# Homework 5

# Lekh Shetty

# Table of contents

																					2
Question 1																					2
Question 2																					12
Question 3																					18

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:

```
packages <- c(
   "dplyr",
   "readr",</pre>
```

```
"tidyr",
 "purrr",
  "broom",
  "magrittr",
  "corrplot",
  "caret",
  "rpart",
  "rpart.plot",
  "e1071",
  "torch",
  "luz"
#renv::install(packages)
sapply(packages, require, character.only=T)
```

### Question 1



9 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

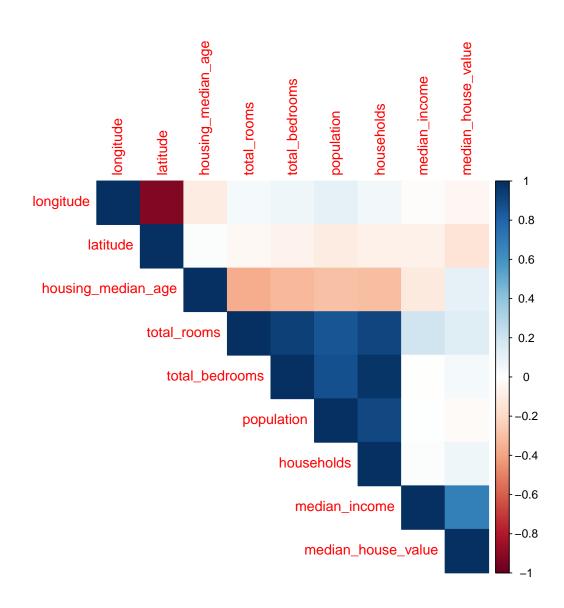
```
path <- "data/housing.csv"</pre>
df <- read_csv(path) %>%
  mutate_if(\(x) is.character(x), as.factor) %>%
  (x)
```

```
names(x) <- tolower(names(x))</pre>
      })
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.
  df <- na.omit(df)</pre>
  head(df)
# A tibble: 6 x 10
  longitude latitude housing_median_age total_rooms total_bedrooms population
      <dbl>
               <dbl>
                                 <dbl>
                                             <dbl>
                                                            <dbl>
                                                                       <dbl>
     -122.
               37.9
                                    41
                                               880
                                                              129
                                                                         322
1
2
     -122.
                                                                        2401
               37.9
                                    21
                                              7099
                                                             1106
3
     -122.
               37.8
                                    52
                                              1467
                                                              190
                                                                         496
4
     -122.
               37.8
                                    52
                                                              235
                                                                         558
                                              1274
     -122.
               37.8
                                    52
                                              1627
                                                              280
                                                                         565
     -122.
               37.8
                                    52
                                                              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()

```
df %>%
  select_if(is.numeric) %>%
  cor() %>%
  corrplot(type = "upper", method = "color")
```



# 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(</pre>
```

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

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

nrow(df_train) + nrow(df_test) == nrow(df)</pre>
[1] TRUE
```

# 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.

```
Estimate Std. Error t value Pr(>|t|)
                         -2.273e+06 9.138e+04 -24.873 < 2e-16 ***
(Intercept)
longitude
                         -2.681e+04 1.060e+03 -25.305 < 2e-16 ***
latitude
                         -2.539e+04 1.047e+03 -24.244 < 2e-16 ***
                          1.074e+03 4.616e+01 23.261 < 2e-16 ***
housing median age
                         -6.159e+00 8.431e-01 -7.306 2.87e-13 ***
total rooms
total bedrooms
                         1.353e+02 4.254e+00 31.804 < 2e-16 ***
population
                         -3.413e+01 9.838e-01 -34.694 < 2e-16 ***
                         3.936e+04 3.573e+02 110.154 < 2e-16 ***
median_income
ocean_proximityINLAND
                         -4.018e+04 1.836e+03 -21.891 < 2e-16 ***
ocean_proximityISLAND
                          1.324e+05 3.442e+04
                                                3.847 0.00012 ***
                         -2.522e+03 2.022e+03 -1.247
ocean_proximityNEAR BAY
                                                       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

```
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

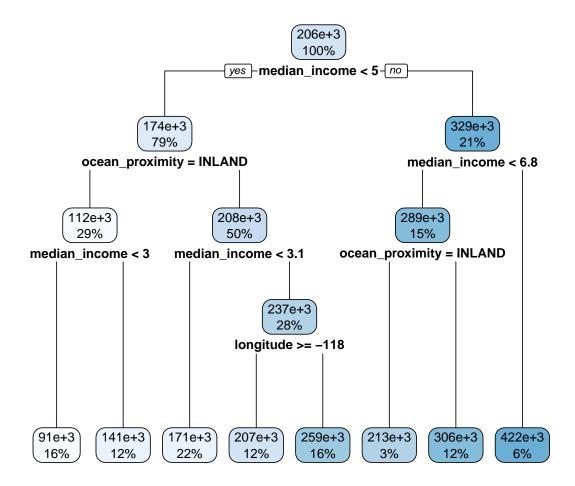
[1] 68339.82

1.6 (5 points)</pre>
```

Fit a decision tree model to predict the median\_house\_value using the same predictors as in 1.4. Use the rpart() function.

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</pre>
```

[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.

#### [1] 69685.65

# 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$hidden3() %>% self$activation() %>%
      self$hidden3() %>% self$activation() %>%
      self$nidden3() %>% self$sigmoid()
```

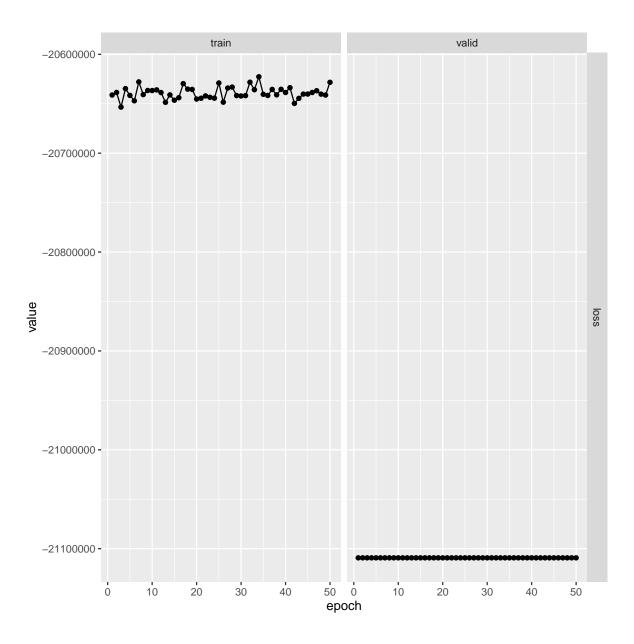
```
)
```

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.

```
M <- model.matrix(median_house_value ~ . -households, data = df_train)</pre>
nnet_fit <- NNet %>%
  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 = 0.1
  ) %>%
  fit(
    data = list(
      model.matrix(median_house_value ~. -households, data = df_train),
      (df_train$median_house_value %>% as.numeric()-1) %>% as.matrix
    ),
    valid_data = list(
      model.matrix(median_house_value ~. -households, data = df_test),
      (df_test$median_house_value %>% as.matrix()-1) %>% as.matrix
    ),
    epochs=50,
    dataloader_options = list(batch_size = 128, shuffle=TRUE),
    verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting
  )
```

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

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



Report the root mean squared error on the test set.

```
nnet_predictions <- predict(
    nnet_fit,
    model.matrix(median_house_value ~ . -households, data = df_test)
)</pre>
```

```
rmse_nnet <- rmse(df_test$median_house_value, nnet_predictions)</pre>
rmse_nnet
```

torch\_tensor 242118 [ CUDAFloatType{} ]



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?

... # Insert your code here

### Question 2



50 points

Spam email classification

The data folder contains the spam.csv dataset. This dataset contains features extracted from a collection of spam and non-spam emails. The objective is to classify the emails as spam or non-spam.

#### 2.1 (2.5 points)

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

1. the variables are of the right data type, e.g., categorical variables are encoded as factors

- 2. all column names to lower case for consistency
- 3. Any observations with missing values are dropped

```
spam <- "data/spambase.csv"</pre>
  df <- read_csv(spam) %>%
    mutate_if(\(x) is.character(x), as.factor) %>%
          names(x) <- tolower(names(x))</pre>
      })
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.
  df <- na.omit(df)</pre>
  head(df)
# A tibble: 6 x 58
  word_freq_1 word_freq_2 word_freq_3 word_freq_4 word_freq_5 word_freq_6
                    <dbl>
                               <dbl>
                                            <dbl>
                                                        <dbl>
        <dbl>
                                                                    <dbl>
         0
                    0.64
                                 0.64
                                               0
                                                         0.32
                                                                     0
1
2
         0.21
                    0.28
                                 0.5
                                                0
                                                         0.14
                                                                     0.28
3
         0.06
                                 0.71
                                                0
                                                         1.23
                                                                     0.19
4
         0
                     0
                                 0
                                                0
                                                         0.63
                                                                     0
5
         0
                     0
                                 0
                                                0
                                                         0.63
                                                                     0
                     0
                                 0
                                                0
                                                         1.85
6
# i 52 more variables: word_freq_7 <dbl>, word_freq_8 <dbl>, word_freq_9 <dbl>,
    word_freq_10 <dbl>, word_freq_11 <dbl>, word_freq_12 <dbl>,
    word_freq_13 <dbl>, word_freq_14 <dbl>, word_freq_15 <dbl>,
   word_freq_16 <dbl>, word_freq_17 <dbl>, word_freq_18 <dbl>,
   word_freq_19 <dbl>, word_freq_20 <dbl>, word_freq_21 <dbl>,
```

word\_freq\_22 <dbl>, word\_freq\_23 <dbl>, word\_freq\_24 <dbl>,
word\_freq\_25 <dbl>, word\_freq\_26 <dbl>, word\_freq\_27 <dbl>, ...

#### 2.2 (2.5 points)

Split the data df into df\_train and df\_split using test\_ind in the code below:

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

df_train <- df[-test_ind, ]
df_test <- df[test_ind, ]</pre>
```

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) {</pre>
 accuracy <- sum(pred_class == true_class) / length(true_class) # Insert your code here</pre>
  error <- 1 - accuracy # Insert your code here
 true_positives <- sum(pred_class=="spam" & true_class=="spam") # Insert your code here</pre>
 true_negatives <- sum(pred_class == "nonspam" & true_class=="nonspam") # Insert your cod
 false_positives <- sum(pred_class="spam" & true_class=="nonspam") # Insert your code her
 false_negatives <- sum(pred_class=="nonspam" & true_class=="spam") # Insert your code he
  true_positive_rate <- true_positives / sum(true_class == "spam") # Insert your code here
 false_positive_rate <- false_positives / sum(true_class == "nonspam") # Insert your cod
 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 = df_train, family = binomial)</pre>
```

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

```
glm_classes <- predict(glm_fit, df_test, type = "response") > 0.5
glm_accuracy <- mean(glm_classes == df_test$spam)
glm_accuracy</pre>
```

[1] 0.923913

2.4 (5 points)

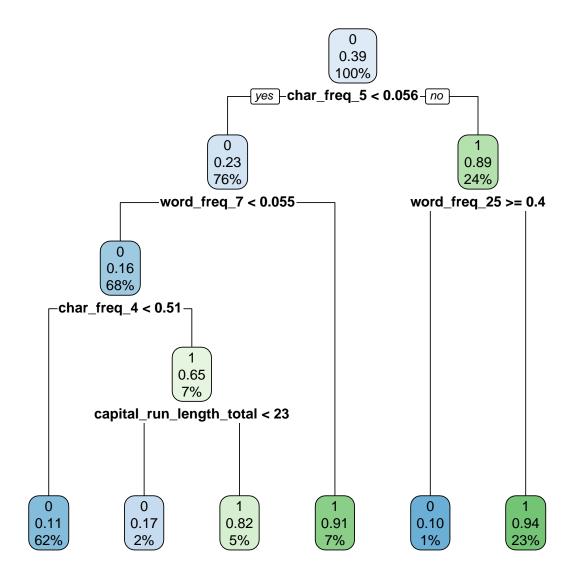
Fit a decision tree model to predict the spam variable using the remaining predictors. Use the rpart() function and set the method argument to "class".

```
rpart_fit <- rpart(
   spam ~ .,
   data = df_train,
   method = "class"
)

rpart_classes <- predict(rpart_fit, df_test, type = "class")</pre>
```

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

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



Report the prediction accuracy on the test set.

```
rpart_test_pred <- predict(rpart_fit, df_test, type="class")
rpart_classes <- mean(rpart_test_pred == df_test$spam)
rpart_classes</pre>
```

[1] 0.8782609

2.5 (5 points)

Fit a support vector machine model to predict the spam variable using the remaining predictors. Use the sym() 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 = df_train, kernel = "linear", type = "C-classification")</pre>
```

Report the prediction accuracy on the test set.

```
svm_classes <- predict(svm_fit, newdata = df_test)</pre>
accuracy <- mean(svm_classes == df_test$spam)</pre>
accuracy
```

[1] 0.9173913

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.

A 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.

```
M <- model.matrix(spam ~ 0+.,data=df_train)</pre>
nnet_fit <- NNet %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam,
  ) %>%
```

```
set_hparams(
  p = ncol(M), q1=32, q2=16, q3=8
) %>%
set_opt_hparams(
  lr = 0.1
) %>%
fit(
  data = list(
    model.matrix(spam ~ 0 + ., data=df_train),
    (df_train$spam %>% as.numeric()-1) %>% as.matrix
  ),
  valid_data = list(
    model.matrix(spam ~ 0 +., data=df_test),
    (df_test$spam %>% as.matrix()-1) %>% as.matrix
  ),
  epochs=50,
  dataloader_options = list(batch_size = 128, shuffle=TRUE),
  verbose = FALSE # Change to TRUE while tuning. But, set to FALSE before submitting
)
```

#### 2.7 (5 points)

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

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

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



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.

A This is a multi-class classification problem

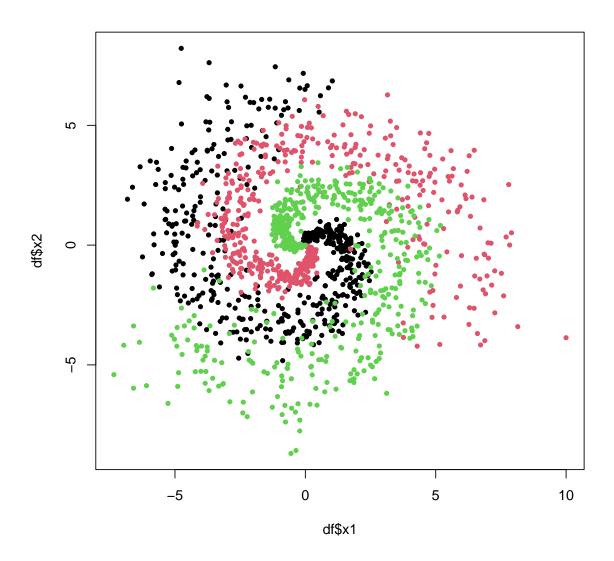
The dataset can be generated using the provided R code below:

```
generate_three_spirals <- function(){</pre>
 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.

```
df <- generate_three_spirals()
plot(
   df$x1, df$x2,
   col = df$y,
   pch = 20
)</pre>
```



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.

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

df_test <- as_tibble(grid)</pre>
```

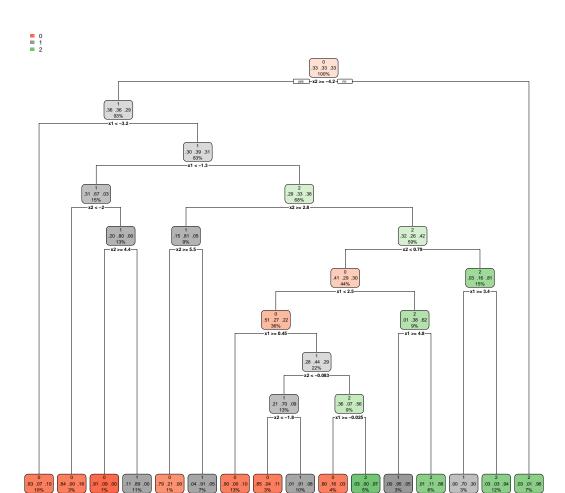
# 3.2 (10 points)

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

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

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

rpart.plot(rpart_fit)</pre>
```

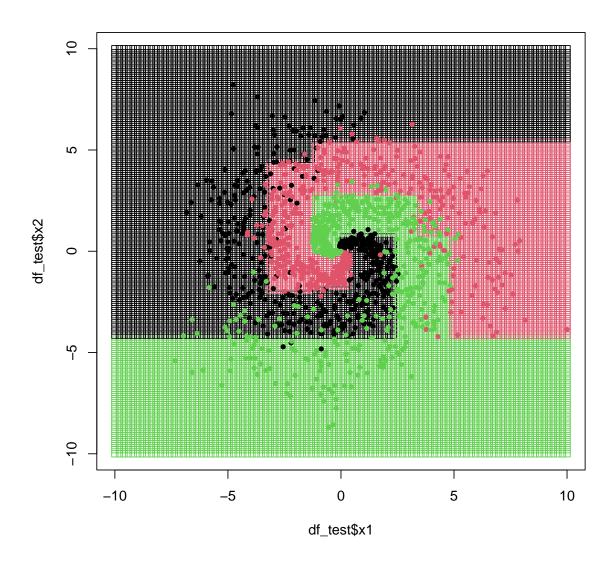


Plot the decision boundary using the following function:

```
plot_decision_boundary <- function(predictions){
  plot(
    df_test$x1, df_test$x2,
    col = predictions,
    pch = 0
)</pre>
```

```
points(
    df$x1, df$x2,
    col = df$y,
    pch = 20
)

plot_decision_boundary(rpart_classes)
```

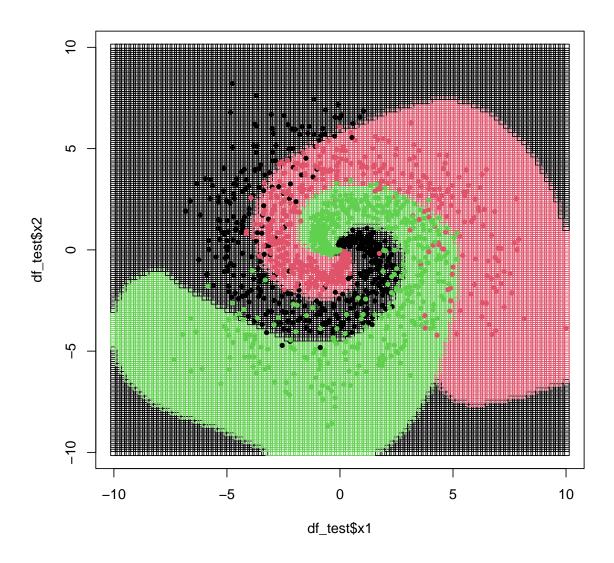


# 3.3 (10 points)

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.

```
svm_fit <- ... # Insert your code here
svm_classes <- ... # Insert your code here
plot_decision_boundary(svm_classes)

svm_fit <- svm(y ~ ., data = df, kernel = "radial", type = "C-classification")
svm_classes <- predict(svm_fit, df_test)
plot_decision_boundary(svm_classes)</pre>
```



#### ⚠ 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(</pre>
  initialize = function(p, q1, o){
    self$hidden1 <- nn_linear(p, q1)</pre>
    self$output <- nn_linear(q1, o)</pre>
    self$activation <- nn_sigmoid()</pre>
  },
  forward = function(x){
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$output()
  }
)
fit_1 <- NN1 %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(
    p = ncol(df_test), q1=10, o=3
  ) %>%
```

```
set_opt_hparams(
    lr = 0.001
) %>%
fit(
    data = list(
        df %>% select(x1, x2) %>% as.matrix,
        df$y %>% as.integer
),
    epochs=50,
    dataloader_options = list(batch_size = 200, shuffle = TRUE),
    verbose = FALSE
)
```

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
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.

```
NNO <- nn_module(
  initialize = function(p, o){
    self$output <- nn_dense(p, o) # Insert your code here
},
  forward = function(x){
    x %>%
    self$output() # Insert your code here
}
)
fit_0 <- NNO %>%
```

```
setup(...) %>%
set_hparams(...) %>%
set_opt_params(...) %>%
fit(...)
```

Plot the results using the plot\_decision\_boundary() function.

3.6 (10 points)

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

```
NN2 <- nn_module(
  initialize = function(p, q1, q2, o){
    ... # Insert your code here
  },
  forward = function(x){
    x %>%
    ... # Insert your code here
  }
)

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

Plot the results using the plot\_decision\_boundary() function.

3.7 (5 points)

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

# i Session Information

Print your R session information using the following command

#### sessionInfo()

R version 4.2.3 (2023-03-15 ucrt)

Platform: x86\_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 19044)

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

#### attached base packages:

[1] stats graphics grDevices datasets utils methods base

#### other attached packages:

[1]	luz_0.4.0	torch_0.10.0	e1071_1.7-13	rpart.plot_3.1.1
[5]	rpart_4.1.19	caret_6.0-94	lattice_0.20-45	ggplot2_3.4.1
[9]	corrplot_0.92	magrittr_2.0.3	$broom_1.0.4$	purrr_1.0.1
[12]	tidur 1 3 0	readr 2 1 4	dnlwr 1 1 1	

#### loaded via a namespace (and not attached):

[1]	nlme_3.1-162	fs_1.6.1	<pre>lubridate_1.9.2</pre>
[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	<pre>prettyunits_1.1.1</pre>	processx_3.8.0
[16]	bit_4.0.5	compiler_4.2.3	cli_3.6.1
[19]	labeling_0.4.2	scales_1.2.1	callr_3.7.3
[22]	proxy_0.4-27	stringr_1.5.0	digest_0.6.31
[25]	rmarkdown_2.21	coro_1.0.3	pkgconfig_2.0.3
[28]	htmltools_0.5.5	parallelly_1.34.0	fastmap_1.1.1
[31]	rlang_1.1.0	farver_2.1.1	generics_0.1.3

[34] jsonlite_1.8.4	vroom_1.6.1	ModelMetrics_1.2.2.2
[37] Matrix_1.5-3	Rcpp_1.0.10	munsell_0.5.0
[40] fansi_1.0.4	lifecycle_1.0.3	stringi_1.7.12
[43] pROC_1.18.0	yaml_2.3.7	MASS_7.3-58.2
[46] plyr_1.8.8	recipes_1.0.5	grid_4.2.3
[49] parallel_4.2.3	listenv_0.9.0	crayon_1.5.2
[52] splines_4.2.3	hms_1.1.3	zeallot_0.1.0
[55] knitr_1.42	ps_1.7.4	pillar_1.9.0
[58] future.apply_1.10.0	reshape2_1.4.4	codetools_0.2-19
[61] stats4_4.2.3	glue_1.6.2	evaluate_0.20
[64] data.table_1.14.8	renv_0.17.2	vctrs_0.6.1
[67] tzdb_0.3.0	foreach_1.5.2	gtable_0.3.2
[70] future_1.32.0	xfun_0.38	gower_1.0.1
[73] prodlim_2019.11.13	class_7.3-21	survival_3.5-3
[76] timeDate_4022.108	tibble_3.2.1	iterators_1.0.14
[79] hardhat_1.2.0	lava_1.7.2.1	timechange_0.2.0
[82] globals_0.16.2	ellipsis_0.3.2	ipred_0.9-14