Using Machine Learning Models to Classify Fake and Real News Text

Melody Chen

UCSB Spring 2022

This report was created as a final project for PSTAT 131 Introduction to Statistical Machine Learning at the University of California, Santa Barbara, with Professor Katie Coburn during Spring 2023.

Introduction

About the Dataset

The purpose of this report is to build a fake news detection model that can detect language patterns that characterize fake and real news. The dataset was downloaded as a `.csv` file from The Information Security and Object Technology (ISOT) Research Group at the University of Victoria in British Columbia, Canada. The `Fake.csv` dataset contains news articles from different sources that were flagged by Wikipedia and Politifact (an American fact-checking organization) as unreliable websites. The `True.csv` dataset contains news articles from Reuters (a reliable news organization).

What is "Fake News"?

Fake news is information that is clearly and *deliberately* fabricated in such a way that appears legitimate and is designed to manipulate people's perception of real events (Center for Information Technology and Society at University of California, Santa Barbara). False and misleading content can be presented in various forms of communication, including written, spoken, and digital/electronic. Importantly, fake news or disinformation is a deliberate form of deception that capitalizes on the intent to manipulate the reader or audience. This notion separates itself from misinformation, which is created regardless of the intent to deceive. When individuals or groups knowingly share or re-post misinformation to deliberately cast doubt on an event, misinformation can easily turn into disinformation.

The problem of "fake news" has been a persistent theme in the headlines for decades, but most notably within the past few years. One of the most relevant examples of a fake news campaign is the collective "Big Lie" narrative that is reinforced by former President Donald Trump about the 2020 election results (Brennan Center for Justice). The National Intelligence Council released a 2021 assessment of foreign threats to the 2020 U.S. federal elections and found that Russian President Vladimir Putin authorized Russian individuals/groups and government organizations to denigrate President Biden's candidacy and the Democratic Party with the intent of eroding public trust in the electoral process.

These campaigns have largely existed in the medley of social media networks where false and misleading information may easily go undetected with the speed in which it spreads, taking into account the appended efforts made by both domestic and foreign actors. In response, social media platforms like Facebook and Twitter are now engaging in fact-checking and labeling misinformation. Yet, these initiatives target the *spread* of misinformation rather than the *source*. Different variations of falsehoods may still exist in the

media. And without identifying and stamping out on the origin of these falsehoods, we may observe a never-ending stream of fake news and illusory efforts.

Why is this model relevant?

For the purposes of our Natural Language Processing (NLP) model — a machine learning technique that programs the computer to interpret and manipulate the human language — our dataset consists of news headlines and texts that were electronically published in written form. Effectively, we target the primary source of these headlines rather than secondary or tertiary efforts made by (social) media users.

About two-thirds of U.S. adults say false and misleading news stories have caused a great deal of confusion about the basic facts of current events, according to a survey of 1,002 U.S. adults conducted by Pew Research Center from Dec. 1-4, 2016. This report was published a year before the collection of our raw dataset and four years before the 2020 presidential election fraud disinformation campaign.

With the upcoming presidential election in 2024, the reversal of content moderation and rise in Generative Artificial Intelligence (AI) may put the public at risk. On June 2, 2023, YouTube announced a reversal of its policy ban on misinformation about the 2020 presidential election. The Republican National Committee created a lookbook video featuring AI-generated images and clips of Biden and Vice President Kamala Harris celebrating at Election Day. These trends dangerously illustrate a future in which truths and falsehoods become much harder to discriminate, making it more pertinent for statisticians and data scientists to explore appropriate algorithms to detect fake news.

Exploratory Data Analysis (EDA)

Prior to modeling, we can inspect what our data looks like. We cannot assume that our raw data is completely clean and ready for modeling. Instead, we need to convert some variables to factors, remove any missing values, and clean our predictor names. Additionally, there will be some text-cleaning manipulation performed on our data for our natural language pre-processing step.

Loading Packages and Data

Let's load in all necessary packages required for this report in R.

Next, we can load our `Fake.csv` and `True.csv` datasets and assign the data to a variable. We then clean our predictor names using `clean_names()` such that all resulting names are unique and consist of the `_` character, numbers, and letters. Afterwards, we create a new column `truth_value` where we assign "fake" or "news" to each dataset, and a target category `Category` identifying Fake News as 0 and Real News as 1. After assigning those values, we can combine the two dataframes by row into a new dataframe called `news`. The purpose behind the new columns `truth_value` and `Category` is to identify whether a news title/text is fake or news in the merged dataframe.

```
# Loading the data as a .csv and assigning the data to a variable
fake_news <- read.csv("Fake.csv")
real_news <- read.csv("True.csv")


# cleaning predictor names
fake_news <- clean_names(fake_news)
real_news <- clean_names(real_news)

# calling head() to see the first five observations
```

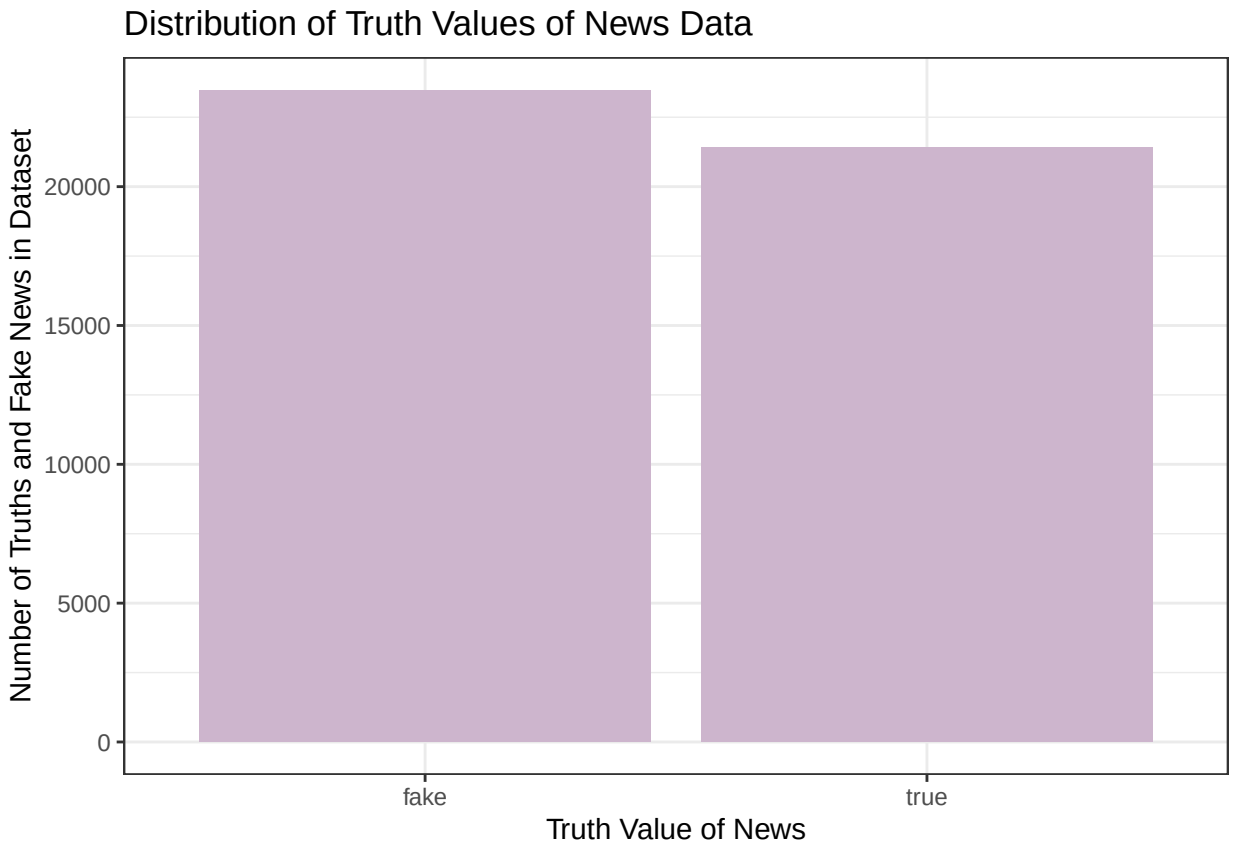
```
# head(fake_news)
# head(real_news)

truth_value = "fake"
fakeNews <- cbind(fake_news, truth_value)

truth_value = "true"
realNews <- cbind(real_news, truth_value)

# Create a Target Category Identifying Fake News as 0 and Real News as 1
fakeNews['Category'] = 0
realNews['Category'] = 1

# Combine the two dataframes into a new dataframe called 'news'
news <- rbind(fakeNews, realNews)

# making sure that the 'news' dataframe contains both fake and true news
# head(news)  # contains truth_value = FAKE
# tail(news)  # contains truth_value = TRUE
```

We can retrieve the dimensions of our new dataframe `news`. Our dataframe consists of over 40,000 observations and exactly 5 variables.

```
dim(news)  # retrieving the dimensions of our new dataframe
```

```
## [1] 44898     6
```

To view the distribution of truth values of our news dataframe, we can create a bar graph using `ggplot`. There are 23,481 fake news observations and 21,417 real news observations. It appears that our dataframe is relatively balanced in both categories.



It is important to determine whether there are null values in our `news` dataframe prior to tidying our data. Using `is.null()`, we observe no null values in our dataframe. We can then proceed to text-cleaning methods in Natural Language Processing (NLP).

```
is.null(news)
```

```
## [1] FALSE
```

Text Cleaning Methods in Natural Language Processing (NLP)

Tidying the Data

Date: One of our variables `date` is in %B %d, %Y format (e.g., June 10, 2023), which is not conventional to use in our classification model. So, we take care to convert date in standard YYYY-MM-DD format.

Lower Case: We then lower case all words in the news article to ensure our text is consistent when performing parsing in the process of NLP preprocessing.

Remove Punctuation and Number We remove all punctuation and numbers in our data. We are only interested in words.

```
# Convert the date in date format
date <- as.Date(news$date, format = "%B %d, %Y")
news$date <- format(date, "%Y-%m-%d")

# Lower case all words in title
```

```
news$title <- tolower(news$title)

# Remove punctuation
news$title <- gsub("[[:punct:]]", "", news$title)

# Remove numbers
news$title <- gsub("[[:digit:]]", " ", news$title)
```

Lemmatize words

Next, we take care to lemmatize the remaining words in our `title` column using `lemmatize_strings()`. Lemmatization is the process of minimizing the words to their root form, which is known as the lemma. The purpose of this process is to standardize and consider different forms of the same word as a single unit, which reduces the complexity of our vocabulary and improves analysis accuracy. For example, lemmatizing the word "bagels" normalize the text to "bagel."

```
news$title <- lemmatize_strings(news$title)
```

Removing Stop Words

Then, we make sure to remove stop words, which eliminates common words like articles, pronouns, and prepositions. These words do not generally carry significant meaning, so this process would highlight important keywords that bear semantic meaning and reduce noise in our dataset.

```
# Create a corpus
corpus <- VCorpus(VectorSource(news$title))

# Remove English stop words
corpus_clean <- tm_map(corpus, removeWords, stopwords("english"))

news_title <- sapply(corpus_clean, as.character)

# Recreate corpus using news_title
corpus <- VCorpus(VectorSource(news_title))

# Remove unncessary letters
corpus_clean <- tm_map(corpus, removeWords, c("s", "t", "k"))

news_title <- sapply(corpus_clean, as.character)
```

Term Frequency-Inverse Document Frequency (TF-IDF)

We can build a term-document matrix, which consists of the frequency of words in all true and fake news title texts. We can control the weighting such that our frequency weight is standardized to the **Term Frequency-Inverse Document Frequency** (TF-IDF), which can be calculated by taking the product of two statistics: term frequency (TF) and inverse document frequency (IDF). We consider the TF as the ratio between the number of times a keyword is found in a document and the number of words in document. The IDF is the natural logarithm of (the total number of documents / number of documents containing a keyword).

```
tdm <- TermDocumentMatrix(corpus_clean,
                          control=list(weighting=weightTfIdf))
m <- as.matrix(tdm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),tf_idf=v)
head(d, 10)
```

```
##                   word    tf_idf
## trump           trump 2754.1124
## video           video 1964.7612
## say               say 1703.9990
## obama           obama 1074.4964
## house           house 1058.5861
## republican republican 1045.0129
## new               new  893.8787
## clinton       clinton  885.0131
## hillary       hillary  856.5789
## watch           watch  852.7406
```

```
tdm
```

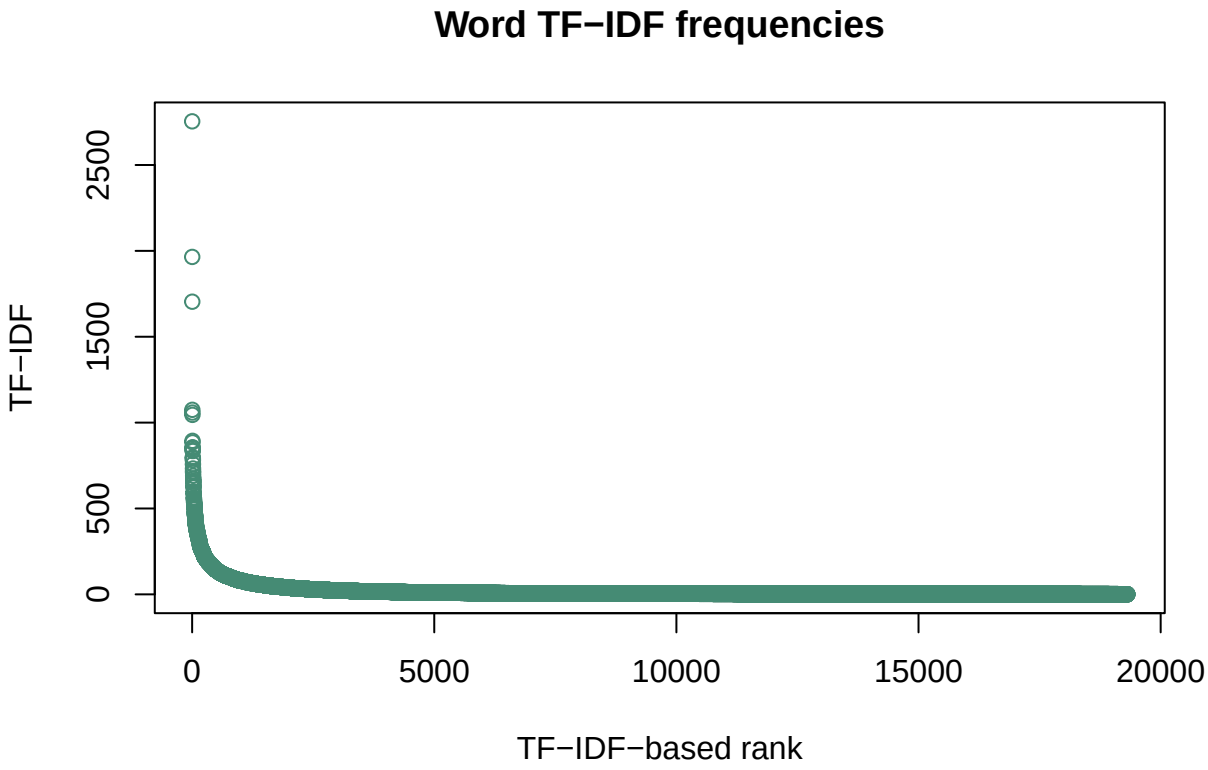
```
## <<TermDocumentMatrix (terms: 19312, documents: 44898)>>
## Non-/sparse entries: 406255/866663921
## Sparsity           : 100%
## Maximal term length: 118
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

```
freq=rowSums(as.matrix(tdm))
head(sort(freq,decreasing=TRUE))
```

```
##      trump      video        say      obama      house republican
##   2754.112   1964.761   1703.999   1074.496   1058.586   1045.013
```

We can extract six terms with the greatest TF-IDF frequencies, ranked from highest to lowest.

```
plot(sort(freq, decreasing = T),col="aquamarine4",main="Word TF-IDF frequencies", xlab="TF-IDF-based ra
```

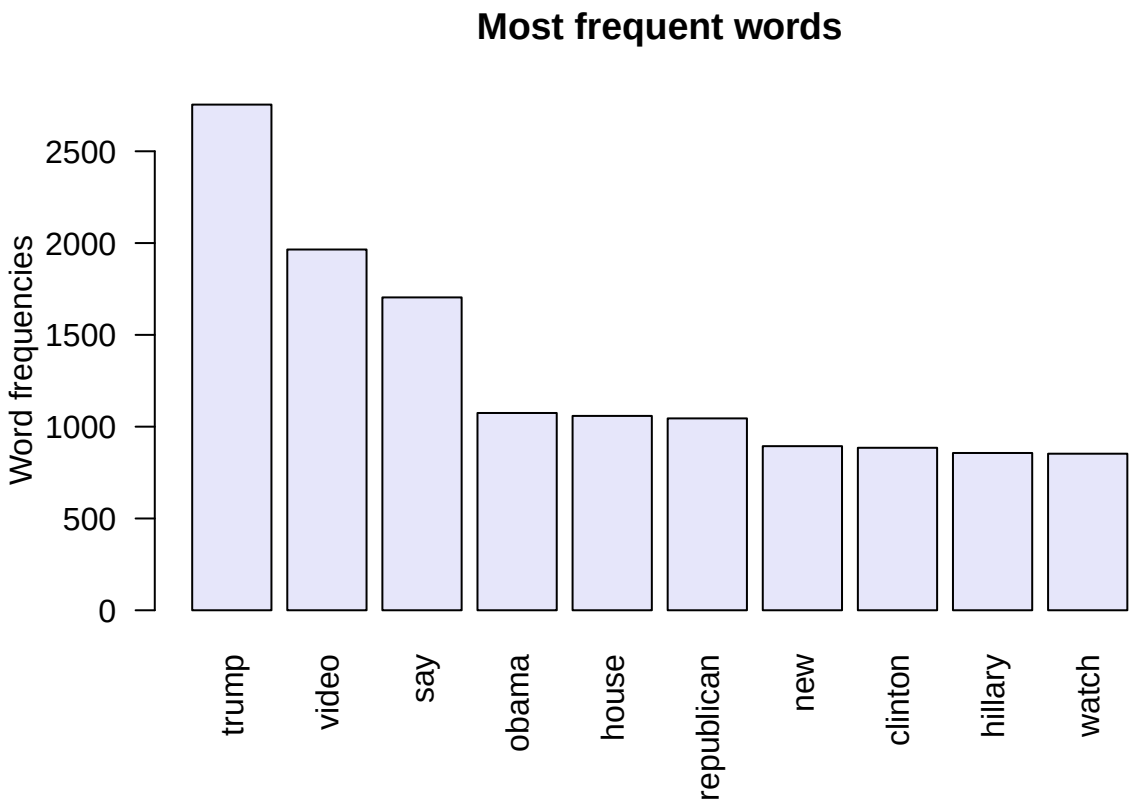


While we will not be using our calculated TF-IDF frequency in our predictive model, it is important to consider the relevancy of the terms with the largest TF-IDF value in each document.

```
# Add a new column 'max_tfidf' to store the largest TF-IDF value for each word
news$max_tfidf <- 0

# Iterate over each word in the dataframe
for (i in 1:nrow(news)) {
  words <- strsplit(news$title[i], "\\s+")
  max_tfidf <- 0

  # Find the largest TF-IDF value for each word in the text
 for (j in 1:length(words)) {
    for (word in words[[j]]) {
      tfidf_value <- d$tf_idf[i]
      if (!is.na(tfidf_value) && tfidf_value > max_tfidf) {
        max_tfidf <- tfidf_value
      }
    }
 }

   # Assign the largest TF-IDF value to the corresponding row in the dataframe
  news$max_tfidf[i] <- max_tfidf
}
```

Based on the preceding dataframe `d`, we can build a barplot consisting of the most frequent words. We can numerically and visually observe that the most frequent word in all new titles is *trump*, followed by *video*, *say*, and *obama*.

```r
barplot(dfreq[1:10,]$freq, las = 2, names.arg = dfreq[1:10,]$word,
        col ="lavender", main ="Most frequent words",
        ylab = "Word frequencies")
```



Text Normalization and White Spaces

Afterwards, we replace all occurrences of one or more whitespace characters with a single space. This ensures that our processed text reduces variation in whitespace usage, so the text is uniformly aligned.

```r
# Replaces all occurrences of 1 or more whitespace characters with a single space
news_title <- gsub("\\s+", " ", news_title)

# Drop news$title and add news_title to our 'news' dataframe
news$title <- news_title
```

After performing all necessary text-cleaning methods, we find ourselves with a standardized title text that optimally represents important features of our data. Ultimately, these methods prioritize computational efficiency and accuracy.

We can then examine the emotional lexicons of the observed words in our title text and match its sentiment with our dictionary terms.

Examine Sentiment and Emotion Lexicons

Now, we will be introducing the `syuzhet` sentiment analysis algorithm package in R. This package consists of a NRC Word-Emotion Association Lexicon which includes positive/negative sentiment values and emotional

categories that can classify a word's association with eight emotions (i.e., anger, anticipation, disgust, fear, joy, sadness, surprise, and trust).

Prior to retrieving the sentiment values for the title text, we consider the `news` dataframe uniquely as "fake" or "true."

```
fake_news <- news[news$truth_value == "fake",]
true_news <- news[news$truth_value == "true",]

f.sentiment <- get_nrc_sentiment(fake_news$title)
```

```
## Warning: `spread_()` was deprecated in tidyr 1.2.0.
## i Please use `spread()` instead.
## i The deprecated feature was likely used in the syuzhet package.
##   Please report the issue to the authors.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
t.sentiment <- get_nrc_sentiment(true_news$title)
```

```
head(f.sentiment)
```

```
##   anger anticipation disgust fear joy sadness surprise trust negative positive
## 1     0            0       0    0   0       1        1     0        1        0
## 2     1            2       1    1   0       1        1     0        2        0
## 3     1            2       0    1   0       0        0     1        3        0
## 4     0            0       0    0   0       0        1     0        0        0
## 5     0            0       0    0   0       0        1     0        0        2
## 6     0            0       1    1   0       1        0     1        2        0
```

```
head(t.sentiment)
```

```
##   anger anticipation disgust fear joy sadness surprise trust negative positive
## 1     1            1       0    2   0       0        0     1        2        1
## 2     0            0       0    1   0       0        0     0        0        0
## 3     0            0       0    0   0       0        0     0        0        1
## 4     0            0       0    0   0       0        0     0        0        0
## 5     0            0       0    0   0       0        1     0        0        0
## 6     0            2       1    0   1       0        0     2        0        3
```

```
# save(fake_news, f.sentiment, file="fake_sentiment.rda")
load("fake_sentiment.rda")

# save(true_news, t.sentiment, file="true_sentiment.rda")
load("true_sentiment.rda")
```

We can examine the top emotion observed in both fake and real news. The top emotion in fake news is "fear"; the top emotion in real news is "trust."

```r
# Show top emotion "fear" in fake news
fear_fake <- which(f.sentiment$anger > 0)  # fear
head(fake_news$title[fear_fake])
```

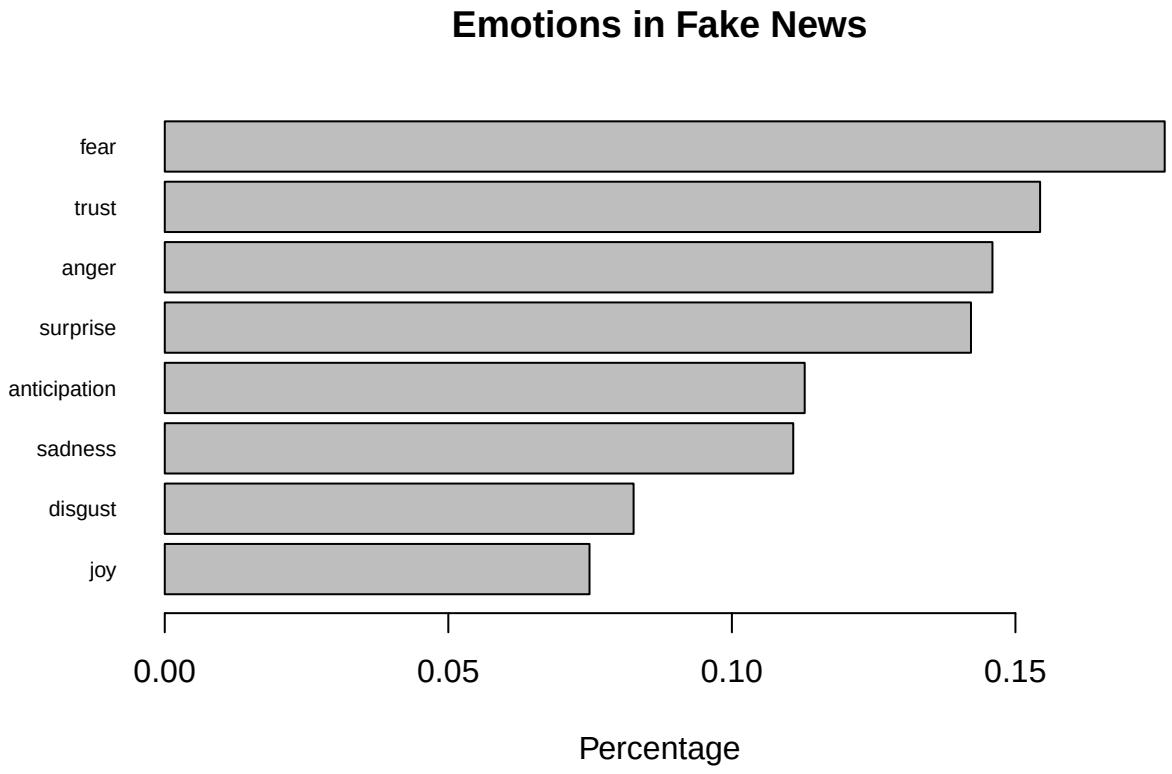
```
## [1] "drink brag trump staffer start russian collusion investigation"
## [2] "sheriff david clarke become internet joke threaten poke people eye"
## [3] "fresh golf course trump lash fbi deputy director james comey"
## [4] "former cia director slam trump un bully openly suggest hes act like dictator tweet"
## [5] "papa johns founder retire figure racism bad business"
## [6] "watch paul ryan just tell us doesnt care struggle family live blue state"
```

```r
# Show top emotion "trust" in real news
trust_true <- which(t.sentiment$trust > 0)  # trust
head(true_news$title[trust_true])
```

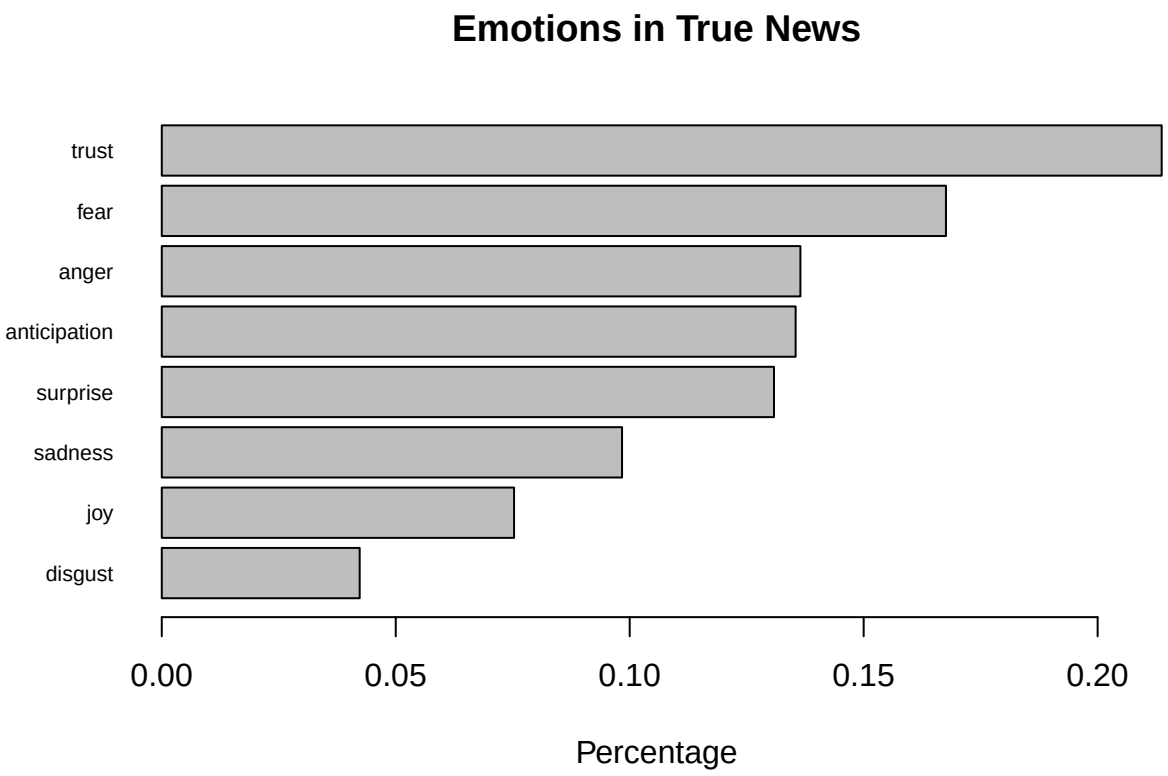
```
## [1] " us budget fight loom republican flip fiscal script"
## [2] "white house congress prepare talk spend immigration"
## [3] "alabama official certify senatorelect jones today despite challenge cnn"
## [4] "jones certify us senate winner despite moore challenge"
## [5] "new york governor question constitutionality federal tax overhaul"
## [6] "man say deliver manure mnuchin protest new us tax law"
```

We can then plot the percentage of each emotion and positive/negative sentiment by counting each column in the `f.sentiment` and `t.sentiment` tables.

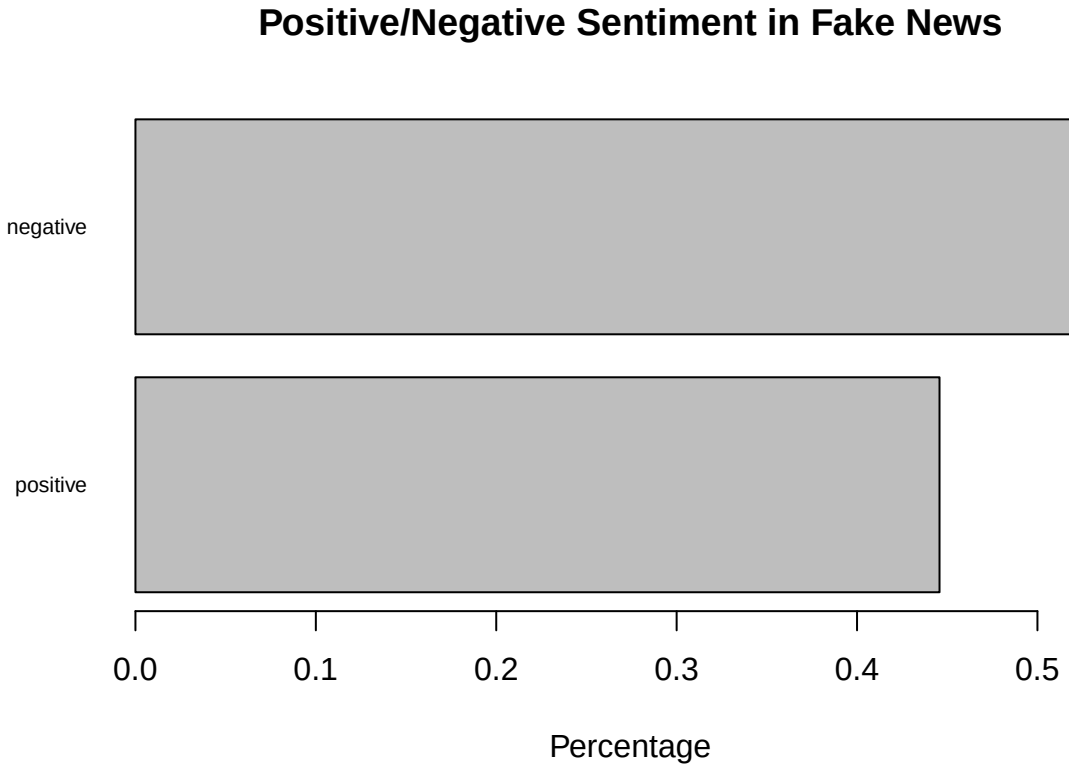
```r
# Bar Plot of Emotions in Fake/True News
barplot(
  sort(colSums(prop.table(f.sentiment[, 1:8]))),
  horiz = TRUE,
  cex.names = 0.7,
  las = 1,
  main = "Emotions in Fake News", xlab="Percentage"
  )
```



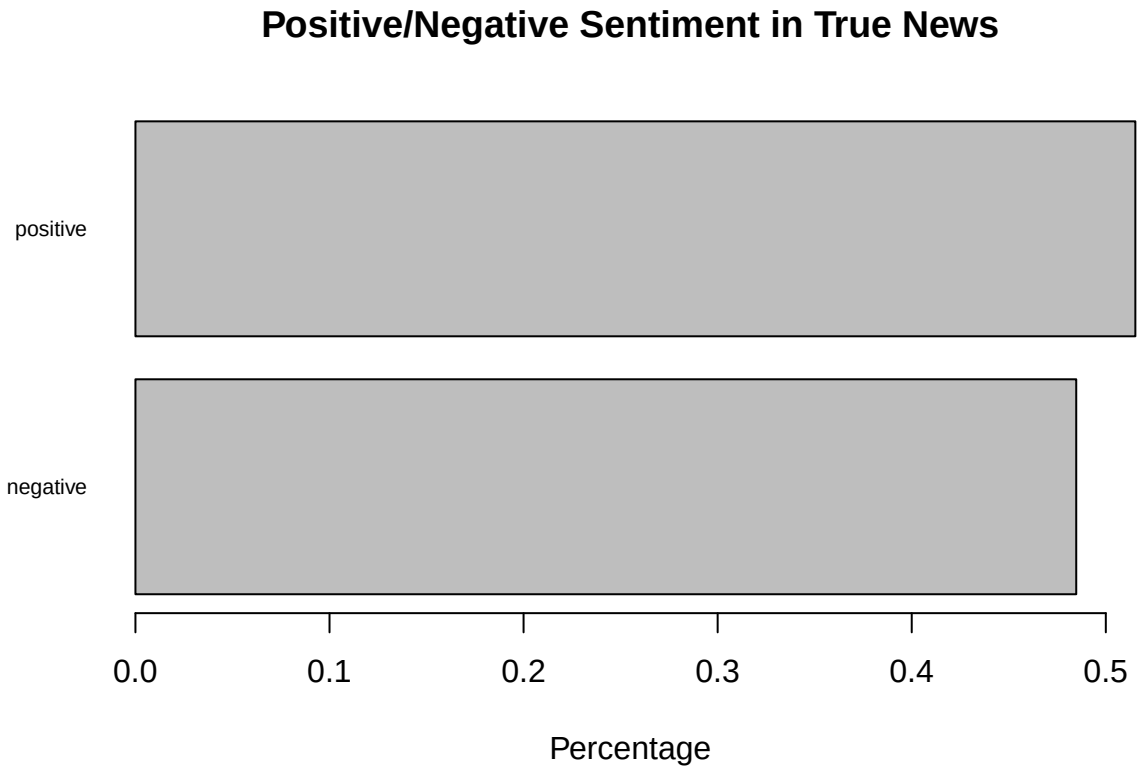

```
barplot(
  sort(colSums(prop.table(t.sentiment[, 1:8]))),
  horiz = TRUE,
  cex.names = 0.7,
  las = 1,
  main = "Emotions in True News", xlab="Percentage"
  )
```




```
# Bar Plot of Positive/Negative Sentiment in Fake/True News

barplot(
  sort(colSums(prop.table(f.sentiment[, 9:10]))),
  horiz = TRUE,
  cex.names = 0.7,
  las = 1,
  main = "Positive/Negative Sentiment in Fake News", xlab="Percentage"
  )
```




```
barplot(
  sort(colSums(prop.table(t.sentiment[, 9:10]))),
  horiz = TRUE,
  cex.names = 0.7,
  las = 1,
  main = "Positive/Negative Sentiment in True News", xlab="Percentage"
  )
```



To set up a new dataframe in which we consider all sentiments as factors, we consider "1" to indicate the presence of a sentiment and "0" as the absence of a sentiment. We then combine the `news` dataframe and `all_sentiment` dataframe into a `final_news` dataframe.

For the purpose of the report, we are focused on the binary values of `all_sentiment`, which we determine based on the `title` of the news articles and NRC Word-Emotion Association Lexicon. Our goal is to build an ML model that closely predicts the `truth_value` of our news dataset.

```
# Consider all sentiments
all_sentiment <- get_nrc_sentiment(news_title)

# Combining the news dataframe to sentiment dataframe
final_news <- cbind(news, all_sentiment)
```

```
# save(all_sentiment, final_news, file="final_news.rda")
load("final_news.rda")
```

Setting Up Models

Prior to setting up our model, we will randomly split our data into training and testing sets, while taking care to stratify our sample on our response variable, i.e., `truth_value`.

Splitting Training and Testing Set

Our first step is to set our seed, which creates reproducible results every time we train our models. We set our `initial_split()` against our `final_news` dataframe, ensuring that 80% of our data is randomly split to our training set and 20% to testing set. I chose a proportion of 0.8 because this allows for more training data while preserving enough data for our test.

```r
set.seed(123)

news_split <- final_news %>% initial_split(prop = 0.8, strata = truth_value)

news_train <- training(news_split)
news_test <- testing(news_split)

news_train <- news_train %>% mutate(truth_value = factor(truth_value, levels = c("fake", "true")))
news_test <- news_test %>% mutate(truth_value = factor(truth_value, levels = c("fake", "true")))
```

The training data has 35,917 observations; the testing data has 8,981 observations.

```r
dim(news_train)
```

```
## [1] 35917    17
```

```r
dim(news_test)
```

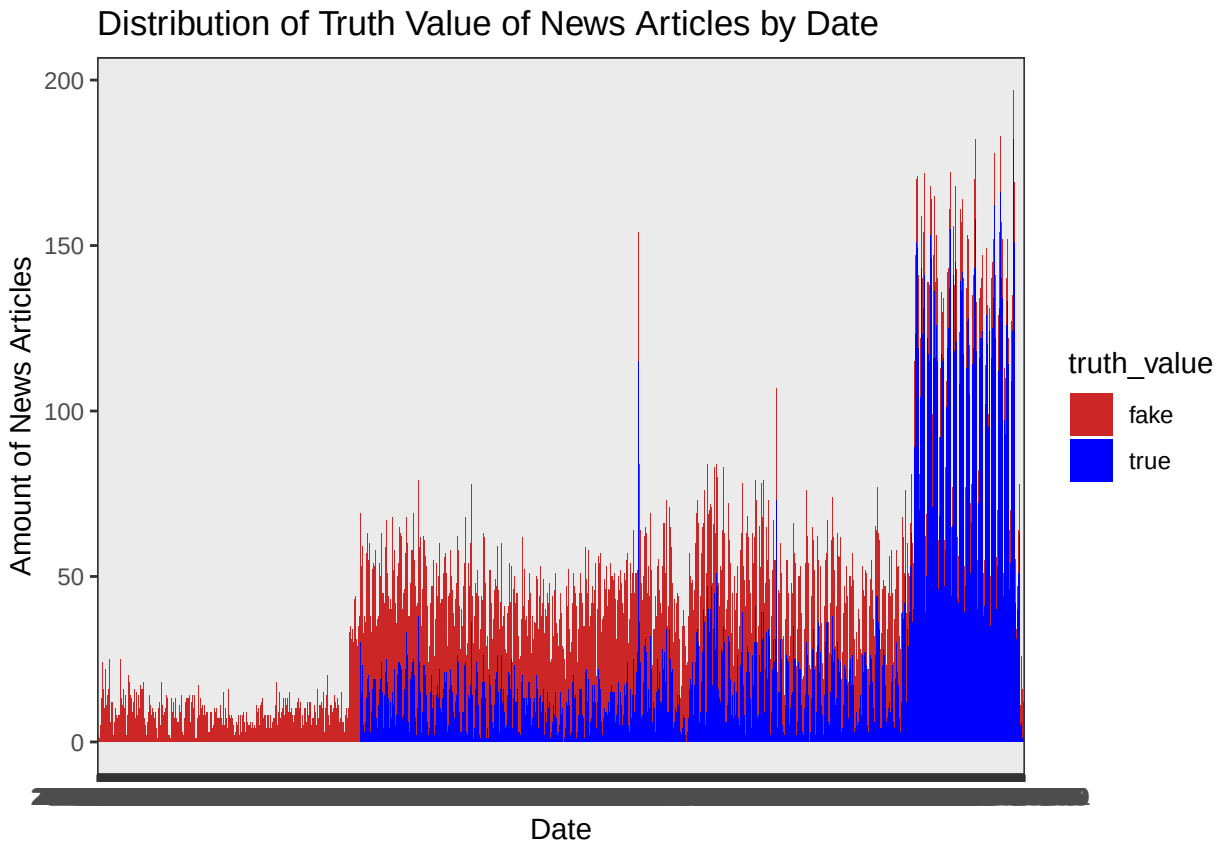
```
## [1] 8981   17
```

Visual EDA and Codebook

***Note*:** We will be excluding the rest of the variables in our original dataset. The `date` and `subject` variables fail to logically explain whether a news article is real or fake. After all, real and fake news occur anytime. We can view the distribution of truth value of news articles by date below:

`Date` Date when the news article was published; originally in %B %d, %Y format (e.g., June 10, 2023). Converted character to YYYY-MM-DD format (e.g., 2023-06-10).

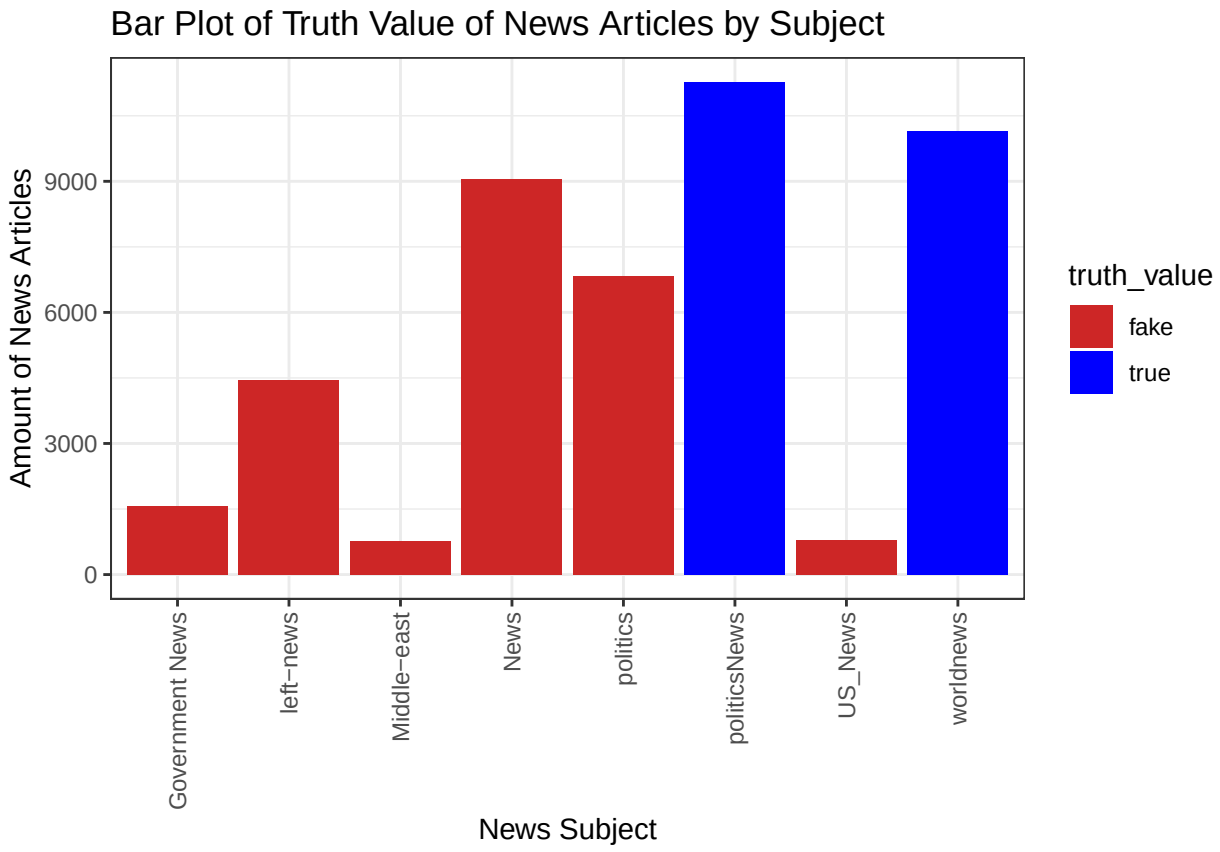
```r
ggplot(news, aes(date)) +
  geom_bar(aes(fill = truth_value)) +
  scale_fill_manual(values = c("firebrick3", "blue")) +
  ggtitle("Distribution of Truth Value of News Articles by Date") +
  xlab("Date") + ylab("Amount of News Articles")
```



In our dataset, all news articles that fall under the subject "Government News," "left-news," "Middle-east," "News," "politics," and "US_News" correspond to a `truth_value = "fake"`. All news articles that contain the subject "politicsNews" and "worldnews" have a "true" `truth_value`. While the `subject` variable appears to have a clear distinction between the truth values in our dataset, we cannot confidently apply this predictive method in our real-world population of news articles. Ultimately and realistically, all news subjects may contain both real and fake information.

`Subject` Character subject category of news article (i.e., Government News, left-news, Middle-east, News, politics, US_News, politicsNews, worldnews)

```
ggplot(news, aes(subject)) +
  geom_bar(aes(fill = truth_value)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_fill_manual(values = c("firebrick3", "blue")) +
  ggtitle("Bar Plot of Truth Value of News Articles by Subject") +
  xlab("News Subject") + ylab("Amount of News Articles")
```

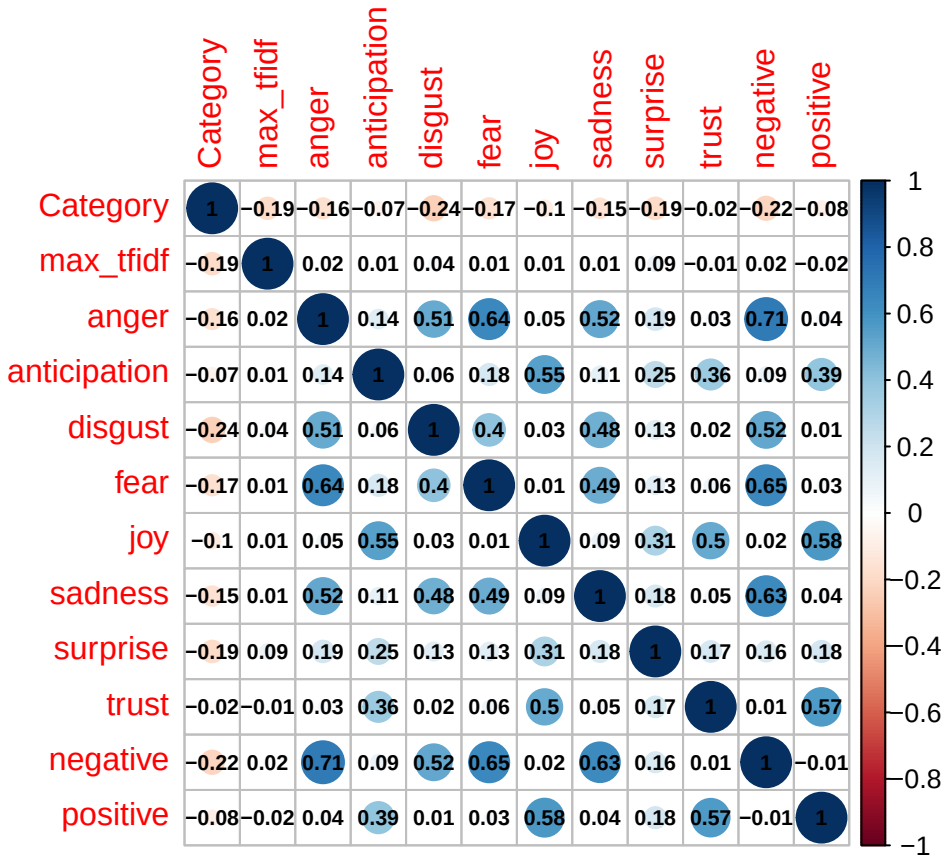


Correlation Matrix

Let's explore the overall relationship between all the numeric, continuous variables using a correlation matrix on our training dataset.

We can first select numeric variables that pass the condition `is.numeric()` to ensure that we only capture numeric variables in our correlation matrix. The strongest linear dependence occurs between `negative` and `anger` with a correlation coefficient of 0.71. Variable pairs with relatively strong linear dependence include `anger` and `fear` with$r^2 = 0.64$, and then `positive` and `joy` with $r^2 = 0.58$ and `positive` and `trust` with $r^2 = 0.57$.

```
# Select columns in news_train that are numeric
num_news_train <- select_if(news_train, is.numeric)

# Create a correlation matrix
M = cor(num_news_train)
corrplot(M, method='circle', addCoef.col = 1, number.cex = 0.7)
```



Building Our Model Recipe

Now, we begin to build our model recipe by taking into account both our predictors (calculated based on the positive/negative sentiment values and emotional categories of the NRC Word-Emotion Association Lexicon) and response variable `truth_value`.

Let's consider all eight emotions (anger, anticipation, disgust, fear, joy, sadness, surprise, trust) and sentiments (positive, negative). We will exclude the `date` and `subject` variables, as we observe our main predictors. Since all variables used in our model are categorical, we make sure to convert these variables into dummy variables, and center and scale our data for model usage.

```
# Building our recipe
news_recipe <- recipe(truth_value ~ anger + anticipation + disgust + fear + joy + sadness + surprise + t
  step_dummy(all_nominal_predictors()) %>%  # create dummy variables
  step_center(all_predictors()) %>% # standardize predictor variables
  step_scale(all_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_interact(terms = ~anger:disgust:fear:sadness:negative + joy:surprise:trust:positive)  # create i

news_recipe %>%
  prep() %>%
  bake(new_data = news_train)
```

```
## # A tibble: 35,917 x 13
##     anger anticipation disgust   fear    joy sadness surprise  trust negative
##     <dbl>        <dbl>   <dbl>  <dbl>  <dbl>   <dbl>    <dbl>  <dbl>    <dbl>
##  1 -0.781       -0.753  -0.528 -0.824 -0.570   0.741    0.535 -0.906  -0.0927
##  2  0.433        1.98    1.22   0.228 -0.570   0.741    0.535 -0.906   0.833
##  3  0.433        1.98   -0.528  0.228 -0.570  -0.673   -0.866  0.224   1.76
##  4 -0.781       -0.753  -0.528 -0.824 -0.570  -0.673    0.535 -0.906  -1.02
##  5 -0.781       -0.753  -0.528 -0.824 -0.570  -0.673    0.535 -0.906  -1.02
##  6 -0.781       -0.753   1.22   0.228 -0.570   0.741   -0.866  0.224   0.833
##  7  0.433       -0.753  -0.528  0.228 -0.570  -0.673    0.535  1.35   -0.0927
##  8  1.65        -0.753  -0.528  2.33  -0.570  -0.673    1.94   1.35    1.76
##  9 -0.781        1.98    1.22   0.228  1.11    0.741    0.535 -0.906  -0.0927
## 10  0.433       -0.753   1.22   0.228 -0.570   0.741   -0.866 -0.906  -0.0927
## # i 35,907 more rows
## # i 4 more variables: positive <dbl>, truth_value <fct>,
## #   joy_x_surprise_x_trust_x_positive <dbl>,
## #   anger_x_disgust_x_fear_x_sadness_x_negative <dbl>
```

K-Fold Cross Validation

We will perform stratified cross validation on our response variable `truth_value` by 5 folds.

```
news_folds <- vfold_cv(news_train, v = 5, strata = truth_value); news_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits               id
##   <list>               <chr>
## 1 <split [28733/7184]> Fold1
## 2 <split [28733/7184]> Fold2
## 3 <split [28733/7184]> Fold3
## 4 <split [28734/7183]> Fold4
## 5 <split [28735/7182]> Fold5
```

Next, we can save our results to an RDA file for future processing. Because model building takes up some computing time, we can effectively return to our RDA file and load it later for future use.

```
# save(news_folds, news_recipe, news_train, news_test, file="News_Modeling_Setup.rda")
load("News_Modeling_Setup.rda")
```

Building Prediction Models

We will be building four different machine learning models to assess our report using the same recipe. Since the model tuning takes up some computing time and power, we will perform the codes in the appendix, and save and load the model separately in our resulting findings. That way, we only run our model once.

The models we will be analyzing are as follows:

- Decision Tree
- Random Forest

- Gradient Boosted Tree
- Elastic Net Regression

Performance Metric

To evaluate performance, we will be applying the metric of performance, Receiver Operating Characteristic Area Under the Curve, `roc_auc`. This metric is commonly used in ML to determine the quality of binary classification models. We can visually represent the metric by plotting the ROC curve, which shows the trade-off between the true positive rate (TPR) and false positive rate (FPR). A model containing a higher ROC AUC value indicates better predictive power of the model, where the ROC curve comes closer to the top-left corner of the plot (i.e., higher TPR and lower FPR). A perfect classifier would result in a ROC AUC value of 1.0.

Model Building Outline

All four of our models tend to follow a similar pattern. The general process can be outlined as such:

1. Set up the model by identifying the type of model, prepare the engine, and set the mode. Since we are working with a classification model, we will set the mode to "classification."
2. Prepare the workflow for the model and add the new model and our established `news_recipe`.
3. Set up the tuning grid, specify the parameters we want to tune and levels of tuning.
4. Tune the model with parameters and/or hyperparameters of choice.
5. Select the most accurate model based on the tuning grid and finalize the workflow with those tuning parameters.
6. Fit the chosen model with our workflow to the training dataset `news_train`.
7. Save our results to an RDA file to ensure efficiency as we load our product back to our main project file.

Model Results

Since our dataset has over 40,000 observations, it is important to save all our models and load our results to inspect each of their individual performances and analysis. This step ensures that we do not re-run any of our models, thus reducing computing time.

```
# save(tune_tree, file="tune_tree.rda")
# save(rf_tune, file = "rf_tune.rda")
# save(tune_bt_class, file = "tune_bt_class.rda")
# save(en_tune, file = "en_tune.rda")

load("tune_tree.rda")
load("rf_tune.rda")
load("tune_bt_class.rda")
load("en_tune.rda")
```

Visualizing Results

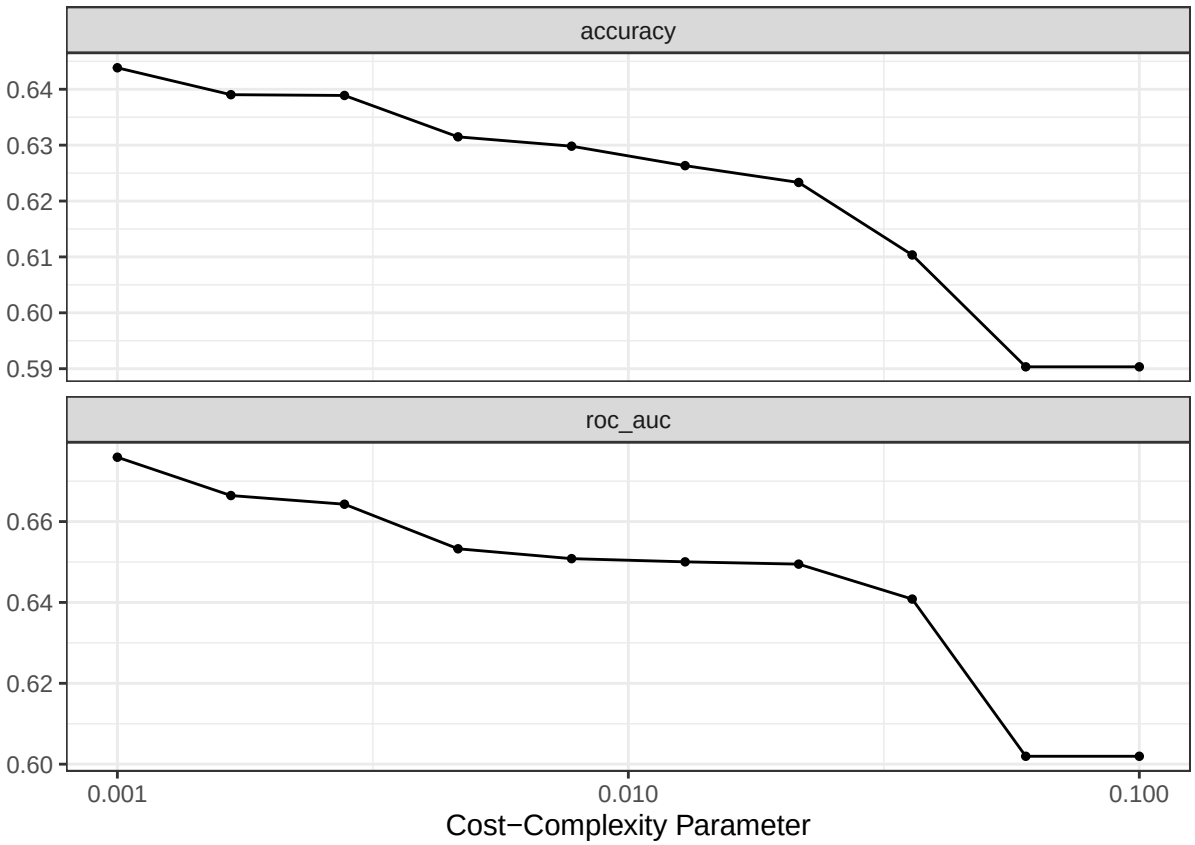
To visualize the results of our tuned model, we will be using the `autoplot` function in R. Since our primary performance metric is `roc_auc`, we will visualize our model performance based on the effects that the change in certain parameters has on `roc_auc`.

Decision Tree

In a pruned decision tree, we would tune the `cost_complexity`, or the range of complexity of the tree, which is specifically used to prevent overfitting. In our grid, we set the range of `cost_complexity` to -3 and -1 with the levels = 10. We notice that as the Cost-Complexity Parameter increased, the ROC AUC value trended lower.

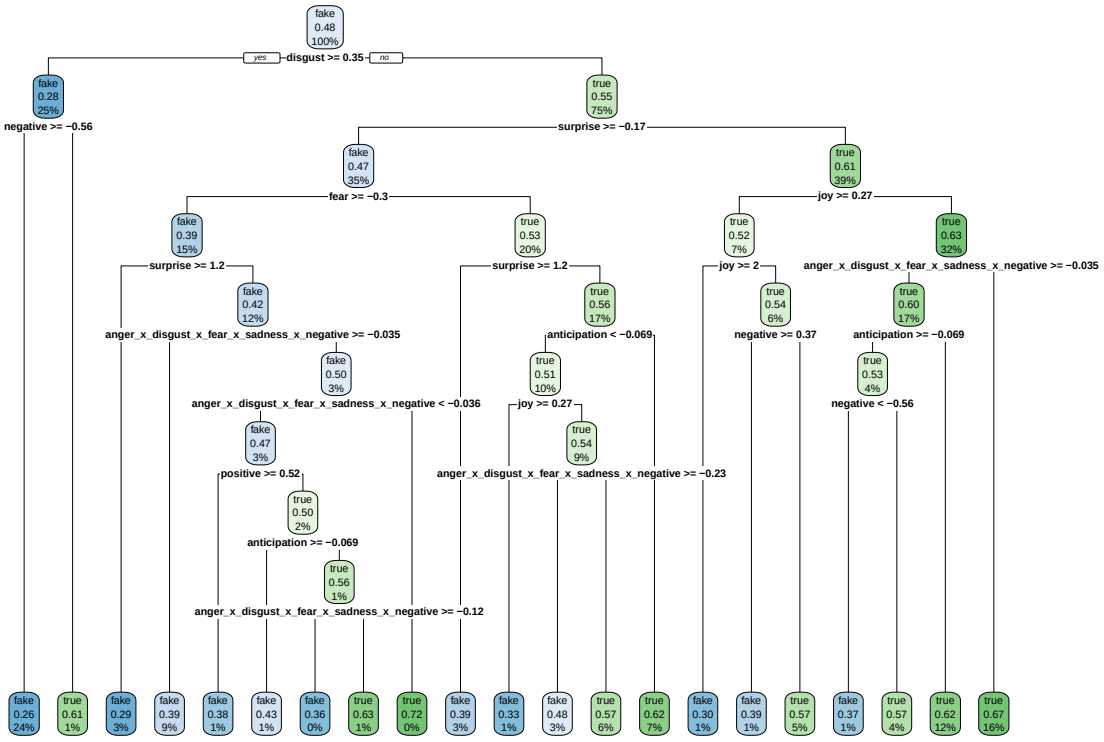
```
# RANDOM FOREST MODEL
rf_class_news <- rand_forest(mtry = tune(),
                             trees = tune(),
                             min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

# Set up a random forest workflow with recipe 'news_recipe'
rf_class_wf <- workflow() %>%
  add_model(rf_class_news) %>%
  add_recipe(news_recipe)

# Create a regular grid with 1 level each
rf_grid <- grid_regular(mtry(range = c(1, 6)),
  trees(range = c(200, 600)),
  min_n(range = c(10, 20)),
  levels = 5)
```

```
# Plot the tuned tree
autoplot(tune_tree)
```



And we can extract the model fit results and view them with the `rpart.plot()` function. We can view the actual splits in this function, which we observe that the optimal value of `cost_complexity` resulted in a relatively deep tree.

```
best_complexity <- select_best(tune_tree, metric = "roc_auc")
# Finalize workflow and fit the final model to training set
tree_final <- finalize_workflow(tree_wf, best_complexity)
tree_final <- fit(tree_final, news_train)
tree_final_fit <- fit(tree_final, data=news_train)

# RPart Plot
tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



Random Forest

In our random forest model, we tuned three unique parameters:

(1) `mtry`, or the number of variables we randomly sample as candidates at each split.

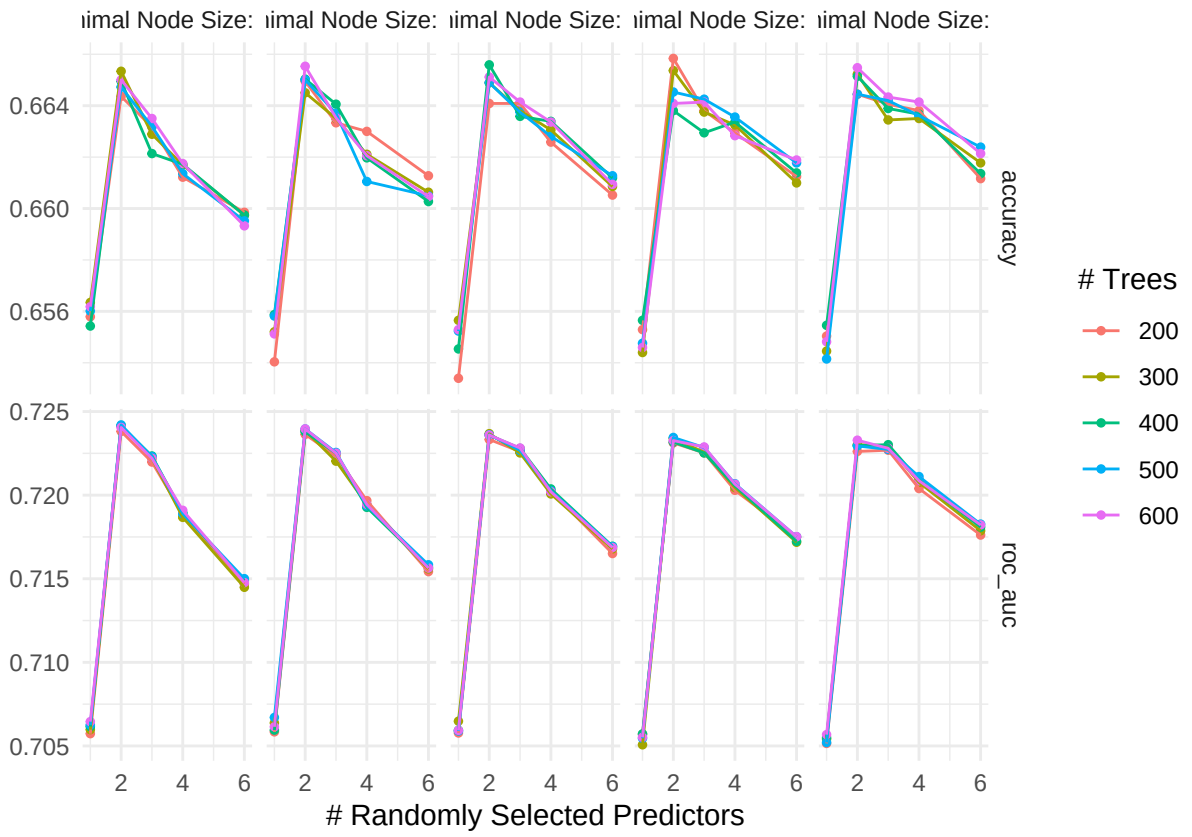
(2) `trees`, or the number of decision trees to be built in the random forest model

(3) `min_n`, or the minimum number of samples in a node that is required to split.

We set our `mtry` within the range of 1 and 6, `trees` within the range of 200 and 600, and `min_n` within the range of 10 and 20. We fit 5 `levels` to our model.

We observe from our plot that generally around the second predictor, the ROC AUC value is the highest. The difference in number of trees did not affect much of the ROC AUC. It appears that our most accurate model has 500 `trees`, 2 `mtry`, and 10 `min_n` with a ROC AUC of 0.7242.

```
autoplot(rf_tune) + theme_minimal()
```




```
best_rf <- select_best(rf_tune, metric = "roc_auc")
show_best(rf_tune, n=1)
```

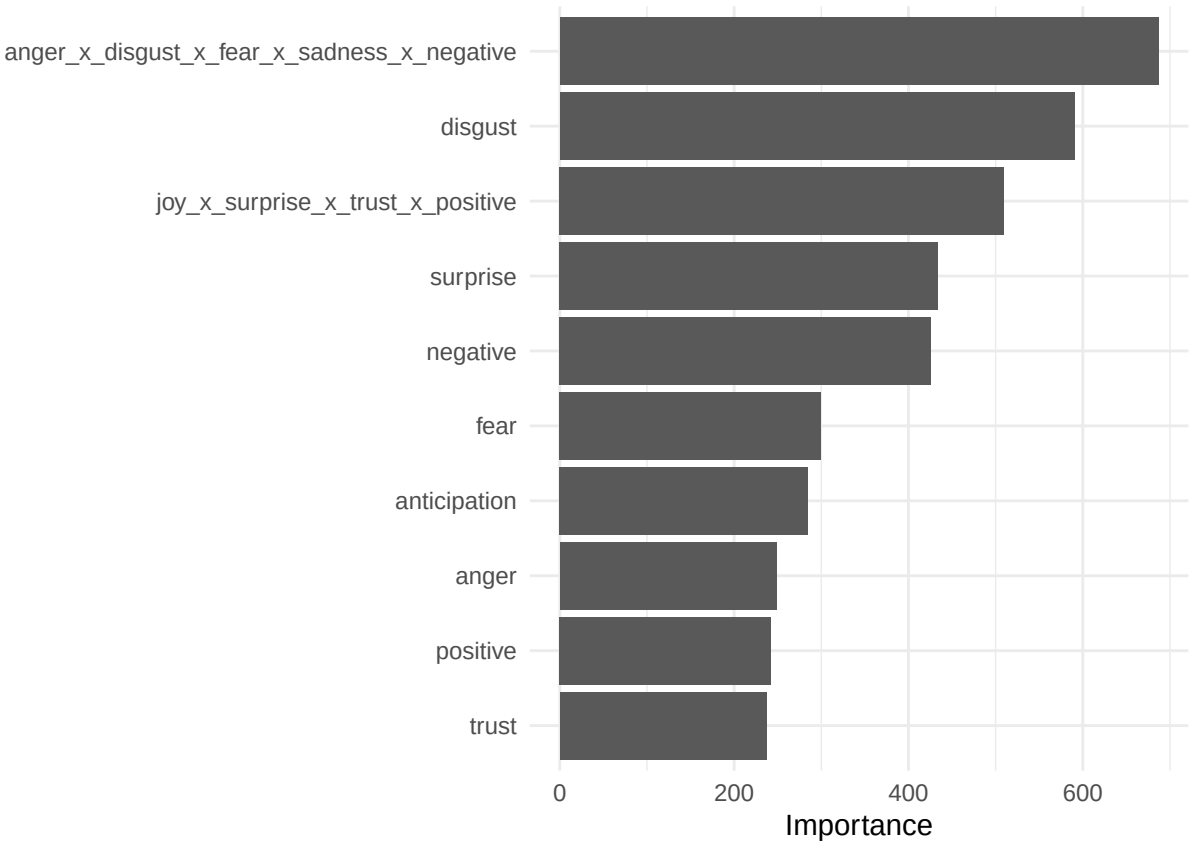
```
## Warning: No value of 'metric' was given; metric 'roc_auc' will be used.
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     2   500    10 roc_auc binary     0.724     5 0.00328 Preprocessor1_Model0~
```

```
# Finalize workflow and fit the final model to training set
rf_final <- finalize_workflow(rf_class_wf, best_rf)
rf_final <- fit(rf_final, news_train)
rf_final_fit <- fit(rf_final, data=news_train)
```

We can also create a Variable Importance Plot (VIP), which provides a list of the most significant variables according to the mean decrease in Gini index, which is determined by measuring the the quality of the split of a node on a variable (feature).

```
# VIP PLOT
rf_final %>% extract_fit_parsnip() %>%
  vip() +
  theme_minimal()
```



We observe that our most significant variable/variable interaction of our random forest model is the negative emotional sentiment, i.e., the interaction among `anger`, `disgust`, `fear`, `sadness`, and `negative`.

Gradient Boosted Trees

In our Gradient Boosted Trees model, we tuned three different parameters:

(1) `mtry`, or the number of variables we randomly sample as candidates at each split.

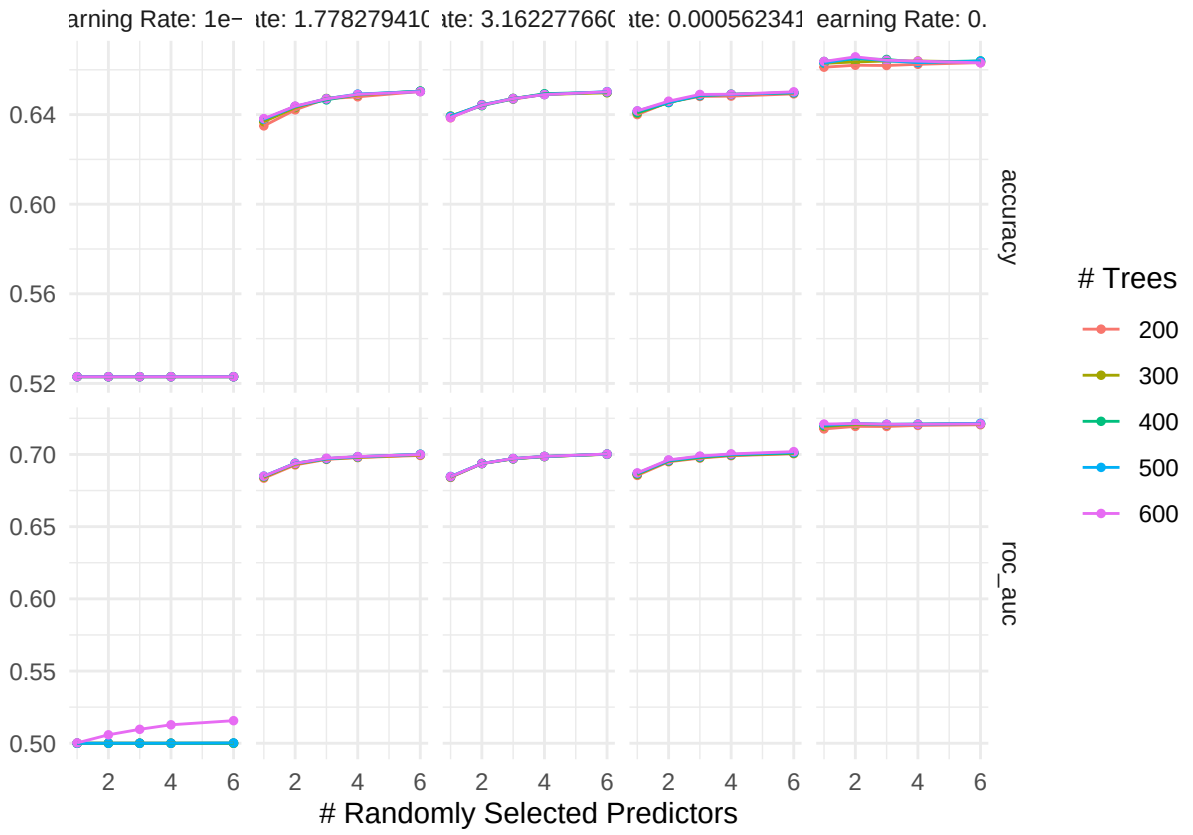
(2) `trees`, or the number of decision trees to be built in the random forest model

(3) `learn_rate`, or the rate of how fast the model learns

We set our `mtry` within the range of 1 and 6, `trees` within the range of 200 and 600, and `learn_rate` within the range of -10 and -1. We fit 5 `levels` to our model.

We observe from our plot that generally as the number of predictors increases, the ROC AUC value increases. Additionally, as the number of trees increases, the ROC AUC value also increases. It appears that our most accurate model has 500 `trees`, `mtry` = 6 and `learn_rate` = 6 with a ROC AUC of 0.7215408.

```
autoplot(tune_bt_class) + theme_minimal()
```




```
show_best(tune_bt_class, n = 1)
```

```
## Warning: No value of ‘metric‘ was given; metric ’roc_auc’ will be used.
```

```
## # A tibble: 1 x 9
##    mtry trees learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     6   500        0.1 roc_auc binary     0.722     5 0.00336 Preprocessor1_M~
```

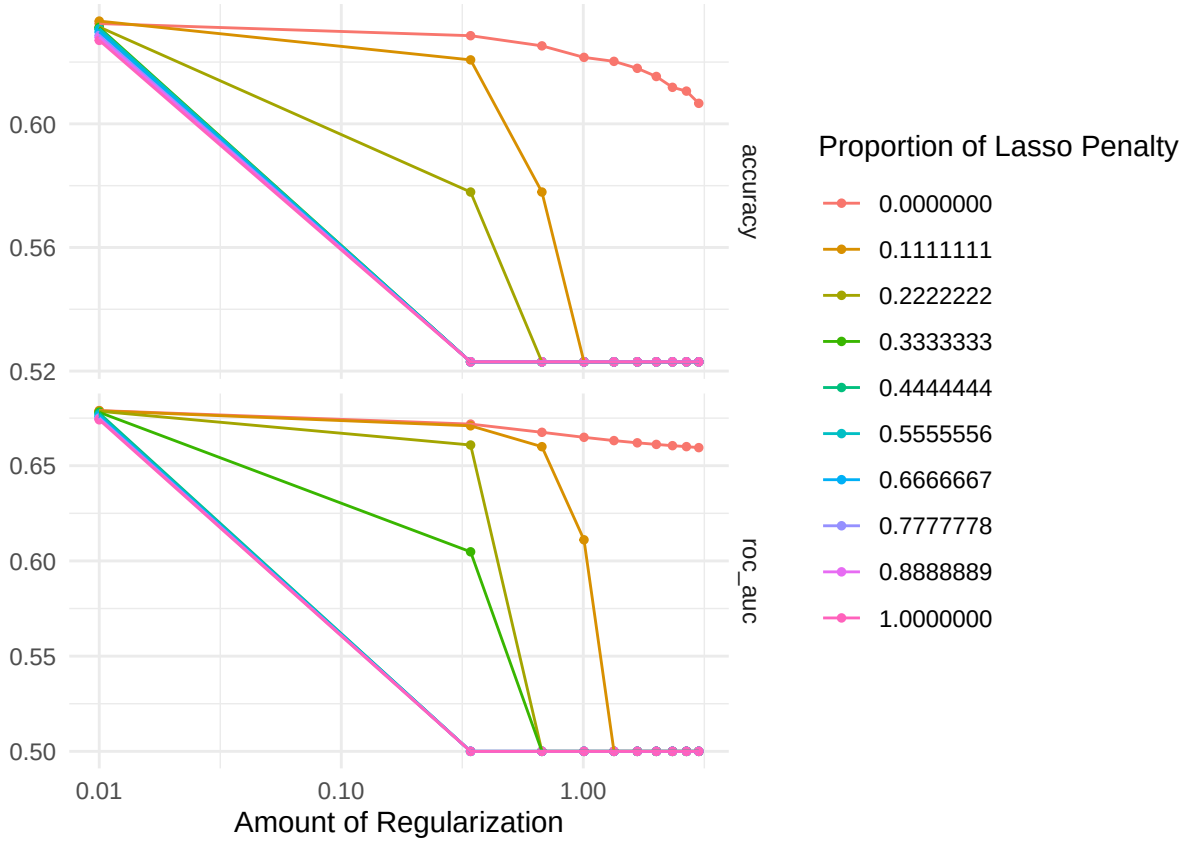
```
best_bt_class <- select_best(tune_bt_class, metric = "roc_auc")

final_bt_model <- finalize_workflow(bt_class_wf, best_bt_class)
final_bt_model <- fit(final_bt_model, news_train)
final_bt_fit <- fit(final_bt_model, data=news_train)
```

Elastic Net Regression

In our elastic net regression, we tuned `penalty` and `mixture` using the `multinom_reg()` with the `glmnet` engine. After setting up the model and workflow, we created a regular grid for `penalty` and `mixture` with 10 levels each: letting `penalty` range from 0.01 to 3 and ‘mixture range from 0 to 1.

We observe that lower values of `mixture` and `penalty` produce higher ROC AUC values. Lower proportions of the lasso penalty correspond to higher ROC AUC values. The `penalty` also produces higher ROC AUC values because regularization increases. Ultimately, our most optimal model has a `penalty` of 0.01 and `mixture` of 0 with a ROC AUC value of 0.6789722.

```
autoplot(en_tune) + theme_minimal()
```




```
best_en <- select_best(en_tune, metric = "roc_auc")
show_best(en_tune, n=1)
```

```
## Warning: No value of ‘metric‘ was given; metric ’roc_auc’ will be used.
```

```
## # A tibble: 1 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    0.01       0 roc_auc binary     0.679     5 0.00164 Preprocessor1_Model001
```

```
final_en_model <- finalize_workflow(en_workflow_news, best_en)
final_en_model <- fit(final_en_model, news_train)
final_en_fit <- fit(final_en_model, data=news_train)
```

Model Accuracy

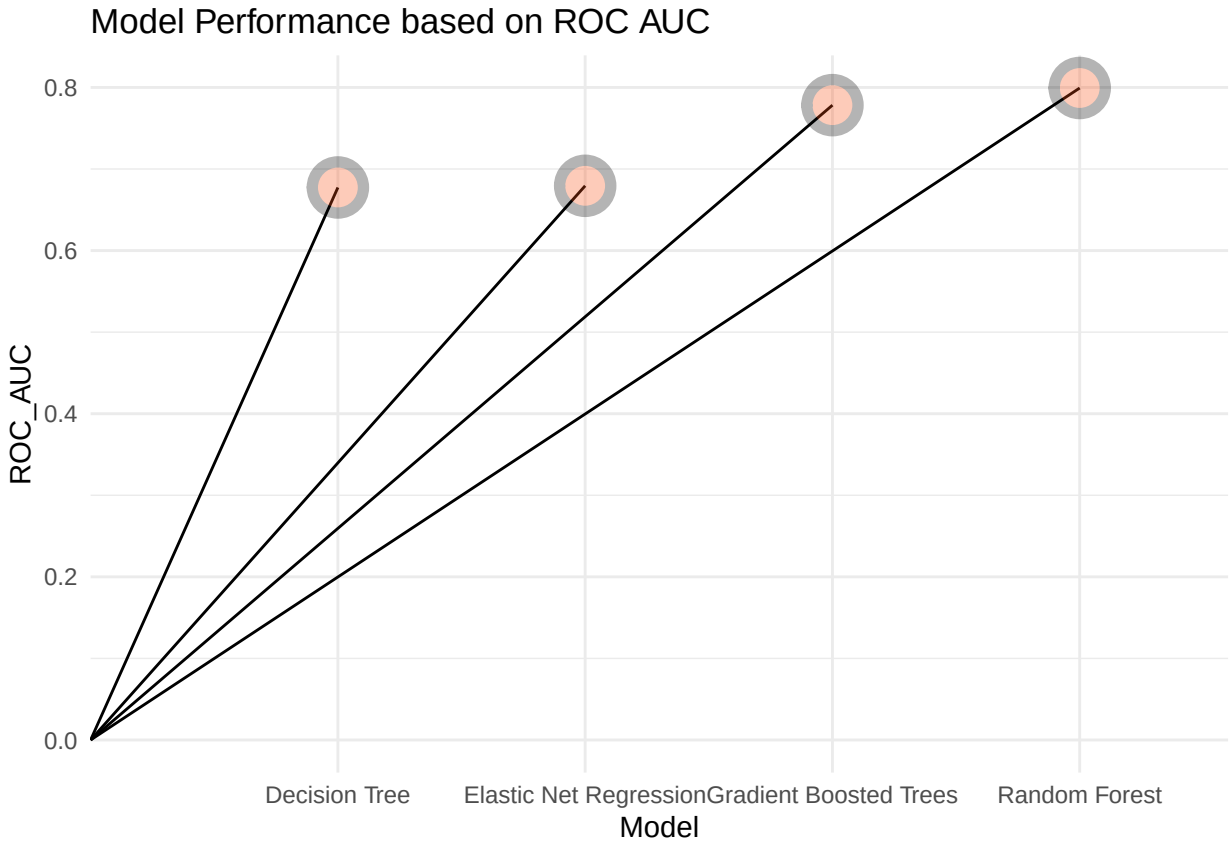
Using the `augment()` function, we can model our fit against our training set and estimate our ROC AUC value. We can create a tibble containing all the ROC AUC estimates of all four models.

```
# DECISION TREE ROC AUC ####
decision_tree_roc <- augment(tree_final_fit, new_data=news_train) %>%
  roc_auc(truth=truth_value, .pred_fake) %>%
  select(.estimate)

# RANDOM FOREST ####
random_forest_roc <- augment(rf_final_fit, new_data = news_train) %>%
  roc_auc(truth=truth_value, .pred_fake) %>%
  select(.estimate)

# GRADIENT BOOSTED TREES #####
bt_roc <- augment(final_bt_model, new_data = news_train) %>%
  roc_auc(truth=truth_value, .pred_fake) %>%
  select(.estimate)


# ELASTIC NET REGRESSION
en_roc <- augment(final_en_model, new_data = news_train) %>%
  roc_auc(truth=truth_value, .pred_fake) %>%
  select(.estimate)


news_mod_names <- c("Decision Tree",
                    "Random Forest",
                    "Gradient Boosted Trees",
                    "Elastic Net Regression")

all_news_roc <- c(decision_tree_roc$.estimate,
                  random_forest_roc$.estimate,
                  bt_roc$.estimate,
                  en_roc$.estimate)

# Create tibble with model and ROC AUC value of optimal model
news_result <- tibble(Model = news_mod_names,
                      ROC_AUC = all_news_roc) %>% arrange(desc(ROC_AUC))
```

We observe that our Random Forest model performed the best overall with a ROC AUC score of 0.7992152, followed by the Gradient Boosted Tree with a ROC AUC score of 0.7789864. We consider these metrics as acceptable discrimination with the first score approximating 0.8, which is considered excellent discrimination. These metrics are only fitted on the training model, so we need to make sure that our models perform on our testing data. We will be moving forward with the Random Forest model, but explore the Gradient Boosted Tree model's performance on the testing data as well. Let's now find the models' true performances on our testing set.

Another way to visually inspect and compare the ROC AUC values for each fitted model is a lollipop plot, which shows the relationship between a numeric (ROC AUC) and categorical variable (Types of Model).

```
news_lollipop_plot <- ggplot(news_result, aes(x = Model, y = ROC_AUC)) +
  geom_segment(aes(x = Model, xend = 0, y = ROC_AUC, yend = 0)) +
  geom_point(size = 7, color = "black", fill = "#FB4F14", alpha = 0.3, shape = 21, stroke = 3) +
  labs(title = "Model Performance based on ROC AUC") +
  theme_minimal()

news_lollipop_plot
```




```
# save(news_result, file = "news_result.rda")
load("news_result.rda")
news_result
```

```
## # A tibble: 4 x 2
##   Model                   ROC_AUC
##   <chr>                     <dbl>
## 1 Random Forest             0.799
## 2 Gradient Boosted Trees    0.779
## 3 Elastic Net Regression    0.679
## 4 Decision Tree             0.677
```

Results From Our Best Models

Let's proceed to analyze the true results of our best fitted model, i.e. a random forest model. We will also explore our second best model and analyze the results of our Gradient Boosted Tree. The Gradient Boosted Tree model is nevertheless an incredibly powerful algorithm that is worth exploring on our testing set.

Random Forest

The random forest model performed the best out of our fitted models. Let's now examine the strength of this model with data the model has not seen. After all, the high ROC AUC score in the tibble represents the model's capability to predict the truth value of the news headline using the same data it was originally trained on.

Best Model: Random Forest 1017

Random forest model #1017 performed the best overall from all the random forest models. This model exceeded all other prediction models we performed on. We can view the model's output, ROC AUC value, and tuned parameters below:

```
show_best(rf_tune, metric = "roc_auc", n=1)
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     2   500    10 roc_auc binary     0.724     5 0.00328 Preprocessor1_Model0~
```

Gradient Boosted Tree

The Gradient Boosted Tree performed the second best out of our fitted models. We can examine the strength of this model by observing its computed ROC AUC value on the training dataset below.

```
show_best(tune_bt_class, metric = "roc_auc", n=1)
```

```
## # A tibble: 1 x 9
##    mtry trees learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     6   500        0.1 roc_auc binary     0.722     5 0.00336 Preprocessor1_M~
```

After observing our optimal (and second optimal) model, we can fit our testing data to the model and assess the actual performance in predicting whether a news headline is true or fake.

Final ROC AUC Results

Random Forest 1017

`ROC AUC`

The resulting ROC AUC value after fitting our testing dataset is 0.7388494. We consider these metrics as acceptable discrimination, as the AUC value ranges between 0.7 and 0.8.

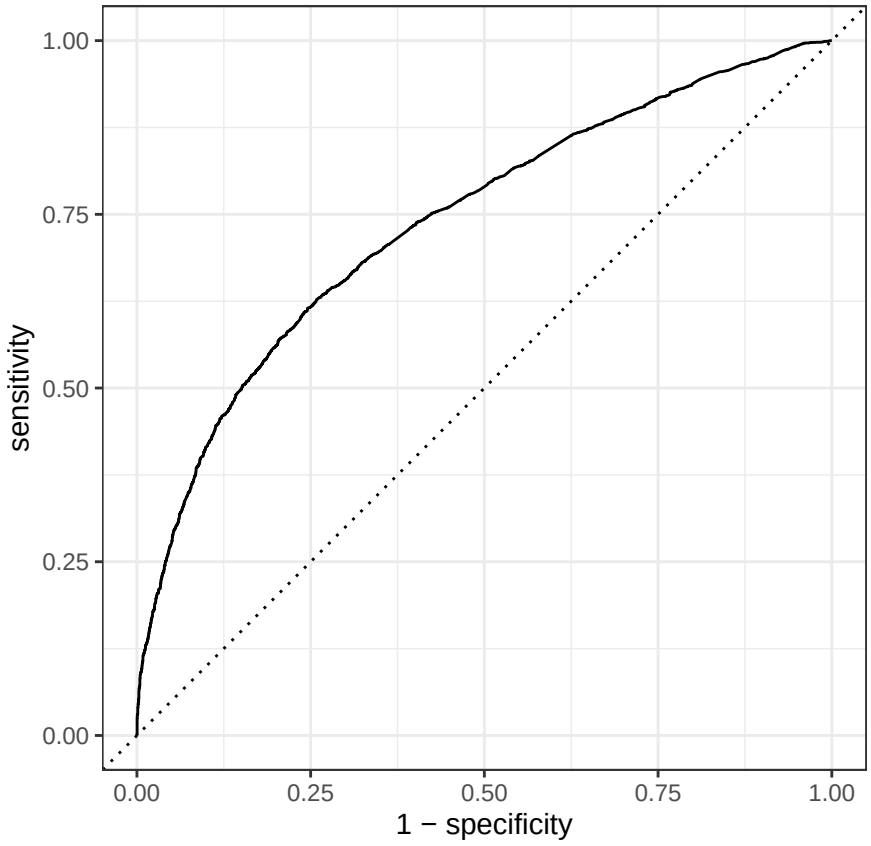
```
random_forest_roc_test <- augment(rf_final_fit, new_data = news_test) %>%
  roc_auc(truth=truth_value, .pred_fake) %>%
  select(.estimate)
random_forest_roc_test
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1     0.739
```

ROC Curve

To visualize the ROC AUC score, it is helpful to plot a ROC curve. The closer the curve approximates to the top left corner, the greater the model's ROC AUC value is and the better the model performs.

```
augment(rf_final_fit, new_data = news_test) %>%
  roc_curve(truth_value, .pred_fake) %>%
  autoplot()
```



Based on the ROC curve, we observe that the plot approximates to the top left. While the plot does not perfectly resemble a perfect top-left right angle, the curve confirms our computed AUC curve.

Gradient Boosted Tree

ROC AUC

The resulting ROC AUC value after fitting our testing dataset to a Gradient Boosted Tree is 0.7356351. We consider these metrics as acceptable discrimination, as the AUC value ranges between 0.7 and 0.8. However, the value falls short of the ROC AUC value of our Random Forest model. This makes sense considering that this model performs secondary to our Random Forest model, according to the computed ROC AUC value on our training set.

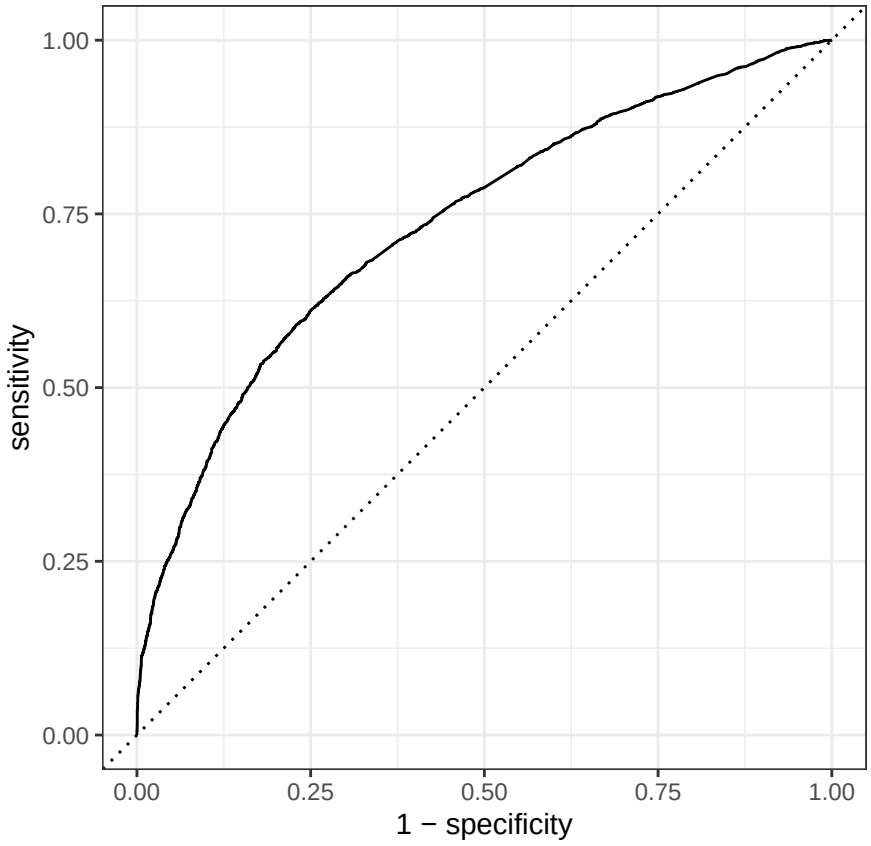
```
bt_roc_test <- augment(final_bt_model, new_data = news_test) %>%
  roc_auc(truth=truth_value, .pred_fake) %>%
  select(.estimate)
bt_roc_test
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1     0.737
```

ROC Curve

Based on the ROC curve, we observe that the plot approximates to the top left. The curve confirms our computed AUC curve.

```
augment(final_bt_fit, new_data = news_test) %>%
  roc_curve(truth_value, .pred_fake) %>%
  autoplot()
```



Putting the Model to the Test

Finally, we can put our Random Forest model to the test and observe whether our model accurately predicts a headline to be true or fake.

Fake News Example

Let's generate a random news headline observation, which we identify as "fake." We observe that the tidied title text is:

"man pen benghazi mom gop convention speech startle degree flip hillary"

The sample observation has the following emotional/sentiment metrics:

- anger = 0
- anticipation = 0
- disgust = 0
- fear = 1
- joy = 0
- sadness = 0
- surprise = 1
- trust = 0
- negative = 1
- positive = 3

```
set.seed(1)
sample <- news_test %>% sample_n(1)
sample
```

```
##                                                                 title
## 1 man pen benghazi mom gop convention speech startle degree flip hillary
##
## 1 The man who wrote the speech for Patricia Smith, a.k.a.  Benghazi Mom,  for the GOP convention has
##   subject       date truth_value Category anger anticipation disgust fear joy
## 1    News 2016-08-17        fake        0     0            0       0    1   0
##   sadness surprise trust negative positive
## 1       0        1     0        1        3
```

Based on our prediction, the model correctly predicts that the news headline is "fake."

```
predict(rf_final_fit, sample, type="class")
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 fake
```

True News Sample

Now, we re-generate a random news headline observation, which we identify as "true." We observe that the tidied title text is:

"republican senator collins say talk tax bill productive"

The sample observation has the following emotional/sentiment metrics:

- anger = 0

- anticipation = 0
- disgust = 0
- fear = 0
- joy = 0
- sadness = 1
- surprise = 0
- trust = 0
- negative = 1
- positive = 2

```
set.seed(2)
sample2 <- news_test %>% sample_n(1)
sample2
```

```
##                                                    title
## 1 republican senator collins say talk tax bill productive
##
## 1 WASHINGTON (Reuters) - Republican U.S. Senator Susan Collins, who has demanded changes to a Republi
##        subject       date truth_value Category anger anticipation disgust fear
## 1 politicsNews 2017-11-28        true        1     0            0       0    0
##   joy sadness surprise trust negative positive
## 1   0       1        0     0        1        2
```

```
predict(rf_final_fit, sample2, type="class")
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 true
```

Based on our prediction, the model correctly predicts that the news headline is "true."

Disclaimer

The fake news detection model is not 100% accurate. We do not expect that *all* sample headlines would be accurately predicted. Indeed, we are only working with emotion/sentiment as the sole predictor of a given headline's truth value. Perhaps if we were to consider the news article text rather than the headline or include more predictors (e.g., commonality of words), our model accuracy would improve immensely. This makes sense as we introduce more data in our existing variable (more text) and/or more predictor variables. However, given our software's limited computing power, we considered a smaller region of data — albeit with over 40,000 observations. Ultimately, this model is well-versed in providing a preliminary analysis of how fake news actors deliberately induce emotion in forms of deception.

Conclusion

Overall, the Random Forest model provided the most accurate prediction of whether a given news headline is fake or true. This model was not entirely perfect. We may consider further extensions that may generate a more improved predictive model.

For instance, we may examine the application of a Naive Bayes classifier, Support Vector Machine (SVM), or neural network. These models may require greater bandwidth, complexity, and versatility in other programming languages like Python. It may be worthwhile to explore several deep learning models and fit different architectures of neural networks, as we train the model to accurately predict the truth value of a text.

As far as the number of observations goes, we have a sufficient sample size. So, rather than increasing the size of our observation, we can take care to improve the complexity of our predictors laterally. By evaluating the news article text rather than the headline alone, we could see an increase in the complexity of terms available. We could also explore various predictors on top of the frequency of emotionally-charged words (e.g., commonality of words) or classify our predictors based on different kinds of word-emotion lexicons. Ultimately, there are many ways we can explore the accuracy of our model performance and maximize the potential that our preliminary report provides. These considerations are compelling as we explore ways to quell false and misleading information in the media.

References

Ahmed H, Traore I, Saad S. "Detecting opinion spams and fake news using text classification", Journal of Security and Privacy, Volume 1, Issue 1, Wiley, January/February 2018.

Ahmed H, Traore I, Saad S. (2017) "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127- 138).

Republican National Committee [GOP]. (2023, April 25). Beat Biden [Video]. YouTube. https://www.youtube.com/watch?v=kLMMxgtxQ1Y.

"An update on our approach to US election misinformation," YouTube Official Blog (2023). https://blog.youtube/inside-youtube/us-election-misinformation-update-2023/

Barthel, Michael, Amy Mitchell, & Jesse Holcomb (2016). "Many Americans Believe Fake News Is Sowing Confusion." Pew Research Center. https://www.pewresearch.org/journalism/2016/12/15/many-americans-believe-fake-news-is-sowing-confusion/.

"Foreign Threats to the 2020 US Federal Election." National Intelligence Council (2021). https://www.dni.gov/files/ODNI/documents/assessments/ICA-declass-16MAR21.pdf.

Meta Business Help Center. "About fact-checking on Facebook and Instagram." https://www.facebook.com/business/help/2593586717571940?id=673052479947730&helpref=uf_permalink.

"The Myth of Voter Fraud." Brennan Center for Justice. https://www.brennancenter.org/issues/ensure-every-american-can-vote/vote-suppression/myth-voter-fraud.

Twitter Help Center. "How we address misinformation on Twitter." https://help.twitter.com/en/resources/addressing-misleading-info.

"What is Fake News?" Center for Information Technology and Society at UC Santa Barbara. https://www.cits.ucsb.edu/fake-news/what-is-fake-news.

Main Dataset: ISOT Fake News Dataset. University of Victoria, Information Security and Object Technology (ISOT) Research Group. (2016-2017). https://onlineacademiccommunity.uvic.ca/isot/2022/11/27/fake-news-detection-datasets/.

Model Appendix

```
# DECISION TREE
tree_spec <- decision_tree(cost_complexity = tune())  %>%
  set_engine("rpart") %>%
  set_mode("classification")

tree_wf <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(news_recipe)

param_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)

# RANDOM FOREST MODEL
rf_class_news <- rand_forest(mtry = tune(),
                             trees = tune(),
                             min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

# Set up a random forest workflow with recipe 'news_recipe'
rf_class_wf <- workflow() %>%
  add_model(rf_class_news) %>%
  add_recipe(news_recipe)

# Create a regular grid with 1 level each
rf_grid <- grid_regular(mtry(range = c(1, 6)),
  trees(range = c(200, 600)),
  min_n(range = c(10, 20)),
  levels = 5)

# BOOSTED TREES MODEL
bt_class_spec <- boost_tree(mtry = tune(),
                            trees = tune(),
                            learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

bt_class_wf <- workflow() %>%
  add_model(bt_class_spec) %>%
  add_recipe(news_recipe)

# SET UP GRID
bt_grid <- grid_regular(mtry(range = c(1, 6)),
                        trees(range = c(200, 600)),
                        learn_rate(range = c(-10, -1)),
                        levels = 5)

# ELASTIC NET REGRESSION
# Fitting and tuning an elastic net
en_news <- multinom_reg(mixture=tune(), penalty=tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
```

```r
# Set up model and workflow
en_workflow_news <- workflow() %>%
  add_recipe(news_recipe) %>%
  add_model(en_news)

# Create a regular grid for 'penalty' and 'mixture'
en_grid <- grid_regular(penalty(range=c(0.01,3), trans=identity_trans()),
  mixture(range=c(0,1)), levels=10)
```

```r
### TUNING

# DECISION TREE
tune_tree <- tune_grid(
  tree_wf,
  resamples = news_folds,
  grid = param_grid
)

# RANDOM FOREST MODEL

# Fit all models to your folded data using 'tune_grid()'
rf_tune <- tune_grid(
  rf_class_wf,
  resamples = news_folds,
  grid = rf_grid
)


# BOOSTED TREES MODEL
# TUNE GRID
tune_bt_class <- tune_grid(
  bt_class_wf,
  resamples = news_folds,
  grid = bt_grid
)

# ELASTIC NET REGRESSION

# Fit model to folded data using 'tune_grid()
en_tune <- tune_grid(
  en_workflow_news,
  resamples = news_folds,
  grid = en_grid
)

best_en <- select_best(en_tune, metric = "roc_auc")

final_en_model <- finalize_workflow(en_workflow_news, best_en)
final_en_model <- fit(final_en_model, news_train)
final_en_fit <- fit(final_en_model, data=news_train)
```