Analyzing Political Polarization in News Media with Natural Language Processing

by
Brian Sharber

A thesis presented to the Honors College of Middle Tennessee State University in partial fulfillment of the requirements for graduation from the University Honors College

Fall 2020

Analyzing Political Polarization in News Media with Natural Language Processing

by
Brian Sharber

APPROVED:

Dr. Salvador E. Barbosa
Project Advisor
Computer Science Department

Dr. Medha Sarkar
Computer Science Department Chair

Dr. John R. Vile, Dean
University Honors College

Department of Computer Science

Middle Tennessee State University; Murfreesboro, Tennessee, USA.

The software is posted on GitHub under the following repository:
https://github.com/briansha/NLP-Political-Polarization

Abstract

Political discourse in the United States is becoming increasingly polarized. A Natural Language Processing (NLP) framework is provided to uncover political polarization, utilizing political blogs and political websites as unique sources of insight into this phenomenon. These aspects of polarization are quantified utilizing different machine learning classifiers and methods of analysis. The creation of an ensemble learner as a voting system is proposed for increased accuracy of text classification of documents between differing political poles. Methods are applied to study polarization across eleven differing political sources from the timeline of January 2020 to July 2020. Findings indicate that discussion of events across this timeline are highly polarized politically, and a list of the top identifying features (words) which may indicate significant slant is provided. This work contributes to a deeper understanding of computational methods for studying polarization in political news media and how differing political groups manifest in language.

Table of Contents

1. Project Introduction

With an ever-growing amount of data stored in electronic form such as news articles, emails, policies, legal documents, public statements, Twitter posts, etc., automatic text classification is becoming increasingly vital to research, organizations, and overall human health. Technologies which utilize Natural Language Processing (NLP), computer processing of natural language, and are becoming increasingly widespread, from phones supporting predictive text and spam detection in email inboxes to machine translation of texts written in another language to English. The goal of NLP is to do some form of analysis, or processing, where the machine can understand, at least to some level, what a body of text means, says, or implies. Utilizing Python as the programming language of choice, NLP has been used to describe and analyze language for scientific, cultural, economic, and social advances. By providing more natural human-machine interfaces and more sophisticated access to stored information, language processing has come to play a central role in the multilingual information society [2].

When we think about political parties and news media in the US, two political poles come to mind: left and right. The difference in these ideologies center around the role of the government and the rights of individuals. Both sides prize rights of individuals, but each has a hierarchy to it which has changed over time. Left-wing beliefs are liberal, believing that society is best served with the government having an expanded role in the economy and providing a range of social services to ensure well-being and equality across society. Individuals on the right are conservative, believing the best outcome for society is achieved when the role of the government is minimized, operating mainly at the state or local level, preferring private sector-based solutions to problems.

When asking someone what the left care about vs. what the right care about, and the type of rhetoric each side uses, differing answers are certain to be provided on a person-to-person basis. To have a definitive answer, one would have to read hundreds upon thousands of articles from each side and put aside one's own bias. Fortunately, a human does not have to bear this burden on their own. For this task, NLP can be utilized to computationally analyze political polarization in news media, such as political blogs and websites, and create a machine learning model that is capable of distinguishing between the language of the left and right.

Prior NLP work has modeled slant in news articles and shown that polarized messages are more likely to be shared, along with certain topics to be more polarizing [1] [11] [6]. It is also known that language is one of the main tools used by politicians to promote their agenda, gain popularity, win elections, and drive societal change [9]. However, we are lacking in a broader understanding of the many ways that polarization can be identified linguistically [4]. This work builds a framework for studying linguistic aspects of polarization in news media, by looking at the political section of different news sources. The linguistic aspects of different political poles are studied by extracting frequently used words and word pairs, seen more often in one political pole than the other, that could indicate significant slant.

2. Background and Related Work

Natural Language Processing allows computers to understand human speech with the help of machine learning. Dating back to 1971, the Defense Advanced Research Projects Agency (DARPA) utilized NLP for Robust Automatic Transcription of Speech (RATS) for tasks related to speech-containing signals received over highly distorted communication channels. From 2011 onward, smart assistants such as Apple's Siri, Amazon's Alexa, and Google's Google Assistant have made breakthroughs for various applications of NLP, utilizing the Python programming language and the *NLTK* library to make these breakthroughs a reality.

The Natural Language Toolkit (*NLTK*) documentation describes it as a "leading platform for building Python programs to work with human language data." It provides access to lexical and corpora resources, such as WordNet, along with text processing libraries for classification, tokenization, lemmatization, etc. for tasks such as working with corpora, text classification, analysis of linguistic structure, and sentiment analysis.

Classification is the task of the machine choosing the correct class label for a given input. A class label is the discrete category whose value is predicted based on the values of other attributes (features). A class involving a specific grouping, such as flowers, will have labels denoting specific categories of that class, such as *Iris* or *Lily*. In this example, features may be the length, width, or number of petals. Training a machine learning model involves using pre-labelled data, typically defined by the user, that the machine learning algorithm can use to learn different associations between pieces of text and that a particular output (tag) is expected for a particular input (text). Supervised classifiers are built based on training corpora containing the correct label for each input

[2]. A key characteristic between supervised and unsupervised classifiers is that supervised learning data must be labeled for outcome-comparison. The more representative the data is to its true target environment, the more accurate the classifier trained using such data will appropriately label data.

Text classification assigns a particular tag or category to a body of text according to its content. The most common example of text classification is a spam detection system for email inboxes. Text is rich in information, but extracting important insights from text can be time-consuming and difficult due to the unstructured nature of it. Businesses turn to text classification for structuring text to enhance decision-making and automate processes.

Sentiment analysis refers to analysis of text or images to extract opinion or feelings regarding almost anything. Amazon uses sentiment analysis in the customer reviews section of their product listings to detect public opinion regarding for-sale products. Based on the context of customer reviews labelled as positive, negative, or neutral, the company moves forward in its decision-making process regarding certain products.

Related works have applied NLP methodology to study aspects of polarization in social media by looking at topic choice, framing, affect, and illocutionary force (speaker intention behind delivering an utterance) across 4.4 million tweets from Twitter on 21 mass shootings [4]. When looking at affect and illocutionary force, the findings suggested Democrats were more likely to assign blame and make calls for action than Republicans. In addition, sadness and trust were more likely to be expressed by Democrats across events, while fear and disgust were more likely to be expressed by Republicans, particularly when the shooter was a person of color [4]. When looking at the setting and the shooter's race, Democrats were more likely to contextualize any mass shooting among school shootings and call white shooters "terrorists" than Republicans, who in turn were more likely to liken any shooting to other violent events pertaining to people of color – whom they were more likely to call "terrorist" than Democrats. Moreover, Democrats were as likely to frame the shooter as mentally ill when they were a person of color, as Republicans were when shooters were white [4]. When looking at induced topics, the findings suggested Republicans preferentially discussed topics about the shooter's identity and ideology, whereas Democrats preferentially discussed solidarity and policy-related topics [4]. The study concluded that reactions to these tragic events are highly polarized politically.

Another study used a collection of blog posts dating from November 2007 to October 2008 from five blogs that focused on American politics to model polarizing topics and see when different political communities responded differently to the same news [1]. The following are the findings from the study regarding six different topics. Two topics from the study resulting in positive comments from Daily Kos (liberal) readers and negative comments from Red State (conservative) readers were "Energy and Environment" (a topic regarding the current administration's energy policy, nuclear and alternative sources of energy, and the BP oil spill off the Gulf coast of the US and its impact on the environment) and "Union Rights and Women's Rights" [1]. Two topics resulting in negative comments from Daily Kos readers, but positive comments from Red State readers were "Senate Procedures" and "Republican Primaries" [1]. Finally, the last two topics, "Economy, Taxes, and Social Security" and "Mid-term Elections", indicated the least amount of contention between differing political parties, resulting in the topics being non-polarized. The study, inspired by research in political psychology that has made it clear that political decision-making is strongly influenced by emotion, also found that topics evoking positive comments attracted a higher volume of comments, whereas topics which attracted negative comments triggered less volume [1]. These findings reinforce domain knowledge about political discourse in the US.

One other study used a corpus of public statements, a total of 134,000 statements, released by members of Congress in the Senate and The House of Representatives along a span of four years (2010-2013) to provide a glimpse into the complex dynamic processes of framing, attention shifts, and agenda setting in US politics [10]. Party discipline is of great interest to political scientists and is typically examined by analysis of roll call votes on bills, but it can also be measured by adherence to party lines in talking points and agenda setting campaigns [10]. In an absence of an official list of talking points, the study proposed a metric of repeated use of similar phrases across public statements to indicate the level of party discipline. For example, the phrase "the American people" appeared in 81 statements by 54 members of Congress, but was only used by two Democrats [10]. Similarly, the phrase "social security benefits" appeared in 123 statements issued by 89 members, 71 of which were Democrat [10]. Examining ownership of total usage of repeated use of similar phrases across statements revealed Republicans do tend to stick to talking points more than Democrats do [10]. This is somewhat "common knowledge" among political scientists, but the findings of the study were able to provide evidence that supported this idea, using a machine to learn human language.

Some of these studies utilized LDA (Latent Dirichlet Allocation), WordNet, and SVM (Support Vector Machine) in their machine learning models. This thesis expands on previous work by proposing a wider variety of classifiers, along with an ensemble learner, in an attempt to increase the accuracy at which a machine learning model can correctly classify sources as being “left or “right”, based on the type of language contained within the source. Inspired by previous related work, this thesis utilizes NLP to detect polarization across the political news sections of several differing political blogs and online news outlets.

3. Approach/Methods

To begin quantifying polarization across differing political communities, data needed to be collected from several political blogs and political websites. All the front page articles, whether opinion or non-opinion based, from the political news sections of each news source were gathered on a daily basis at varying times, with the date the article was created and the contents of each article catalogued. Left-wing political blogs and websites looked at include: HuffPost, Talking Points Memo, Salon, AlterNet, The Raw Story, and Slate. Right-wing political blogs and websites looked at include: National Review, Red State, The Bulwark, The Washington Examiner, and NewsMax. The timeline of the list compiled relates to articles published from January 1, 2020 – July 31, 2020.

After data had been gathered, the pre-processing phase could begin. This phase is important for ensuring a good set of features is produced for decent classification of text. A clean dataset allows the model to train on meaningful data and not overfit on irrelevant data noise. A model will only ever be as good as the data it gets to work with, so the data gathered needs to be a good representation of the labels assigned to individual entries. In this phase, stopwords which do not provide any significant contextual meaning (such as "the", "a", "an", "in") were removed, words were lowercased so their capitalized forms were not considered as separate entities, punctuation and special characters such as "@" were removed as they provide almost no significant meaning, text was tokenized by separating it into individual words, and lemmatization was considered to reduce words to their more common forms (such as "am" and "are" into "be").

With the pre-processing done, different classifiers could be used to train different models. For classification, Bernoulli Naïve Bayes, Multinomial Naïve Bayes, Adaboost, Binary Logistic Regression, LinearSVC, Random Forest, and SGD classifiers were settled on. RBF SVM, K-nearest neighbors, NuSVC, SVC, and Decision Tree were considered, but the runtimes were considerable and early results showed far less accuracies in classification of text.

After deciding on the classifiers to use for each new training and testing run, the models trained on the input data: articles from different leaning political news sources. CountVectorizer(), TfidfTransformer(), and Pipeline objects were utilized. Once trained, the models were analyzed for accuracy by checking the percentages across the number of articles in the data sets that were correctly classified with a class label of "left" or "right", and further tweaked by testing different parameters. The accuracy of each model was averaged over 30 runs and the top 3 performing models were used within a VotingClassifier ensemble learner (similar to a voting system) to see whether this new model would result in an increase of accuracy and be the top-performing model, which was hypothesized before any testing began. The confusion matrix, F1-score, precision, and recall of each model was shown after each full training session, indicating the top performers.

4. Conduct of Experiments

4.1 Pre-Processing

All the articles gathered from the corresponding blogs and websites were originally stored into .txt files and later converted to Excel files. Short articles fewer than 800 characters were not considered, as they are not as helpful for classification purposes than longer articles with more thought put into them. Only unique articles were kept within the data, to prevent the model from training on duplicate data. The software library, *pandas*, is built on top of the Python programming language and contains many data structures and operations which are useful for data analysis and classification tasks. A dataframe, accessed from the *pandas* library, was utilized as the main storage object for all articles and their respective dates of publication and class labels; this storage object can be thought of as an Excel table.

Data from individual blogs and websites was loaded into a dataframe containing three columns labelled *date*, *article*, and *pole*. Each individual entry (row) in the dataframe contained the date the article was published, the entire article itself, and the political polarity of the article. In the *pole* column, left-wing articles took on a label of 0, while right-wing articles took on a label of 1. An *unfiltered_articles* column was made after loading the data into the three columns. This column contained row entries correlating to the original articles loaded into the *article* column. As each entry in the *article* column was to be pre-processed for feature finding purposes, this column was necessary for being able to properly test models against the original source material gathered. After the *unfiltered_articles* column was made, all text in each entry in the *article* column was lower-cased. This would help the model train on the true distribution of words, such as “healthcare”, when every instance of the word was made into its lower-cased form.

To prevent data noise, any entry seen to have a null value (or empty) in the *date* or *article* columns was replaced with a space. Replacing any null value present in the *article* column with a space would remove the article from being considered, as it was less than 800 characters. Replacing any null value present in the *date* column with a space would keep an article that was still useful to train upon, but with an unknown date. This process of fixing null values in the data set was deemed reasonable.

To maintain a balance for training purposes, sources had their number of articles reduced to the minimum number of articles between the left and right sources considered. When lowering the number of sources to the minimum amount, articles were randomly chosen and balanced. Dates, in general, within the articles were removed, as they were not useful features to consider. To deal with contractions, a list relating to commonly seen ones replaced each contraction with its two-word pair, such as “do not” for the contraction “don’t”. See Appendix for more.

4.2 Classifiers

The process of quantifying polarization began by considering different classifiers available for training machine learning models. In keeping with the philosophy of starting with the simplest tool available, the Naïve Bayes algorithm was the first algorithm used for training purposes: a basic algorithm, but a popular one used for text classification purposes as a good starting point. Utilizing the most fundamental probability knowledge, the algorithm makes a naïve assumption that all features of a data set are independent and then attempts to classify data.

Bernoulli Naïve Bayes models the presence/absence of a feature, whereas Multinomial Naïve Bayes models the number of counts of a feature. The way these two classifiers classify text made them useful starting tools, although the naïve assumption they utilize did not make them top performers (low tier).

Adaboost is a machine learning approach that attempts to classify data using a sequential *ensemble* method. It combines several classifiers to improve the final predictive performance. To put it simply, the idea is to set weights to the classifiers and data in such a manner that forces the classifiers to focus on observations difficult to correctly classify. Adaboost uses the algorithm known as AdaBoost-SAMME [7]. This algorithm performed better than the Bayes classifiers, but was still on the lower end of performance.

Binary Logistic Regression models can only be used to distinguish between two different categories, such as "left" or "right". This algorithm attempts to find a relationship between features and the probability of a particular outcome. Differing from linear regression, logistic regression does not directly fit a straight line to data. Instead, it fits an S-shaped curve, called *Sigmoid*, to the observations. This model was in the mid-tier of performance, as it is a binary classification model, making it a sound choice for this classification task.

SVC is the implementation of the Support Vector Machine classifier using *libsvm* [3]. LinearSVC (Linear Support Vector Classification) is similar to SVC, but is implemented in terms of *liblinear* rather than *libsvm* [5]. Thus, it has more flexibility in the choice of penalties and loss functions and scales better to data sets containing large numbers of samples. NuSVC is similar to SVC but uses a parameter to control the number of support vectors. LinearSVC ended up being a high-performing model, but NuSVC and SVC were both too low, performance-wise, in early test runs to be considered. The fact that LinearSVC scales better to larger data sets was a contributing factor.

Decision Trees are non-parametric supervised learning methods used for classification and regression tasks. These classifiers can be thought of as flowchart-like structures in which the path taken from root node to leaf node represents classification rules. Each internal node in the tree represents a test on a feature and each leaf node represents a class label (the decision made after flowing down the tree and computing all features). This classifier was close to Bernoulli NB in accuracy, but was unnecessary to use on its own, as Random Forest, which utilizes multiple Decision Trees in its classification, is almost always more accurate.

Random Forest is a supervised machine learning algorithm based on ensemble learning, combining multiple algorithms of the same type (multiple Decision Tree classifiers). This results in a forest of trees, hence the name. The basic steps in the algorithm are as follows: pick *n* random records from the dataset, build a decision tree based on these records, choose the number of trees you want in your algorithm, repeat steps 1 and 2 until the desired number of trees have been made, assign the new record a category (in the case of a classification problem) that is based on the majority vote of what each tree in the forest predicted the record to be classified as. The classifier in this research utilized 200 Decision Trees and was one of the top performers. Although more

Decision Trees could have been used (resulting in an increase for runtime completion), the increase in accuracy was not significant: a model with 250 Decision Trees resulted in a performance increase of less than 1%.

SGD classifiers are linear classifiers with SGD (Stochastic Gradient Descent) training. By default, the model it fits is a linear Support Vector Machine (SVM). Gradient descent is an iterative algorithm that starts from a random point on a function and travels down its slope in steps until it reaches the lowest point of that function. This is useful in cases where optimal points cannot be found by equating the slope of the function to zero. Stochastic Gradient Descent includes randomness in the descent algorithm by randomly picking one data point from the dataset at each iteration to reduce the total number of computations required to complete the algorithm, making it more useful on larger data sets. This classifier was one of the top performers.

Once each individual classifier was assessed for performance, the top 3 performing classifiers were found (LinearSVC, SGD, and Random Forest). With these in mind, an ensemble learner was created called *VotingClassifier*, which combined the top 3 classifiers in an attempt to increase classification accuracy with a voting system. In theory, this custom-built classifier would provide the highest accuracy, as it would combine votes from all 3 classifiers, with a majority vote deciding the outcome of classification on entries in the dataset.

4.3 Training the Models

With the dataframe set up, data pre-processed, and classifiers decided on, models could be properly trained. Considering the resources available, different models were trained utilizing a different number of sources. Two good databases were found: HuffPost on the left and NewsMax on the right. There was plenty of data (~ 4,300 articles from each side) to train with, but only for these two sources. The next highest source contained around 2,000 articles to work with. To see what led to an increase in performance, different models were setup to train on data utilizing two sources, four sources, and six sources. The sources that had the highest number of unique articles were chosen to train on, as the more data to train on, the better.

Models which used two sources for training utilized HuffPost for the left and NewsMax for the right (~ 4,300 articles from each). Models which used four sources for training utilized HuffPost and Salon for the left and NewsMax and RedState for the right (~ 1,987 articles from each). Models which used six sources for training utilized HuffPost, Salon, and RawStory for the left and NewsMax, RedState, and Washington Examiner for the right. (~ 983 articles from each).

To begin training of models, the dataset was randomly shuffled each time before splitting 80% of the data into a training set and 20% into a testing set. This was so a new set of training and testing data could be used each time, thus preventing oddities or off-weeks of political news reporting from corrupting the dataset. Individual classifiers were trained on the data 30 times and the accuracies the classifier had against the testing sets after the 30 runs were averaged. Averages represented how accurate the classifiers were, as the idea behind the 30 runs was based on the Central Limit Theorem, which states that when there is a sufficiently large sample size, the sampling distribution of the mean for a variable will approximate a normal distribution regardless of that variable's distribution in the population. A sufficiently large sample size, according to statisticians, is typically of size 30 for most distributions [8].

During each separate run, each model was saved as a Pipeline object which does three things: turns text into a matrix of token counts using CountVectorizer(), transforms the count matrix into a normalized term frequency representation using TfidfTransformer(), and trains the classifier using the normalized tf-representation. The CountVectorizer() function is necessary to retrieve the number of occurrences for each unique token or word pair, while the TfidfTransformer() is important for transforming the number of occurrences for each unique token or word pair into their corresponding TF-IDF (Term Frequency-Inverse Document Frequency) weights.

Using IDF weights, all terms will not be considered equally important when assessing for relevance and meaning, whereas raw term frequency suffers from this dilemma. The idea behind IDF is to reduce the effect of terms that appear with high frequency in a collection of words when establishing meaningfulness and relevance. Thus, the IDF of rare terms is high, whereas the IDF for frequent terms is low. IDF weights are used to help indicate and extract important words from text which could have significant meaning or slant to them. Once IDF weights have been calculated, TF-IDF weights can be calculated by multiplying IDF weights by term frequency.

Algorithms rely on numerical features, which is why converting text into something the machine can understand is important in allowing complex machine learning tasks on different types of data. With CountVectorizer(), raw text is converted into a numerical vector representation of words and *n-grams*. N-grams are a set of words or sequential words within a body of text. From the sentence "*fox news said*", unigrams (individual words) would be “fox”, “news”, and “said”, bigrams (sequences of two words) would be “fox news” and “news said”, and the trigram would be “fox news said”. Going any higher than trigrams is typically not necessary, as the features that could possibly be extracted from higher n-grams would not provide much additional meaning.

For each article in the corpus of text, CountVectorizer() was used to pre-process and tokenize the data, as well as represent the entire vocabulary as a sparse matrix along the way. The function lowercased the text (so capitalized forms of words are not counted as separate words), used UTF-8 encoding, removed stopwords from being considered as features (using the built-in English stop word list from *sklearn*), performed tokenization through word level tokenization (converting text to smaller units of text, treating each word as a separate token), and ignored single characters during the tokenization process (words such as “a” and “I”).

The function was also used to remove words that either appear too frequently or too infrequently from being considered as features. After much testing, words appearing in 75% or more of documents available across the entire dataset were not considered as useful features, as they occur too frequently to provide any significant contextual meaning. Similarly, words appearing in four or less documents across the entire dataset were not to be considered, as these words occurred too infrequently, such as an individual’s name or the misspelling of a word. Vocabulary size was restricted to contain only the top 5,000 n-grams, to reduce runtime and keep terms which could be significant.

A key parameter of CountVectorizer() is *n_gram*, which can be used when the function is to consider unigrams, bigrams, or any size of n-gram. Using *n_gram* = (1, 3), the function will consider trigrams, bigrams, and unigrams when converting text into a sparse matrix representation. Likewise, *n_gram* = (1, 2) considers bigrams and unigrams, and the default (no parameter provided) considers only unigrams. All combinations were considered in the making of different models for keyword extraction purposes.

With the data pre-processed and transformed, TfidfTransformer() was used to convert the matrix created from CountVectorizer() into a matrix of TF-IDF features. IDF correlates to a weight of how common a word is seen; the more frequent the word is seen across documents, the lower the score, and the less important the word becomes. Words such as “the” will have a very low score, as they appear in almost all text and carry very little topic information. Figure 1 illustrates how IDF is computed.

$$IDF = log\frac{N}{DF_t}$$

Figure 1: IDF formula

N correlates to the total number of documents in your corpora of text, *DFt* is the number of documents containing the term *t*, and *t* is any word in the vocabulary. This formula is used to boost scores of words that are unique to a document in the hopes that high information words characterizing the document will surface and words that carry little weight to them will be suppressed.

IDF values start at 1.00 and proceed upwards. Once IDF values have been calculated, TF-IDF values can be calculated using the formula: *TF = TF * IDF* where *TF* stands for term frequency (number of times a word occurs in a piece of text). TF-IDF values start at 0.00 and proceed upwards. Table 1 showcases some of the top TF-IDF values across a model training session which utilized LinearSVC. The higher the value, the more significant the feature is considered to be.

Value	Feature
3.3251	dems
3.1868	healthcare
2.7198	democrat
1.8509	riots
1.3413	criminals

Table 1: Some TF-IDF values for right-wing documents

After the Pipeline object was created, the models were fit on the training data with the appropriate labels in the *pole* column of 0 indicating left and 1 indicating right-leaning articles. Once trained, models were tested upon entries within the *unfiltered_articles* column, as each entry correlated to the original articles that had no pre-processing performed on them, allowing models to be accurately tested once trained.

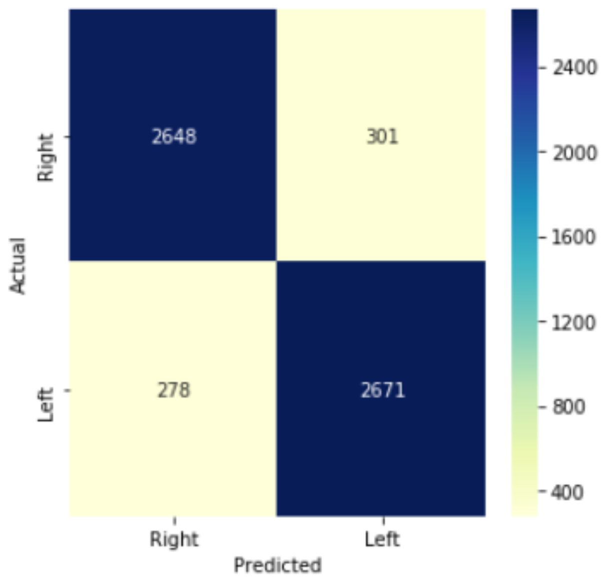
After a few training sessions, it became clear that some words needed to be removed from consideration by creating a custom word removal list. Presidential candidate names, variations on the current president's name, and the names of the political blogs and websites themselves were excluded from being considered as features.

Once models were re-trained, removal of proper nouns was undertaken to see whether an increase in performance occurred. This was done before lower-casing the entire text of each article, as proper nouns are detected mostly by seeing when text is in upper-case. These models were trained after removing the proper nouns within each article by utilizing *NLTK*'s POS tag of "NNP". A part-of-speech tagger, POS-tagger, processes a sequence of words and attaches a part-of-speech tag to each word; the POS tag "NNP" represents proper nouns.

Once these models were trained, lemmatization was undertaken to see whether an increase in classification accuracy occurred. With lemmatization, the idea is to reduce a given word to its root form (*plays*, *playing*, *played* => *play*; *am*, *are*, *is* => *be*). The root word is called a lemma and is the canonical form, dictionary form, or citation form of a set of words. This process helps with normalization of text. The WordNet Lemmatizer that uses the WordNet Database was utilized to look up lemmas of words.

4.4 Evaluating the Models

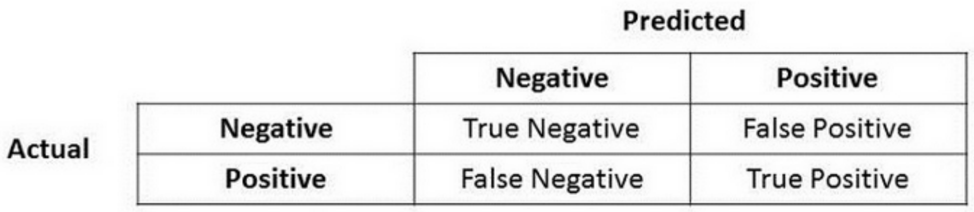
Once the average accuracy of each classifier was noted across 30 sequential runs, a confusion matrix was showcased along with the precision, recall, and f1-scores. The general idea behind a confusion matrix is to illustrate the number of times class A (left) was correctly classified as class A (left), the number of times class A (left) was mistakenly classified as class B (right), and vice versa. In Figure 3, the confusion matrix, along with its corresponding precision, accuracy, and f1-scores, illustrates the results of testing one of the models utilizing six sources against each article gathered from those six sources, keeping a balance of documents in mind.



	Precision	Recall	F1-score	Support
Right	0.90	0.90	0.90	2949
Left	0.90	0.91	0.90	2949
Accuracy			0.90	5898

Figure 3: Confusion Matrix and Corresponding Statistics.

Figure 3 illustrates a model that had 90% accuracy in classifying documents correctly. This particular model was tested only against the sources it was trained upon and not the entire dataset of 11 sources. 2,648/2,949 documents from the right were correctly classified as being from right-wing sources and 2,671/2,949 documents pertaining to the left were correctly classified as left-wing sources. In addition, 301/2,949 documents from the right were incorrectly classified as being left-wing sources, while 278/2,949 documents from the left were incorrectly labelled as being right-wing sources.



		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 4: Confusion Matrix, broken down.

Figure 4 showcases another way of looking at the layout of a confusion matrix. *Precision* refers to how precise the model is by looking at how many positive predictions actually are positive. The formula is: *Predicted = True Positive / (True Positive + False Positive).* This is a good measure to use when you do not want the costs of False Positives to be high (such as marking a non-spam email as spam).

Recall is calculated thusly: *Recall = True Positive / (True Positive + False Negative)*. This metric is used to select the best model when there is a high cost associated with False Negatives (such as sick patient detection).

F1-score is calculated as follows: *F1-Score = 2x (Precision*Recall) / (Precision + Recall)*. This metric is used when a balance is sought between precision and recall and there exists an uneven class distribution (large number of Actual Negatives).

Taking the best-performing model based on overall accuracy, the model was saved as a pickle file so it would not need to be re-trained every time it needed to be used to classify text. Instead, it could simply be loaded from the pickle file with the *.pkl* file type extension, now that it had been trained.

Different parameters considered for different model runs are thus: lemmatization vs. no lemmatization of text, unigram only vs unigram and bigram vs unigram, bigram, and trigram, and proper noun removal vs no proper noun removal. This resulted in 36 models being created for comparison across the 11 different news sources. Naming convention for different models followed the outline in Table 2.

Model Parameters	Descriptive properties
hsr_and_nrw	Left: HuffPost, Salon, RawStory Right: NewsMax, RedState, Washington Examiner
hs_and_nr	Left: HuffPost, Salon Right: NewsMax, RedState
h_and_n	Left: HuffPost Right: NewsMax
trigram	Trigrams, bigrams, and unigrams considered
bigram	Bigrams and unigrams considered
onegram	Unigrams considered
sr	Stopword removal
l	Lemmatization
nol	No lemmatization
pr	Proper noun removal
nopr	No proper noun removal

Table 2: Machine Learning Model Parameters and their meanings.

An example of one of the models saved as a pickle file (*.pkl*) would be "hsr_and_nrw_trigram_sr_nol_nopr.pkl". This means that: (1) The model trained on data from HuffPost, Salon, and RawStory (left sources) and NewsMax, RedState, and Washington Examiner (right sources). (2) Trigrams, bigrams, and unigrams were considered. (3) Stopwords were removed from the corpus before training. (4) No lemmatization was performed on the corpus. (5) Removal of proper nouns from the corpus (looking for words tagged with 'NNP') did not occur.

5. Results

Blog/Website	**Political alignment**	**#Posts**	**#Unique Posts**
HuffPost	Liberal	4314	4304
Talking Points Memo	Liberal	2820	1896
Salon	Liberal	2278	1987
AlterNet	Liberal	931	929
The Raw Story	Liberal	1159	1159
Slate	Liberal	1498	1156
National Review	Conservative	605	578
Red State	Conservative	2207	2111
The Bulwark	Conservative	1405	578
The Washington Examiner	Conservative	1104	983
NewsMax	Conservative	5637	5588

Table 3: Dataset Statistics

Table 3 indicates the dataset statistics for each source gathered. Something noteworthy for TalkingPointsMemo is that around 2,800 articles were gathered, but only 1,800 were unique; for Bulwark, around 1,400 articles were gathered, but only 578 were unique. With this much data being removed from the two sources for testing purposes, the balance between weeks of reporting was thrown off, thus creating corruption within the data sets. Therefore, these two sources were tossed from consideration when testing the overall accuracy of a model, as it became risky that the two data sets could skew or misrepresent a classifier's overall accuracy. Table 4 illustrates the results from testing against the entire data set of 11 sources, excluding TalkingPointsMemo and Bulwark, for accuracy once every model had been trained.

Left			Right			Attributes			Accuracy
Huffington Post	Salon	RawStory	NewsMax	RedState	Washington Examiner				
X	X	X	X	X	X	Lemmatization	Proper Noun Removal	Unigram	79.423
								Bigram	59.619
								Trigram	55.925
							No Proper Noun Removal	Unigram	82.596
								Bigram	82.675
								Trigram	82.522
						No Lemmatization	Proper Noun Removal	Unigram	81.099
								Bigram	80.947
								Trigram	80.853
							No Proper Noun Removal	Unigram	83.469
								Bigram	83.348
								Trigram	83.729
X	X		X	X		Lemmatization	Proper Noun Removal	Unigram	75.757
								Bigram	75.729
								Trigram	75.525
							No Proper Noun Removal	Unigram	76.983
								Bigram	78.046
								Trigram	77.934
						No Lemmatization	Proper Noun Removal	Unigram	76.314
								Bigram	77.064
								Trigram	76.857
							No Proper Noun Removal	Unigram	77.776
								Bigram	78.453
								Trigram	78.042
X			X			Lemmatization	Proper Noun Removal	Unigram	69.163
								Bigram	70.302
								Trigram	69.044
							No Proper Noun Removal	Unigram	69.764
								Bigram	70.811
								Trigram	58.886
						No Lemmatization	Proper Noun Removal	Unigram	70.855
								Bigram	69.260
								Trigram	69.798
							No Proper Noun Removal	Unigram	59.094
								Bigram	70.996
								Trigram	70.815

Table 4: All model accuracies against entire dataset.

The best performing model is listed as the 12th model in Table 4 at 83.729% accuracy. For the most part, models tended to increase in accuracy as they had more sources to train on. This makes sense, as the model is being exposed to a wider array of content from multiple sources, helping it to learn on more sources and increase its ability to generalize against future unknown sources. There were a few oddities in the 2nd and 3rd models indicated in Table 4 that used lemmatization along with proper noun removal, which significantly worsened the models that took from six news sources, compared to their four and two source counterparts. Something was causing noise when proper nouns were removed, as the 5th and 6th models indicated in Table 4 performed significantly better. This could mean the proper nouns that are being removed with these parameters are significant for text classification purposes, even though the goal of being able to generalize across different political timelines with different presidents and presidential candidates would prefer that proper nouns be removed.

Each of the 32 best-performing classifiers utilized the VotingClassifier, with the lowest 4 best-performers utilizing the RandomForest classifier. With the best-performing model known, the features it used were ready to be checked. Unfortunately, a Pipeline object classifier cannot be accessed to see what weights it is assigning to each of its 5,000 features. Therefore, some models needed to be trained utilizing the 3rd best option, LinearSVC, as RandomForest, along with the VotingClassifier, does not have a *coef_* function which can be utilized to show weights assigned to each feature.

The top 40 features across all 18 models (40 * 18 = 720) which removed proper nouns from consideration and utilized LinearSVC were researched for significant slant. Some of the features appeared multiple times as the top features across models; each unique feature was considered only once. Looking at the top overall features for each political pole gives the best idea of a grouping of words which could have significant slant or political charge to them. With these words in hand, frequency for each unique term was calculated across a balanced set of articles from HuffPost, Salon, RawStory, NewsMax, RedState, and Washington Examiner. Figure 5 showcases the results for the top feature frequencies (above a ratio of 1.50:1) for each political side. Unigrams, bigrams, and trigrams are present, with some words having their original frequencies scaled down to a threshold when generating the word cloud, so as not to overshadow other words (such as *right wing* correlating to an original frequency of 74:1 scaled down to 20:1).

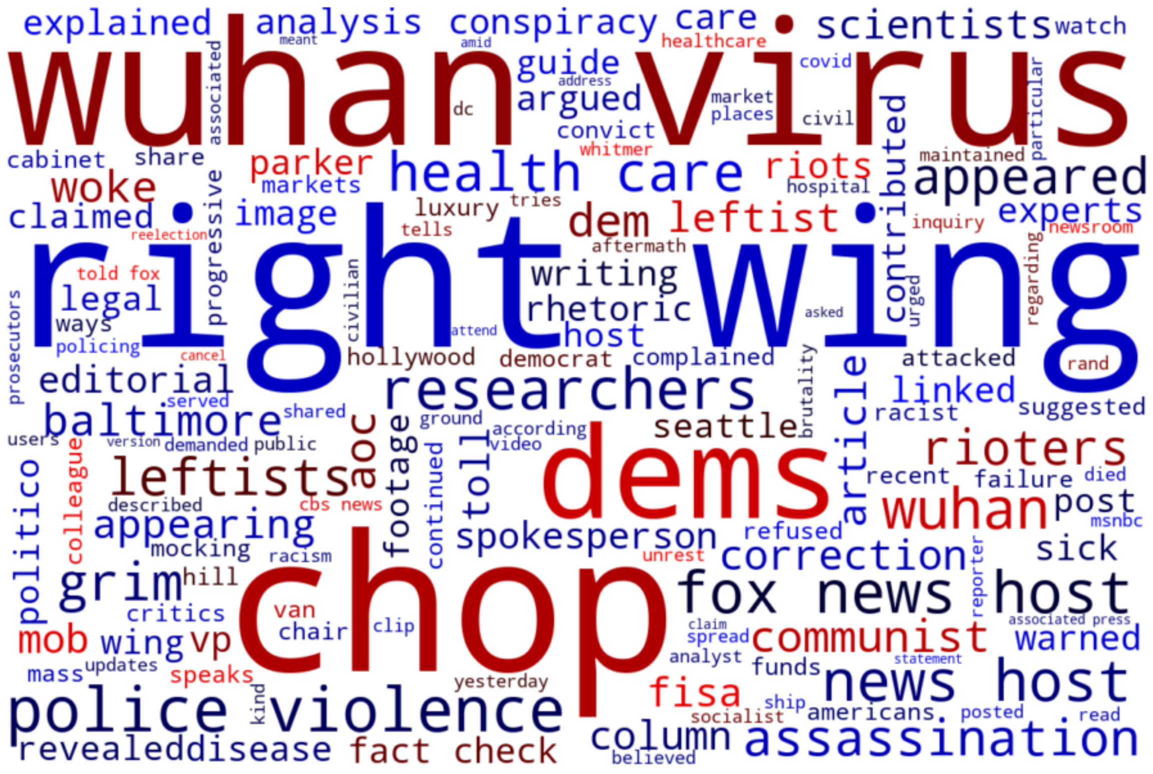


Figure 5: Word Cloud for the top 148 word frequencies (Right: Red, Left: Blue).

The existence of some of the top features in the findings, such as “healthcare” on the right, are discriminating and a good sign because that topic is one of which the right is usually more concerned with than the left (depending on the time period). The fact that the bigram “health care” is used more frequently by the left than the right here likely refers to articles that mention “health care officials”. However, a detailed analysis of the interplay between the different features is left for political scientists and beyond the scope of this thesis.

6. Conclusion and Future Work

This thesis proposed a framework for distinguishing and analyzing the language between differing political poles. The top-performing model having an accuracy of 83.729% is indicative that the model has somewhat successfully learned the language that the left and right utilize in their rhetoric. When models were trained by searching text for terms, some were found to be more discriminating and polarizing than others, with some indicative of significant slant. The top-performing model utilized the VotingClassifier, as expected, and was the proposed creation of a custom ensemble learner which used the three top-performing classifiers as a voting system for classification of documents.

Some limitations of this research are as follows: a limiting amount of articles were gathered. If the model had more data to train on, higher performance would result when classifying documents. With general classification purposes in mind for classifying text across different years, proper nouns were attempted to be removed as they are indicative of the time period (different presidents, etc.). However, keeping them in showcased an increase in accuracy, and the fact is, the language of the left and right has changed over the years, thus limiting the models' usefulness by time period. This is a very time-limited view of the vocabulary of the left and right, and the models trained may not be useful in 2, 5, or 10 years. Re-training the models with a new time period would be required to keep them up to date. Future work may involve training and testing new models with a larger database of articles with a new timeline.

Language is one of the most powerful tools, heavily influencing political decision-making. The set of measures used by the models trained are able to assist in studying the different facets of polarization through the lens of language, thus creating a clearer picture of the polarization that permeates public life. The proposed framework and results can be further used and interpreted by political scientists and communication scholars.

References

[1] Balasubramanyan, Ramnath, et al. "Modeling polarizing topics: When do different political communities respond differently to the same news?." *ICWSM*. 2012.

[2] Bird, Steven, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[3] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: A library for support vector machines." *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011): 1-27.

[4] Demszky, Dorottya, et al. "Analyzing polarization in social media: Method and application to tweets on 21 mass shootings." *arXiv preprint arXiv:1904.01596* (2019).

[5] Fan, Rong-En, et al. "LIBLINEAR: A library for large linear classification." *Journal of machine learning research* 9.Aug (2008): 1871-1874.

[6] Gentzkow, Matthew, and Jesse M. Shapiro. "What drives media slant? Evidence from US daily newspapers." *Econometrica* 78.1 (2010): 35-71.

[7] Hastie, Trevor, et al. "Multi-class adaboost." *Statistics and its Interface* 2.3 (2009): 349-360.

[8] Kwak, Sang Gyu, and Jong Hae Kim. "Central limit theorem: the cornerstone of modern statistics." *Korean journal of anesthesiology* 70.2 (2017): 144.

[9] Luntz, Frank. *Words that work: It's not what you say, it's what people hear*. Hachette UK, 2007.

[10] Tsur, Oren, Dan Calacci, and David Lazer. "A frame of mind: Using statistical models for detection of framing and agenda setting campaigns." *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015.

[11] Zafar, Muhammad Bilal, Krishna P. Gummadi, and Cristian Danescu-Niculescu-Mizil. "Message Impartiality in Social Media Discussions." *ICWSM*. 2016.

Appendix

Contractions and Characters Replaced

Characters to replace	Replacement
u.s.	united states
what's	what is
's	
've	have
're	are
'll	will
can't	can not
aren't	are not
couldn’t	could not
didn’t	did not
doesn’t	does not
don’t	do not
hadn't	had not
hasn’t	has not
haven't	have not
isn't	is not
shouldn’t	should not
wasn’t	was not
weren't	were not
won't	will not
wouldn’t	would not
mustn't	must not
i'm	i am