CLASSIFYING POLITICAL ARTICLES FROM FOX NEWS AND CNN USING RANDOM FORESTS

by

MICHAEL INGRAM

B.S. STATISTICS, COLORADO STATE UNIVERSITY, 2017

A report submitted to the

Department of Mathematical and Statistical Sciences

at the University of Colorado Denver in partial fulfillment

of the requirements for the degree of

Master of Science

Statistics

Spring 2020

This project for the Master of Science degree by

Michael Ingram

has been approved for the

Statistics Program

by

Dr. Stephanie Santorico, Chair

Dr. Audrey Hendricks

Dr. Burton Simon

Michael Ingram

Classifying Political Articles from Fox News and CNN Using Random Forests

I. Abstract

Media bias in national news sources has been a hot topic in recent years, especially with regard to the political sphere. While national news sources are supposed to be unbiased sources of news, many people claim this is not the case. This project will look at two different national media sources, Fox News and CNN, which were chosen based on popular beliefs that the former has a conservative political bias with the latter having a liberal political bias. The two news sources were web scraped using Python's "newspaper" package. Ten articles from each news sources' political page were scraped per day during the weeks of September 16, 2019 – October 4, 2019, which amounted to 300 total articles being scraped with 150 articles from each news source. Natural language processing was then used to remove stop words, convert text to lowercase, remove punctuation, tokenize, perform lemmatization, and vectorize using both Term Frequency – Inverse Document Frequency (TF-IDF) and Bag of Words methods. Finally, the random forest algorithm and logistic regression were used to classify an article's news source based on the article's text and titles. The random forest model using TF-IDF classified the article's text with the highest accuracy of all models. While the models used for classification of an article's title were lower than their text counterparts, the random forest model using TF-IDF still had the highest accuracy of all the other models used for article title classification. The top five tokens with the highest feature importance from the most accurate model, listed from highest to lowest, were "click", "president donald", "donald", "contributed" and "report".

Table of Contents

II. Introduction

Natural language processing has been a hot area of research in recent years due to the increased availability of big data and machine learning methods. Natural language processing allows for computers to interpret and analyze human speech. Having computers understand human language has become increasingly important for both developments in AI and for analyzing new data. This project will focus on an application of natural language processing and the random forest algorithm to news data.

Fake news as well as media bias has been a controversial topic with the recent presidential administration. This project looks at two different news sources, Fox News and CNN, to see how well a random forest algorithm can classify the news sources based on the text, authors, keywords, and titles of the article. These two news sources were chosen based on popular belief that the news source, Fox News, has a bias towards the republican political party and while CNN has a bias towards the democratic political party. The web scraped articles were specifically related to politics and were chosen from the political section of each news sources' website.

News source classification is not a new idea. Specifically related to political news, "Classifying the Political Leaning of News Articles and User Votes" published in 2011 by Zhou D et. al, used semi-supervised learning methods to classify articles as liberal or conservative as well as whether liberal users vote for liberal articles more often than conservative users and vice versa. Their best algorithm performed extremely well generating 99.6% accuracy. This research and many others related to news sources fall under the sentiment analysis side of natural language processing. "Sentiment Analysis in the News" published in 2013 by Balahur A, et al., specifically looked at news opinion mining as well as "ASNA: An intelligent agent for retrieving and classifying news on the basis of emotion-affinity" published in 2007 by Al Masum S, et al. In the ASNA paper, the researchers created a news aggregator that fetches news employing several RSS news feeds and auto-categorizes news according to affect sensitivity.

The other area of news classification research has been related to accuracy, or "fake news". "Social Media and Fake News" published by Moretti E in 2017 in the American Economic Review, looked at false news stories that were shared on social media leading up to the 2016 election. They found only 14% of Americans considered social media their primary election news source and for a single fake news article to have changed the election it would need 36 times the persuasive effect of a television campaign ad. Research has also been done on the natural language side of news accuracy. One example, "Truth of Varying Shades: Analyzing Language in Fake News and Political Fact Checking," by Rashkin H, et al. (2017), performed an analytical study of language in news articles in the context of political fact checking and fake news detection where news articles were compared with those specifically written as hoaxes and satires.

The research presented here differs from previous works by looking at two specific national news sources who many consider to have a liberal and conservative bias. The goal of this research project will be to see how accurately the supervised learning algorithm, random forests, can classify both the article text and titles to which news source they originate. The other goal will be to see which words contributes the most to the prediction. National news media sources are supposed to be unbiased so intuition would assume it would be hard to classify the media from the two different news sources if they were truly unbiased. In addition, the results from the top predicting words could give surprising results if they end being something related to a specific political party, political figure, or business.

III. Methods

This project utilized multiple methods. The methods are outlined briefly below and then discussed more thoroughly in the following sections. The project was done in Python v 3.6.9 and code and packages mentioned in the following sections refers to this version of Python.

i. Web Scraping

The data for the project was collected by web scraping Foxnews.com/politics and cnn.com/politics. Web scrapers are an Application Programming Interface (API) developed to extract data from websites. The scrapers fetch (download) the web page and extract information from it. The web scraper used for this project is Newspaper3k developed by Lucas Ou-Yang. The package was specifically designed for web scraping news articles and allows the user to build classes of multiple articles or scrape specific articles. Scraping the articles allows the user to parse out useful information from the article such as text, title, author, url, publish date, image, movie, etc.

In this project, articles were chosen individually, and in a way to be as unbiased as possible. Ten articles a day were chosen to be web scraped from both Fox and CNN. The articles were chosen from the political tabs of each news source. To make it as unbiased as possible, articles were chosen from the top story and then continuing on downward on Fox. Since CNN's political tab layout was more horizontal, the top story was chosen then more articles were chosen from left to right. Ten articles were chosen each day because on a slow news day, ten articles included almost all of the new articles posted to the political tabs of each news source. Video only articles were skipped in the choosing of articles to scrape. Articles were gathered Monday – Friday for three weeks starting from September 16, 2019 to October 4, 2019. This resulted in 100 articles a week, with 50 from each source for a total of 300 articles, with 150 from each source.

Once the articles were fetched and parsed a data frame was built that contained the author(s), title, text, keywords, URL and publish date. While there is a lot of interesting analyses that could be performed with this information, only the text and title were used in the analysis since performing classification on author(s) or URL would give an obvious result and not provide any new insight. However, for more information on the algorithm used to parse keywords from the article see the documentation section.

ii. Natural Language Processing

Natural language processing is a subfield of computer science, artificial intelligence and linguistics that is concerned with how computers can process natural human languages. Specifically, natural language processing was used to get raw text data from news articles into numerical vectors the computer can understand. This is a multi-step process and the methodology used is defined below.

A. Removing Punctuation and Lowercase Conversion with Regular Expressions

The first natural language processing method performed on the text and titles is the removal of punctuation, new lines, and special characters. Then the text and titles are converted to lowercase. This is done because the punctuation, new lines, and special characters (such as ! , . ‘ $ @ \n & etc.) are extra noise in the data and do not provide any substantial information. Similarly, words with capital such as Fox or FOX have the same meaning as fox and so every word in the titles and article text are set to lowercase.

This was done using regular expressions which is also referred to as regex. Regex is a sequence of characters that form a search pattern. This is done in Python using the *re* package. A regular expression was used to search for punctuation, new lines, and special characters. When one was found it was replaced with an empty character or, in the case of new lines, white space. The most common functions used in the re package of Python are as follows:

1. Findall: Returns a list containing all matches
2. Search: Returns a Match Object if there is a match anywhere in the string
3. Split: Returns a list where the string has been split at each match
4. Sub: Replaces one or many matches with a string

To remove the punctuation, new lines and special characters from the article titles and text, the sub function is used. The regular expression used was r ‘[^ \w \s]’ . The r and first quotation mark indicates to Python this is a raw string inside the quotes. The ^ inside the brackets in a regular expression indicate the regular expression should match with characters except those inside the

brackets or, in other words, it indicates to match with the complement of the characters inside the bracket. The \w is a regular expression symbol for any words characters a-Z as well as 0-9 and the underscore characters. The \s symbol is a regular expression symbol for a white space character. The full regular expression therefore indicates to replace any character that is not a-Z, 0-9, an underscore or a white space with an empty character. To remove new lines another regular expression was written that uses '\n' to indicate new lines and the new line was replaced with a white space. Regular expressions can get as simple or as complicated as needed. For an in-depth guide into regular expressions see the regular expressions documentation in section VII Code.

B. Removal of Stop Words

Stop words are the most common words that are removed because they do not provide any useful information. Some common examples in English would be words like a, the, I, he, they, etc. There is no universal list of stop words and they are typically chosen specifically for the needs of each project. During this analysis, the stop words used, in addition to the package defaults, were fox, cnn, foxs, cnns, and news. These were included as stop words because classification of fox and cnn political articles becomes trivial if the word fox or cnn is in the article itself. The other remaining stop words come from the default stop words of the spaCy package. SpaCy is a package in Python specifically built for natural language processing. Including those words, a total of 335 stop words were removed from the article text and titles. The list of 335 stop words used is shown in the python code which can be accessed in section VII.

C. Lemmatization

Lemmatization is the process of reducing a word to its base form or lemma. This is done so words that have the same meaning, but different conjugation, are reduced to the same word. For example, watch, watching, and watched all have the same general meaning behind the word. The lemmatization process reduces these words to the word watch. What is unique about lemmatization is

the method uses the morphological structure, part of speech and context of a sentence to reduce a word to its base form. This is not a trivial process and research is ongoing in improving the accuracy of lemmatization algorithms.

Commonly confused with lemmatization is stemming. Stemming is where a word is reduced to its base form through removing the suffix or prefix of a word. Table 1 below shows an example of the difference between stemming and lemmatization for the word study.

Table 1. An example of stemming versus lemmatization

Stemming		
Form	**Suffix**	**Stem**
Studies	-es	Studi
Studying	-ing	Study
Lemmatization		
Form	**Morphological Information**	**Lemma**
Studies	Third Person, Present Tense, Singular Number of the verb study	Study
Studying	Gerund of the verb study	Study

Lemmatization is usually preferred because of its improved accuracy. However, it comes at the cost of increased computing power. Since this project did not have a large amount of data, lemmatization was used based on the spaCy package. Both article titles and text were lemmatized.

D. Converting from Words to Vectors

In order to use machine learning classification algorithms, the text needs to be transformed in some way to numerical data. While there are increasingly more complex ways to do this, two common

methods were used for this project: the Bag of Words method and the Term Frequency – Inverse Document Frequency method also known as TF-IDF.

a. Bag of Word – Count Vectorizer

Bag of words is a method that creates a vocabulary of the unique words in a document. It is a collection of words that represent the word count of all the words in a document and disregards the order and context in which the words appear. This is best explained through an example originally from Dr. Jason Brownlee.

The following sentences are from a Tale of Two Cities by Charles Dickens:

"It was the best of times.

It was the worst of times.

It was the age of wisdom.

It was the age of foolishness."

In this example treat each line as a separate document and the four lines as the entire corpus of text.

Now we can make a vocabulary of all the unique words present in the corpus:

- "it"
- "was"
- "the"
- "best"
- "of"
- "times"
- "worst"
- "age"
- "wisdom"
- "foolishness"

There are 10 total unique words present in the corpus. Now the sentences can be represented as vectors. This is done by giving the word in the vocabulary a Boolean value of 1 if the word is present in the sentence or 0 if it is not. For the first sentence:

- “it” = 1
- “was” = 1
- “the” = 1
- “best” = 1
- “of” = 1
- “times” = 1
- “worst” = 0
- “age” = 0
- “wisdom” = 0
- “foolishness” = 0

As vectors the four sentences would look like:

“it was the best of times” = (1, 1, 1, 1, 1, 1, 0, 0, 0, 0)

“it was the worst of times” = (1, 1, 1, 0, 1, 1, 1, 0, 0, 0)

“it was the age of wisdom” = (1, 1, 1, 0, 1, 0, 0, 1, 1, 0)

“it was the age of foolishness” = (1, 1, 1, 0, 1, 0, 0, 1, 0, 1)

While a relatively simple approach, the obvious disadvantage to this methodology is that for a corpus with a very large vocabulary the resulting vectors are very sparse, i.e., contain a lot of zeros. Large empty vectors require a lot of memory in the computer and increase computational time, especially for algorithms that are already computationally expensive. This comes back to the reason behind why natural language processing was done prior to codifying the text. Removing stop words, lower case conversion, and removal of punctuation all reduce the amount of unique words that do not contain

useful information in the bag of words vocabulary. The example above only used Boolean values for simplicity, but the words can be scored with counts or frequencies. Counts would be where words are codified by the number of times they occur in the document, or as in the example above, in the sentence. Frequencies would be a proportion of the number of times the word occurred out of all the documents. In the example above, "it" would have a frequency of 4/4 =1 and "foolishness" would have a frequency of 1/4.

b. Term Frequency – Inverse Document Frequency (TF-IDF)

One of the problems with the bag of words method is that words that appear frequently have more weight than those that are less frequent but are more domain specific and have more informational content. The solution to this is to weight words by how often they appear in all documents as is done with Term Frequency – Inverse Document Frequency (TF-IDF) methodology. "Term frequency" stands for how often the word occurs in the document whereas "inverse document frequency" or IDF represents how rare the word is across all documents. The IDF of a rare token will be high while the IDF of a common token will be low. For a term t:

$$TF - IDF = TF * IDF$$

$$TF = \frac{Number\ of\ times\ term\ t\ appears\ in\ document}{Total\ number\ of\ terms\ in\ document}$$

$$IDF = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it}\right)$$

An example from tf-df.com is helpful for understanding. Suppose there is a document containing 100 words where the word cat appears 3 times. The term frequency for cat is then (3/100) = 0.03. Suppose the total corpus is 10,000,000 documents where the word cat appears in 1,000 of these documents. Then the inverse document frequency would be log (10,000,000 / 1,000) = 4. So the TF-IDF for cat would be 0.03 *4 = 0.12. The log dampens the effects of the inverse document frequency for a very large

corpus. The general idea behind TF-IDF is to give more weight to words that are relevant by highly weighting words that appear frequently in a single document AND are rare in the entire corpus.

c. CountVectorizer() and TfidfVectorizer() Methods

Two methods were used in Python to transform the words to vectors, CountVectorizer() and TfidfVectorizer(). CountVectorizer() simply performs bag of words using word counts. TfidfVectorizer() uses TF-IDF to weight the words when transforming them. In addition to transforming the text, the vectorizer methods also perform a number of the natural language processing methods that were previous described such as lowercase conversion, removal of stop words, pre-processing, tokenizing, and more. Specifically, the parameters used for the project were ngram_range, stop words, and max features. Ngram_range is a method to specify how many N grams the method should use. N grams are how the document should be tokenized. This is where a sentence is broken into its separate words and white space is removed. The previous examples have all used 1-grams where the sentences are split into each individual word. A 2-gram would be where sentences are split into two-word tokens. A 3-gram would be three-word tokens and so on. The ngram_range parameter allows for the specification of a range of n-grams such (1,2). This means the method will use both 1 and 2 n-grams. Table 2 show an example of different n-grams below.

Table 2. Example of N-grams for n=1, 2, 3, 4

n	N-gram example
1 - Gram	“Bike”
2 - Gram	“Bike Ride”
3 - Gram	“Cool Bike Ride”
4 - Gram	“Super Cool Bike Ride”

The stop_words parameter allows for the vectorizer to remove the provided stop words from the text and titles. The stop words used were those previously declared in section ii, subsection B of the methods section. The max_features parameter specifies the maximum number of unique words to consider. When max_features parameter is set to a value x, then the x most frequent words are used from the entire corpus.

Grid search cross validation was used with the vectorizers to find the optimum ngram_range as well as max_features. The ngram_ranges that were used during grid search cross validation were (1,1), (1,2), (1,3), (2,2), and (1,4). The max_features used during the grid search cross validation were 100, 2000, and 5000. Grid search cross validation was used for other things besides optimizing the parameters of the vectorizers and will be discussed in greater detail in the cross-validation section.

iii. Classification

Two different machine learning classification algorithms were used to classify the data to its news source, either Fox News or CNN. These algorithms were logistic regression and random forest. Logistic regression was chosen as a baseline method for comparison. This algorithm is traditionally used to classify binary data. Random forest was chosen as a modern classification algorithm and because, unlike with some machine learning methods, it has the property of feature importance. Feature importance specifies tokens that are contributing the most weight to the classification. The news source was labeled as 1 for Fox and 0 for CNN. This was done because the goal of the analysis was to see how accurately the articles can be classified, in addition to which words would have the most weight in classification. The original data set of 300 articles was split into a training and test set with an 80/20% split, respectively. This resulted in 240 articles in the training set and 60 articles in the test set. The train and test split of the data was performed on the text and title data that had been cleaned with lowercase conversion, removal of punctuation and new line characters.

A. Logistic Regression

Logistic regression is a regression method commonly used for classification because its output results in values between 0 and 1 for all input values of X. The multiple logistic regression model is defined as:

$$log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \quad (1)$$

The left-hand side of the equation is known of the log-odds or logit function. This function results in an S-shaped curved that stretches between zero and one. In logistic regression, a one unit increase in X_i changes the log odds by β_i with the values of the predictors held constant. The β_i coefficients are estimated through maximum likelihood estimation. Closed form solutions for maximum likelihood estimates are not always possible but can be estimated computationally using iteratively reweighted least squares.

Logistic Regression in many ways can be thought of as an extension on linear regression. In fact, it is part of a group of methods known as generalized linear models (GLMs). Generalized linear models expand on linear regression allowing the error distributions to no longer be normal. There are 3 parts to a GLM:

1. An Exponential family of probability distributions.
2. A linear predictor.
3. A link function.

In the case of logistic regression, the link function is the logit function. This is shown as the left-hand side of the equation (1) above. The assumptions that are important to logistic regression are that the response variable should be binary (0 or 1). There should also be a linear relationship between the logit of the response and each predictor. It's also important that there little to no multicollinearity in the data.

Logistic regression was used in this project as a comparison method of classification for the random forest algorithm. This is because logistic regression is frequently used, and many people are

familiar with this method as it is a natural extension of linear regression data for a binary response. Since it is so closely related to linear regression it shares much of the same advantages, such as easy interpretability, and disadvantages, such as failing to solve non-linear problems. This has only been a brief introduction to the topic of logistic regression because it is not the main focus of this project. For more information there are many other resources on logistic regression such as *Elements of Statistical Learning* by Hastie, Tibshirani, Friedman, which covers a machine learning context, or *Categorical Data Analysis* by Agresti, which cover a classical statistics approach. However, for interpreting the results of logistic regression used as a classification method for prediction see the section iii subsection F Classification Metrics and Confusion Matrix.

B. Decision Trees

In order to understand the random forest algorithm, it is important to first discuss decision trees. Decision trees are a method that can be used for both regression and classification. They involve splitting the predictor space into different regions. To make a prediction, the mean of the region (for regression) or most common case of the region to which the observation belongs (for classification) is used. Visually, the splits can be shown as a tree like diagram, hence, the name decision tree (e.g., Figure 1)

Figure 1. Decision tree for the annual salary of baseball players using hits and years of experience

Figure 1 is an example of a decision tree from *Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani. In this example, the decision tree is used for predicting the log salary, in thousands of dollars, of baseball players based on their number of years playing and their number of hits. Notice at the splits, the conditions, if true, are the left branch and, if false, are the right branch. The numbers at the bottom of the tree are known as leaves or terminal nodes. This is a regression tree and so the terminal nodes are the mean of the region. The regions for this data are years less than 4.5, years greater than or equal to 4.5 with hits less than 117.5, and years greater than or equal to 4.5 with hits greater than or equal to 117.5. The node appearing at the split, in this case the Hits <117.5, are known as internal nodes. The algorithm for a decision tree involves two steps.

1. Divide the predictor space – that is, the set of possible values for $X_1, X_2, \dots, X_p$ – into J distinct and non-overlapping regions, $R_1, R_2, \dots, R_J$.

2. For every observation that falls into region R_j, the same prediction is made, which is simply the mean of the response values for the training observations in R_j.

For regression decision trees, the goal is to minimize the residual sums of squares (RSS). Specifically, for any j and s, the pair of half planes is defined as

$$R_1(j,s) = \{X|X_j < s\}\ and\ R_2\ (j,s) = \{X|X_j \geq s\}$$

where the goal is to find the value of j and s that minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i:x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2$$

Here, s is known as the cut point, $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j,s)$, and $\hat{y}_{R_2}$ is the mean response for the training observation in $R_2(j,s)$. The process is then repeated looking for the best predictor and cut point. The difference this time is that only the two previous split regions is considered to be split instead of the entire predictor space. The process continues until a set criterion is reached.

For classification decision trees, the algorithm is very similar but instead of using residual sums of squares, the Gini Index is used. The Gini Index is defined as follows.

$$G_m = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Where $\hat{p}_{mk}$ is the proportion of training observations in the *m*th region that are from the *k*th class. The Gini Index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or 1. A small value of the Gini Index indicates a node contains observations predominantly from a single class.

The biggest disadvantages to simple decision trees are that they typically do not have the same predictive accuracy as other regression and classification methods such as linear regression, LASSO, logistic regression, etc. The other disadvantage is the trees are not robust (James, et. Al, 2017 p 315). These disadvantages are solved by aggregating many decision trees in methods such as bagging, boosting and random forests. While bagging will be briefly explained because it is necessary to understand random forests, boosting will not be covered because it is not necessary to understand the methodology or results of this project.

C. Bagging

Bagging is a method that is short for boostrap aggregation. Since decision trees tend to suffer from high variance, creating many different trees and averaging them is a plausible way to reduce variance. However, it is not always feasible to split the data into enough training sets to create multiple decision trees to average. This is where bootstrapping comes in to play by taking B different datasets bootrapped from the single training data set. The model is trained on the *b*th bootstrapped training set to get $\hat{f}^{*b}(x)$ and then averaged over all predictions. This is defined mathematically below.

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

Bagging, in short, is a bias-variance trade off method for decision trees. It is used to reduce the high variance of decision trees at the cost of a small amount of bias by taking the average of B boostrapped decision trees.

D. Random Forest

Bagged trees are useful but also have a major flaw. If there is one strong predictor in the data set then most or all of the bagged trees will use this top predictor when considering the split. This causes many of the trees to look quite similar and the predictions will be highly correlated. Random forests are a solution to decorrelate the bagged trees. This is done by only considering only m of the total p predictors at each split. At each split a new sample of m out of p predictors is chosen with a good rule of thumb being $m \approx \sqrt{p}$. By sub-setting the number of predictors at each split, the predictors that are not as strong will have more of a chance to influence the model. For this project, predictors correspond to the tokens created from the article text and title. Hence, the models had a large number of predictors, especially in the case of article text. This is one of the advantages of using the random forest algorithm for classification is that it gives tokens that might not be the strongest a chance to

influence the model. Unlike bagging, which averages correlated results, averaging uncorrelated results leads to a larger reduction in variance.

E. Feature Importance

Since both bagging and random forests are no longer using a single decision tree, interpretation of the results, by taking the mean (regression) or most common case (classification) of the region the observation belongs, is no longer possible. With a collection of bootstrapped trees, the way the results of the model are interpreted are by looking at feature importance. For regression, feature importance is calculated by taking the total amount RSS is decreased due to splits over a given predictor which is then averaged over all B trees. In the case of classification, the total amount of reduction in the Gini Index is used in place of RSS. In both cases, a large mean decrease indicates an important feature. This project was specifically looking at feature importance as one of its goals because features here indicates what words are the most important predictors in the model to classify articles text and titles to their source.

F. Classification Metrics and Confusion Matrix

Results for binary classification algorithms are reported in a number of different ways. These metrics apply to both the random forest algorithm and the logistic regression. The most common and most interpretable is accuracy.

$$Accuracy = \frac{Number\ Correct}{Number\ of\ Predictions}$$

While accuracy is a useful metric, it makes no distinction between different classes. This is where a confusion matrix provides more information. A confusion matrix shows the actual class versus the predicted class as shown in Table 3.

Table 3. Example of a confusion matrix

[redacted]	PREDICTED CLASS OF OBSERVATION	
	Class 1	Class 2

TRUE CLASS OF OBSERVATION	Class 1	True Positive (TP)	False Negative (FN)
	Class 2	False Positive (TP)	True Negative (TN)

Table 3 has two classes. For this project, class 1 is the news source Fox News whereas class 2 is the news source CNN. The true positive (TP) rate, in this case, would refer to article text or title that is from Fox News and was successfully predicted as being from Fox News. Similarly, true negative (TN) rate refers to article text or titles from CNN which was successfully predicted as being from CNN. The false positive (FP) rate indicates that the observation, either text or title, was originally from CNN but was wrongly predicted as being from Fox News. False negative (FN) rate would then be an observation that was originally from Fox News but was predicted as being from CNN. Note that accuracy can be defined in terms of TP, TN, FP and FN.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Other classification metrics used to measure the results of classification are precision and recall. Recall is the total number of correctly classified positive or $Recall = \frac{TP}{TP+FN}$. Another way to think about recall is that high recall would indicate a low false negative rate. While it makes more sense for the results of a medical test, the recall for this project would indicate the number of articles or titles correctly classified as Fox News.

Precision, on the other hand, means that an observation labeled as positive is actually positive or has a small number of false positives or $Precision = \frac{TP}{TP+FP}$. That means an article or text that is labeled as being from Fox News is actually from Fox News.

Precision and Recall share a special kind of relationship. If the prediction results have high recall and low precision, this indicates most of the positive examples are correctly classified but there are a lot of false positives. This means that article text or titles that were from Fox are being correctly classified as

being from Fox but there are also a lot of CNN article text or titles being classified as a Fox News source. If the results show low recall and high precision, this means a lot of positive examples were misclassified but those that were classified as positive are actually positive. In the sense of this project, a low recall and high precision result would indicate that a lot of article text or titles that were originally from Fox were misclassified as being from CNN but observations that were classified as being from Fox are actually from Fox.

In order to measure the relationship between precision and recall, a classification metric known as the F_1 score is used. The F_1 score can be defined as the harmonic mean of the precision and recall. In a more general sense, it is the percentage of positive predictions that were correct. This is similar to accuracy but the F_1 score is usually lower than accuracy since precision and recall are embedded in the calculation.

$$F_1\ Score = \frac{2 * Precision * Recall}{Recall + Precision}$$

The F_1 score uses the harmonic mean instead of the arithmetic mean so that extreme values are penalized. A higher F_1 score indicates a better model.

G. Grid Search Cross Validation

The data is initially split into a testing and training set. The training set was chosen to be 80% of the entire data set while 20% was used for testing. The training set is where the model is fit or trained. The testing set is then used for validating the results. In order to choose the best hyper parameters for the logistic regression and random forest models, grid search cross validation was used. Grid search cross validation in Python is implemented using the GridSearchCV package. Grid search cross validation with K=5 was implemented so that a number of different models using different hyper parameters were compared. In K=5-fold cross validation, the data set is split into 5 different sets or folds. The model is then trained using four of the folds and tested using the one that is left out. In K=5-fold cross validation, this process would be repeated 5 times so that each fold gets a chance to be left out. The results are

then averaged for all K cases. Note that grid search cross validation is performed on the training set only to find the best parameters. Once the best parameters are found, the model is retrained using those parameters on the training set and the results are validated using the test set. Figure 2 shows a visual representation of the K=5-fold cross validation process used for grid search cross validation. Figure 3 shows a visual representation of the entire cross validation workflow. Both Figure 2 and 3 are from the sklearn documentation.

Hyper parameters for both random forests, logistic regression, and the word vectorization process were tuned in the grid search cross validation process. In the random forests algorithm, the hyper parameters tuned were n_estimators and max_depth. For the n_estimators, the values 10, 50, and 100 were used. The max_depth hyper parameter used 100, 500, and 1000. The min_sample_split was set at 100 and the max_leaf_nodes was set at "None". During the grid search cross validation for the word vectorization process, both count vectorization and TF-IDF vectorization used the same hyper parameters for optimization. Hyper parameters during the word vectorization process were max_features, ngram_range, and stop_words. The values used for the respective hyperparameters were 100, 2000, and 5000 for max_features. The ranges (1, 1), (1, 2), (2, 2), (1,3), and (1,4) were used for ngram_range and the stop word list previously discussed was used for stop_words. During the logistic regression grid search cross validation, word vectorization hyper parameters were optimized but there were no hyperparameters specifically used for logistic regression.

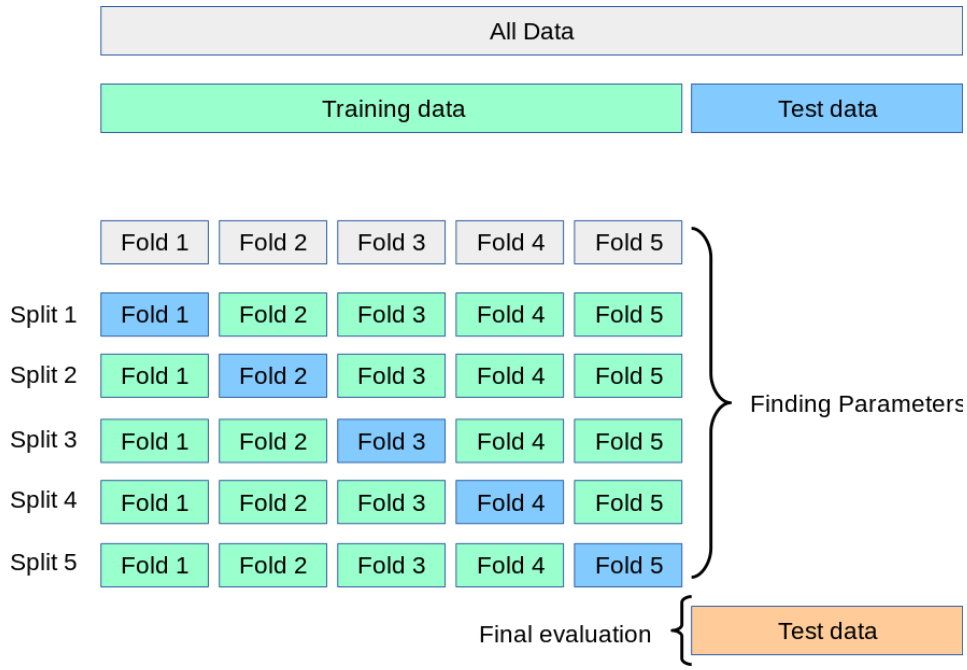

Figure 2. K=5-fold grid search cross validation to find optimum parameters

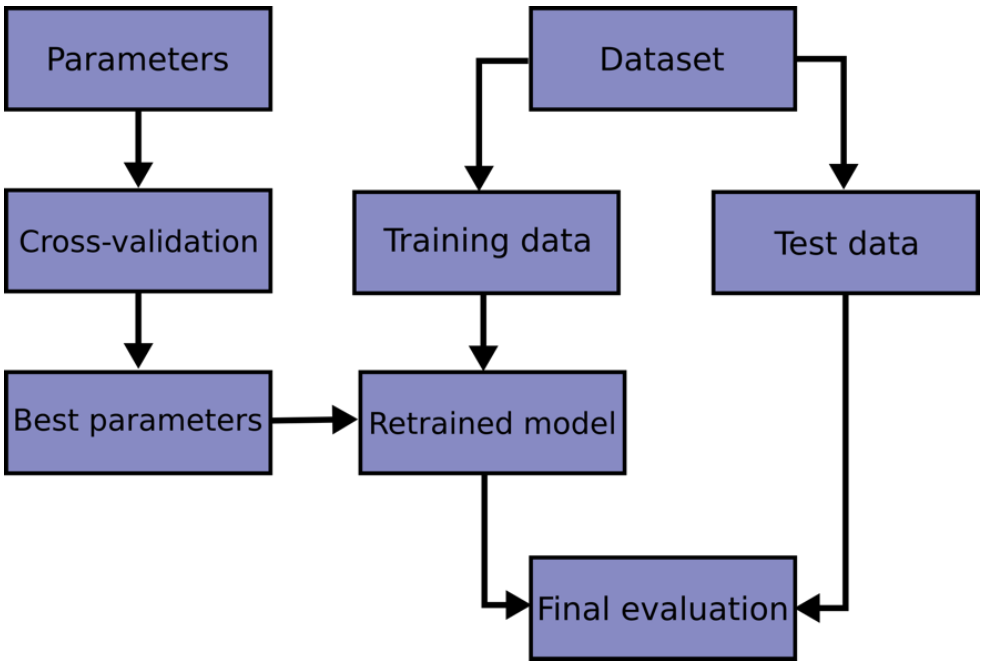

Figure 3. Order of splitting data and cross validation

H. Pipeline

There were a number of different methods used to get from the initial web scraping of the news sources to the final classification results. In order to optimize the process, a pipeline was created. In Python, this is done with the Pipeline package. The pipeline method was not used for all of the methods, but only for the methods involving the cross validation and fitting of the different models. The overall pipeline of the project is listed below. Although the steps use the article text for the purpose of illustration, the process is also performed a second time for the article titles.

1. Web scrape the article text from Fox News and CNN
2. Remove punctuation, new lines, and perform lowercase conversion on the article text
3. Train-test split the cleaned article text in an 80/20 split

4. Tune the hyper parameters using grid search K=5-fold cross validation
    a. Apply TF-IDF or CountVectorizer to transform the data from words to vectors
    b. During the word vectorization process, stop words were removed and the text was tokenized
    c. Fit the classifying algorithm

Box 1. Finding the optimum hyperparameters for use in the model

5. Fit the model to the training data set using the optimized parameters and then test results using the test set
    a. Apply TF-IDF or CountVectorizer to transform the data from words to vectors
    b. During the word vectorization process, stop words were removed and the text was tokenized
    c. Fit the classifying algorithm

Box 2. Running the model with the optimum hyper parameters found from grid search cross validation

6. Evaluate Results

The pipeline method was set up to streamline steps 4 and 5. Pipelines were set up for both word vectorizing methods to be used with logistic regression and the random forests algorithm. These pipelines were then created for both of the article text data and the article titles

IV. Results

i. Article Text

The following results in this section pertain specifically to the article text that was web scraped and not to article titles.

A. Random Forest using TF-IDF

The best hyper parameters found using the grid search cross validation are shown in Table 3.

Table 3. Optimum hyper parameters for the random forest model using TF-IDF for article text classification

Random Forest: Max_depth	100
Random Forest: Max_leaf_nodes	None
Random Forest: Min_sample_split	100
Random Forest: N_estimators	50
TF-IDF: Max_features	2000
TFIDF: Ngram_range	(1,4)

The hyper parameters given in Table 3 were then used in a final model. The confusion matrix of the results for the final random forest model are shown in Table 4.

Table 4. Confusion matrix of results from the random forest model using TF-IDF for article text classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	31	2
Actual: Fox News	0	27

The precision, recall, F_1 score and accuracy for the final model using the optimized parameters are shown in Table 5.

Table 5. Precision, recall, F_1 score, and accuracy for the random forest model using TF-IDF for article text classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	1.00	0.94	0.97
1 (Fox News)	0.93	1.00	0.96
Weighted Average	0.97	0.97	0.97
Accuracy	0.9666		

The 20 tokens with the highest feature importance are shown from top to bottom in Table 6.

Table 6. Top 20 tokens with the highest feature importance from the random forest model using TF-IDF for article text classification

Token	Feature Importance
“click”	0.0671
“president donald”	0.0554

“donald”	0.0527
“contributed”	0.0487
“report”	0.0407
“percent”	0.0372
“president donald trump”	0.0312
“president trump”	0.0292
“read”	0.0286
“contributed report”	0.0263
“press contributed report”	0.0201
“democrats”	0.0200
“wrongdoing joe”	0.0147
“administration”	0.0140
“click app”	0.0139
“vice president”	0.0133
“ukranian”	0.0128
“app”	0.0126
“foxbusinesscom”	0.0122
“wrongdoing joe hunter biden”	0.0119

The actual tree depth for all 50 estimators is shown in the code. A quick summary of the tree depths are the maximum was 8, the minimum was 2, the median was 4, the mean was 4.12, and standard deviation was 1.409.

B. Logistic Regression TF-IDF

The best hyper parameters found using grid search cross validation for the logistic regression using TF-IDF were 2000 for the max_features parameter and (1,4) for the Ngram_range parameter.

The optimum hyperparameters were then used in a final logistic regression model using TF-IDF vectorization. The confusion matrix for the final model is shown in Table 7.

Table 7. Confusion matrix of results for the logistic regression using TF-IDF model for article text classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	28	5
Actual: Fox News	2	25

The precision, recall, F_1 score, and accuracy for the final logistic regression model using TF-IDF is presented in Table 8.

Table 8. Precision, recall, F_1 score, and accuracy for logistic regression using TF-IDF model for article text classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.93	0.85	0.89
1 (Fox News)	0.83	0.93	0.88
Weighted Average	0.89	0.88	0.88
Accuracy	0.88333		

C. Random Forest with Count Vectorizer

The best parameters found using grid search cross validation for the random forest algorithm using count vectorizer are shown in Table 9.

Table 9. Optimum hyper parameters for the random forest using count vectorizer model for article text classification

Random Forest: Max_depth	1000
Random Forest: Max_leaf_nodes	None
Random Forest: Min_sample_split	100
Random Forest: N_estimators	100
Bag of Words: Max_features	2000
Bag of Words: Ngram_range	(1,3)

Using the hyper parameters found in Table 9, a final random forest model using count vectorizer was fit. The results of the model are presented in a confusion matrix in Table 10.

Table 10. Confusion matrix of results from the random forest using count vectorizer model for article text classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	31	2
Actual: Fox News	3	24

The precision, recall, F_1 score, and accuracy for the final model random forest model using count vectorization is shown in Table 11.

Table 11. Precision, recall, F_1 score, and accuracy for the random forest using count vectorizer model for article text classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.91	0.94	0.93
1 (Fox News)	0.92	0.89	0.91
Weighted Average	0.92	0.92	0.92
Accuracy	0.91666		

The top 20 tokens from the final random forest model using count vectorizer listed in terms of highest feature importance to lowest is displayed in Table 12.

Table 12. Top 20 tokens with the highest feature importance from the random forest model using count vectorizer for article text classification

Token	Feature Importance
“click”	0.0740
“click app”	0.0564
“contributed”	0.0481
“app”	0.0411
“read”	0.0360
“president donald”	0.0337
“contributed report”	0.0221
“dcalif”	0.0220
“president trump”	0.0213

“donald trump”	0.0190
“associated press contributed”	0.0180
“click allnew”	0.0158
“allnew foxbusinesscom”	0.0157
“washington”	0.0153
“wrongdoing joe hunter”	0.0137
“campaign”	0.0130
“allnew”	0.0129
“evidence wrongdoing”	0.0125
“bernie”	0.0122
“president donald trump”	0.0118

D. Logistic Regression with Count Vectorizer

The optimum hyper parameters found using grid search cross validation for the logistic regression using count vectorizer were 5000 for the max_features parameter and (2,2) for the Ngram_range parameter. A final logistic regression model using the optimum hyper parameters was fit and the confusion matrix of the results is displayed in Table 13 below.

Table 13. Confusion matrix of results from the logistic regression model using count vectorizer for article text classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	31	2
Actual: Fox News	2	25

The precision, recall, F_1 score and accuracy for the final logistic regression model using count vectorizer is shown in Table 14.

Table 14. Precision, recall, F_1 score, and accuracy for the logistic regression model using count vectorizer for article text classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.94	0.94	0.94
1 (Fox News)	0.93	0.93	0.93
Weighted Average	0.93	0.93	0.93
Accuracy	0.93333		

ii. Article Titles

The results in this section pertain only to the article titles that were web scraped. These models were performed separately from the article text and so the only connection between these results and the results of the article text presented in the previous section is that each article has an associated title.

A. Random Forest using TF-IDF

The best hyper parameters found using grid search cross validation for the random forest model using TF-IDF are shown in Table 15.

Table 15. Optimum hyper parameters for the random forest model using TF-IDF for article title classification

Random Forest: Max_depth	500
Random Forest:	None

Max_leaf_nodes	
Random Forest: Min_sample_split	100
Random Forest: N_estimators	100
TF-IDF: Max_features	100
TFIDF: Ngram_range	(1,2)

Hyper parameters shown in Table 15 were then used in a final random forest model using TF-IDF. The confusion matrix for the final random forest model is presented in Table 16 while the precision, recall, F_1 score and accuracy of the final model are shown in Table 17.

Table 16. Confusion matrix of results from the random forest model using TF-IDF for article title classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	21	12
Actual: Fox News	6	21

Table 17. Precision, recall, F_1 score, and accuracy for the random forest model using TF-IDF for article title classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.78	0.64	0.70
1 (Fox News)	0.64	0.78	0.70

Weighted Average	0.71	0.70	0.70
Accuracy	0.70		

The 20 tokens with the highest feature importance form the final random forest model are shown in Table 18 and are listed from highest to lowest feature importance.

Table 18. Top 20 tokens with the highest feature importance from the random forest model using TF-IDF for article title classification

Token	Feature Importance
“whistleblower”	0.0975
“trump”	0.0807
“probe”	0.0583
“dems”	0.0404
“impeachment”	0.0371
“house”	0.0313
“trumps”	0.0288
“campaign”	0.0286
“sanders”	0.0260
“ukraine”	0.0218
“says”	0.0172
“impeachment inquiry”	0.0170
“gop”	0.0170
“kavanaugh”	0.0143

“pelosi”	0.0141
“claims”	0.0138
“dc”	0.0137
“inquiry”	0.0132
“president”	0.0128
“stop”	0.0126

The actual tree depth for all 100 estimators is shown in the code. A quick summary of the tree depths are the maximum was 23, the minimum was 4, the median was 13, the mean was 12.8, and standard deviation was 3.59.

B. Logistic Regression using TF-IDF

The optimum hyper parameters found using grid search cross validation for the logistic regression model using TF-IDF were 2000 for the max_features parameter and (1,2) for the Ngram_range parameter. Those hyper parameters were used for the final logistic regression model with TF-IDF vectorization. The confusion matrix of the results of the final model are presented in Table 19 and the precision, recall, F_1 score and accuracy are shown in Table 20.

Table 19. Confusion matrix of results from the logistic regression model using TF-IDF for article title classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	15	18
Actual: Fox News	4	23

Table 20. Precision, recall, F_1 score, and accuracy for the logistic regression model using TF-IDF for article title classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.79	0.45	0.58
1 (Fox News)	0.86	0.85	0.68
Weighted Average	0.69	0.63	0.62
Accuracy	0.63333		

C. Random Forest using Count Vectorizer

The optimum hyper parameters found using grid search cross validation for the random forest model using count vectorization are shown in Table 21.

Table 21. Optimum hyper parameters for the random forest model using count vectorizer for article title classification

Random Forest: Max_depth	1000
Random Forest: Max_leaf_nodes	None
Random Forest: Min_sample_split	100
Random Forest: N_estimators	50
Bag of Words: Max_features	2000
Bag of Words: Ngram_range	(1,3)

The hyper parameters in Table 21 were used in a final random forest model using count vectorization. For this final model, the confusion matrix of the results is displayed in Table 22 and the precision, recall, F_1 score and accuracy are all presented in Table 23.

Table 22. Confusion matrix of results from the random forest model using count vectorizer for article title classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	20	13
Actual: Fox News	6	21

Table 23. Precision, recall, F_1 score, and accuracy for the random forest model using count vectorizer for article title classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.77	0.61	0.68
1 (Fox News)	0.62	0.78	0.69
Weighted Average	0.70	0.68	0.68
Accuracy	0.683333		

The 20 tokens with the highest feature importance from the model are listed in Table 24 in order of highest feature importance to lowest.

Table 24. Top 20 tokens with the highest feature importance from the random forest model using count vectorizer for article title classification

Token	Feature Importance

“probe”	0.0398
“sanders”	0.0327
“campaign”	0.0244
“dems”	0.0213
“whistleblower”	0.0179
“house”	0.0148
“trump”	0.0138
“committee”	0.0132
“lawmaker”	0.0098
“kavanaugh”	0.0090
“donald”	0.0084
“gop”	0.0084
“trumps”	0.0082
“stop”	0.0081
“debate”	0.0080
“congress”	0.0080
“primary”	0.0079
“slams”	0.0073
“bid”	0.0058
“new”	0.0057

D. Logistic Regression using Count Vectorizer

The optimum hyper parameters found using grid search cross validation for the logistic regression model using count vectorization were 5000 for the max_features hyper parameter and (2,2) for the

Ngram_range hyper parameter. A final logistic regression model using the optimum hyper parameters was fit and the confusion matrix of the results is displayed in Table 25 below.

Table 25. Confusion matrix of results from the logistic regression model using count vectorizer for article title classification on the test data set

	Predicted: CNN	Predicted: Fox News
Actual: CNN	19	14
Actual: Fox News	7	20

The precision, recall, F_1 score and accuracy for the final logistic regression model using count vectorizer is shown in Table 26.

Table 26. Precision, recall, F_1 score, and accuracy for the logistic regression model using count vectorizer for article title classification on the test data set

	Precision	Recall	F_1 Score
0 (CNN)	0.73	0.58	0.64
1 (Fox News)	0.59	0.74	0.66
Weighted Average	0.67	0.65	0.65
Accuracy	0.65		

iii. Model Comparison and Discussion

For comparison, Table 27 displays the accuracy from the four models used for the article text while Table 28 shows the accuracy from the four models used for the article titles.

Table 27. Accuracy of article text classification models

Model	Accuracy
Random Forest using TF-IDF	0.96666
Logistic using TF-IDF	0.88333
Random Forest using Count Vectorization	0.91666
Logistic using Count Vectorization	0.93333

Table 28. Accuracy of article title classification models

Model	Accuracy
Random Forest using TF-IDF	0.70000
Logistic using TF-IDF	0.63333
Random Forest using Count Vectorization	0.68333
Logistic using Count Vectorization	0.65000

For the article text, the model with the highest accuracy is the random forest model using TF-IDF. For the article titles, the random forest model using TF-IDF is also the model with highest accuracy. In both the classification of the article text and article titles, the logistic model performed better with count vectorization instead of TF-IDF. Conversely, the random forest model performed better when using the TF-IDF instead of the count vectorization in both the article titles and article text. The other result that stands out is the accuracy for the article text classification is much higher than the article titles. There are two likely causes for this. The first one is that the text classification had a lot more tokens to use in the learning process. The second was the overfitting problem for the article titles. When scoring on only the training data sets, many of the models for article titles achieved 100% accuracy while the accuracy on the test data was only around 60%. This is a classic example of overfitting where the model fits the training data set so well that it doesn't generalize well when using a new data set, in this

case, the test data. Table 29 shows the training and testing accuracy scores for each of the models. While training scores are usually higher, a vast difference between training and testing scores indicates an overfitting problem.

Table 29. Train and test accuracy scores for all models

	Training Score	Testing Score
Random Forest using TF-IDF for Text Classification	0.9708	0.9167
Logistic Regression using TF-IDF for Text Classification	0.9500	0.8833
Random Forest using Count Vectorizer for Text Classification	0.9667	0.95
Logistic Regression using Count Vectorizer for Text Classification	0.9999	0.9333
Random Forest using TF-IDF for Title Classification	0.8167	0.6833
Logistic Regression using TF-IDF for Title Classification	1.0000	0.6333
Random Forest using Count Vectorizer for Title Classification	0.9043	0.5833
Logistic Regression using Count Vectorizer for Title Classification	1.0000	0.6500

The feature importance also yielded results worth discussing. In the models used to classify article text a few tokens appeared in both models. Various forms of “president donald trump” and

"wrongdoing joe hunter biden" and "click app" are common in both models. In the article text models, forms of "trump" are once again common and also "dems", "probe" and "sanders". These results may have more to do with the collection period of the news articles. During the three weeks of web scraping, political events in the news were the start of impeachment proceedings for the president. It is, perhaps, unsurprising then that the president is so high in feature importance in all of the random forest models. However, the classification problem was to classify Fox News articles from CNN articles. One way to interpret the feature importance results are that these are tokens important to the model to classify Fox News articles from CNN news articles. In that sense it is surprising that the various tokens of Trump have so much importance when he is prevalently mentioned in both CNN and Fox News articles. An initial hypothesis of which words that would be high in feature importance were more in line with specific lingo to each newspaper or authors that work for each news source. One such example is the "dems" token that was common in the article title models. This is a specific way to abbreviate the democratic party that seemed common in Fox News articles whereas the word was typically spelled out in full in CNN articles. Further investigation into the tokens with high feature importance could be expanded on in future work.

V. Conclusions and Future Work

The random forest model using TF-IDF classified articles with higher accuracy than the logistic model. In the case of the article text, both models achieved a very high rate of success with almost all having >90% accuracy. With the test set size, this translates to only a few of the articles being misclassified. This is pretty amazing considering the articles taken from both news sources covered a lot of the same news material during those three weeks. Yet, the classification algorithms could still find enough of a difference to classify with high accuracy.

Although the random forest model performed better in this case, some further tweaking of the logistic model, such as l_1 or l_2 penalization, could see the logistic model improve. Other future work

includes increasing the number of articles and collection time. The results, especially the feature importance tokens, were heavily influenced by what was occurring in the news. Being able to run the experiment for a longer time, such as a year, would yield different results. In addition, collection of more articles could fix the overfitting problem found in the article title classification. The biggest focus of this project, at least algorithm wise, was on using the random forests to classify the article text. There could be a number of ways to improve the article title classification as well the logistic models such as making sure multicollinearity isn't affecting the logistic models or doing a more thorough job of cleaning the article titles.

Lemmatization was done in the initial cleaning process but was not used in the final models. The vectorizers were having problems taking in already tokenized data that the lemmatize methods created and the results for the lemmatized text were not much of an improvement or worse. Therefore, it was decided not to lemmatize the text in this project. However, future work could look further into this problem and, perhaps with more articles, the lemmatization would result in a bigger benefit to the results.

This project was a very specific case and could easily be expanded into a bigger project. The number of news sources as well as the different algorithms used to classify the data could be expanded upon. In addition to logistic and random forest algorithms, support vector machines were also tried for classification of this data. However, due to time constraints it was decided to not pursue those models further. While random forest and logistic regression did classify the article text pretty accurately, there are a number of other classification methods worth trying that could classify with even higher accuracy than those investigated in this project.

VI. Code

Two separate Jupyter notebook files, one for the web scraping and the other for the analysis portion of the project are available are the GitHub link below. Code was done using python version 3.6.9

GitHub: https://github.com/mjingram/MastersPoliticalNLP

Python Package Documentation:

Newpaper3k: https://newspaper.readthedocs.io/en/latest/

Re: https://docs.python.org/2/library/re.html

SpaCy: https://spacy.io/usage/spacy-101

Sklearn: https://scikit-learn.org/stable/documentation.html

VII. References

Agresti, A. (2014). *Categorical Data Analysis*. Hoboken: Wiley.

Balahur, Steinberger, Kabadjov, Zavarella, van der Goot, Halkia, … Belyaeva. (2013). Sentiment Analysis in the News.

Brownlee, J. (2019, August 7). A Gentle Introduction to the Bag-of-Words Model. Retrieved September 25, 2019, from https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

Hastie, T., Friedman, J., & Tisbshirani, R. (2017). *The Elements of statistical learning: data mining, inference, and prediction*. New York: Springer.

idf :: A Single-Page Tutorial - Information Retrieval and Text Mining. (n.d.). Retrieved September 25, 2019, from http://www.tfidf.com/.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning: with applications in R*. New York: Springer.

Masum, S. A., Islam, M., & Ishizuka, M. (2006). ASNA: An Intelligent Agent for Retrieving and Classifying News on the Basis of Emotion-Affinity. *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA06)*. doi: 10.1109/cimca.2006.51

Mehler, A., Bao, Y., Li, X., Wang, Y., & Skiena, S. (2006). Spatial Analysis of News Sources. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 765–772. doi: 10.1109/tvcg.2006.179

Moretti, E. (2017). Social Media and Fake News. *American Economic Review*, *107*, 716–718. doi: 10.1257/aer.107.5.716

Rashkin, H., Choi, E., Jang, J. Y., Volkova, S., & Choi, Y. (2017). Truth of Varying Shades: Analyzing Language in Fake News and Political Fact-Checking. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. doi: 10.18653/v1/d17-1317

Zhou, D., Resnick, P., & Mei, Q. (2011). Classifying the Political Leaning of News Articles and Users from User Votes. Retrieved from www.aaai.org