Homework 5

HX

Table of contents

Link to the Github repository

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

Please read the instructions carefully before submitting your assignment.

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

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

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

We will need the following packages:

```
rm(list=ls())
packages <- c(
  "dplyr",
  "readr",
```

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

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

Question 1

60 points

Prediction of Median House prices

1.1 (2.5 points)

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

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

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

```
path <- "/Users/alex/Desktop/SP2023/STAT380/hw-5/data/housing.csv"

df <- read.csv(path)
df$ocean_proximity<-as.factor(df$ocean_proximity)
str(df)
```

```
'data.frame':   20640 obs. of  10 variables:
 $ longitude         : num  -122 -122 -122 -122 -122 ...
 $ latitude          : num  37.9 37.9 37.9 37.9 37.9 ...
 $ housing_median_age: num  41 21 52 52 52 52 52 52 42 52 ...
 $ total_rooms       : num  880 7099 1467 1274 1627 ...
 $ total_bedrooms    : num  129 1106 190 235 280 ...
 $ population        : num  322 2401 496 558 565 ...
 $ households        : num  126 1138 177 219 259 ...
 $ median_income     : num  8.33 8.3 7.26 5.64 3.85 ...
 $ median_house_value: num  452600 358500 352100 341300 342200 ...
 $ ocean_proximity   : Factor w/ 5 levels "<1H OCEAN","INLAND",..: 4 4 4 4 4 4 4 4 4 4 ...
```

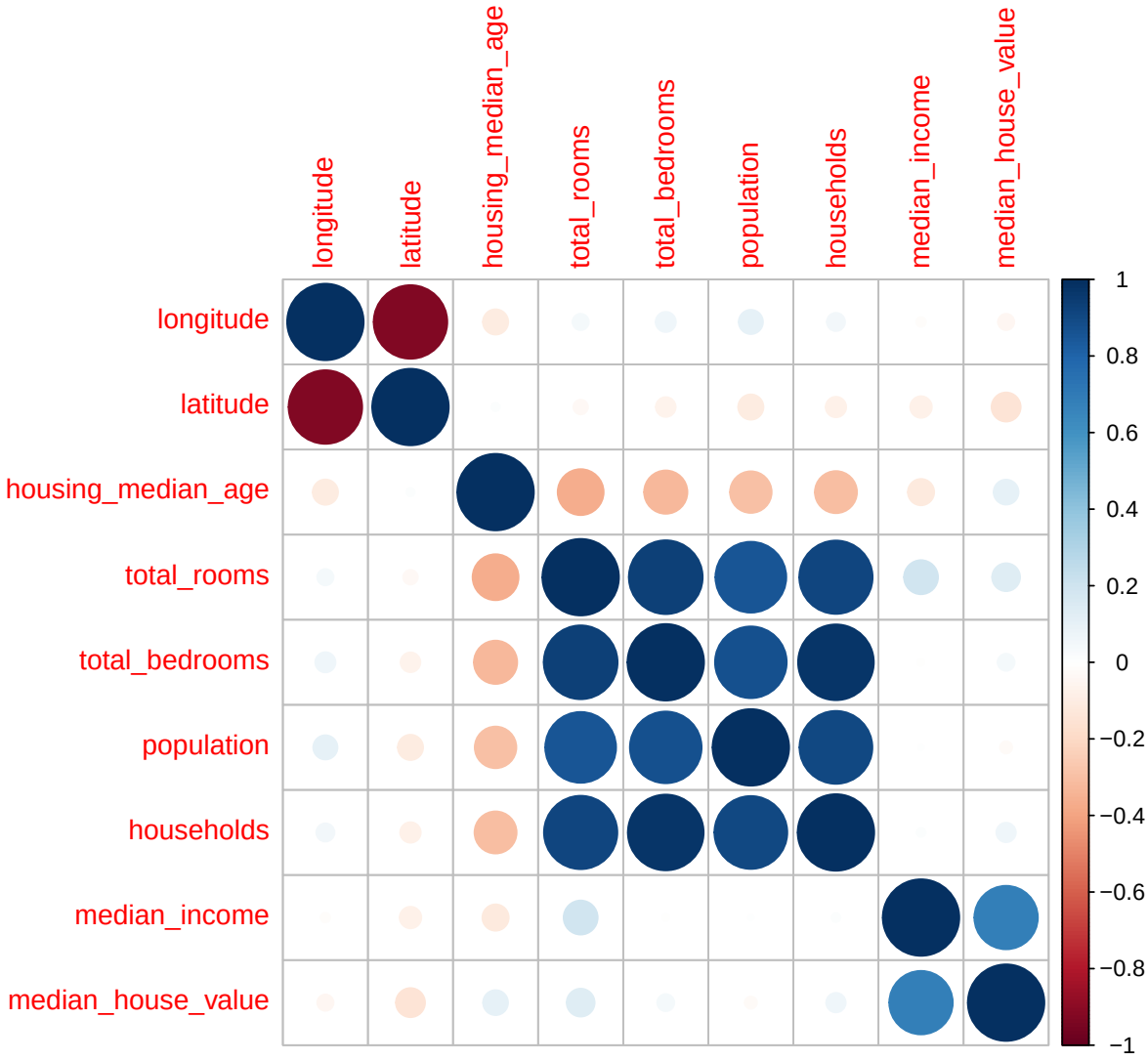
```r
names(df) <- tolower(names(df))

df <- na.omit(df)
```

1.2 (2.5 points)

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

```r
dfnum <- df%>%
  select(!ocean_proximity)
dfcor <- cor(dfnum)
corrplot(dfcor)
```



1.3 (5 points)

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

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

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

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

1.4 (5 points)

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

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

Interpret the coefficients and summarize your results.

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

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

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

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)               -2.273e+06  9.138e+04 -24.873  < 2e-16 ***
latitude                  -2.539e+04  1.047e+03 -24.244  < 2e-16 ***
longitude                 -2.681e+04  1.060e+03 -25.305  < 2e-16 ***
housing_median_age         1.074e+03  4.616e+01  23.261  < 2e-16 ***
total_rooms               -6.159e+00  8.431e-01  -7.306 2.87e-13 ***
total_bedrooms             1.353e+02  4.254e+00  31.804  < 2e-16 ***
population                -3.413e+01  9.838e-01 -34.694  < 2e-16 ***
median_income              3.936e+04  3.573e+02 110.154  < 2e-16 ***
ocean_proximityINLAND     -4.018e+04  1.836e+03 -21.891  < 2e-16 ***
ocean_proximityISLAND      1.324e+05  3.442e+04   3.847  0.00012 ***
ocean_proximityNEAR BAY   -2.522e+03  2.022e+03  -1.247  0.21226
ocean_proximityNEAR OCEAN  4.349e+03  1.658e+03   2.622  0.00875 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

1.5 (5 points)

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

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

lm_predictions <- predict(lm_fit, newdata = df_test)
lm_rmse <- rmse(df_test$median_house_value, lm_predictions)
lm_rmse
```

```
[1] 68339.82
```

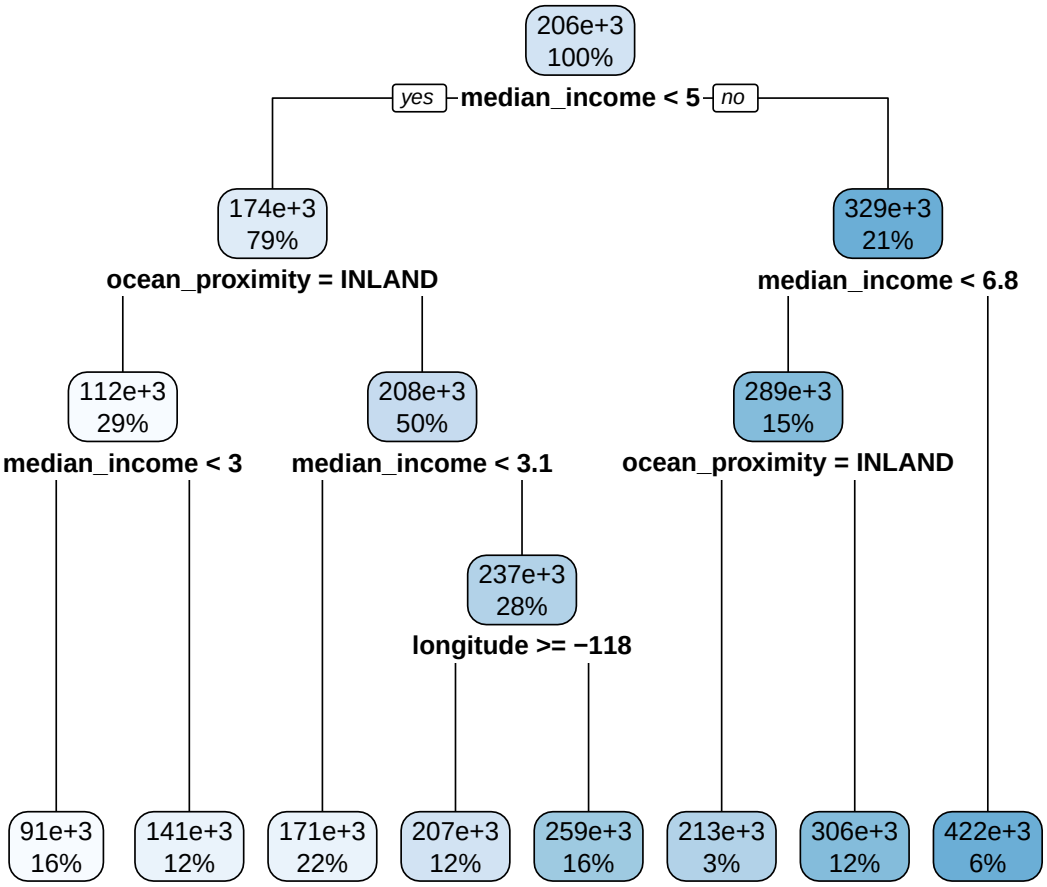
1.6 (5 points)

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

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

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

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



Report the root mean squared error on the test set.

```
rpart_predictions <- predict(rpart_fit, newdata = df_test)
rpart_rmse <- rmse(df_test$median_house_value, rpart_predictions)
rpart_rmse
```

```
[1] 75876.87
```

1.7 (5 points)

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

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

```
[1] 56678.84
```

1.8 (25 points)

Initialize a neural network model architecture:

```
NNet <- nn_module(
  initialize = function(p, q1, q2, q3) {
    self$fc1 <- nn_linear(p, q1)
    self$fc2 <- nn_linear(q1, q2)
    self$fc3 <- nn_linear(q2, q3)
    self$relu <- nn_relu()
    self$dropout <- nn_dropout(p = 0.5)
  },
  forward = function(x) {
    x %>%
```

```
      torch$nn$functional$linear(self$fc1, .) %>%
      self$relu(.) %>%
      self$dropout(.) %>%
      torch$nn$functional$linear(self$fc2, .) %>%
      self$relu(.) %>%
      self$dropout(.) %>%
      torch$nn$functional$linear(self$fc3, .)
  }
)
```

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.

```
nnet_fit <- NNet %>%
  setup(
    ... # Insert your code here
  ) %>%
  set_hparams(
    ... # Insert your code here
  ) %>%
  set_opt_params(
    ... # Insert your code here
  ) %>%
  fit(
    ... # Insert your code here
    dataloader_options = ... # Insert your code here
    verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting

  )
```

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

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

Report the root mean squared error on the test set.

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

> **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?

Lower values of RMSE indicate better fit. Therefore, SVM is better.

```
lm_rmse <- rmse(df_test$median_house_value, lm_predictions)
rpart_rmse <- rmse(df_test$median_house_value, rpart_predictions)
svm_rmse <- rmse(df_test$median_house_value, svm_predictions)
results <- data.frame(
  Model = c("Linear Regression", "Decision Tree", "Support Vector Machine"),
  RMSE = c(lm_rmse, rpart_rmse, svm_rmse)
)
print(results)
```

```
                   Model     RMSE
1      Linear Regression 68339.82
2          Decision Tree 75876.87
3 Support Vector Machine 56678.84
```

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

```
df2 <- read.csv("/Users/alex/Desktop/SP2023/STAT380/hw-5/data/spambase.csv")
invisible(lapply(df2,as.numeric))

names(df2) <- tolower(names(df2))

df2 <- na.omit(df2)
```

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(df2),
  floor( nrow(df2)/10 ),
  replace=FALSE
)

df2_train <- df2[-test_ind, ]
df2_test <- df2[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 <- sum(pred_class == true_class) / length(true_class)
  error <- 1 - accuracy
  true_positives <- sum(pred_class == true_class & true_class == 1)
  true_negatives <- sum(pred_class == true_class & true_class == 0)
```

```
  false_positives <- sum(pred_class != true_class & true_class == 0)
  false_negatives <- sum(pred_class != true_class & true_class == 1)
  true_positive_rate <- true_positives / sum(true_class == 1)
  false_positive_rate <- false_positives / sum(true_class == 0)
  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.

```
glm_fit <- glm(spam ~ ., data = df2_train, family = "binomial")
```

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

```
glm_probs <- predict(glm_fit, newdata = df2_test, type = "response")
glm_classes <- ifelse(glm_probs > 0.5, 1, 0)
accuracy <- mean(glm_classes == df2_test$spam)
accuracy
```

```
[1] 0.923913
```

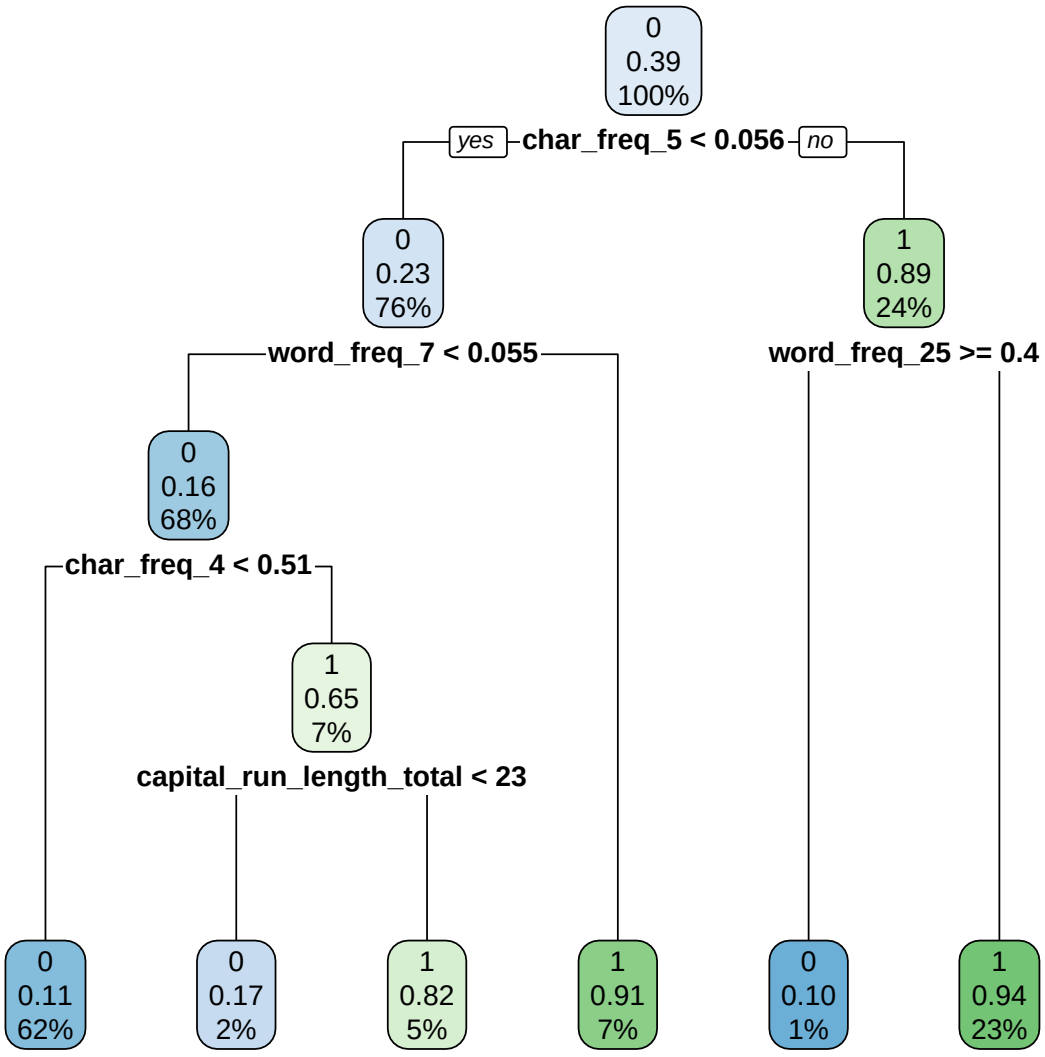
2.4 (5 points)

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

```
rpart_fit2 <- rpart(spam ~ ., data = df2_train, method = "class")
rpart_probs2 <- predict(rpart_fit2, newdata = df2_test, type = "prob")
rpart_classes2 <- ifelse(rpart_probs2[, 2] > 0.5, 1, 0)
```

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

```
rpart.plot(rpart_fit2)
```



Report the prediction accuracy on the test set.

```
rpart_classes2 <- predict(rpart_fit2, newdata = df2_test, type = "class")
rpart_acc2 <- mean(rpart_classes2 == df2_test$spam)
rpart_acc2
```

```
[1] 0.8782609
```

2.5 (5 points)

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

```
if (!is.factor(df2_train$spam)) {
  df2_train$spam <- as.factor(df2_train$spam)
}
svm_fit2 <- svm(spam ~ ., data = df2_train, kernel = "radial")
```

Report the prediction accuracy on the test set.

```
svm_classes2 <- predict(svm_fit2, newdata = df2_test)
svm_acc2 <- mean(svm_classes2 == df2_test$spam)
svm_acc2
```

```
[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.

```
X_train <- model.matrix(spam ~ ., data = df2_train)[,-1]  # Remove the intercept
Y_train <- df2_train$spam

X_test <- model.matrix(spam ~ ., data = df2_test)[,-1]
Y_test <- df2_test$spam

normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

X_train <- as.data.frame(apply(X_train, 2, normalize))
X_test <- as.data.frame(apply(X_test, 2, normalize))

nn_model <- luz::net() %>%
  input(57) %>%  # 57 predictors
  dense(32) %>%
  relu() %>%
  dense(16) %>%
  relu() %>%
  dense(8) %>%
  relu() %>%
  dense(1) %>%
  sigmoid()

set.seed(42)
nn_fit <- nn_model %>%
  luz::optimizer("adam", lr = 0.001) %>%
  luz::loss("binary_crossentropy") %>%
  luz::train(
    X_train,
    Y_train,
    batch_size = 32,
    epochs = 100,
    validation_data = list(X_test, Y_test)
  )

nn_probs <- predict(nn_fit, X_test)

nn_classes <- ifelse(nn_probs > 0.5, 1, 0)
```

```r
nn_accuracy <- mean(nn_classes == Y_test)
nn_accuracy
```

2.7 (5 points)

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

Accuracy of Support vector machine and log regression are the same.

```r
results2 <- data.frame(Model = c('Logistic Regression', 'Decision Tree', 'Support Vector M
                       Accuracy = c(accuracy,rpart_acc2,svm_acc2))
results2
```

```
                  Model  Accuracy
1    Logistic Regression 0.9239130
2          Decision Tree 0.8782609
3 Support Vector Machine 0.9239130
```

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

Question 3

60 points

Three spirals classification

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

This is a multi-class classification problem

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

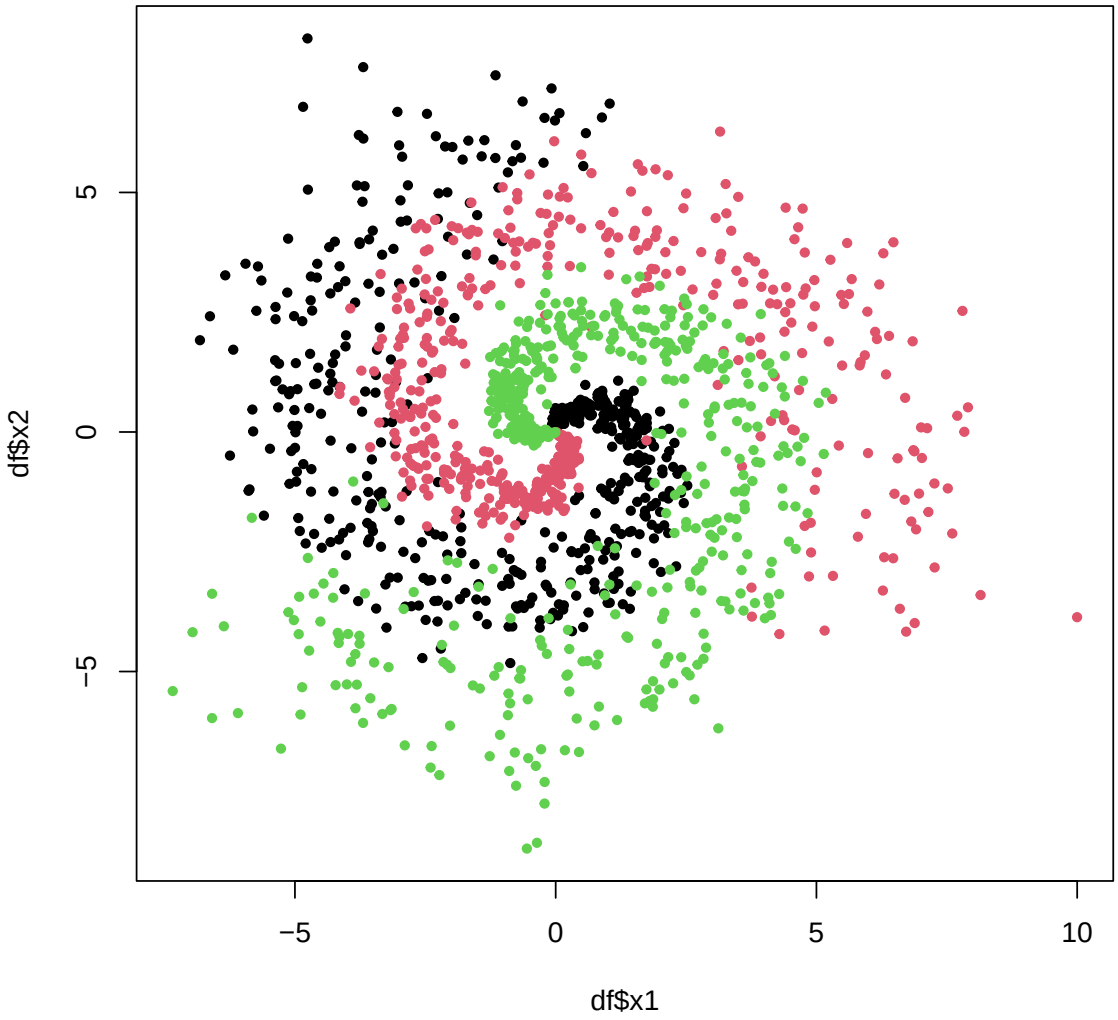
```r
generate_three_spirals <- function(){
  set.seed(42)
  n <- 500
  noise <- 0.2
  t <- (1:n) / n * 2 * pi
  x1 <- c(
      t * (sin(t) + rnorm(n, 0, noise)),
      t * (sin(t + 2 * pi/3) + rnorm(n, 0, noise)),
      t * (sin(t + 4 * pi/3) + rnorm(n, 0, noise))
    )
  x2 <- c(
      t * (cos(t) +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

3.1 (5 points)

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

```r
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`.

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

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

3.2 (10 points)

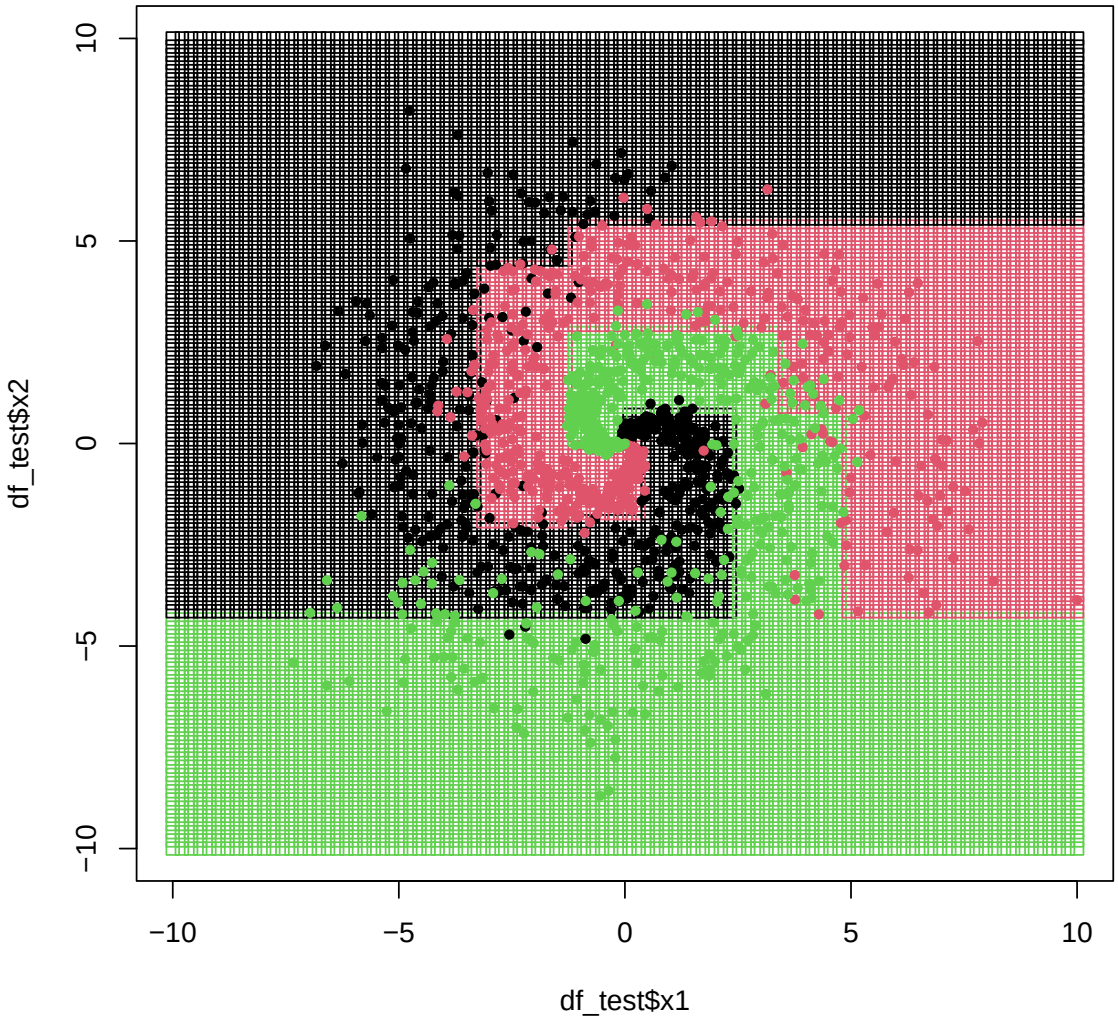
Fit a classification tree model to predict the `y` variable using the `x1` and `x2` predictors, and plot the decision boundary.

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

Plot the decision boundary using the following function:

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

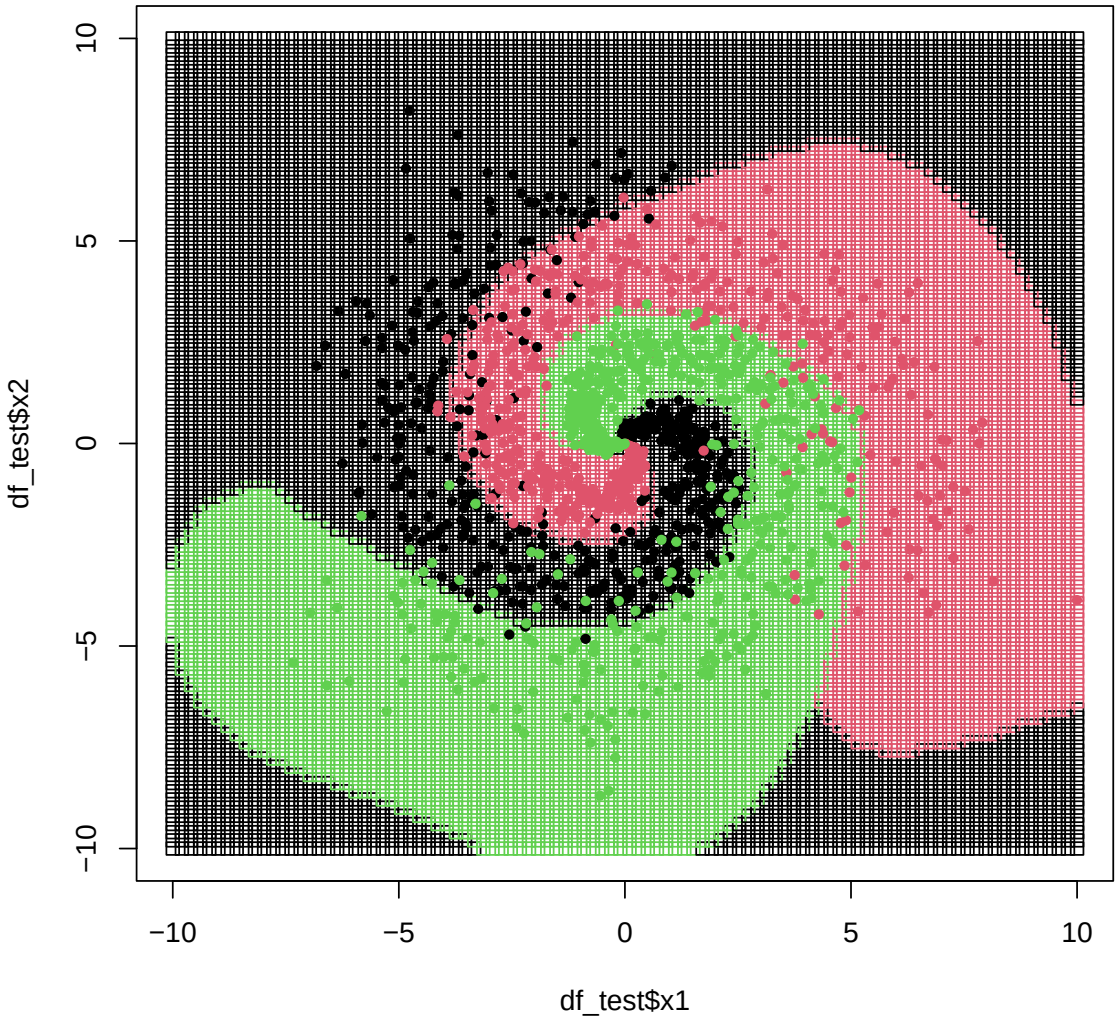
```r
plot_decision_boundary(rpart_classes)
```



3.3 (10 points)

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

```r
svm_fit <- svm_fit <- svm(y ~ x1 + x2, data = df, kernel = "radial", type = "C-classificat
svm_classes <- predict(svm_fit, newdata = df_test)
plot_decision_boundary(svm_classes)
```



Instructions

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

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

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

3.4 (10 points)

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

```
library(torch)
library(luz)
NN1 <- nn_module(
  initialize = function(p, q1, o){
    self$hidden1 <- nn_layer(q1, p)
    self$output <- nn_layer(o, q1)
    self$activation <- nn_sigmoid()
  },
  forward = function(x){
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$output()
  }
)

fit_1 <- NN1 %>%
  setup(
    input_shape = ncol(df) - 1,
    output_shape = 3,
    hidden_shapes = list(q1 = 10)
  ) %>%
  set_hparams(
```

```
    epochs = 50,
    learning_rate = 0.001
  ) %>%
  set_optimizers(
    "adam",
    lr = 0.001
  ) %>%
  fit(
    data = list(
      df %>% select(x1, x2) %>% as.matrix,
      df$y %>% as.integer
    ),
    dataloader_options = list(batch_size = 128, shuffle = TRUE),
    verbose = FALSE
  )

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

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

plot_decision_boundary(fit_1_predictions)
```

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

```
test_matrix <- df_test %>% select(x1, x2) %>% as.matrix
library(ramify)
fit_1_predictions <- predict(fit_1, test_matrix) %>%
  argmax(2) %>%
  as.integer()
```

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$output <- nn_layer(o)
  },
  forward = function(x){
    x %>%
    self$output()
  }
)

fit_0 <- NN0 %>%
  setup(input_size = 2, output_size = 3) %>%
  set_hparams(units = NULL, loss = nn_ce_loss, epochs = 50) %>%
  set_optimizers(adam_optimizer(learning_rate = 0.01)) %>%
  fit(
    data = list(
      df %>% select(x1, x2) %>% as.matrix,
      df$y %>% as.integer
    ),
    dataloader_options = list(batch_size = 128, shuffle = TRUE),
    verbose = FALSE
  )
```

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

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

plot_decision_boundary(fit_0_predictions)
```

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$hidden1 <- nn_linear(q1, p)
    self$hidden2 <- nn_linear(q2, q1)
```

```
    self$hidden3 <- nn_linear(q3, q2)
    self$output <- nn_linear(o, q3)
    self$activation <- nn_functional$relu
  },
  forward = function(x){
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$hidden2() %>%
      self$activation() %>%
      self$hidden3() %>%
      self$activation() %>%
      self$output()
  }
)

fit_2 <- NN2 %>%
  setup(
    input_size = 2,
    output_size = 3
  ) %>%
  set_hparams(
    hidden_sizes = c(10, 10, 10),
    activation_function = "relu",
    learning_rate = 0.01
  ) %>%
  set_optimizers(
    optimizer = "adam"
  ) %>%
  fit(
    X_train = df %>% select(x1, x2) %>% as.matrix(),
    Y_train = df$y %>% as.factor() %>% as.numeric() %>% as.matrix(),
    n_epoch = 50,
    batch_size = 128,
    shuffle = TRUE,
    verbose = FALSE
  )
```

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

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

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

plot_decision_boundary(df_test, fit_2_predictions)
```

3.7 (5 points)

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

The three models are different in terms of the number of hidden layers used to make predictions. As the complexity of the decision boundary increases, the model becomes better able to classify points accurately.

Session Information

Print your R session information using the following command

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur ... 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

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

other attached packages:
 [1] tibble_3.2.1     luz_0.4.0        torch_0.10.0     e1071_1.7-13
 [5] rpart.plot_3.1.1 rpart_4.1.19     caret_6.0-94     lattice_0.20-45
 [9] ggplot2_3.4.2    corrplot_0.92    magrittr_2.0.3   broom_1.0.4
[13] purrr_1.0.1      tidyr_1.3.0      readr_2.1.4      dplyr_1.1.1

loaded via a namespace (and not attached):
 [1] nlme_3.1-162         fs_1.6.1             lubridate_1.9.2
 [4] bit64_4.0.5          progress_1.2.2       tools_4.2.3
 [7] backports_1.4.1      utf8_1.2.3           R6_2.5.1
[10] colorspace_2.1-0     nnet_7.3-18          withr_2.5.0
[13] tidyselect_1.2.0     prettyunits_1.1.1    processx_3.8.0
[16] bit_4.0.5            compiler_4.2.3       cli_3.6.1
[19] scales_1.2.1         callr_3.7.3          proxy_0.4-27
[22] stringr_1.5.0        digest_0.6.31        rmarkdown_2.21
[25] coro_1.0.3           pkgconfig_2.0.3      htmltools_0.5.5
[28] parallelly_1.35.0    fastmap_1.1.1        rlang_1.1.0
[31] rstudioapi_0.14      generics_0.1.3       jsonlite_1.8.4
[34] ModelMetrics_1.2.2.2 Matrix_1.5-3         Rcpp_1.0.10
[37] munsell_0.5.0        fansi_1.0.4          lifecycle_1.0.3
```

```
[40] stringi_1.7.12       pROC_1.18.0          yaml_2.3.7
[43] MASS_7.3-58.2        plyr_1.8.8           recipes_1.0.5
[46] grid_4.2.3           parallel_4.2.3       listenv_0.9.0
[49] crayon_1.5.2         splines_4.2.3        hms_1.1.3
[52] zeallot_0.1.0        knitr_1.42           ps_1.7.4
[55] pillar_1.9.0         future.apply_1.10.0  reshape2_1.4.4
[58] codetools_0.2-19     stats4_4.2.3         glue_1.6.2
[61] evaluate_0.20        data.table_1.14.8    renv_0.16.0-53
[64] vctrs_0.6.1          tzdb_0.3.0           foreach_1.5.2
[67] gtable_0.3.3         future_1.32.0        xfun_0.38
[70] gower_1.0.1          prodlim_2023.03.31   class_7.3-21
[73] survival_3.5-3       timeDate_4022.108    iterators_1.0.14
[76] hardhat_1.3.0        lava_1.7.2.1         timechange_0.2.0
[79] globals_0.16.2       ipred_0.9-14
```