
Train metrics: Loss: 0.7812 - Acc: 0.712
Epoch 9/50
Train metrics: Loss: 0.769 - Acc: 0.6895
Epoch 10/50
Train metrics: Loss: 0.7563 - Acc: 0.68
Epoch 11/50
Train metrics: Loss: 0.7509 - Acc: 0.6882
Epoch 12/50
Train metrics: Loss: 0.7429 - Acc: 0.6936
Epoch 13/50
Train metrics: Loss: 0.7355 - Acc: 0.7052
Epoch 14/50
Train metrics: Loss: 0.7358 - Acc: 0.697
Epoch 15/50
Train metrics: Loss: 0.7362 - Acc: 0.7065
Epoch 16/50
Train metrics: Loss: 0.7282 - Acc: 0.7126
Epoch 17/50
Train metrics: Loss: 0.7295 - Acc: 0.7181
Epoch 18/50
Train metrics: Loss: 0.7287 - Acc: 0.7174
Epoch 19/50
Train metrics: Loss: 0.7225 - Acc: 0.7201
Epoch 20/50
Train metrics: Loss: 0.719 - Acc: 0.7269
Epoch 21/50
Train metrics: Loss: 0.721 - Acc: 0.7255
Epoch 22/50
Train metrics: Loss: 0.7154 - Acc: 0.7276
```

```
Epoch 23/50
Train metrics: Loss: 0.7134 - Acc: 0.7364
Epoch 24/50
Train metrics: Loss: 0.7164 - Acc: 0.7317
Epoch 25/50
Train metrics: Loss: 0.7163 - Acc: 0.7371
Epoch 26/50
Train metrics: Loss: 0.715 - Acc: 0.7385
Epoch 27/50
Train metrics: Loss: 0.7109 - Acc: 0.7439
Epoch 28/50
Train metrics: Loss: 0.7106 - Acc: 0.7446
Epoch 29/50
Train metrics: Loss: 0.7135 - Acc: 0.7466
Epoch 30/50
Train metrics: Loss: 0.7152 - Acc: 0.7459
Epoch 31/50
Train metrics: Loss: 0.7072 - Acc: 0.7595
Epoch 32/50
Train metrics: Loss: 0.71 - Acc: 0.7486
Epoch 33/50
Train metrics: Loss: 0.7106 - Acc: 0.7548
Epoch 34/50
Train metrics: Loss: 0.7025 - Acc: 0.7643
Epoch 35/50
Train metrics: Loss: 0.7031 - Acc: 0.7704
Epoch 36/50
Train metrics: Loss: 0.702 - Acc: 0.7636
Epoch 37/50
Train metrics: Loss: 0.7032 - Acc: 0.7629
Epoch 38/50
Train metrics: Loss: 0.7041 - Acc: 0.7711
Epoch 39/50
Train metrics: Loss: 0.7 - Acc: 0.7724
Epoch 40/50
Train metrics: Loss: 0.6998 - Acc: 0.7765
Epoch 41/50
Train metrics: Loss: 0.7002 - Acc: 0.7704
Epoch 42/50
Train metrics: Loss: 0.6937 - Acc: 0.784
Epoch 43/50
Train metrics: Loss: 0.6997 - Acc: 0.7826
Epoch 44/50
```

```
Train metrics: Loss: 0.6967 - Acc: 0.7819
Epoch 45/50
Train metrics: Loss: 0.6909 - Acc: 0.7948
Epoch 46/50
Train metrics: Loss: 0.6986 - Acc: 0.7867
Epoch 47/50
Train metrics: Loss: 0.6858 - Acc: 0.8071
Epoch 48/50
Train metrics: Loss: 0.681 - Acc: 0.805
Epoch 49/50
Train metrics: Loss: 0.6924 - Acc: 0.7894
Epoch 50/50
Train metrics: Loss: 0.6886 - Acc: 0.7969
```

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

```
fit_2_predictions <- max.col(predict(fit_2, test_matrix))
plot_decision_boundary(fit_1_predictions)
```



3.7 (5 points)

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

as the number of hidden layers increases, the neural network becomes more expressive

---

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] luz_0.4.0         torch_0.12.0      e1071_1.7-14      rpart.plot_3.1.2
 [5] rpart_4.1.21      caret_6.0-94      lattice_0.21-9    ggplot2_3.4.4
 [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_1.2.0     timeDate_4032.109    farver_2.1.1
 [4] fastmap_1.1.1        pROC_1.18.5          digest_0.6.34
 [7] timechange_0.3.0     lifecycle_1.0.4      ellipsis_0.3.2
[10] survival_3.5-7       processx_3.8.3       compiler_4.3.2
[13] progress_1.2.3       rlang_1.1.3          tools_4.3.2
[16] utf8_1.2.4           yaml_2.3.8           data.table_1.15.0
[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_4.3.2           stats4_4.3.2
[28] fansi_1.0.6          colorspace_2.1-0     future_1.33.1
[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_2.25       generics_0.1.3
[40] rstudioapi_0.15.0    future.apply_1.11.1  reshape2_1.4.4
[43] tzdb_0.4.0           proxy_0.4-27         stringr_1.5.1
[46] splines_4.3.2        parallel_4.3.2       coro_1.0.4
[49] vctrs_0.6.5          hardhat_1.3.1        Matrix_1.6-1.1
[52] jsonlite_1.8.8       callr_3.7.3          hms_1.1.3
[55] bit64_4.0.5          listenv_0.9.1        foreach_1.5.2
[58] gower_1.0.1          recipes_1.0.10       glue_1.7.0
[61] parallelly_1.37.1    ps_1.7.6             codetools_0.2-19
[64] lubridate_1.9.3      stringi_1.8.3        gtable_0.3.4
[67] munsell_0.5.0        pillar_1.9.0         htmltools_0.5.7
[70] ipred_0.9-14         lava_1.8.0           R6_2.5.1
[73] vroom_1.6.5          evaluate_0.23        backports_1.4.1
[76] class_7.3-22         Rcpp_1.0.12          nlme_3.1-163
[79] prodlim_2023.08.28   xfun_0.41            fs_1.6.3
[82] pkgconfig_2.0.3      ModelMetrics_1.2.2.2
```