```
## R Script

# Set working directory
setwd("~/Documents/YEAR 3 MODULES/EC349 Data Science/
EC349AssignementRowan")

# Load libraries
library(jsonlite)
library(caret)
library(dplyr)
library(glmnet)
library(ggplot2)
library(tidyverse)
library(textdata)
library(tidytext)
library(corrplot)
# For random forest
library(ranger)
library(vip)
# For corpus text analysis
library(tm)
library(SnowballC)
library(wordcloud)

# Import datasets - I do not use the tip dataset after determining
it provides no useful information above the other datasets.
load("yelp_review_small.Rda")
business_data <-
stream_in(file("yelp_academic_dataset_business.json"))
checkin_data  <-
stream_in(file("yelp_academic_dataset_checkin.json"))
user_data <- stream_in(file("yelp_academic_dataset_user.json"))

# Add column to check in dataset which shows number of checkin dates
per observation - label this DateCount, also rename date column in
checkin dataset to CheckInDates
checkin_data$DateCount <-
sapply(strsplit(as.character(checkin_data$date), ","), length)
colnames(checkin_data)[colnames(checkin_data) == "date"] <-
"CheckInDates"

# Merge datasets
merge_data_1 <- merge(user_data, review_data_small, by = "user_id")
merge_data_2 <- merge(merge_data_1, business_data, by =
"business_id")
merge_data_3 <- merge(merge_data_2, checkin_data, by =
"business_id")

## Clean data

# Check there are not any duplicated reviews
anyDuplicated(merge_data_3$review_id)
```

```
# Remove unnecessary columns and columns with lots of missing data
or which would lead to overfitting
clean_data <- subset(merge_data_3, select = -c(`address`,
`postal_code`, `latitude`, `longitude`, `attributes`, `hours`))

# Remove any observations with missing data
clean_data<-na.omit(clean_data)

# Rename the outcome variable to review_star
clean_data <- clean_data %>%
  rename(review_star = stars.x)

# Rename other important variables
clean_data <- clean_data %>%
  rename(average_stars_user = average_stars)
clean_data <- clean_data %>%
  rename(average_stars_business = stars.y)
clean_data <- clean_data %>%
  rename(review_count_user = review_count.x)
clean_data <- clean_data %>%
  rename(review_count_business = review_count.y)

# Correct the datatypes
clean_data$is_open<-as.factor(clean_data$is_open)
clean_data$review_star<-as.factor(clean_data$review_star)
clean_data$name.y<-as.factor(clean_data$name.y)
clean_data$state<-as.factor(clean_data$state)

# Set seed for reproducibility
set.seed(1)
# Create a smaller sample of 200,000 for computational time
yelp_sample <- clean_data[sample(nrow(clean_data), 200000, replace =
FALSE), ]


# Add a variable showing the number of words in each review
yelp_sample$word_count <- str_count(yelp_sample$text, "\\w+")

# Add a variable showing the user's number of friends
yelp_sample$friend_count <- sapply(strsplit(yelp_sample$friends, ",
"), length)

# The following code is inspired by Silge and Robinson (2017, ch.2)
- see bibliography at the end of markdown report
# Note, some observations that do not have corresponding sentiments
in the AFINN or NRC lexicon are dropped (~4,000)
# Add a variable showing the sentiment score for each review using
the 'AFINN' lexicon.
yelp_sample %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(review_id) %>%
  summarize(sentiment_score = sum(value)) -> sentiment_scores
# Merge the sentiment score variable with yelp_sample
```

```
yelp_sample <- merge(sentiment_scores, yelp_sample, by="review_id")
# Add a variable showing the emotional sentiment score for each
review using the 'NRC' lexicon
yelp_sample %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments("nrc"), relationship = "many-to-many")
%>%
  group_by(review_id) %>%
  summarize(
    anticipation = sum(sentiment == "anticipation"),
    joy = sum(sentiment == "joy"),
    positive = sum(sentiment == "positive"),
    trust = sum(sentiment == "trust"),
    surprise = sum(sentiment == "surprise"),
    negative = sum(sentiment == "negative"),
    anger = sum(sentiment == "anger"),
    fear = sum(sentiment == "fear"),
    # called disgust_sentiment since later on, a variable from the
corpus text analysis is called disgust
    disgust_sentiment = sum(sentiment == "disgust"),
    sadness = sum(sentiment == "sadness"),
  ) -> sentiment_scores_nrc_2
# Merge the NRC lexicon variables with yelp_sample
yelp_sample <- merge(sentiment_scores_nrc_2, yelp_sample,
by="review_id")

# Create new variables for year and month of yelping since
yelp_sample$yelping_since <-
ym(format(ymd_hms(yelp_sample$yelping_since), "%Y-%m"))
yelp_sample$yelping_since_year <-
as.numeric(format(yelp_sample$yelping_since,"%Y"))
yelp_sample$yelping_since_month <-
as.numeric(format(yelp_sample$yelping_since,"%m"))

# Create new variables for year and month of review
yelp_sample$date <- as.Date(yelp_sample$date, format = "%Y-%m-%d")
yelp_sample$date_year <- as.numeric(format(yelp_sample$date, "%Y"))
yelp_sample$date_month <- as.numeric(format(yelp_sample$date, "%m"))




# Create training and test data - 10% of observations (~19,620) will
be the test data
set.seed(1)
test_index <- createDataPartition(y = yelp_sample$review_star, p =
0.1, list = FALSE, times = 1)
train_data <- yelp_sample[-test_index, ]
test_data <- yelp_sample[test_index, ]

# Plot the count of review_star by level (training data)
ggplot(train_data, aes(x = review_star)) +
  geom_bar(stat = "count", fill = "blue") +
```

```
  geom_text(stat = 'count', aes(label = after_stat(count)), color =
"white", vjust=1.5) +
  ggtitle("Review_star by level") +
  xlab("review_star") +
  ylab("Count") +
  theme_light()

# Show the review_star class distribution in test data
prop.table(table(test_data$review_star))


## Monthly averages of **review_star** by year
# For line graph, make review_star numeric
train_data$review_star <- as.numeric(train_data$review_star)
# Plot monthly averages of review_star by year
ggplot(train_data, aes(x = date_month, y = review_star, group =
date_year, color =  as.factor(date_year))) +
  stat_summary(fun = "mean", geom = "line", size = 1) +
  labs(x = "Date", y = "Average review_star", title = "Average
review_star by Month") +
  # Use facet_wrap to plot all graphs in a grid
  facet_wrap(~ date_year, scales = "free_y", ncol = 4) +
  # For a clearer graph
  theme_light()

# Line graph of average review_star by sentiment score
ggplot(train_data, aes(x = sentiment_score, y = review_star)) +
  stat_summary(fun = "mean", geom = "line", linewidth = 1) +
  labs(x = "Sentiment score", y = "Average review_star", title =
"Average review_star by Sentiment Score (AFINN)") +
  theme_light()

# Scatter plot of review_star by sentiment score
ggplot(train_data, aes(x = sentiment_score, y = review_star)) +
  geom_point(alpha = 0.5) +
  labs(x = "Sentiment Score", y = "review_star", title = "Scatter
Plot of review_star by Sentiment Score (AFINN)") +
  theme_light()

# Convert review_star back to a factor variable for box plot
train_data$review_star <- as.factor(train_data$review_star)
# Box plot of sentiment score by review_star
f <- ggplot(train_data, aes(review_star, sentiment_score)) +
ggtitle("Sentiment score (AFINN) by review_star") + theme_light()
f + geom_boxplot()

# Convert review_star to numeric
train_data$review_star<-as.numeric(train_data$review_star)
key_numeric_predictors <- c("sentiment_score", "negative",
"positive", "joy", "anger", "trust", "sadness", "disgust_sentiment",
"anticipation", "fear", "surprise", "average_stars_user",
"word_count", "DateCount", "friend_count")
# Print residuals of each variable in key_numeric_predictors
for (predictor in key_numeric_predictors) {
```

```
  # Linear regression
  model <- lm(review_star ~ train_data[[predictor]], data =
train_data)
  r_squared <- summary(model)$r.squared
  cat("R² for", predictor, ":", r_squared, "\n")
}

# Convert review_star to factor for box plot
train_data$review_star<-as.factor(train_data$review_star)
# Box plot of average user rating by review_star
e <- ggplot(train_data, aes(review_star, average_stars_user)) +
ggtitle("Average user rating by review_star") + theme_light()
e + geom_boxplot()

# Convert review_star to numeric for correlation matrix
train_data$review_star<-as.numeric(train_data$review_star)
# Extract only the numeric and integer columns in the training data
numeric_integer_columns <- yelp_sample[, sapply(yelp_sample,
function(x) is.numeric(x) || is.integer(x))]
# Calculate correlations
correlation_matrix <- cor(numeric_integer_columns)
# Plot correlation matrix
corrplot(correlation_matrix, method = "color",tl.cex = 0.5)


## Impurity-based variable importance graph (Boehmke and Greenwell,
2020, ch.11)
train_data$review_star<-as.factor(train_data$review_star)
# Impurity-based variable importance graph
rf_impurity <- ranger(
  formula = review_star ~. -user_id -business_id -friends -text
-review_id - name.x -name.y -elite -yelping_since -categories
-CheckInDates -city -date,
  data = train_data,
  # default
  num.trees = 500,
  # default
  mtry = 6,
  min.node.size = 1,
  sample.fraction = .80,
  replace = FALSE,
  importance = "impurity",
  respect.unordered.factors = "order",
  # omit unwanted updates in output (about how much time is left)
  verbose = FALSE,
  seed  = 1
)

# Plot impurity graph
p1 <- vip::vip(rf_impurity, num_features = 50, bar = FALSE)
p1


## Untuned ranger
```

```
# Set seed for reproducibility
set.seed(1)
# Use the ranger package to run a random forest of review_star on
all the predictors, except ones which would cause over fitting
yelp_rf <- ranger(
  review_star ~ .-user_id -business_id -friends -text -review_id -
name.x -name.y -elite -yelping_since -categories -CheckInDates -city
-date,
  data = train_data,
  respect.unordered.factors = "order",
  verbose = FALSE
)

# Run the model on the test data
test_predictions <- predict(yelp_rf, test_data)

# Store predictions as factor
predicted_classes <- as.factor(test_predictions$predictions)

# Confusion matrix to test accuracy of model on test data.
confusionMatrix(predicted_classes, test_data$review_star)
$overall["Accuracy"]


# Plot the confusion matrix after converting it to a data frame
conf_matrix <-confusionMatrix(predicted_classes,
test_data$review_star)
conf_matrix_df <- as.data.frame(as.table(conf_matrix$table))
ggplot(conf_matrix_df, aes(x = Reference, y = Prediction, fill =
Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = sprintf("%d", Freq))) +
  scale_fill_gradient(low = "white", high = "blue") +
  labs(title = "Confusion Matrix for mtry=6, num.trees = 500",
       x = "Actual",
       y = "Predicted")

within_range <- 1

# Calculate accuracy within the specified range
accuracy_within_range <-
sum(ifelse(abs(as.numeric(predicted_classes) -
as.numeric(test_data$review_star)) <= within_range, 1, 0)) /
length(predicted_classes)

# Print the accuracy
cat(sprintf("How often the prediction is within 1 of the actual
rating: %s\n", accuracy_within_range))


## Corpus text analysis

# Convert text data to corpus
corpus <- Corpus(VectorSource(yelp_sample$text))
```

```
# Convert all words to lowercase
corpus <- tm_map(corpus, PlainTextDocument)
corpus <- tm_map(corpus, tolower)

# Remove punctuation
corpus <- tm_map(corpus, removePunctuation)

# Remove stopwords (is, me, our etc.)
corpus <- tm_map(corpus, removeWords, stopwords("english"))

# Stem the corpus to reduce the number of inflectional forms of
words appearing in the text. For example, "recommend", "recommends",
"recommended" etc. are reduced to their common stem "recommend"
corpus <- tm_map(corpus, stemDocument)

# Create a Document Term Matrix which shows the frequencies of words
dtm <- DocumentTermMatrix(corpus)

# Remove words which are sparse in the dtm. In this case, terms
(words) that occur in less than 0.25% of documents (reviews) will be
removed.
dtm_sparsed <- removeSparseTerms(dtm, 0.9975)

# Convert the matrix to a dataframe for modeling
df_corpus <- as.data.frame(as.matrix(dtm_sparsed))

# Make column names valid
colnames(df_corpus) <- make.names(colnames(df_corpus))

# Add the review_id column
df_corpus$review_id<- yelp_sample$review_id

# Create a new training dataframe called df_corpus_train which
combines the review_star column in the previous training data with
the word variables in df_corpus by review_id.
df_corpus_train<-merge(df_corpus,train_data[,c("review_id",
"review_star", by="review_id")]) %>%
  select(-review_id, -review_id.1)

## There are too many variables in the training dataset - select the
200 word variables which are most correlated with review_star
# Convert the outcome variable review_star to numeric for
correlations
df_corpus_train$review_star<-as.numeric(df_corpus_train$review_star)

# Calculate the correlations between each word variable and
review_star
correlations <- sapply(names(df_corpus_train), function(word)
cor(df_corpus_train[[word]], df_corpus_train$review_star))

# Create a dataframe called correlation_df to store the correlations
correlation_df <- data.frame(word = names(correlations), correlation
= correlations)
```

```
# Order the dataframe by absolute correlation values in descending
order
correlation_df <- correlation_df[order(-
abs(correlation_df$correlation)), ]

# Extract the top 200 words based on the ordered correlation
dataframe (also extracts review_star)
top_200_words <- head(correlation_df, 201)

# Remove review_star from the data frame
top_200_words_clean <- top_200_words[-1,]

# Select only these top 200 words from the training data
df_train_top_200 <- df_corpus_train[, top_200_words_clean$word, drop
= FALSE]

# Create a new training dataframe called df_corpus_train which adds
the 100 words which are most highly correlated with review_star (in
absolute terms) to the original training data.
train_data_corpus <- bind_cols(train_data, df_train_top_200)

# Add columns for these same 200 words to the test dataframe
df_corpus_test<-merge(df_corpus,test_data[,c("review_id",
"review_star", by="review_id")]) %>%
  select(-review_id, -review_id.1)
df_test_top_200 <- df_corpus_test[, top_200_words_clean$word, drop =
FALSE]
test_data_corpus <- bind_cols(test_data, df_test_top_200)

# Convert the outcome variable review_star in the training and
testing data back to a factor variable
train_data_corpus$review_star<-
as.factor(train_data_corpus$review_star)
test_data_corpus$review_star<-
as.factor(test_data_corpus$review_star)

# the top 10 most correlated words have correlations of:
# Calculate correlations of the 200 words
correlation_matrix_words <- cor(df_train_top_200)
# Convert to a vector
values <- as.vector(correlation_matrix_words)
# Extract top 10 correlation values (note 1:200 are correlations of
a variable on itself so is 1)
top_10_values <- order(values, decreasing = TRUE)[201:210]
# Print results as a dataframe
data.frame(Value = values[top_10_values])

# Untuned ranger after corpus (same process as earlier)
set.seed(1)
yelp_rf2 <- ranger(
  review_star ~ .-user_id -business_id -friends -text -review_id -
name.x -name.y -elite -yelping_since -categories -CheckInDates
-city-date,
```

```
  data = train_data_corpus,
  respect.unordered.factors = "order",
  verbose = FALSE
)

test_predictions2 <- predict(yelp_rf2, test_data_corpus)

predicted_classes2 <- as.factor(test_predictions2$predictions)

confusionMatrix(predicted_classes2, test_data_corpus$review_star)
$overall["Accuracy"]


within_range <- 1

# Store accuracy within 1
accuracy_within_range2 <-
sum(ifelse(abs(as.numeric(predicted_classes2) -
as.numeric(test_data_corpus$review_star)) <= within_range, 1, 0)) /
length(predicted_classes2)

# Print the accuracy within 1
cat(sprintf("How often the prediction is within 1 of the actual
rating: %s\n", accuracy_within_range2))


## tuning ranger by mtry values
# Create vector with mtry values to model on
mtry_values <- c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
# Create an emptry list to store results for each mtry
results_list <- list()
# For loop - run ranger random forest for each value of mtry
specified above and store accuracy on test data in the list
for (mtry_value in mtry_values) {
  # Train a ranger model with the current mtry value
  set.seed(1)
  yelp_rf4 <- ranger(review_star~.-user_id -business_id -friends
-text -review_id - name.x -name.y -elite -yelping_since -categories
-CheckInDates -city -date, data = train_data_corpus,
respect.unordered.factors = "order", mtry = mtry_value, num.trees =
500, min.node.size = 1, verbose = FALSE)
  test_predictions4 <- predict(yelp_rf4, test_data_corpus)
  predicted_classes4 <- as.factor(test_predictions4$predictions)
  # Evaluate the model (replace this with your actual evaluation
metric)
  accuracy_tree <- confusionMatrix(predicted_classes4,
test_data_corpus$review_star)$overall["Accuracy"]

  # Store the results
  results_list[[as.character(mtry_value)]] <- accuracy_tree
}

# convert list to a data frame
mtry <- data.frame(mtry = as.numeric(names(results_list)), accuracy
```

```
= unlist(results_list))
# print accuracy by mtry
print(mtry)
# plot accuracy by mtry
plot(mtry)




# The final tuned model (same process as earlier)
set.seed(1)
yelp_rf3 <- ranger(
  review_star ~ .-user_id -business_id -friends -text -review_id -
name.x -name.y -elite -yelping_since -categories -CheckInDates
-city-date,
  data = train_data_corpus,
  respect.unordered.factors = "order",
  mtry=60,
  min.node.size = 1,
  verbose = FALSE
)

test_predictions3 <- predict(yelp_rf3, test_data_corpus)

predicted_classes3 <- as.factor(test_predictions3$predictions)

confusionMatrix(predicted_classes3, test_data_corpus$review_star)
$overall["Accuracy"]

within_range <- 1

# Store accuracy within 1
accuracy_within_range3 <-
sum(ifelse(abs(as.numeric(predicted_classes3) -
as.numeric(test_data_corpus$review_star)) <= within_range, 1, 0)) /
length(predicted_classes3)

# Print accuracy within 1
cat(sprintf("How often the prediction is within 1 of the actual
rating: %s\n", accuracy_within_range3))

# Plot confusion matrix
conf_matrix <-confusionMatrix(predicted_classes3,
test_data_corpus$review_star)
conf_matrix_df <- as.data.frame(as.table(conf_matrix$table))
ggplot(conf_matrix_df, aes(x = Reference, y = Prediction, fill =
Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = sprintf("%d", Freq)), vjust = 1) +
  scale_fill_gradient(low = "white", high = "blue") +
  labs(title = "Confusion Matrix for final model mtry=60, num.trees
= 500",
       x = "Actual",
       y = "Predicted") +
```

```
  theme_minimal()


## Plot wordcloud for review_star = 1 and review_star = 5
# Add review_star variable to training data with just the corpus
word variables
df_train_top_200$review_star <- df_corpus_train$review_star
# Create a plotting grid
par(mfrow = c(1, 2))
# Subset data where review_star == 1
subset_data <- subset(df_train_top_200, review_star == 1)
# Count frequency of words
word_freq <- colSums(subset_data[, -ncol(subset_data)])
# Plot wordcloud
wordcloud(words = names(word_freq), freq = word_freq, min.freq = 10,
scale = c(2, 0.5), color = "red",
          main = "Word Cloud for review_star = 1")
title("Word Cloud for review_star = 1")

# Subset data where review_star == 5
subset_data2 <- subset(df_train_top_200, review_star == 5)
# Count frequency of words
word_freq2 <- colSums(subset_data2[, -ncol(subset_data2)])
# Plot wordcloud
wordcloud(words = names(word_freq2), freq = word_freq2, min.freq =
10, scale = c(2, 0.5), color = "green",
          main = "Word Cloud for review_star = 5")
title("Word Cloud for review_star = 5")
```