Machine Learning Assignment

Gina Fornasari - Max Hall

Due: 2024-09-27

STA3043S – Intro to Machine Learning Assignment

Question 2

Question 2(a)

```
# Question 2A
par(mfrow = c(1,2))
plot(tweets ~ connectedness, data = data, pch = 16, cex = 0.8, col = "red"
             , xlab = "User Connectedness", ylab = "Number of Tweets")
model <- lm(tweets ~ connectedness, data = data)
abline(model, col = "blue", lwd = 2, lty = 'dashed')
legend("topleft", legend = "Line of Best Fit", lty = 2,
                  lwd = 2, cex = 0.6, col = 'blue',  bg="white") #cex makes it smaller
plot(tweets ~ engagement, data = data, pch = 16, cex = 0.8, col = "darkgreen"
             , xlab = "User Engagement", ylab = "Number of Tweets")
model <- lm(tweets ~ engagement, data = data)
abline(model, col = "blue", lwd = 2, lty = 'dashed')
legend("topleft", legend = "Line of Best Fit", lty = 2,
                  lwd = 2, cex = 0.6, col = 'blue',  bg="white")
```

```
# Users who haven't posted automatically get classed as neutral
cleaned_sentiment_data <- data.frame(tweets = data[data$tweets > 0,]$tweets,
                           sentiment = data[data$tweets > 0,]$sentiment)

par(mfrow = c(1,2))
boxplot(data$tweets ~ data$sentiment, pch = 16, cex = 0.8, col = "red",
                      xlab = "User Sentiment", ylab = "Number of Tweets",
                      ylim = c(0,max(data$tweets)))
boxplot(cleaned_sentiment_data$tweets ~ cleaned_sentiment_data$sentiment,
                                pch = 16, cex = 0.8, col = "darkgreen",
                                xlab = "User Sentiment",
                                ylab = "Number of Tweets",
                                ylim = c(0,max(cleaned_sentiment_data$tweets)))
```

Question 2(b)

It can be seen using the scatter plots (Figure 1) that for both continuous predictors (User Connectedness and User Engagement) there shows to be an observed positive correlation with the number of tweets. As the

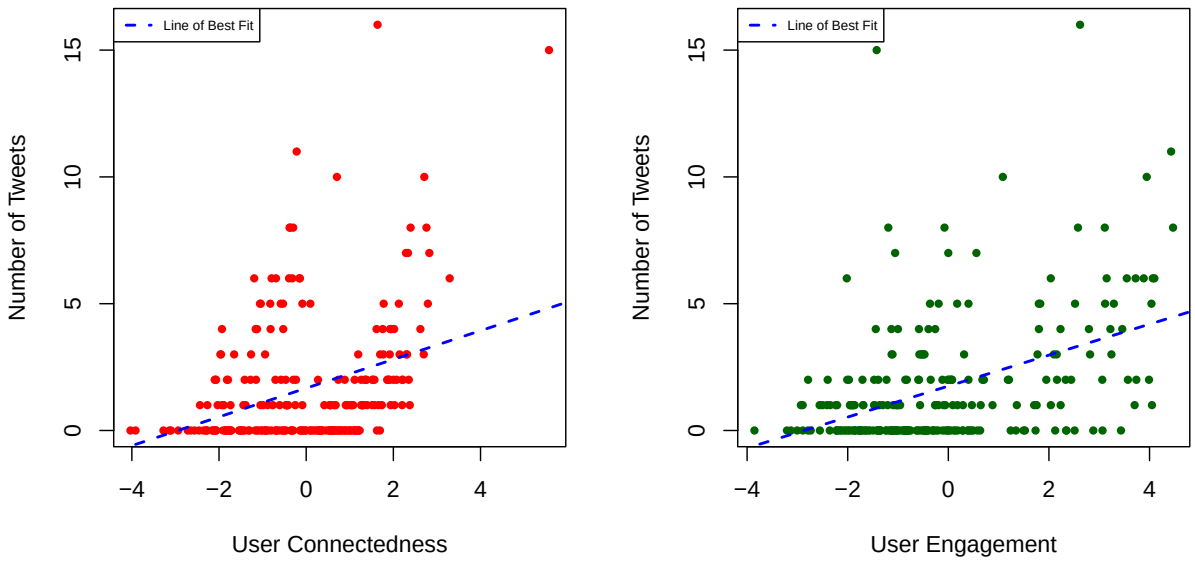


Figure 1: Scatterplot of Tweets as a function of Connectedness and Follower Engagement.

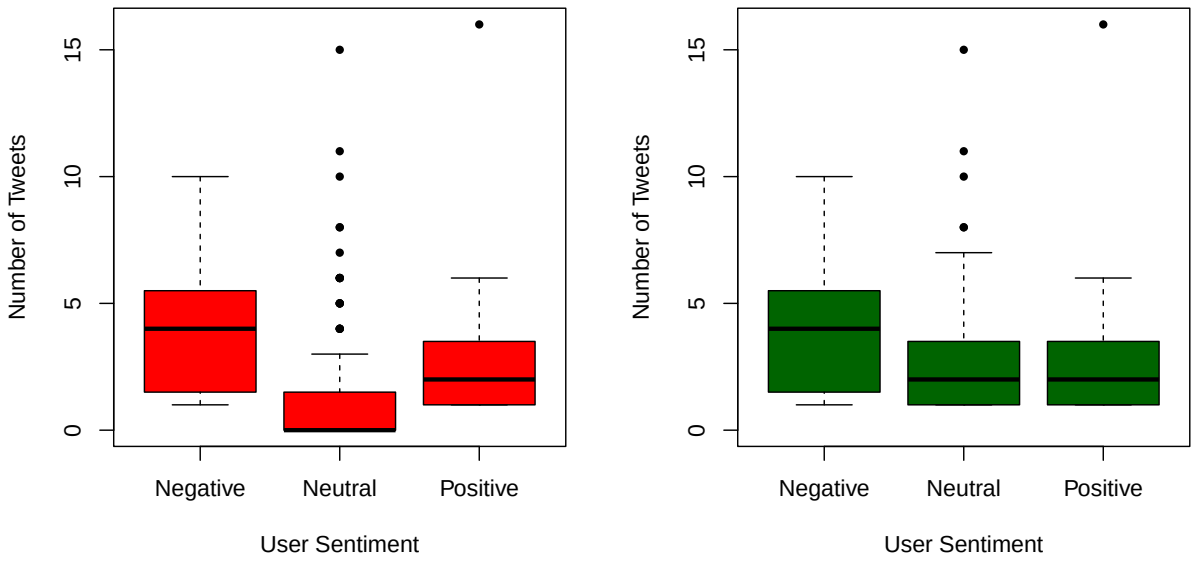


Figure 2: Boxplot of Tweets by level of User Sentiment. The left-hand plot removes the users who haven't posted as their sentiment is logically unknown.

predictors increase, the number of tweets on average increase. This is best observed using the line of best fit, shown on both plots. It must be noted however that there is still large amounts of variance to the line of best fit, indicating patterns in the data have not been observed fully.

For the categorical predictors, upon initial inspection of the box-plot (Figure 2) it appears that there is a difference between the groups. However, if users automatically classed as neutral due to not posting are removed (Figure 2), then it is observed that there is very little difference between groups, with maybe a slightly higher number of tweets from negative users, this would need to be investigated further with newer data to confirm. There also appears to be mainly outliers in the neutral sentiment group.

Question 3

Question 3(a)

```
# Question 3A
set.seed(1)
model = rpart(tweets~sentiment + connectedness + engagement,
              data = data, control = rpart.control(minsplit = 4,cp = 0.04))
```

Question 3(b)

```
# Question 3B
par(mfrow = c(1,2))
plotcp(model)
abline(v = 2, col = 'red', lty = 2, lwd = 2)

# Pruning Tree with optimal cp value
model_prune <- prune.rpart(model, cp = 0.16)
rpart.plot(model_prune)
```

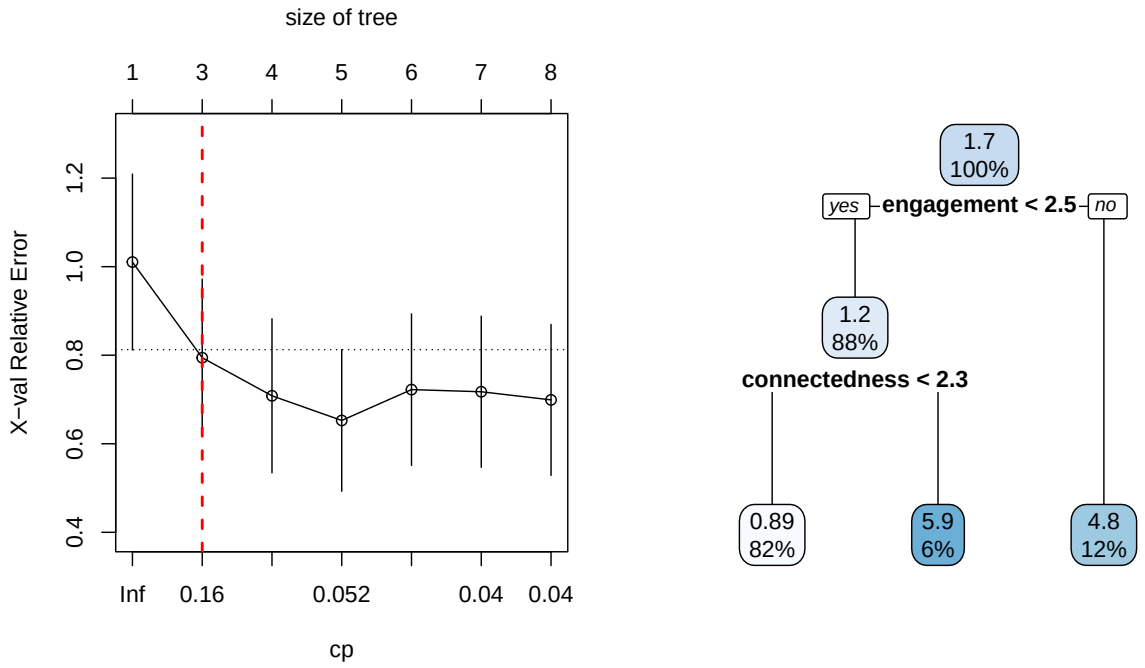


Figure 3: Validation Analysis Plot with the Pruned Tree.

Question 3(c)

The pruned plot (Figure 3) shows for an optimal cp value of 0.16, selected using the simplest model within one standard deviation of the model with the lowest error, follower Engagement and Correctness are influential factors with an initial split in the users on Engagement. If Engagement is below 2.5 then they can be split

again on Connectedness based on if it is above or below 2.3. Sentiment is not used as a contributing factor in the tree, meaning that the influence of Sentiment was not large enough to be considered as a splitting point, making it redundant for the purposes of predicting the number of tweets.

It should be noted that if the original tree is plotted, it can be seen that the un-pruned model also dismisses sentiment as an influential factor. This is further evidence against Sentiment being an significant factor. This also supports the experimental data analysis which saw little change in the tweets based on sentiment, shown in the boxplot in Figure 2.

Question 3(d)

```
# Setting up the validation data split
N <- dim(data)[1]
trainingSet <- sample(1:N, floor(0.8*N), replace = FALSE)
trainingData <- as.h2o(data[trainingSet,])
ValidationData <- as.h2o(data[-trainingSet,])

# Get sequence of examples of L2 Parameters
L2Parameters <- exp(seq(-10,-3, length = 20))
validation_errors <- rep(NA, 20)

# Running the neural network for different L2 Parameters
create_NN <- function(L2_Parameter)
{
  neural_network_model <- h2o.deeplearning(
            x = 2:4, y = 1,
            training_frame = trainingData,
            validation_frame = ValidationData,
            reproducible = TRUE, standardize = TRUE, hidden = c(5,5),
            activation = 'tanh', loss = 'Quadratic',
            rate = 0.01, adaptive_rate = FALSE,
            epochs = 1000, seed = 2,
            l2 = L2_Parameter)
  return (neural_network_model)
}

# Storing the different validation errors
for (i in 1:20)
{
    current_model <- create_NN(L2Parameters[i])
    validation_errors[i] <- h2o.mse(current_model, valid = TRUE)[1]
}
# Finding the optimal L2 Parameter
minL2Parameter <- L2Parameters[which.min(validation_errors)]

# Plotting the validation errors against the different parameters
plot(validation_errors ~ L2Parameters, ylab = 'Validation Errors'
                          , xlab = 'L2 Regularisation Parameter')
abline(v = minL2Parameter, col = 'red', lwd = 2, lty = 2)
text(minL2Parameter, 0.85, pos = 4, col = 'black', cex = 0.8, font = 2,
    labels = paste0('parameter = ', round(minL2Parameter,3)), offset = 0.5)
```

L2 regularization specifies a threshold/limit on the sum of squares of the weights of each parameter in the model. By starting with an overly complex model and constraining it with regularization, the complexity of

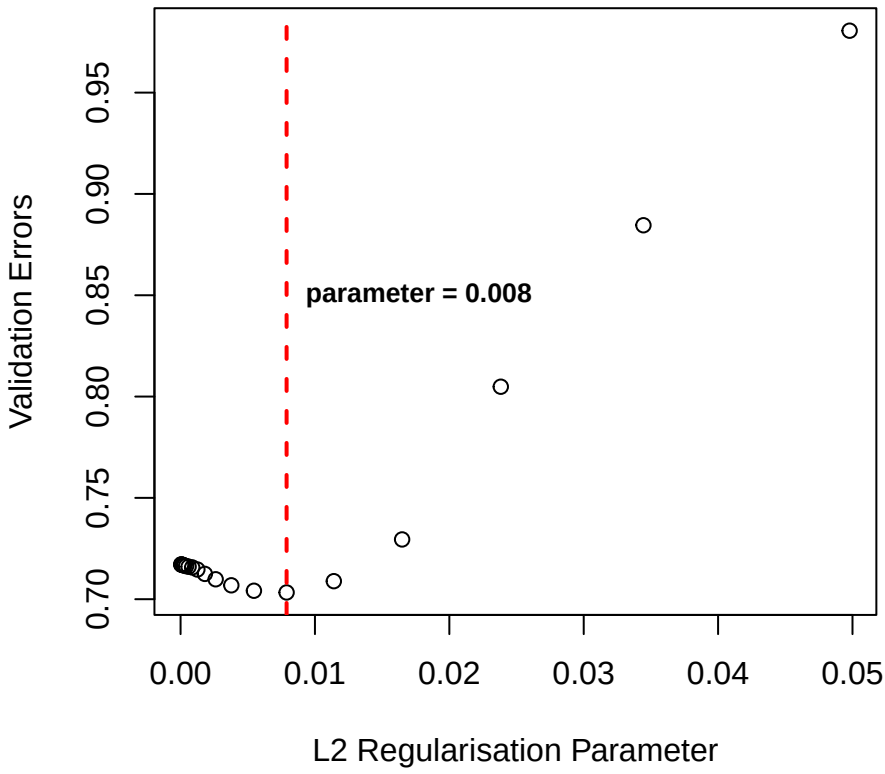


Figure 4: Plot of Neural Network Validation Set Errors as a Function of the Regularisation Parameter.

the model is decreased but the model can still extract the underlying pattern of the data in order to make accurate predictions. This is relevant in determining the existence and nature of relationships between the predictors and responses because it assigns appropriate weights to each parameter, which can be interpreted as the "measure" of influence of each parameter.

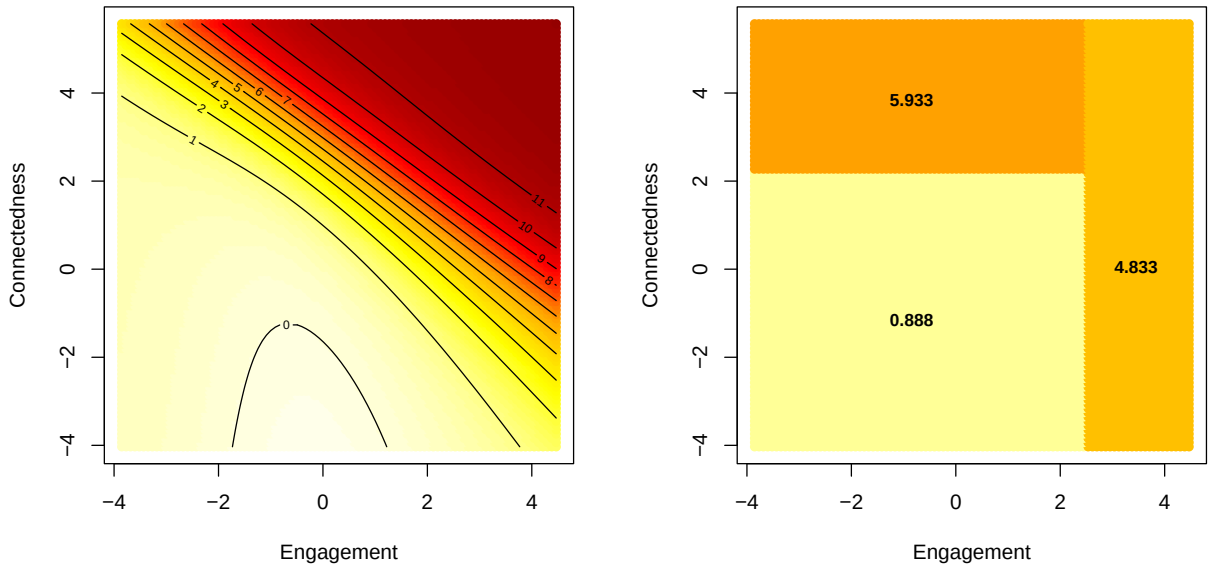
The appropriate L2 Regularization parameter used is the one which minimizes the Validation Error, therefore 0.008 is chosen for the L2 parameter as shown in Figure 4.

Question 4(a)

For a tree-based model, the response curve should be constructed using the pruned tree. Due to the validation analysis done, the pruned model's relative error lies within one standard deviation of the minimum. This means that the pruned model's predictive performance will be almost indistinguishable to that of optimal performance but is a much less complex model, making it less computationally expensive to use.

Question 4(b)

```
# Contour plot
M = 100
engagement_seq = seq(min(data$engagement), max(data$engagement), length = M)
connect_seq = seq(min(data$connectedness), max(data$connectedness), length = M)

# Setting up the points to test across
engagement = rep(engagement_seq, M)
connectedness = rep(connect_seq, each = M)
Lat = data.frame(engagement=engagement, connectedness=connectedness
                                                , sentiment='Neutral')

# Getting predictions from the neural network and the tree
responsePredictionsNN = as.matrix(h2o.predict(selected_NN_Model, as.h2o(Lat)))
responsePredictionsTree = as.matrix(predict(selected_Tree_Model, Lat))

# Matrix to aid in matching Predictions to Connectedness and Engagement values
pred_matrix = matrix(as.vector(responsePredictionsNN),
                     nrow = length(connect_seq), ncol = length(engagement_seq))

# Plots the points with gradiented colour - For NN Model
plot(Lat[-3], pch = 16, col = colour.gradient(pred_matrix), xlab = "Engagement",
                         ylab = "Connectedness")

# Plots the contour plot of the predictions over the colour
contour(engagement_seq, connect_seq, pred_matrix, add = TRUE)

# Matrix to aid in matching Predictions to Connectedness and Engagement values
pred_matrix = matrix(as.vector(responsePredictionsTree),
                     nrow = length(connect_seq), ncol = length(engagement_seq))

# For Tree Model
plot(Lat[-3], pch = 16, col = colour.gradient(pred_matrix), xlab = "Engagement",
                         ylab = "Connectedness")
text(2.8, 0, labels = paste0(4.833), pos = 4, col = 'black', cex = 0.8,
     font = 2)
text(-1.5, -1.2, labels = paste0(0.888), pos = 4, col = 'black', cex = 0.8,
     font = 2)
text(-1.5, 3.8, labels = paste0(5.933), pos = 4, col = 'black', cex = 0.8,
     font = 2)
```



(a) Neural Network Model Predictions.

(b) Tree Model Predictions.

Figure 5: Response Curve over the Engagement and Connectedness for an individual with Neutral Sentiment.

Question 4(c)

It is observed in Figure 5(a) that the (5,5) neural network model predicts zero tweets specifically around an engagement value of 0 and negative connectedness. The model predicts for an increase in tweets with the combination of increase in connectedness and engagement. A low value of either connectedness or engagement results in a low number of predicted tweets regardless of the value of the other measure. It can also be seen that the model predicts negative values. This makes sense as it is a non-linear combination of weights and inputs, thus in some cases, when extrapolating, a return value is outside of the reasonable bounds of the response. This could be remedied by amending the output function to be strictly positive, however the h2o library does not allow for such modifications to the output activation function.

The Tree-based model observed in Figure 5(b) unsurprisingly gives very limited feedback based on the predictions, since there are only 3 leaves to the optimal tree, all of the data is broken into 3 separate categories. and thus 3 'levels' of predicted tweets. It can be observed that high levels of connectedness generally results in a medium numbers of tweets, however if engagement is also high then the number of tweets are expected to decrease slightly. In the case of low connectedness and engagement there are few to no tweets expected from the user by the model. This model is relatively poor at predicting refined number of tweets of users, however can be useful in generally grouping users and gaining a general overview of the expected tweets.

Question 5

Question 5(a)

```
# Question 5A
set.seed(2)
trainingSet  <- sample(1:N, floor(0.8*N), replace = FALSE)
trainingData <- data[trainingSet,]
validData    <- data[-trainingSet,]
```

Question 5(b)

```
# Question 5B
cpValues <- exp(seq(-8,-4, length = 100))

# Function to calculate the log likelihood
negative_log_likelihood <- function(cp, data_train, data_valid)
{
  sum = 0
  model <- rpart(tweets~., data = data_train,
                 control = rpart.control(minsplit = 4,cp = cp))

  # Calculates the log likelihood
  for (i in 1:dim(data_valid)[1])
  {
    pred_y   <- as.numeric(predict(model, data_valid[i,]))
    actual_y <- data_valid$tweets[i]
    sum = sum - (actual_y*log(pred_y) - pred_y - log(factorial(actual_y)))
  }
  return (sum)
}

# Creates space
LLValues <- rep(NA, length(cpValues))

# Getting the LLValues for the set of cpValues
for (i in 1:length(cpValues))
{
  LLValues[i] = negative_log_likelihood(cpValues[i], trainingData, validData)
}

# Validation Analysis Plot with Log-Likelihood Value
plot(LLValues~cpValues, type = 'b')
abline(v = max(cpValues[which(LLValues == min(LLValues, na.rm = TRUE))]),
       col = 'red', lwd = 2, lty = 2)
bestCPValue <- max(cpValues[which(LLValues == min(LLValues, na.rm = TRUE))])
text(bestCPValue, 50, labels = paste0(round(bestCPValue,3)),
     pos = 4, col = 'black', cex = 0.8, font = 2)
```

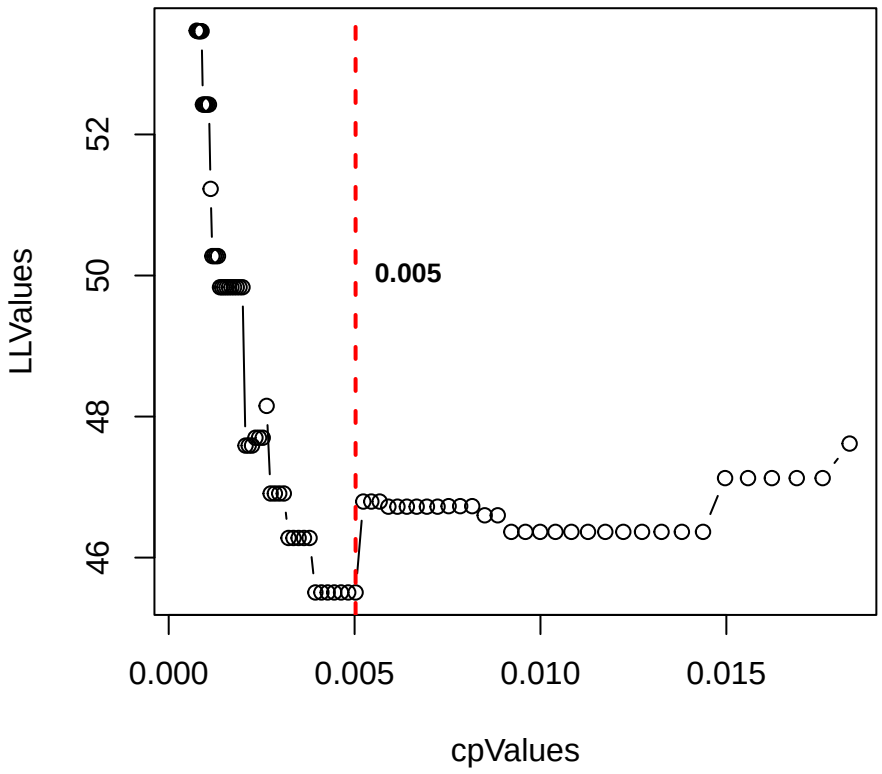


Figure 6: Log-Likelihood Validation Analysis Plot.

Question 5(c)

Yes, using a tree diagram with the optimal parameter from this analysis leads to a very complicated model even with selecting the furthest right cp value of the minimum log likelihoods. Not doing the cross validation analysis means that the value is only selected based on it performing for this specific split in the data into a validation and training set. In this case the current split in the data poorly represents the data as a whole and thus the cp value leads to a poorly representing model.

Question 4 would allow a more complicated model plot, allowing groupings of new data more similar to the the training data used to train the model, however this doesn't guarantee that it represents the underlying pattern of the data. The model cp parameter using the cross validation analysis results in a model with 3 leaves. The model cp parameter using the log likelihood optimisation has 20 leaves, this compromising of a much more complicated model, which is likely overfitted on the training set.

Question 6

Question 6(a)

```r
# Question 6A
k_means <- function(X, K = 1, iter = 15)
{
  # Initialisations:
  set.seed(2024)
  N <- dim(X)[1]
  d <- dim(X)[2]
  startingIndicies <- sample(1:N, K, replace = FALSE)
  Mus <- X[startingIndicies,]
  ones  = matrix(1, N, 1)
  dists = matrix(NA, N, K)
  error = rep(NA, iter)

  if (K == 1)
  {
    Mus <- colSums(as.matrix(X))/N
    dists[,1] <- rowSums((X-ones%*%Mus)^2)
    error <- N*sum((dists[,1])^2)
    return(list(D = dists, SSQ = error, z = ones))
  }

  for (i in 1:iter)
  {
    # Assigning points to clusters
    dists <- dists*0
    for (k in 1:K)
    {
      dists[,k] <- rowSums((X-ones%*%Mus[k,])^2)
    }
    cluster_indicies <- apply(dists, 1, which.min)

    # Calculating error
    err = 0
    cluster_dists <- apply(dists, 1, min)
    for (k in 1:K)
    {
      obs_from_cluster <- which(cluster_indicies == k)
      nk <- length(obs_from_cluster)
      err <- err + nk*sum(cluster_dists[obs_from_cluster])

      # Setting new cluster means
      if (any(cluster_indicies == k))
      {
        Mus[k,] <- colSums(as.matrix(X[obs_from_cluster,]))/nk
        if (nk == 1) {Mus[k,] = X[obs_from_cluster, ]} # For single Obs
      }
      else # If empty cluster
      {
        furtherest_point <- which.max(apply(dists, 1, min))
```

```
        Mus[k,] <- X[furtherest_point,]
      }
    }
    error[i] <- err
  }
  return(list(D = dists, SSQ = error, z = cluster_indicies))
}
```

Question 6(b)

```
# Question 6B
SSC = function(res_k_means)
{
  if (dim(res_k_means$D)[2] == 1)
  {
    return(list(s=NA, s_bar_k = NA, s_bar_overall = NA))
  }

  s <- rep(NA,dim(res_k_means$D)[1])
  for (i in 1:length(s))
  {
    # Getting nearest and second nearest cluster distances
    r_2 <- min(res_k_means$D[i,][-res_k_means$z[i]])
    r_1 <- res_k_means$D[i,][res_k_means$z[i]]

    # Calculating the simplified Silhouette Scores
    s[i] <- (r_2-r_1)/r_2
  }

  # Getting the average simplified Silhouette Scores per cluster
  s_bar_k <- rep(NA,dim(res_k_means$D)[2])
  for (k in 1:length(s_bar_k))
  {
    s_bar_k[k] <- mean(s[which(res_k_means$z == k)])
  }
  # Getting the overall average Silhouette Scores
  s_bar_overall <- mean(s_bar_k)

  return(list(s=s, s_bar_k = s_bar_k, s_bar_overall = s_bar_overall))
}
```

Question 6(c)

```
# Question 6C
K_results = matrix(NA, nrow=6, ncol=2)
par(mfrow = c(1, 2))
for(i in 1:6)
{
  res_k_means = k_means(X, i, 15)
  ssc = SSC(res_k_means)
  K_results[i,1] = res_k_means$SSQ[length(res_k_means$SSQ)]
```

```
  K_results[i,2] = ssc$s_bar_overall
}
plot(1:6, K_results[,1], xlab="K Values",
                         ylab = "Within cluster sum of squares error", type='l')
plot(1:6, K_results[,2], xlab="K Values",
                         ylab = "Overall average Silhouette Scores", type='l')
```

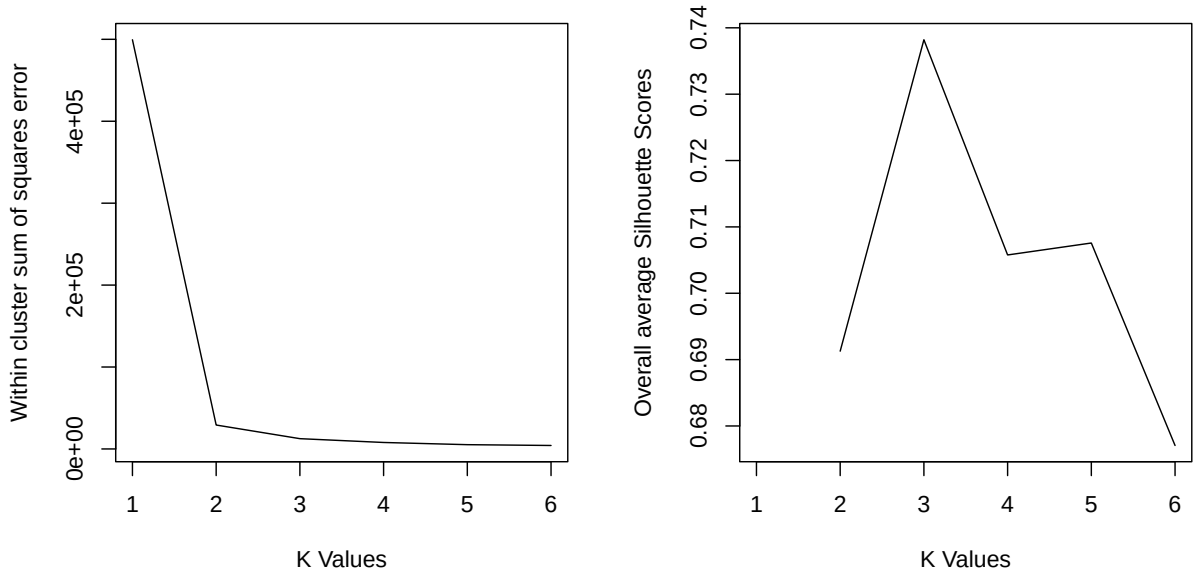


Figure 7: Sillhoette Scores and SSE Scores per K number of Clusters.

The analysis conducted calculates the Overall Simplified Silhouette Scores for an increasing number of clusters of the data. The Overall Simplified Silhouette Scores are metrics that measure how well data fits within the clusters. A high silhouette score reflects well grouped data, meaning the clusters are well defined and far apart. Coupled with this, the within-cluster sum of squares is calculated. This measures the difference between the cluster mean value and the actual observed value. It is used to measure how well the model fits the data. A small sum of squares reflects a well-fit model as the predicted values are close to the actual observations.

The overall average silhouette scores for three clusters is the highest with $K = 3$ (three clusters). The within-cluster sum of squares decreases as k increases, meaning it is lowest at $K = 6$ and would continue to decrease up until $K = N$. Looking at the plots above, the analysis recommends splitting the data into three groups. Having more groups would drastically decreases the silhouette score, but only marginally improves the sum of squares error, thus making three groups the optimal number.

This means that the retailer should separate the wines into three groups based on these allocations as these wines are considered similar across alcohol content, colour intensity and hue and therefore can be grouped as suggested.

Question 6(d)

With $K = 3$, the overall average simplified silhouette score is the closest to 1 (0.738). The overall average simplified silhouette score measures how distinctly the clusters are grouped on average. The closer the score

is to 1, the more distinct the clusters are. It indicates clear groupings leading to a more decisive nature in the model which is desirable. The simplified silhouette score is more influential than the internal-cluster sum of squares, therefore the compromise of $K = 3$ was made to ensure the highest possible silhouette score as a mediocre sum of squares score. This means that there will be some error within the cluster fittings, meaning the clusters will be relatively tightly packed, however still have some spread. This means that clusters are well separated with individual cluster silhouette score averages at 0.747, 0.725 and 0.743 for each of the three clusters all indicating good separation.

Appendix

```
# Libraries required for assignment
require(knitr)
require(rpart)
require(rpart.plot)
require(h2o)
require(colorspace)
require(cluster)
rm(list=ls())
#this chunck sets a 1:1 aspect ratio
knitr::opts_chunk$set(fig.width=5, fig.height=5, fig.align = 'center')
data <- read.table("TweetsDataB_2024.txt", header = TRUE)
data <- data.frame(tweets = data$Tweets, engagement = data$Follower_Engagement
                   , connectedness = data$Connectedness
                   , sentiment = as.factor(data$Sentiment))
# Question 2A
par(mfrow = c(1,2))
plot(tweets ~ connectedness, data = data, pch = 16, cex = 0.8, col = "red"
              , xlab = "User Connectedness", ylab = "Number of Tweets")
model <- lm(tweets ~ connectedness, data = data)
abline(model, col = "blue", lwd = 2, lty = 'dashed')
legend("topleft", legend = "Line of Best Fit", lty = 2,
                  lwd = 2, cex = 0.6, col = 'blue',  bg="white") #cex makes it smaller
plot(tweets ~ engagement, data = data, pch = 16, cex = 0.8, col = "darkgreen"
              , xlab = "User Engagement", ylab = "Number of Tweets")
model <- lm(tweets ~ engagement, data = data)
abline(model, col = "blue", lwd = 2, lty = 'dashed')
legend("topleft", legend = "Line of Best Fit", lty = 2,
                  lwd = 2, cex = 0.6, col = 'blue',  bg="white")

# Users who haven't posted automatically get classed as neutral
cleaned_sentiment_data <- data.frame(tweets = data[data$tweets > 0,]$tweets,
                           sentiment = data[data$tweets > 0,]$sentiment)

par(mfrow = c(1,2))
boxplot(data$tweets ~ data$sentiment, pch = 16, cex = 0.8, col = "red",
                      xlab = "User Sentiment", ylab = "Number of Tweets",
                      ylim = c(0,max(data$tweets)))
boxplot(cleaned_sentiment_data$tweets ~ cleaned_sentiment_data$sentiment,
                                pch = 16, cex = 0.8, col = "darkgreen",
                                xlab = "User Sentiment",
                                ylab = "Number of Tweets",
                                ylim = c(0,max(cleaned_sentiment_data$tweets)))
# Question 3A
set.seed(1)
model = rpart(tweets~sentiment + connectedness + engagement,
              data = data, control = rpart.control(minsplit = 4,cp = 0.04))

# Question 3B
par(mfrow = c(1,2))
plotcp(model)
abline(v = 2, col = 'red', lty = 2, lwd = 2)
```

```r
# Pruning Tree with optimal cp value
model_prune <- prune.rpart(model, cp = 0.16)
rpart.plot(model_prune)

# Question 3D
# Initialising h2o for Neural Network functionality
h2o.init()
h2o.no_progress()
set.seed(2024)
# Setting up the validation data split
N <- dim(data)[1]
trainingSet <- sample(1:N, floor(0.8*N), replace = FALSE)
trainingData <- as.h2o(data[trainingSet,])
ValidationData <- as.h2o(data[-trainingSet,])

# Get sequence of examples of L2 Parameters
L2Parameters <- exp(seq(-10,-3, length = 20))
validation_errors <- rep(NA, 20)

# Running the neural network for different L2 Parameters
create_NN <- function(L2_Parameter)
{
  neural_network_model <- h2o.deeplearning(
            x = 2:4, y = 1,
            training_frame = trainingData,
            validation_frame = ValidationData,
            reproducible = TRUE, standardize = TRUE, hidden = c(5,5),
            activation = 'tanh', loss = 'Quadratic',
            rate = 0.01, adaptive_rate = FALSE,
            epochs = 1000, seed = 2,
            l2 = L2_Parameter)
  return (neural_network_model)
}

# Storing the different validation errors
for (i in 1:20)
{
    current_model <- create_NN(L2Parameters[i])
    validation_errors[i] <- h2o.mse(current_model, valid = TRUE)[1]
}
# Finding the optimal L2 Parameter
minL2Parameter <- L2Parameters[which.min(validation_errors)]

# Plotting the validation errors against the different parameters
plot(validation_errors ~ L2Parameters, ylab = 'Validation Errors'
                          , xlab = 'L2 Regularisation Parameter')
abline(v = minL2Parameter, col = 'red', lwd = 2, lty = 2)
text(minL2Parameter, 0.85, pos = 4, col = 'black', cex = 0.8, font = 2,
    labels = paste0('parameter = ', round(minL2Parameter,3)), offset = 0.5)

# Question 4B
# Renaming Models As See Fit
selected_NN_Model   <- create_NN(minL2Parameter)
```

```
selected_Tree_Model <- model_prune

#plot(selected_NN_Model)
# Function to allow for colour grading on the contour plot
colour.gradient = function(x,
                  colors=c('white', 'yellow', 'orange','red','darkred'),
                  colsteps=100)
{
  colpal = colorRampPalette(colors)
  return(colpal(colsteps)[findInterval(x, seq(-0.5,12, length=colsteps))])
}
# Contour plot
M = 100
engagement_seq = seq(min(data$engagement), max(data$engagement), length = M)
connect_seq = seq(min(data$connectedness), max(data$connectedness), length = M)

# Setting up the points to test across
engagement = rep(engagement_seq, M)
connectedness = rep(connect_seq, each = M)
Lat = data.frame(engagement=engagement, connectedness=connectedness
                                                , sentiment='Neutral')

# Getting predictions from the neural network and the tree
responsePredictionsNN = as.matrix(h2o.predict(selected_NN_Model, as.h2o(Lat)))
responsePredictionsTree = as.matrix(predict(selected_Tree_Model, Lat))

# Matrix to aid in matching Predictions to Connectedness and Engagement values
pred_matrix = matrix(as.vector(responsePredictionsNN),
                     nrow = length(connect_seq), ncol = length(engagement_seq))

# Plots the points with gradiented colour - For NN Model
plot(Lat[-3], pch = 16, col = colour.gradient(pred_matrix), xlab = "Engagement",
                         ylab = "Connectedness")

# Plots the contour plot of the predictions over the colour
contour(engagement_seq, connect_seq, pred_matrix, add = TRUE)

# Matrix to aid in matching Predictions to Connectedness and Engagement values
pred_matrix = matrix(as.vector(responsePredictionsTree),
                     nrow = length(connect_seq), ncol = length(engagement_seq))

# For Tree Model
plot(Lat[-3], pch = 16, col = colour.gradient(pred_matrix), xlab = "Engagement",
                         ylab = "Connectedness")
text(2.8, 0, labels = paste0(4.833), pos = 4, col = 'black', cex = 0.8,
     font = 2)
text(-1.5, -1.2, labels = paste0(0.888), pos = 4, col = 'black', cex = 0.8,
     font = 2)
text(-1.5, 3.8, labels = paste0(5.933), pos = 4, col = 'black', cex = 0.8,
     font = 2)

# Question 5A
set.seed(2)
```

```r
trainingSet  <- sample(1:N, floor(0.8*N), replace = FALSE)
trainingData <- data[trainingSet,]
validData    <- data[-trainingSet,]

# Question 5B
cpValues <- exp(seq(-8,-4, length = 100))

# Function to calculate the log likelihood
negative_log_likelihood <- function(cp, data_train, data_valid)
{
  sum = 0
  model <- rpart(tweets~., data = data_train,
                 control = rpart.control(minsplit = 4,cp = cp))

  # Calculates the log likelihood
  for (i in 1:dim(data_valid)[1])
  {
    pred_y   <- as.numeric(predict(model, data_valid[i,]))
    actual_y <- data_valid$tweets[i]
    sum = sum - (actual_y*log(pred_y) - pred_y - log(factorial(actual_y)))
  }
  return (sum)
}

# Creates space
LLValues <- rep(NA, length(cpValues))

# Getting the LLValues for the set of cpValues
for (i in 1:length(cpValues))
{
  LLValues[i] = negative_log_likelihood(cpValues[i], trainingData, validData)
}

# Validation Analysis Plot with Log-Likelihood Value
plot(LLValues~cpValues, type = 'b')
abline(v = max(cpValues[which(LLValues == min(LLValues, na.rm = TRUE))]),
       col = 'red', lwd = 2, lty = 2)
bestCPValue <- max(cpValues[which(LLValues == min(LLValues, na.rm = TRUE))])
text(bestCPValue, 50, labels = paste0(round(bestCPValue,3)),
     pos = 4, col = 'black', cex = 0.8, font = 2)

# Question 6
# Reading in Wine Data
rm(list=ls())
data = read.table('wine2023.txt', header=TRUE)
X = cbind(data$Alcohol,data$Color_Intensity, data$Hue)

# Question 6A
k_means <- function(X, K = 1, iter = 15)
{
  # Initialisations:
  set.seed(2024)
  N <- dim(X)[1]
```

```
  d <- dim(X)[2]
  startingIndicies <- sample(1:N, K, replace = FALSE)
  Mus <- X[startingIndicies,]
  ones  = matrix(1, N, 1)
  dists = matrix(NA, N, K)
  error = rep(NA, iter)

  if (K == 1)
  {
    Mus <- colSums(as.matrix(X))/N
    dists[,1] <- rowSums((X-ones%*%Mus)^2)
    error <- N*sum((dists[,1])^2)
    return(list(D = dists, SSQ = error, z = ones))
  }

  for (i in 1:iter)
  {
    # Assigning points to clusters
    dists <- dists*0
    for (k in 1:K)
    {
      dists[,k] <- rowSums((X-ones%*%Mus[k,])^2)
    }
    cluster_indicies <- apply(dists, 1, which.min)

    # Calculating error
    err = 0
    cluster_dists <- apply(dists, 1, min)
    for (k in 1:K)
    {
      obs_from_cluster <- which(cluster_indicies == k)
      nk <- length(obs_from_cluster)
      err <- err + nk*sum(cluster_dists[obs_from_cluster])

      # Setting new cluster means
      if (any(cluster_indicies == k))
      {
        Mus[k,] <- colSums(as.matrix(X[obs_from_cluster,]))/nk
        if (nk == 1) {Mus[k,] = X[obs_from_cluster, ]} # For single Obs
      }
      else # If empty cluster
      {
        furtherest_point <- which.max(apply(dists, 1, min))
        Mus[k,] <- X[furtherest_point,]
      }
    }
    error[i] <- err
  }
  return(list(D = dists, SSQ = error, z = cluster_indicies))
}

# Question 6B
SSC = function(res_k_means)
```

```
{
  if (dim(res_k_means$D)[2] == 1)
  {
    return(list(s=NA, s_bar_k = NA, s_bar_overall = NA))
  }

  s <- rep(NA,dim(res_k_means$D)[1])
  for (i in 1:length(s))
  {
    # Getting nearest and second nearest cluster distances
    r_2 <- min(res_k_means$D[i,][-res_k_means$z[i]])
    r_1 <- res_k_means$D[i,][res_k_means$z[i]]

    # Calculating the simplified Silhouette Scores
    s[i] <- (r_2-r_1)/r_2
  }

  # Getting the average simplified Silhouette Scores per cluster
  s_bar_k <- rep(NA,dim(res_k_means$D)[2])
  for (k in 1:length(s_bar_k))
  {
    s_bar_k[k] <- mean(s[which(res_k_means$z == k)])
  }
  # Getting the overall average Silhouette Scores
  s_bar_overall <- mean(s_bar_k)

  return(list(s=s, s_bar_k = s_bar_k, s_bar_overall = s_bar_overall))
}
# Question 6C
K_results = matrix(NA, nrow=6, ncol=2)
par(mfrow = c(1, 2))
for(i in 1:6)
{
  res_k_means = k_means(X, i, 15)
  ssc = SSC(res_k_means)
  K_results[i,1] = res_k_means$SSQ[length(res_k_means$SSQ)]
  K_results[i,2] = ssc$s_bar_overall
}
plot(1:6, K_results[,1], xlab="K Values",
                         ylab = "Within cluster sum of squares error", type='l')
plot(1:6, K_results[,2], xlab="K Values",
                         ylab = "Overall average Silhouette Scores", type='l')
```