~~~
---
title: "Individual Project EC349"
author: "Rowan Vinayak u2100546"
date: "2023-12-2"
output: pdf_document
header-includes:
  - |
    ```{=latex}
    \usepackage{fvextra}
    \DefineVerbatimEnvironment{Highlighting}{Verbatim}{
      showspaces = false,
      showtabs = false,
      breaklines,
      commandchars=\\\{\}
    }
    ```
---

word count (excluding code chunks): 1341

```{r include=FALSE}
knitr::opts_chunk$set(comment = NA)
```

## Introduction

In this project, I predict and analyse yelp review ratings through
business and user characteristics, as well as sentiment and text
analysis.

I adopt a flexible approach to the **CRISP-DM** methodology. It
provides a structured, comprehensive and cyclical framework,
supporting the iterative nature of data science (DSPA, 2021). My
flexible approach ensured that I constantly iterated through
processes, allowing me to build an initial model that I could then
improve from. I value the CRISP-DM methodology's emphasis on the
initial *understanding* phase which helped me focus on the specific
problem. Its five phases (deployment is not relevant to this
assignment) are intuitive and encouraged me to critically think and
ask questions throughout about my analysis.




### Classification or regression problem?

Since user reviews are discrete, I employed a classification
approach by transforming the outcome variable **review_star** into a
factor variable with 5 levels.

I utilise confusion matrices to evaluate the random forest model’s
performance on the test dataset.

This report contains 4 sections:
~~~

~~~
1.  Data preparation
2.  Exploratory data analysis (EDA) and model selection
3.  Model implementation, tuning and accuracy *(and most difficult
challenge)*
4.  Model evaluation




## Data preparation

```{r, include=FALSE}
#Set working directory
setwd("~/Documents/YEAR 3 MODULES/EC349 Data Science/
EC349AssignementRowan")
```

I merged all datasets except the tip dataset (it contained no useful
information above the others).


```{r, message=FALSE, include=FALSE}
# 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")

```

### Data Cleaning



To enhance data quality and completeness, I removed the:

* 39 attributes, each with at least 95,000 missing observations.
* hours variables - each having at least 80,000 missing
observations.
* variables with high cardinality (address, postal code, latitude,
longitude) since these may cause overfitting. Geographic impacts are
already covered by the state variable.

I removed observations (54) with missing data.

```{r, include=FALSE}
# Check there are not duplicated reviews
anyDuplicated(merge_data_3$review_id)
# Remove unnecessary columns and columns with lots of missing data
or which would lead to overfitting
clean_data <- subset(merge_data_3, select = -c(`address`,
`postal_code`, `latitude`, `longitude`, `attributes`, `hours`))
# Remove any observations still with missing data
clean_data<-na.omit(clean_data)
# Rename the outcome variable to review_star
clean_data <- clean_data %>%
  rename(review_star = stars.x)
# Rename other importnat variables
clean_data <- clean_data %>%
  rename(average_stars_user = average_stars)
clean_data <- clean_data %>%
  rename(average_stars_business = stars.y)
clean_data <- clean_data %>%
  rename(review_count_user = review_count.x)
clean_data <- clean_data %>%
  rename(review_count_business = review_count.y)
# Correct the datatypes
clean_data$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)
~~~~

~~~
```

Finally, I produced a smaller sample of 200,000 (for computational
time).

```{r, include=FALSE}
# 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), ]
```


I explore the impact of 9 additional variables on **review_star**:

1. Length of review (words)
2. User's number of friends
3. Sentiment of review using AFINN lexicon<sup>1</sup>
(sentiment_score)
4. Emotional sentiment of review using NRC lexicon<sup>1</sup>
5. Month a user has been yelping since
6. Year a user has been yelping since
7. Month of review
8. Year of review
9. Business's number of checkin dates (DateCount)

<sup>1</sup>I follow the text mining process outlined by Silge and
Robinson (2017, ch.2).

The NRC lexicon provides data about the emotions in text, producing
the variables positive, negative and 8 emotion variables (anger,
sadness, fear etc.).


```{r, message=FALSE, warning=FALSE, include=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"))
```


```{r, include=FALSE}
# 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, ]
```
~~~

~~~
## Exploratory data analysis (EDA) and model selection



```{r, echo=FALSE}
# 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()
```

In the training dataset, **review_star** has uneven class
distribution with 5 being the most common rating. A naive model
predicting 5 for each review would have an accuracy of 0.4617 on the
test dataset:


```{r, echo=FALSE}
# Show the review_star class distribution in test data
prop.table(table(test_data$review_star))
```




### Relationship between review_star and key predictors

### Review Date

The graphs below illustrate monthly averages of **review_star** by
year. Monthly fluctuations suggest some seasonality in
**review_star**. For most years, the average of **review_star** dips
between September and December.


```{r, echo=FALSE, warning=FALSE, message=FALSE}
# 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()

```




### Sentiment score (AFINN)

The graphs below show a positive relationship between sentiment
score and **review_star** for most observations. However, some
observations with low sentiment scores have a high **review_star**
and vice versa which may challenge the model. The box plot
illustrates that sentiment_score does not distinguish well between 4
and 5 star reviews.

```{r, echo=FALSE, message=FALSE}
# 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()

```





Many of the R<sup>2</sup>s from a linear regression of
**review_star** on predictors are low, highlighting the highly non-
linear relationship between **review_star** and several predictors.
~~~~

~~~
```{r, message=FALSE, echo=FALSE, warning=FALSE}
# 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

  # Print R-squared
  cat("R² for", predictor, ":", r_squared, "\n")
}
```
~~~

The graph below shows the strong positive relationship between a user's average stars and **review_star**.

~~~
```{r, echo=FALSE}
# Convert review_star to factor for box plot
train_data$review_star<-as.factor(train_data$review_star)

# Box plot of a user's average 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()
```
~~~

The correlation matrix below shows many predictors are highly correlated:

~~~
```{r, echo=FALSE}
train_data$review_star<-as.numeric(train_data$review_star)
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
corrplot(correlation_matrix, method = "color",tl.cex = 0.5)
~~~

~~~~
```

Below is an impurity-based variable importance graph (Boehmke and
Greenwell, 2020, ch.11). The feature importance is determined by
calculating the average reduction in the loss function (in our case
the **Gini index**) due to a certain feature across all trees.
average_stars_user and sentment_score are the most important
predictors.

```{r, echo=FALSE, message=FALSE}
# Convert review_star to factor variable
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,
  num.trees = 500,
  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
)

p1 <- vip::vip(rf_impurity, num_features = 50, bar = FALSE)
p1

```

### Model selection

My EDA above supports the random forest model choice for 5 main
reasons:

1. The correlation matrix shows that many **predictors are
correlated** with each other. For example, useful.x and cool.x
have .99 correlation. Random forests reduce the impact of correlated
predictors by randomly selecting a subset of features at each split
(Hastie et al., 2009). Random forests therefore decrease the
correlation between trees without increasing variance too much.
~~~~

Hastie et al. (2009, p.588) show that with Bagging, the variance of the average prediction is $\rho \sigma^2 + (1 - \rho) B \sigma^2$. While bagging does not reduce $\rho \sigma^2$, random forests randomly select a subset of features at each split, reducing the correlation across predictions ($\hat{Y}$) and so reducing $\rho \sigma^2$. Hence, with correlated predictors random forest is effective.

2. The impurity graph shows that **multiple variables are important** in predicting review_star. Random forest is most effective when there are multiple good predictors.

3. Random forest is well suited to **classification** problems with uneven distribution of classes.

4. Random forest is effective at dealing with numerical and **categorical datatypes**.

5. Random forest handles **non-linear relationships** well.

Additionally, later I create 200 word variables that are moderately correlated to each other, reinforcing 1.

## Model implementation, tuning and accuracy *(and most difficult challenge)*

The most difficult challenge in carrying out this project was the computational burden of random forest. The **randomForest** package caused R to crash (due to the number of observations and predictors). To overcome this, I took inspiration from Wright's and Ziegler's (2017) use of the **ranger** package. This package is a fast implementation of random forests for high dimensional data.

An untuned ranger with 500 trees and mtry = 6 has an accuracy of 0.6208461 on the test dataset:

```{r, message=FALSE}
# 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"]
```



```{r, echo=FALSE}
# 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")
```

```{r, echo=FALSE}
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

A corpus is a structured set of documents for pre-processing text
data (Welbers et al. 2017). The code is extensive, so I give a
summary of my method:

I converted the text data to a corpus, cleaned the corpus (e.g.
converted words to lowercase), stemmed the corpus and converted the
corpus to a Document Term Matrix showing the frequencies of each
~~~

word. After removing words that appear in less than 0.25% of all documents, I converted the matrix to a data frame for analysis and combined this with the training dataset. However, there were too many word variables (~ 2,000) which was a burden on computational power. Therefore, I kept only the top 200 words most correlated with **review_star** in the training data. I also added these word variables to the test data.

~~~
```{r, message=FALSE, warning=FALSE, echo=FALSE}
# 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)
~~~

```{r, message=FALSE, warning=FALSE, echo=FALSE}
# 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 0.5% 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. Let's select
the 100 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 dataframe
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 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:

```{r, message=FALSE, echo=FALSE}
correlation_matrix_words <- cor(df_train_top_200)
values <- as.vector(correlation_matrix_words)
indices <- order(values, decreasing = TRUE)[201:210]
data.frame(Value = values[indices])
```

Therefore, random forest is an appropriate model (predictors are
correlated - see section 2)
~~~~

An untuned ranger (num.trees = 500, mtry = 15) after corpus text
analysis has an accuracy of:

```{r, message=FALSE, echo=FALSE}
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"]
```

```{r, echo=FALSE}
within_range <- 1

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

```

### HyperTuning

For random forest classification, the default mtry (number of
variables randomly sampled at each decision tree split) is $$
\sqrt{number~of~features~(k)}$$ Among popular machine learning
algorithms, random forests display the least variability in
prediction accuracy when tuned (Probst et al., 2018). However, a
higher mtry increases the likelihood that decision trees choose
important features in many splits (Ellis, 2022). Since some of the
predictors have lower predictive power, a higher (than default) mtry

produces higher accuracy. The optimal mtry is 60 with an accuracy of 0.6456677.

(n.b. increasing num.trees from 500 had no effect on accuracy).

~~~
```{r, message=FALSE, echo=FALSE}
mtry_values <- c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)

results_list <- 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
}

mtry <- data.frame(mtry = as.numeric(names(results_list)), accuracy
= unlist(results_list))
print(mtry)
plot(mtry)
```
~~~

### The final tuned model

~~~
```{r, message=FALSE}
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"]
```

```{r, echo=FALSE}
within_range <- 1

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

```

```{r, echo=FALSE}
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()
```




## Model evaluation


Although the model's accuracy on the test dataset improved with
sentiment analysis, sentiment scores were sometimes largely positive
when the actual review was 1 star, perhaps failing to pick up
sarcasm in text. Therefore, the model could benefit from similar
approaches to Bharti et al. (2016) who use sarcasm sentiment
detection in tweets.

The corpus text analysis, although increasing model accuracy, could
have been better implemented and refined. The word cloud shows that
'get' and 'one' are common in both 1 and 5 star reviews, potentially
impacting the model's predictive power. The removal of more
~~~

'stopwords' could improve this.

~~~
```{r, message=FALSE, echo=FALSE, warning=FALSE}
df_train_top_200$review_star <- df_corpus_train$review_star

par(mfrow = c(1, 2))
subset_data <- subset(df_train_top_200, review_star == 1)
word_freq <- colSums(subset_data[, -ncol(subset_data)])
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_data2 <- subset(df_train_top_200, review_star == 5)
word_freq2 <- colSums(subset_data2[, -ncol(subset_data2)])
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")

```
~~~

Additionally, the model could benefit from cross-validation for robustness and to further prevent overfitting.

Overall, new variables such as the year/ month of a review and yelping since, word count and friend count strengthened the model. The final model achieved an accuracy of 64.57% on the test dataset and correctly predicted within 1 star 89.79% of the time. There is potential for improved accuracy with more refined text and sentiment analysis.

### Bibliography

* Bharti, S.K., Vachha, B., Pradhan, R.K., Babu, K.S. and Jena, S.K. (2016). Sarcastic sentiment detection in tweets streamed in real time: a big data approach. Digital Communications and Networks, [online] 2(3), pp.108–121. doi:https://doi.org/10.1016/j.dcan.2016.06.002.
* Boehmke, B. and Greenwell B. (2020). Chapter 11 Random Forests | Hands-on Machine Learning with R. [online] Github.io. Available at: https://bradleyboehmke.github.io/HOML/random-forest.html
* DSPA. 2021. Evaluating CRISP-DM For Data Science. Data Science Process Alliance
* Ellis, C. (2022). Mtry in random forests. [online] Crunching the Data. Available at: https://crunchingthedata.com/mtry-in-random-forests/

∗ Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning, second edition : data mining, inference, and prediction. 2nd ed. New York: Springer
∗ Probst, P., Boulesteix, A.-L. and Bischl, B. (2019). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. Journal of Machine Learning Research, [online] 20, pp.1–32.
∗ Silge, J. and Robinson, D. (2017). Text Mining with R. 'O'Reilly Media, Inc.'
∗ Wright, M.N. and Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software, 77(1). doi:https://doi.org/10.18637/jss.v077.i01