Paul Darnaby
ISYE 6740 Spring 2022
Final Project

Table of Contents

Problem Statement

News and information is spread at an incredibly fast rate that only increases with each passing day. Each time someone looks down at their phone, thousands of bits of information, and potentially misinformation, pass by each person in an instant. Within the past decade, the term "Fake news" has become fairly prominent due to the proliferation of baseless claims being spread at an alarming rate. The 2016 election was a recent case where there was a noticeable increase in awareness of this phenomenon. This recently became an issue of public health with widespread misinformation regarding the COVID-19 pandemic and the effects/efficacy of mRNA vaccines. At the start of the Ukrainian War there were wide reports of Russian also utilizing fake news even going so far as claiming that news reports about the war are merely media fabrications that include footage of interviews with "crisis actors." My goal with this project is to create an algorithm that can accurately classify these fake news sources as they arise.

While there are many different methods through which news is distributed, I will narrow the scope of my report to focus on text-based news. I will be utilizing a multilevel modeling approach through word vectorization and a collection of nonlinear classification techniques to identify fake and real news with the highest level of accuracy. Topic modeling will also be explored through the usage of Latent Dirichlet Allocation, to explore the process and examine the distribution of topics among fake and real news.

An additional aspect worth noting is the dimensionality of the text. Text data is inherently high-dimensional, especially once vectorized, for reasons I'll explain shortly. The main focus of this report will be finding a logical tradeoff between the computational expense of preprocessing steps and algorithms and model accuracy.

Data

Source

The Fake News dataset analyzed in this project was downloaded via the open-source data website, Kaggle. It consists of 20,800 articles posted during the months leading to the 2016 U.S. Presidential Election (exact dates were not divulged on the website by the provider of the data). The authenticity of each article was verified by journalists and given a classification as either real or fake.

While in my original proposal I mentioned the usage of another data source, the fake news net repository. I opted not to include it in this project. The Kaggle dataset was collected roughly 6 years ago, while fake news datasets was collected from the past couple years. Given how ephemeral news stories are, and the topics these stories span, it seems like this would be the best option to develop algorithms for fake and real news detection within a more condensed timeframe. Including articles that could be as much as 5 years apart seemed to unnecessarily introduce additional error/noise into text data that is already inherently noisy. This truncation of sources should ideally help with the exploration of topic modeling techniques as well. The sample size was small enough - there are only 1100 entries in that dataset, compared to the 20,800 in Kaggle's - that the impact on the resulting dataset should be minimal as well.

Methodology

Preprocessing Text Data

The original dataset provided four columns:

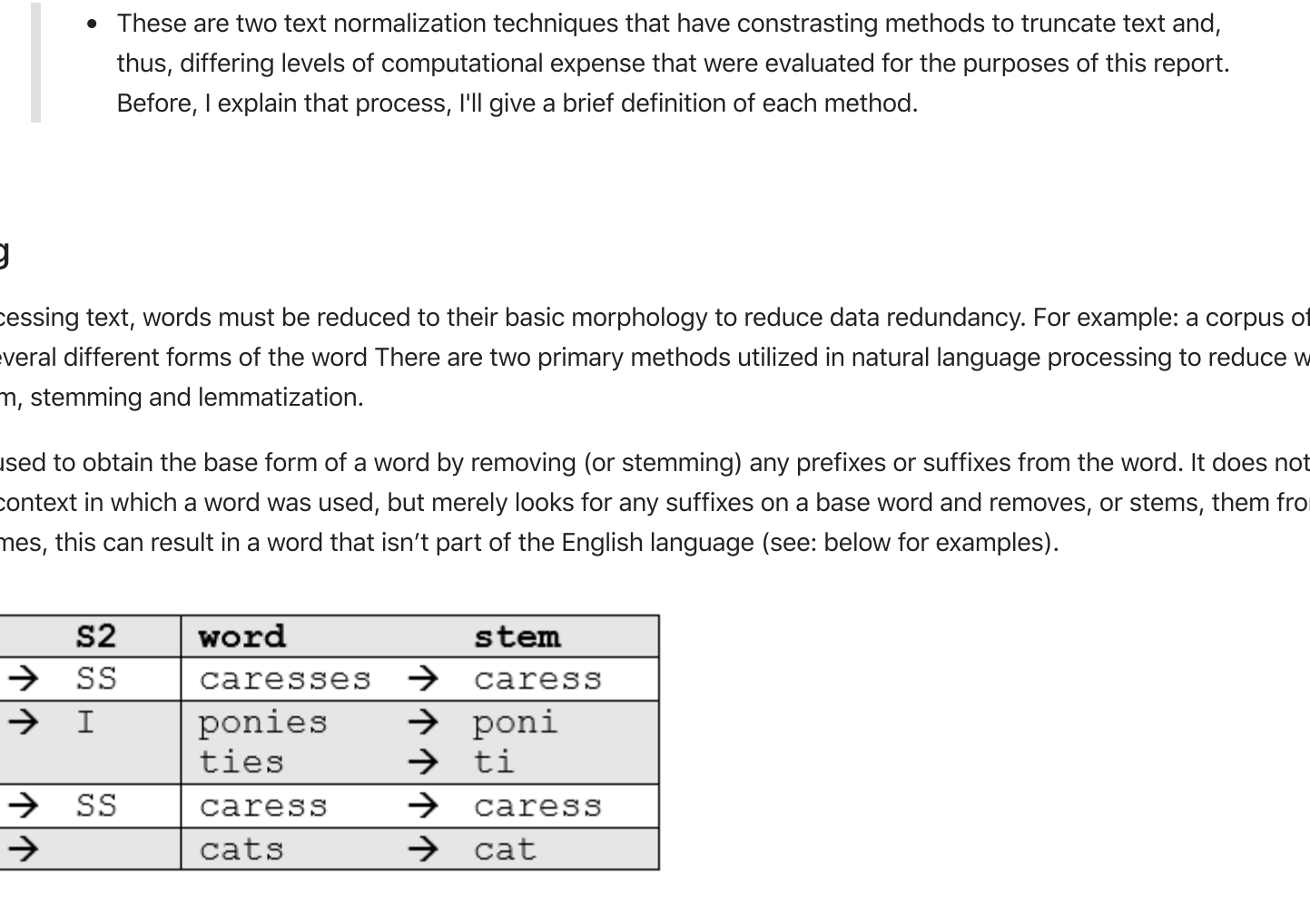
- *Title*: The title of the article
- *Author*: The author of the article
- *Text*: The body of the article
- *Label*: The classification as a fake or real article ((0: real, 1: fake))

	id	title	author	text	label
0	0	House Dem Aide: We Didn't Even See Comey's Let...	Darrell Lucus	House Dem Aide: We Didn't Even See Comey's Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

For the purposes of this analysis, a concatenation of the author and title was used in the subsequent models. The text body was disregarded due to the similarity/overlap within the article title and its body. The risk of duplicate entries of words could cause disproportionate value to certain text that could skew the analysis. Another consideration was limiting the dimensionality of the data by analyzing fewer words/features.

After the concatenation of author and title (entries I'll refer to as documents for the remainder of the report), further processing of the data is necessary before being fed to any models. The noise in the data (punctuation, similar words with different casings, etc.) must be handled. Therefore, the typical steps for text preprocessing were completed:

1. Tokenization
2. Lowercasing
3. Removal of Stopwords
4. Stemming/Lemmatization

> 1. Tokenization - breaking down each document into its component pieces, or tokens. See: below for a good example of this process on the phrase "We're Moving to L.A.!", compliments of Jose Portilla's NLP course on Udemy.
>
> 
>
>
> As can be seen there are several steps in this process:
>
> - Split on whitespace to individual words
> - Prefixes, or characters at the beginning of a word (e.g. "$,(etc.) are removed.
> - Suffixes, characters at the end of a word (e.g. `km) , . ! ") are removed
> - Infixes, like hyphens between words like bow-legged, are removed You'll also notice an exception to this rule in the image above where 'L.A.' is left whole despite the presence of two periods in it. The tokenization library in spacy has taken words like this into account to preserve their integrity.
>
> 1. Lowercasing of words
> 2. Removal of Stopwords (The full list of stopwords can be found here)
> 3. Stemming/Lemmatization
>
> > - These are two text normalization techniques that have constrasting methods to truncate text and, thus, differing levels of computational expense that were evaluated for the purposes of this report. Before, I explain that process, I'll give a brief definition of each method.

Stemming

Whenever preprocessing text, words must be reduced to their basic morphology to reduce data redundancy. For example: a corpus of text might have several different forms of the word There are two primary methods utilized in natural language processing to reduce words to their base form, stemming and lemmatization.

Stemming is used to obtain the base form of a word by removing (or stemming) any prefixes or suffixes from the word. It does not consider the context in which a word was used, but merely looks for any suffixes on a base word and removes, or stems, them from the word. Sometimes, this can result in a word that isn't part of the English language (see: below for examples).

S1		S2	word		stem
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
			ties	→	ti
SS	→	SS	caress	→	caress
S	→		cats	→	cat

The two most popular algorithms used for stemming are Snowball and Porter's. Differences between the two are deemed marginal at best among the industry, so Porter's was used due to its semi-relative simplicity. This was completed with the PorterStemmer() method from the nltk package in Python.

See: below for a comparison of the stemmed and regular version of the first entry in the fake news dataset. I should also note that the other steps taking place during the stemming process, namely:

- Removal of punctuation
- Conversion to lower case
- Tokenization (defined: above)
- Removal of stop words

Regular: Darrell Lucus House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It

Stemmed: darrel lucu hous dem aid didnt even see comey letter jason chaffetz tweet

It's clear to see that this method has some difficulty preserving the integrity of some of the original words (E.g. Lucus->lucu, house->hous,etc.). Whether that will have an impact on the accuracy of subsequent models is yet to be seen. It's also of note that the module spacy doesn't even include a stemming option in its library because they've deemed it irrelevant when compared to lemmatization. One attractive aspect to this process is the processing speed of the task, though. When running the algorithm on the 20,800 entries in the fake news dataset, it took an elapsed time of 68.93 seconds.

Lemmatization

In contrast to stemming, lemmatization looks beyond word reduction, and considers a language's full vocabulary to apply a morphological analysis to words. The lemma of 'was' is 'be' and the lemma of 'mice' is 'mouse'. Further, the lemma of 'meeting' might be 'meet' or 'meeting' depending on its use in a sentence.

Naturally, this process is more computationally expensive than stemming due to its relative sophistication. Again, we can look at the comparison in the first entry to its lemmatized counterpart:

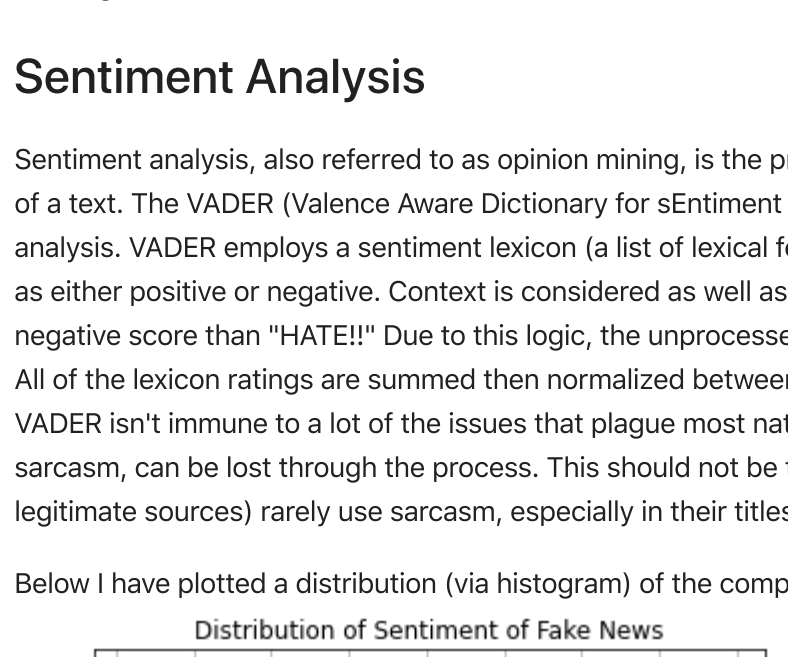
- Regular: Darrell Lucus House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It
- Lemmatized: darrell lucus house dem aide comey letter jason chaffetz tweeted

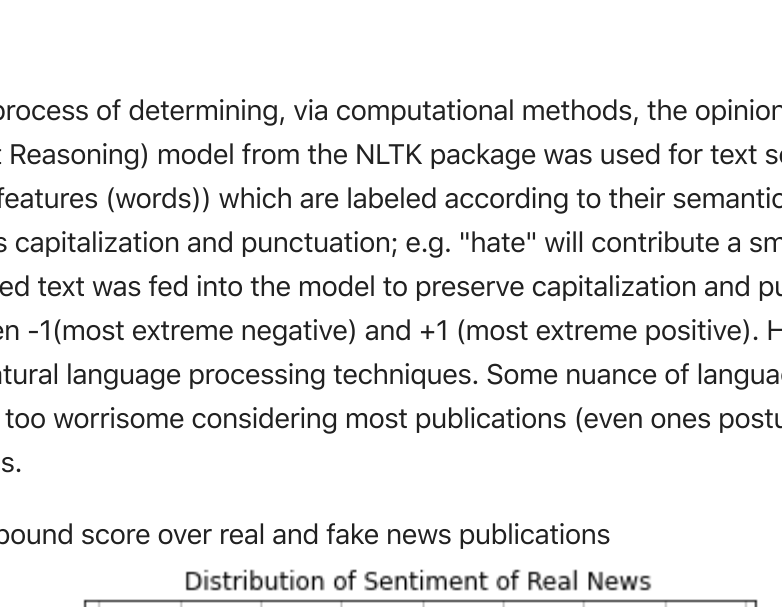
At first glance, this appears to have preserved more of the integrity of the original text than its stemmed counterpart. However, this comes with a significantly longer processing time as well. When the this algorithm is run over the dataset (again, through the nltk package) the elapsed time for running the lemmatization algorithm was 209.79 seconds. Almost 3 times as long as the stemming process.

EDA

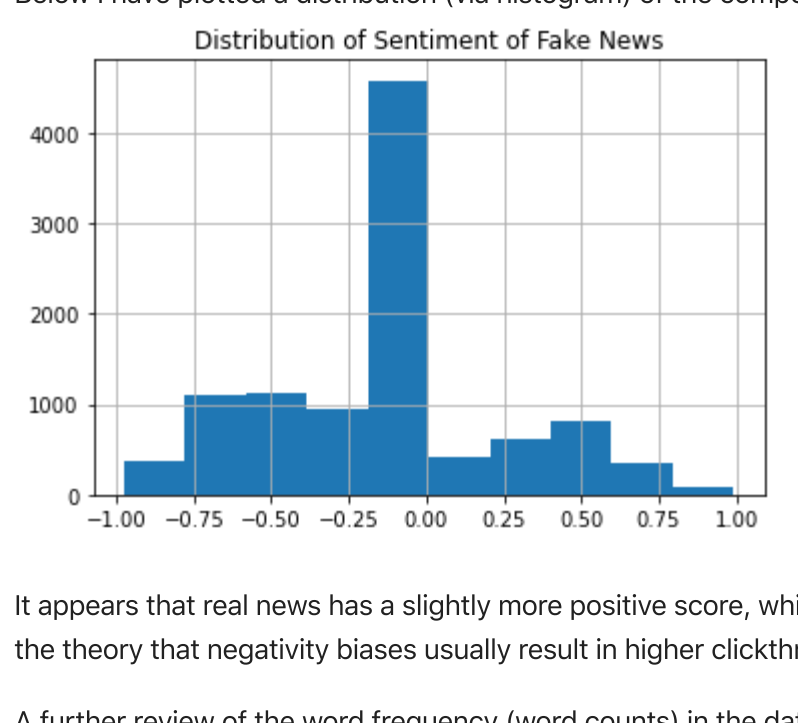
With the text preprocessing step now complete, the data is now in a condition more suitable for exploration and sentiment analysis.

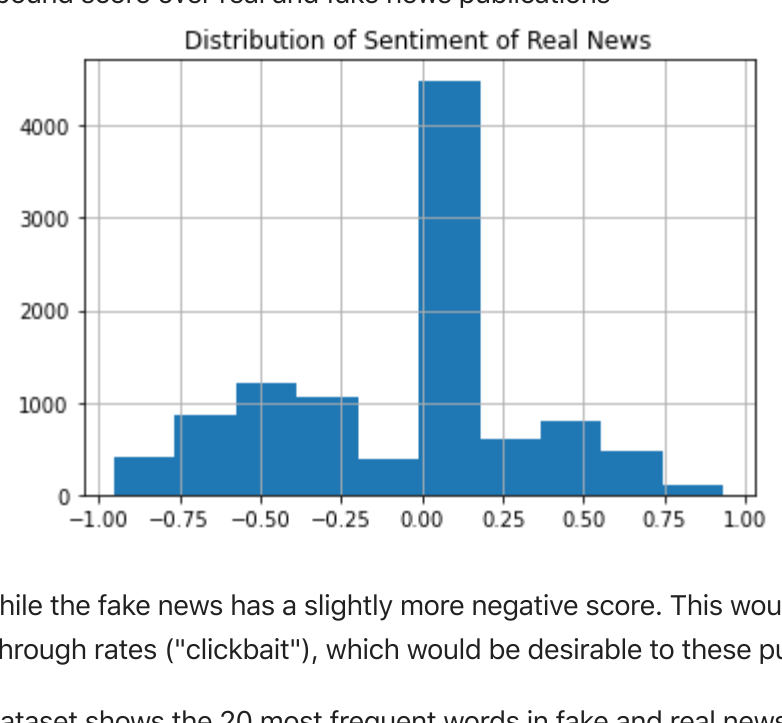
The distribution of length (number of words in a document) and punctuation (number of punctuation marks in each document) was plotted for the fake and real news





The fake news clearly has more punctuation inside it, which would align with clickbait wanting to invoke more emotion through the use of significant punctuation



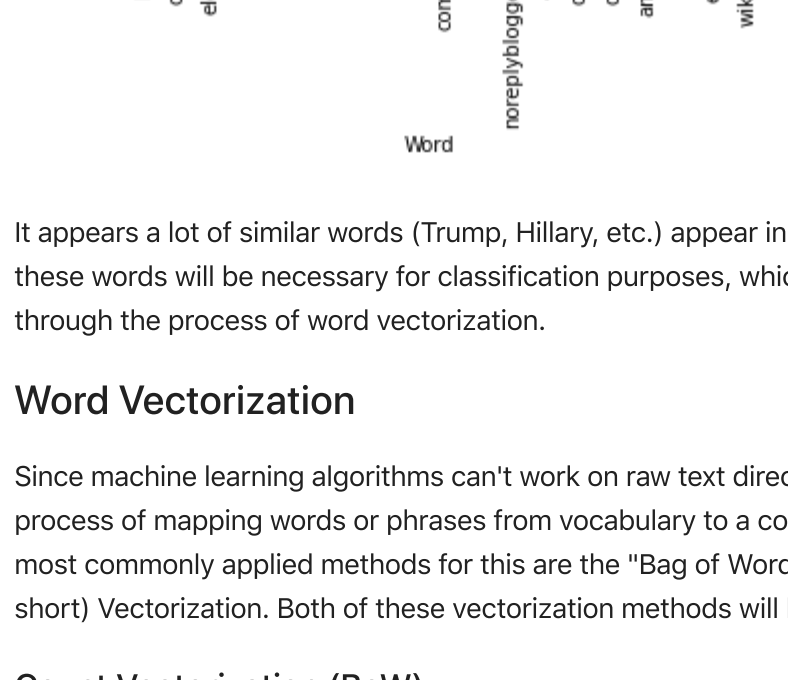


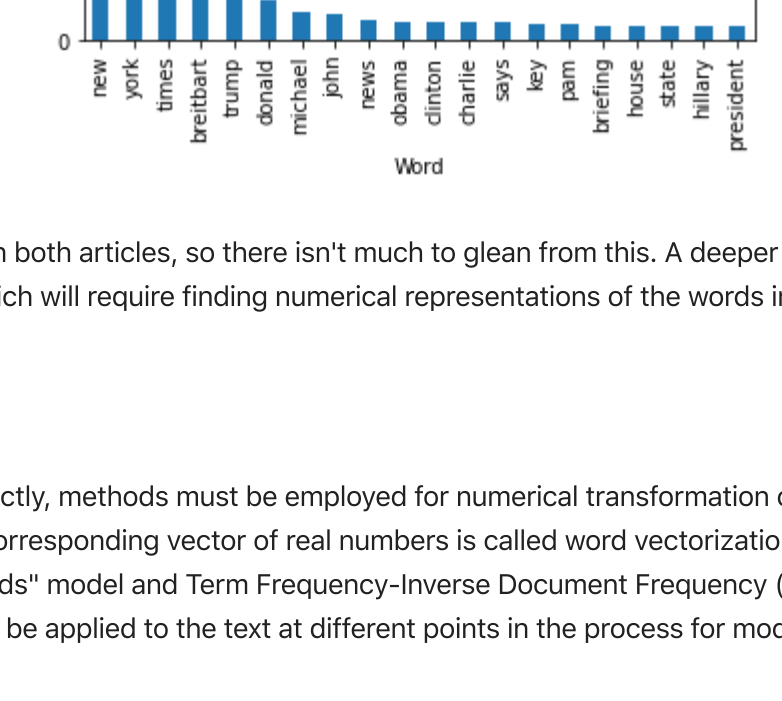
It also appears that fake news is generally longer than its real counterpart. It's interesting that the titles of fake news would be longer than real. Intuitively, once would think that getting a short, concise point across in a headline would lead to a higher level of clickthrough. Clicking

Sentiment Analysis

Sentiment analysis, also referred to as opinion mining, is the process of determining, via computational methods, the opinion or sentiment of a text. The VADER (Valence Aware Dictionary for sEntiment Reasoning) model from the NLTK package was used for text sentiment analysis. VADER employs a sentiment lexicon (a list of lexical features (words)) which are labeled according to their semantic oritentation as either positive or negative. Context is considered as well as capitalization and punctuation; e.g. "hate" will contribute a smaller negative score than "HATE!!" Due to this logic, the unprocessed text was fed into the model to preserve capitalization and punctuation. All of the lexicon ratings are summed then normalized between -1(most extreme negative) and +1 (most extreme positive). However, VADER isn't immune to a lot of the issues that plague most natural language processing techniques. Some nuance of language, like sarcasm, can be lost through the process. This should not be too worrisome considering most publications (even ones posturing as legitimate sources) rarely use sarcasm, especially in their titles.

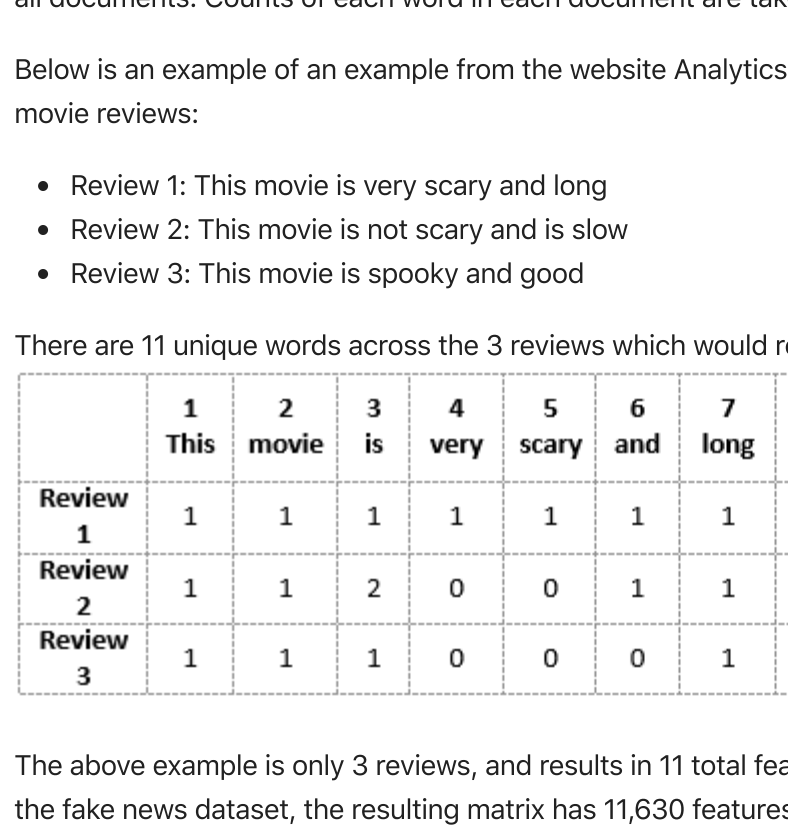
Below I have plotted a distribution (via histogram) of the compound score over real and fake news publications

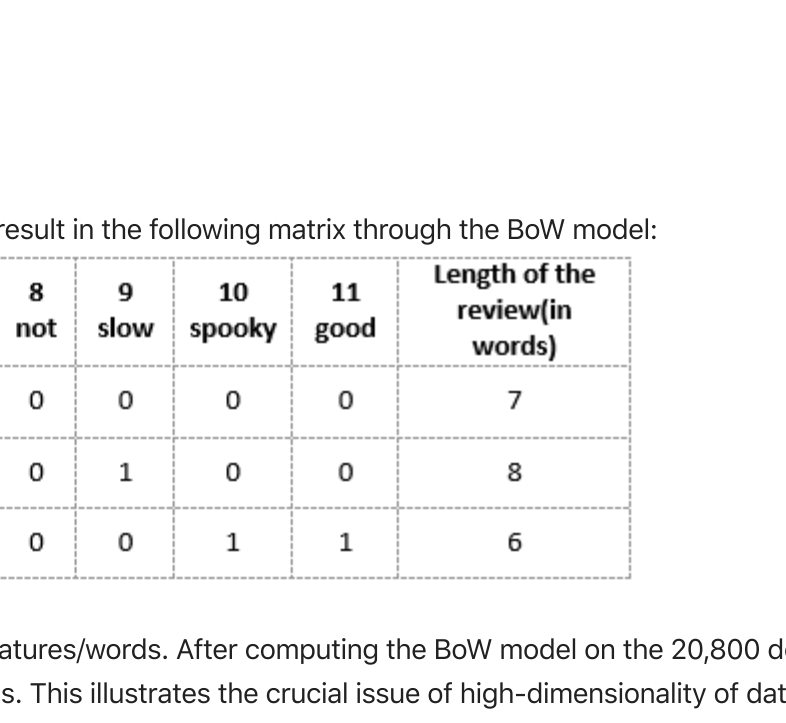




It appears that real news has a slightly more positive score, while the fake news has a slightly more negative score. This would align with the theory that negativity biases usually result in higher clickthrough rates ("clickbait"), which would be desirable to these publications.

A further review of the word frequency (word counts) in the dataset shows the 20 most frequent words in fake and real news publications.





It appears a lot of similar words (Trump, Hillary, etc.) appear in both articles, so there isn't much to glean from this. A deeper dive into these words will be necessary for classification purposes, which will require finding numerical representations of the words in this article through the process of word vectorization.

Word Vectorization

Since machine learning algorithms can't work on raw text directly, methods must be employed for numerical transformation of words. The process of mapping words or phrases from vocabulary to a corresponding vector of real numbers is called word vectorization. The two most commonly applied methods for this are the "Bag of Words" model and Term Frequency-Inverse Document Frequency (TF-IDF, for short) Vectorization. Both of these vectorization methods will be applied to the text at different points in the process for modeling.

Count Vectorization (BoW)

In the Count Vectorization (or the "Bag of Words") model, each document is represented by a "bag", or multiset of words. No consideration for grammar or context is taken into account in this instance. The number of features is the amount of unique words across all documents. Counts of each word in each document are taken.

Below is an example of an example from the website Analytics Vidhya that makes this point clear. Given the following three hypothetical movie reviews:

- Review 1: This movie is very scary and long
- Review 2: This movie is not scary and is slow
- Review 3: This movie is spooky and good

There are 11 unique words across the 3 reviews which would result in the following matrix through the BoW model:

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

The above example is only 3 reviews, and results in 11 total features/words. After computing the BoW model on the 20,800 documents in the fake news dataset, the resulting matrix has 11,630 features. This illustrates the crucial issue of high-dimensionality of data when processing text. The rate at which information (and misinformation) is published through the internet and methods such as Twitter magnifies the logistical and computation hurdles the must be considered when launching a model to analyze text.

TF-IDF

An issue with the CountVectorizer model above is its lack of consideration for the importance of words in a document set. Very common words in the entire corpus have a disproportionate weight appied to them. The TF-IDF algorithm addresses this problem

TF-IDF is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient). The equation is denoted below:

$$\text{tf}(t,d) * \text{idf}(t,D)$$

Below are definitions of the terms used in the equation:

- Term Frequency (TF): the amount of times a term (i) appears in a document (j) divided by the total number of words in the document.
    > - $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$
- Inverse Document Frequency (IDF): This is the This measures the importance of a term by calculating the logarithm of the ratio of number of documents to the number of documents with term (t). $\text{idf}(t,D) = log\frac{N}{|\{d \in D: t \in d\}|}$ If a word appears in all documents, its IDF score will be 0, while if it only appears in one document, it will have the highest possible IDF score.

While this algorithm accounts for the importance of each word in each document, it still doesn't account for context of the words. When conducted on our text data it left us with a matrix of dimensions 20800,11630. The TF-IDF model was employed on the text data prior to being fed into the models, due to its ability to preserve more of the context/importance of the text it analyzes.

Models Used

Given the nature of the binary classification problem (Real vs. Fake), 3 classification models were used on the vectorized text data for classification purposes:

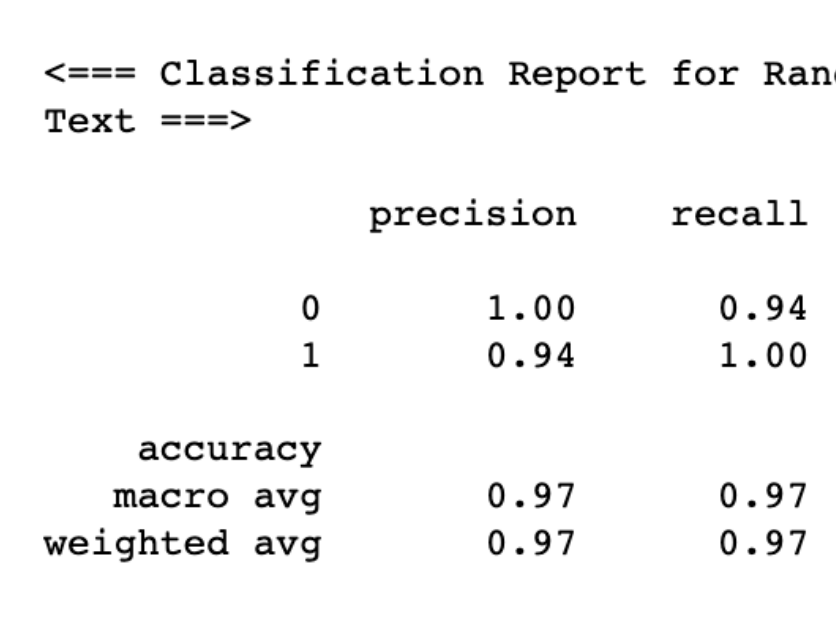
1. Random Forest Classifier
2. Logistic Regression
3. Multinomial Naive Bayes

These were chosen based on their general acceptance as being relatively balanced between accuracy and interpretability.

The data was given a train-test split of 80-20 for training and testing evaluation. Each model had 5-fold cross validation performed on it through the StratifiedKFold Module in sklearn. Hyperparameter tuning was also completed through the GridSearchCV() on the following condensed list (due to time/processing constraints) parameters for each model:

- Random Forest Classifier
    > - Criterion: Gini vs. Entropy
    > - Max Tree Depth: tuned over a range from 1 to 100 (step size:1)
- Logistic Regression
    > - Solver: Newton-cg, Limited-memory Broyden–Fletcher–Goldfarb–Shanno (lbfgs), liblinear
    > - Penalty: l1 (lasso) vs l2(ridge)
    > - C = tuned over a range from 1 to 1000
- Multinomial Naive Bayes
    > - Alpha: range from 0.1 to 1

Prior to determining the metrics that would be used for determining model performance, the distribution of responses was plotted to check whether it was a balanced or imbalanced dataset



Given the balanced dataset, accuracy will used as the performance metrics of the models. The accuracy of the model will be reported as the mean of the cross-validated scores for each fold of each model. The following metrics will also be evaluated:

1. Classification Metrics
2. Confusion Matrix
3. Processing Speed

The classification models were run on both stemmed and lemmatized text to compare the accuracy. Speed was not considered due to the fact that both the stemmed and lemmatized matrices will have almost identical dimensions, thus will have similar processing time.

Random Forest Classifier

Optimal Parameters Found Through Cross Validation

The best accuracy was achieved using the 'gini' criterion with a forest depth of 49 trees.

Accuracy

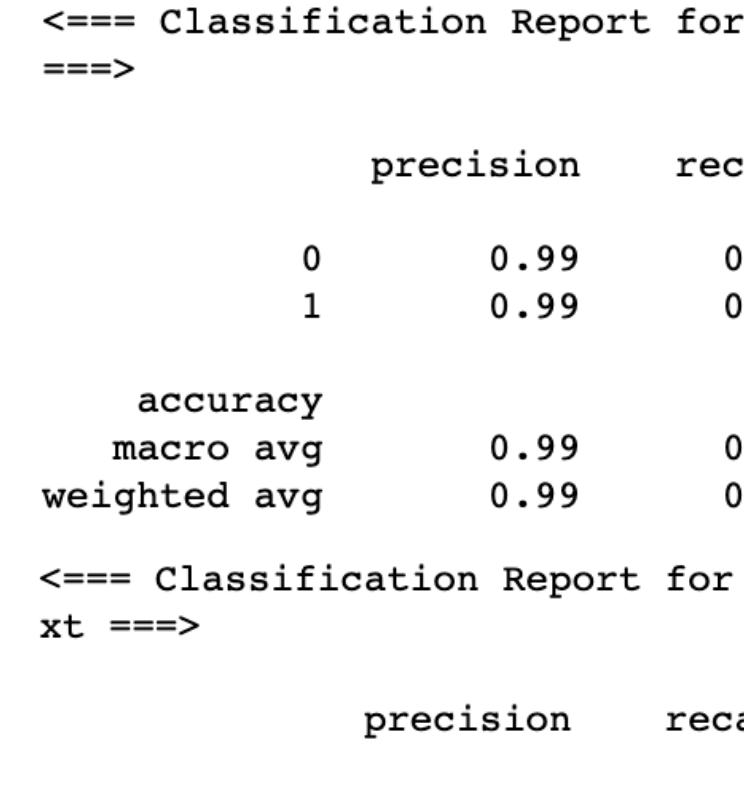
The accuracy score of the Random Forest Classifier model on stemmed text was 96.61% as compared to 96.96% on Lemmatized text.

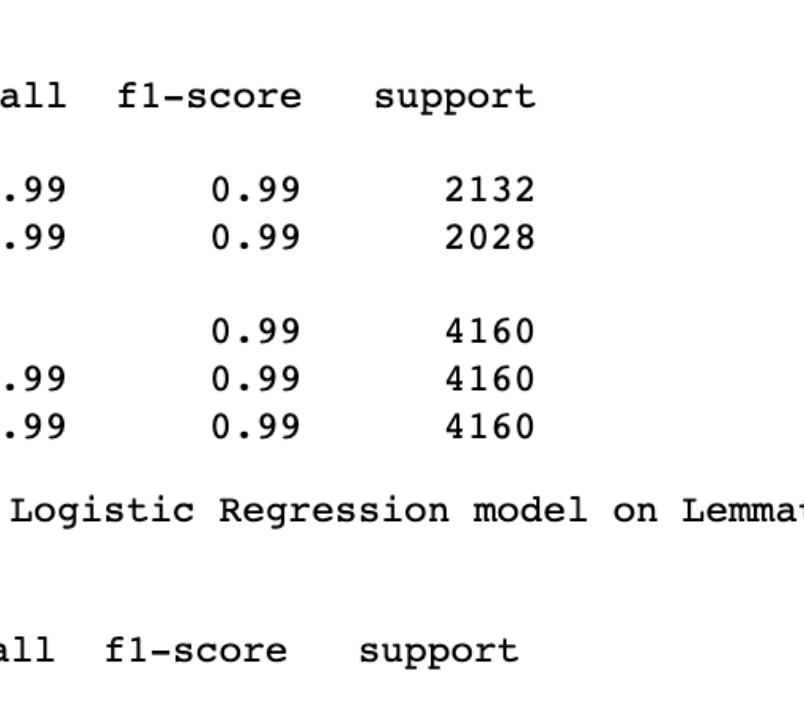
Classification Metrics

```
<=== Classification Report for Random Forest Classifier model on Stemmed
Text ===>

              precision    recall  f1-score   support

           0       1.00      0.94      0.97      2132
           1       0.94      1.00      0.97      2028

    accuracy                           0.97      4160
   macro avg       0.97      0.97      0.97      4160
weighted avg       0.97      0.97      0.97      4160
```

```
<=== Classification Report for Random Forest Classifier model on Lemmatiz
ed Text ===>

              precision    recall  f1-score   support

           0       1.00      0.94      0.97      2132
           1       0.94      1.00      0.97      2028

    accuracy                           0.97      4160
   macro avg       0.97      0.97      0.97      4160
weighted avg       0.97      0.97      0.97      4160
```

Confusion Matrix





Processing Speed

The time taken to run the Random Forest Classifier model on stemmed text was an average of roughly 270 seconds on both stemmed and lemmatized text.

Logistic Regression

Accuracy

The accuracy score of the Logistic Regression model on stemmed text was 99.2% as compared to 99.3% on lemmatized text.

Optimal Parameters Found Through Cross Validation

The best accuracy was achieved in the Logistic Regression model with a l2 (Ridge regression, squared regularization term) penalty and a C parameter of 1000.0 utilizing the 'newton-cg' solver.

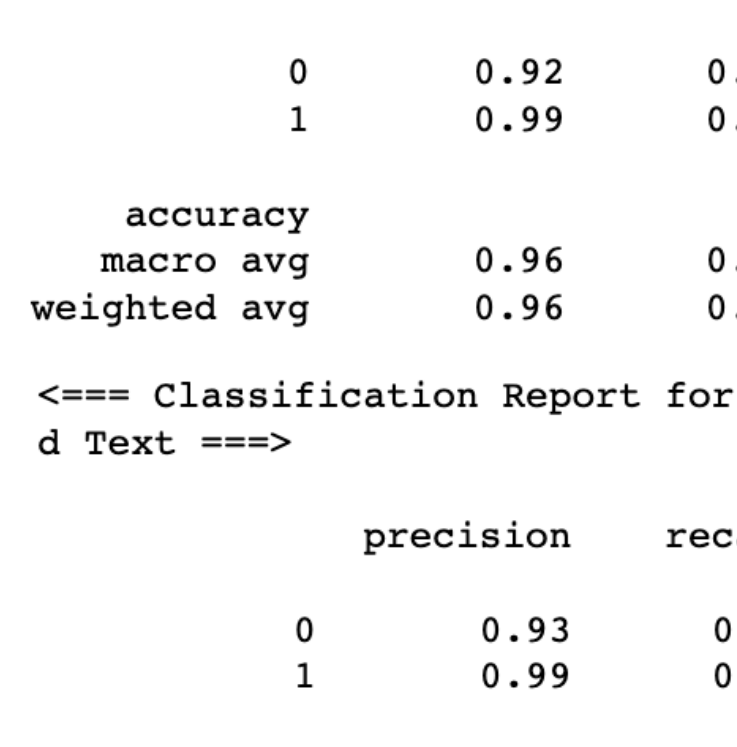
> - Interesting, because this would calculate the hessian of the gradient and is more computationally expensive due to this

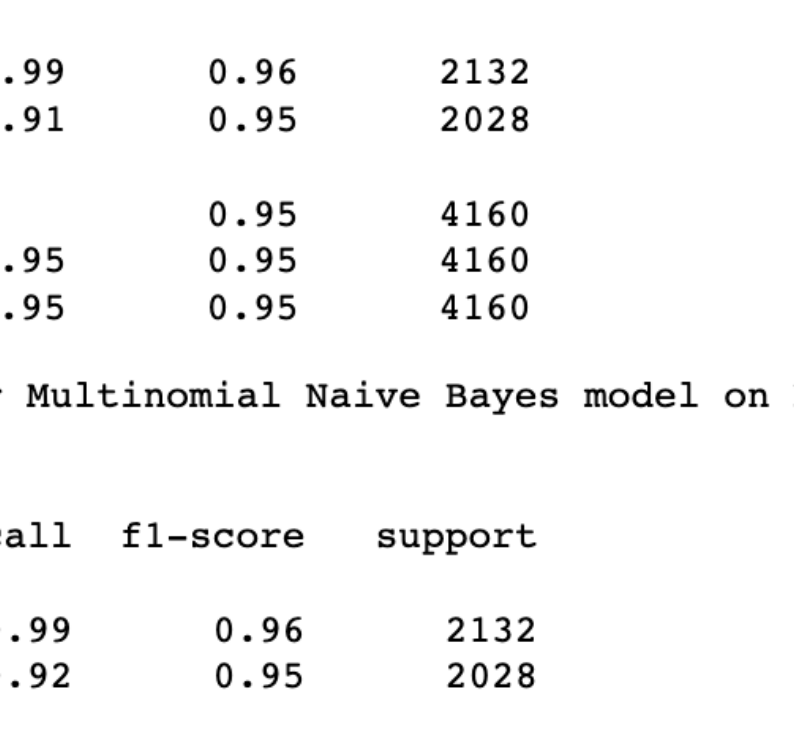
Classification Metrics

```
<=== Classification Report for Logistic Regression model on Stemmed Text
===>

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2132
           1       0.99      0.99      0.99      2028

    accuracy                           0.99      4160
   macro avg       0.99      0.99      0.99      4160
weighted avg       0.99      0.99      0.99      4160
```

```
<=== Classification Report for Logistic Regression model on Lemmatized Te
xt ===>

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2132
           1       0.99      0.99      0.99      2028

    accuracy                           0.99      4160
   macro avg       0.99      0.99      0.99      4160
weighted avg       0.99      0.99      0.99      4160
```

Confusion Matrix





Processing Speed

The time taken to run the Logistic Regression model on stemmed text was 24.23 seconds as opposed to 23.85 seconds on the lemmatized text.

Multinomial Naive Bayes

Accuracy

The accuracy score of the Multinomial Naive Bayes model on stemmed text was 95.3% as compared to 95.52% on lemmatized text.

Optimal Parameters Found Through Cross Validation

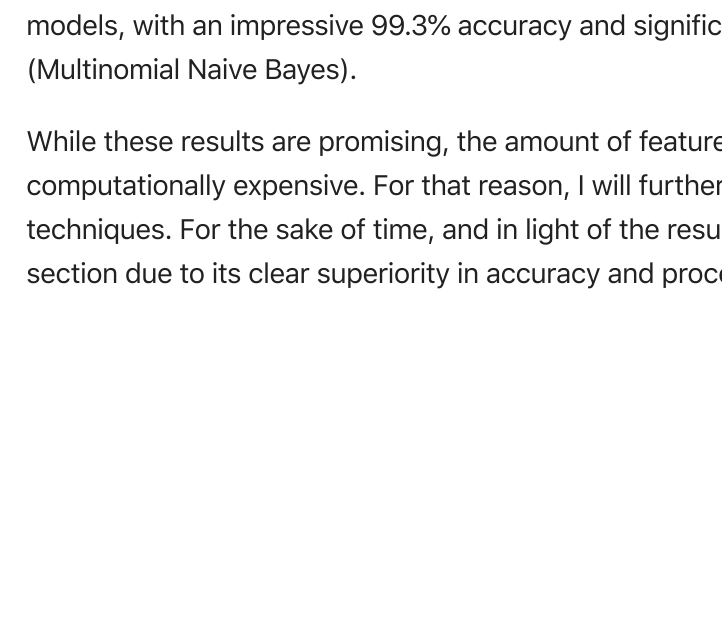
The best accuracy for the Multinomial Naive Bayes model was achieved with an alpha parameter of 0.7061224489795919.

Classification Metrics

```
<=== Classification Report for Multinomial Naive Bayes model on Stemmed T
ext ===>

              precision    recall  f1-score   support

           0       0.92      0.99      0.96      2132
           1       0.99      0.91      0.95      2028

    accuracy                           0.95      4160
   macro avg       0.96      0.95      0.95      4160
weighted avg       0.96      0.95      0.95      4160
```

```
<=== Classification Report for Multinomial Naive Bayes model on Lemmatize
d Text ===>

              precision    recall  f1-score   support

           0       0.93      0.99      0.96      2132
           1       0.99      0.92      0.95      2028

    accuracy                           0.96      4160
   macro avg       0.96      0.96      0.96      4160
weighted avg       0.96      0.96      0.96      4160
```

Confusion Matrix



Processing Speed

The time taken to run the Multinomial Naive Bayes model on stemmed text was 59.08398914337158 seconds as compared to 45.1137580871582 seconds for the lemmatized text.

Model Results

After computing the accuracy over each model on stemmed vs lemmatized text, we're left with the following results.

	Model Accuracy on full TF-IDF matrix		
	Random Forest	Logistic Regression	Multinomial Naive Bayes
Stemmed	96.61%	99.2%	95.3%
Lemmatized	96.95%	99.3%	95.5%

So there is a marginal increase in accuracy across all models when using lemmatized text, but not much worth noting. Also, an improvement in all metrics in the confusion matrix: precision, recall, and F1 score. Logistic Regression was the clear winner across all models, with an impressive 99.3% accuracy and significantly faster processing speed - almost 2x faster than the next closest algorithm (Multinomial Naive Bayes).

While these results are promising, the amount of features in the dataset (11,630) still leaves a fairly high-dimensional task that is computationally expensive. For that reason, I will further evaluate the complexity-accuracy tradeoff through dimensionality reduction techniques. For the sake of time, and in light of the results from the previous models, I've only utilized logistic regression in the following section due to its clear superiority in accuracy and processing speed when compared to Random Forest and Naive Bayes.

Dimensionality Reduction

With the feature count of the TF-IDF matrix being 11630, a logical step would be to reduce the dimensionality of our data to improve processing speed. This can also have an additional benefit of removing some of the noise from our data, an issue that can plague text classification algorithms. Matrix factorization can be utilized via Singular Value Decomposition, due to the rectangular shape of the TF-IDF matrix. This can be truncated to achieve a desired rank level that preserves the "essence" of the data. The process for such is as follows:

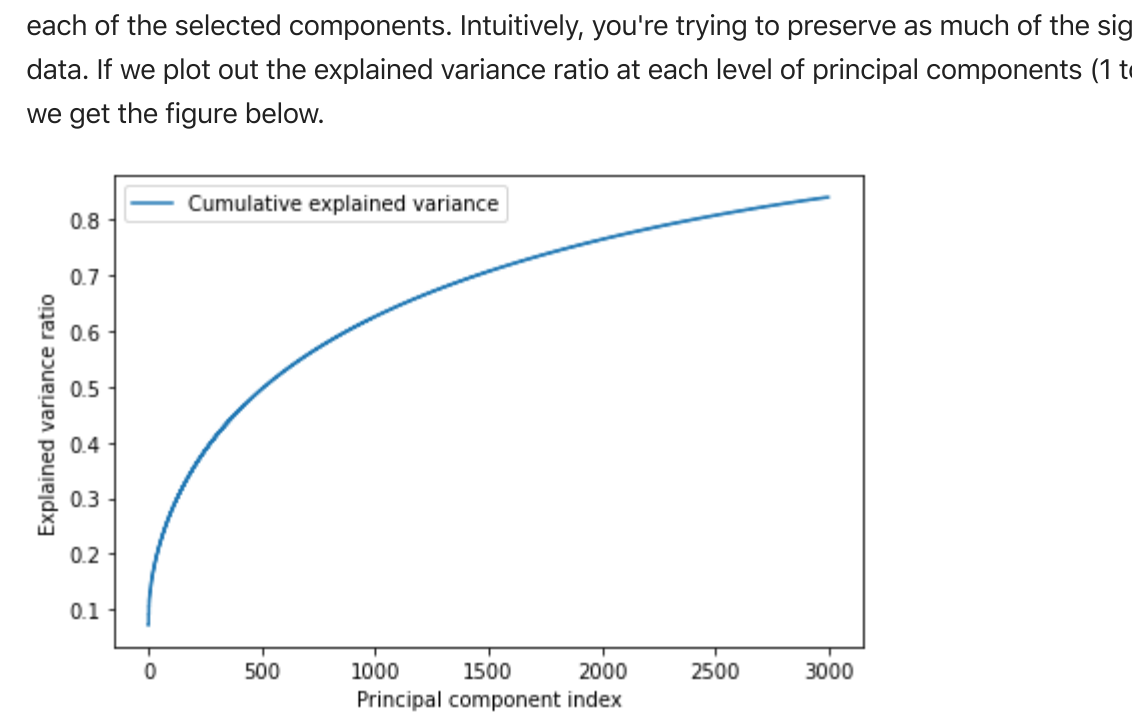
- Let X be our rectangular TF-IDF matrix with dimensions (mxn)
- The SVD of the mxn matrix can be denoted as follows:

$$X = U\Sigma V^*$$

- Once the X matrix has been factorized to these components, you can truncate the $U\Sigma V^*$ matrices appropriately to a lower rank, r, after the appropriate number of components has been decided

$$X \approx U_r \Sigma_r V_r^*$$

The logical question at this point is what is the ideal number of components? Traditionally, heuristics and naive methods have been utilized to identify the ideal number of components. Explained variance can be utilized to identify the appropriate number of components to keep in the $U\Sigma V^*$ matrices. A relatively naive method for finding the optimal number of components is finding the threshold where at least 70-80% of the variance is explained in the model. The explained variance ratio is the percentage of variance that is attributed by each of the selected components. Intuitively, you're trying to preserve as much of the signal in the data while reducing the size of the data. If we plot out the explained variance ratio at each level of principal components (1 to 3,000 was due to computational constraints), we get the figure below.



This would put us at somewhere between 1500 to 2500 components in our truncation, which would be about an 8-10 fold reduction in features. This is far from marginal, but there is also a more scientific way to How do we choose the optimal number of components (r)

I've ran the logistic regression model on the TF-IDF matrix that has been factorized through SVD and set to rank 1500 and 2500. This resulted in accuracies

- r=1500: 98.6% accuracy
- r=2500: 98.7% accuracy

We essentially lose 0.5% accuracy shifting down from full dimensionality to a lower rank (1500-2500).

The issue with this technique is it comes down to a "look-and-feel" approach that is semi-subjective and relies on rules of thumb.. In 2014, a paper published by Gavin and Donaho developed a new technique to determine the appropriate value for r called Optimal Singular Value Hard Threshold (optimal SVHT). The goal is to find a value, τ that is the Singular Value Hard Threshold (SVHT). This tells us where exactly to truncate our data. There are two methods to utilize for this technique, one where the noise level, σ is known and one where it is unknown. Since the noise in unknown in this instance, we will follow that method. The calculation for approximating τ ($\hat{\tau}_*$) when the noise level σ is unknown is as follows:

$$\hat{\tau}_* = \omega(\beta) * y_{med}$$

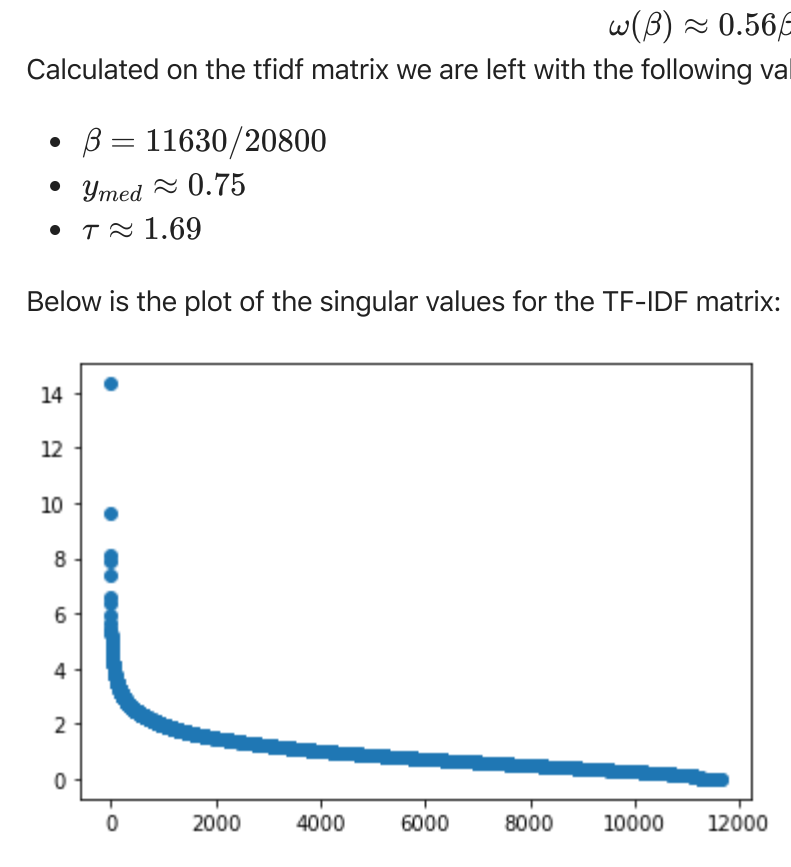
where $\beta = m/n$ and y_{med} is the median singular value of the matrix X. And $\omega(\beta)$ can be approximated with the equation:

$$\omega(\beta) \approx 0.56\beta^3 - 0.95\beta^2 + 1.82\beta + 1.43$$

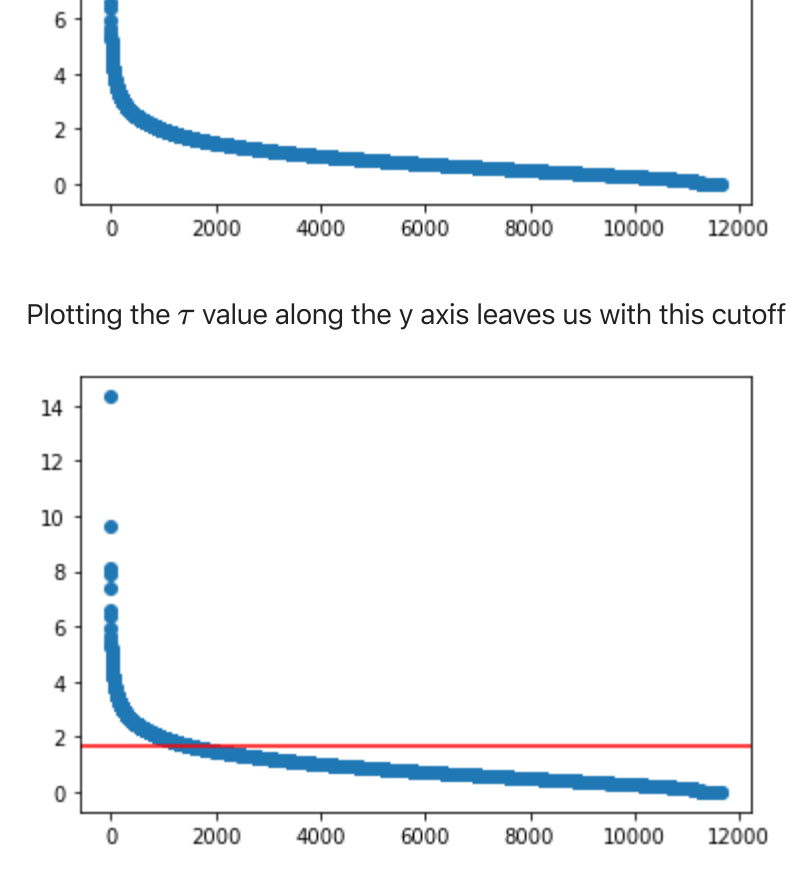
Calculated on the tfidf matrix we are left with the following values:

- $\beta = 11630/20800$
- $y_{med} \approx 0.75$
- $\tau \approx 1.69$

Below is the plot of the singular values for the TF-IDF matrix:



Plotting the τ value along the y axis leaves us with this cutoff for singular values



The logic being any values below that τ threshold are essentially noise in the data. Utilizing this logic, we're left with 1474 components, which is pretty close to the 1500 we used from the heuristic. Sure enough, the accuracy for the model with 1474 components comes out to approximately 98.6% accuracy as well. This is only a .6% reduction in accuracy after reduction in features by 88%. That's a fairly impressive retention of the signal in the data for such a large reduction in data size.

Latent Dirichlet Allocation

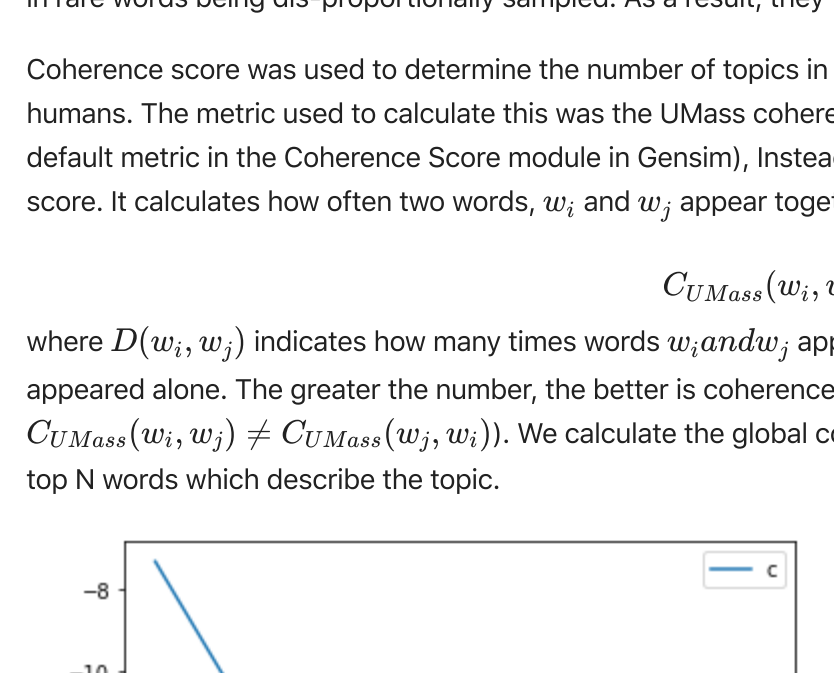
LDA is an unsupervised machine learning technique that is used to group documents across a large collection according to similar terminology used in them. LDA's approach to topic modeling is it considers each document as a collection of topics in a certain proportion. And each topic as a collection of keywords, again, in a certain proportion. Once you provide the algorithm with the number of topics, all it does it to rearrange the topics distribution within the documents and keywords distribution within the topics to obtain a good composition of topic-keywords distribution.

CountVectorizer was used on the data since LDA is based on term count and document count. Fitting LDA with TfidfVectorizer will result in rare words being dis-proportionally sampled. As a result, they will have greater impact and influence on the final topic distribution

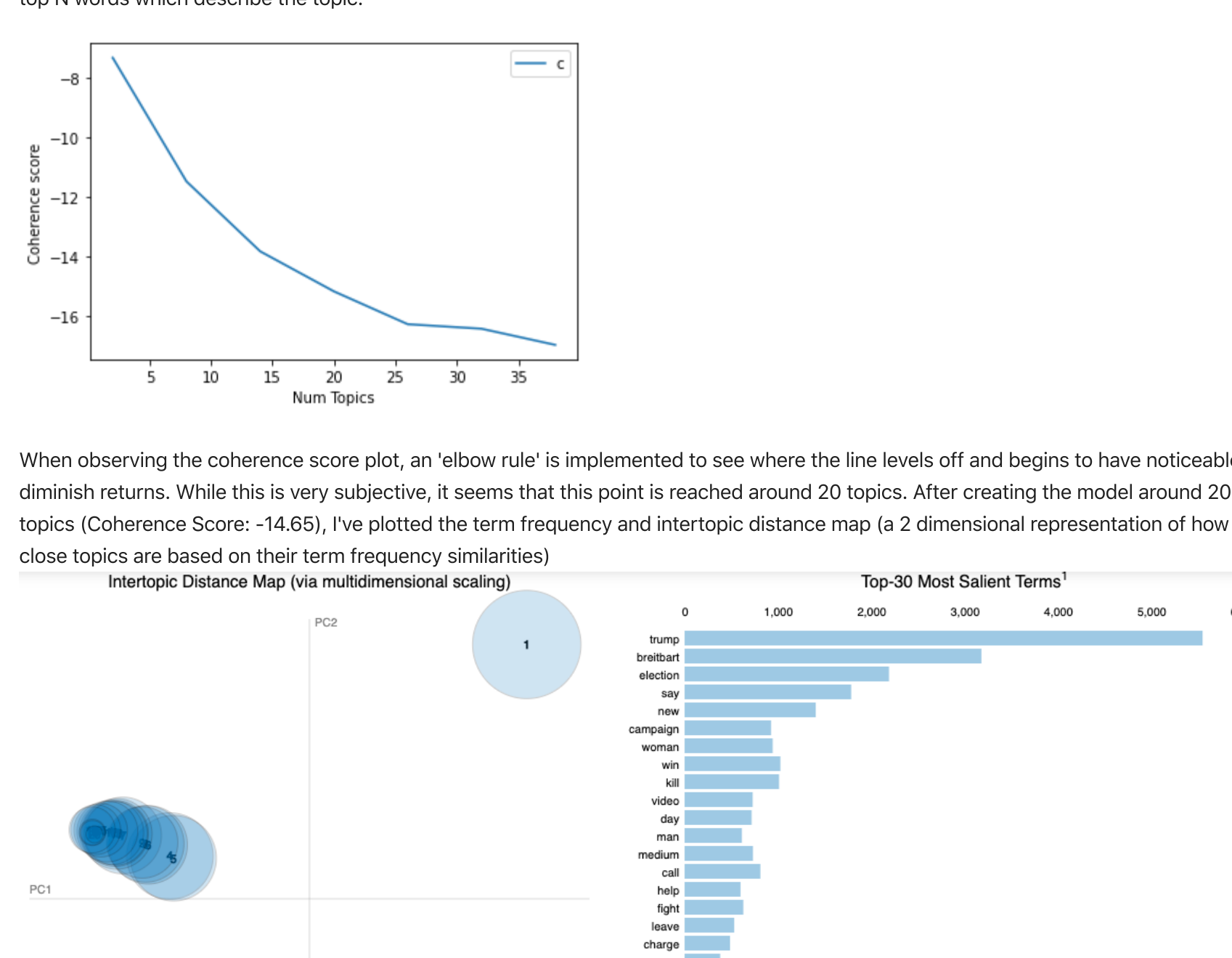
Coherence score was used to determine the number of topics in the model. This is used to measure how interpretable topics are to humans. The metric used to calculate this was the UMass coherence score. While there is a preference among most to utilize cv (it is the default metric in the Coherence Score module in Gensim), Instead of using the CV score, we recommend using the UMass coherence score. It calculates how often two words, w_i and w_j appear together in the corpus and it's defined as

$$C_{UMass}(w_i, w_j) = \log \frac{D(w_i, w_j)+1}{D(w_i)},$$

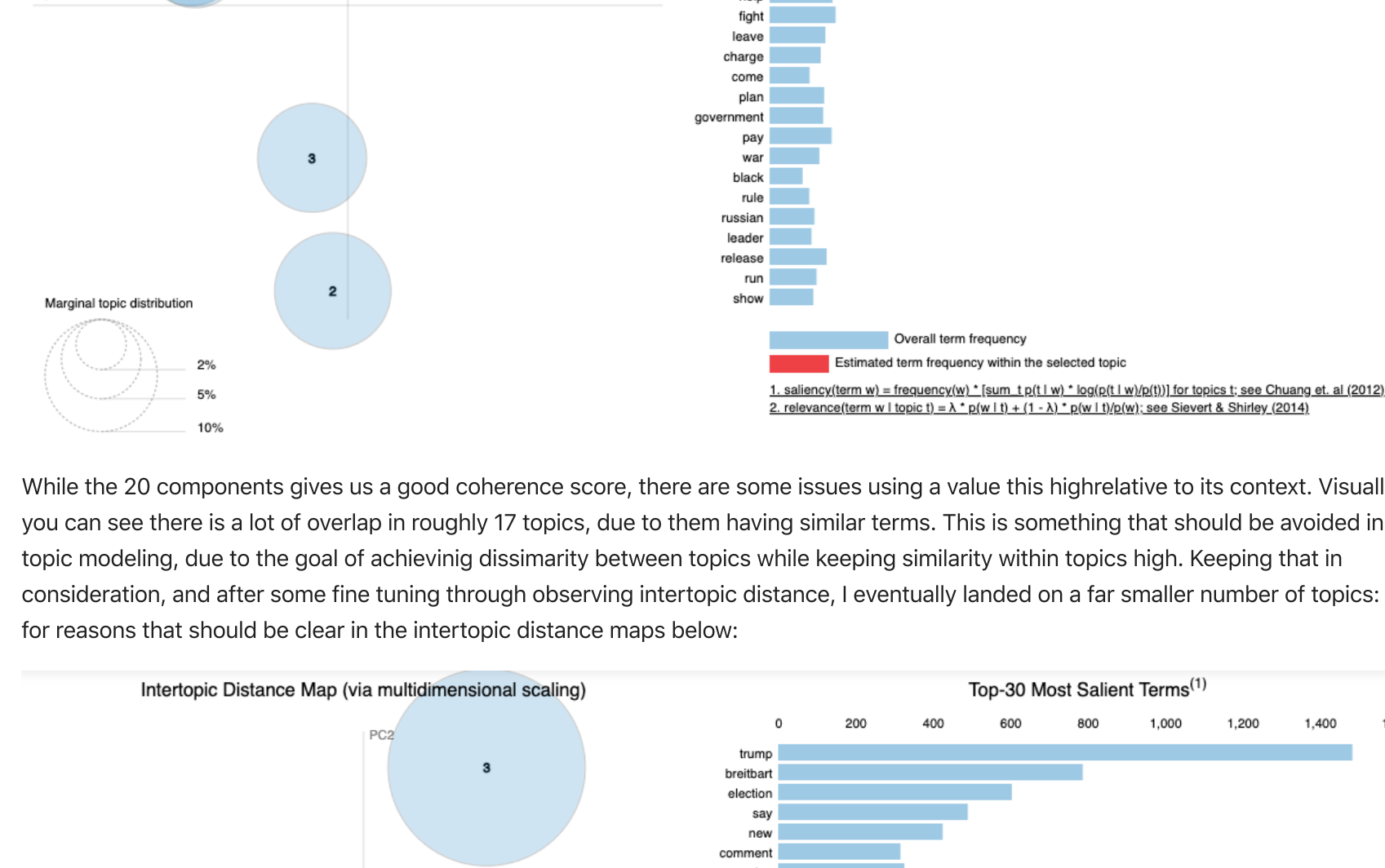
where $D(w_i, w_j)$ indicates how many times words $w_i and w_j$ appear together in documents, and $D(w_i)$ is how many time word w_i appeared alone. The greater the number, the better is coherence score. Note: this measure isn't symmetric($C_{UMass}(w_i, w_j) \neq C_{UMass}(w_j, w_i)$). We calculate the global coherence of the topic as the average pairwise coherence scores on the top N words which describe the topic.

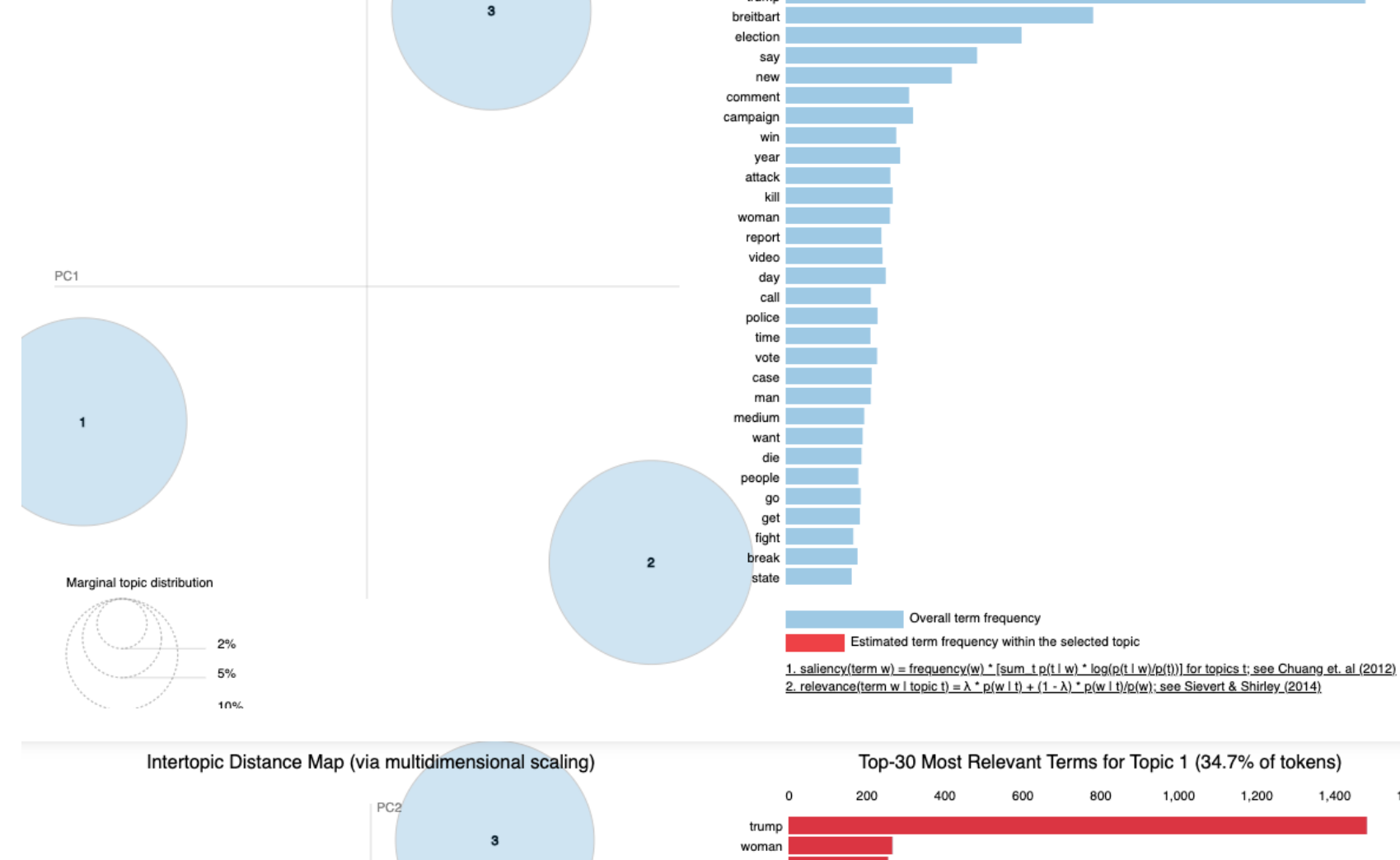


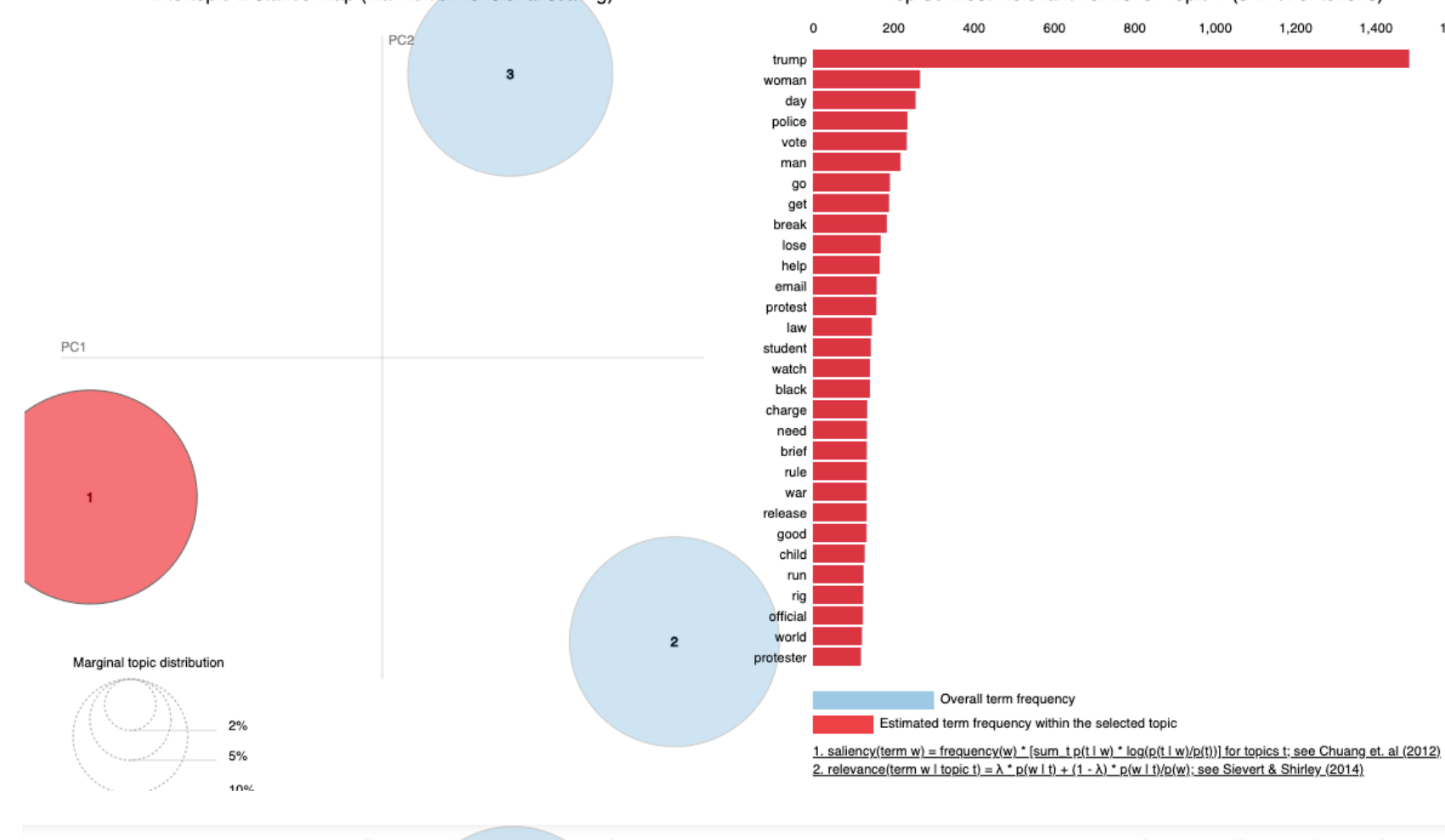
When observing the coherence score plot, an 'elbow rule' is implemented to see where the line levels off and begins to have noticeable diminish returns. While this is very subjective, it seems that this point is reached around 20 topics. After creating the model around 20 topics (Coherence Score: -14.65), I've plotted the term frequency and intertopic distance map (a 2 dimensional representation of how close topics are based on their term frequency similarities)

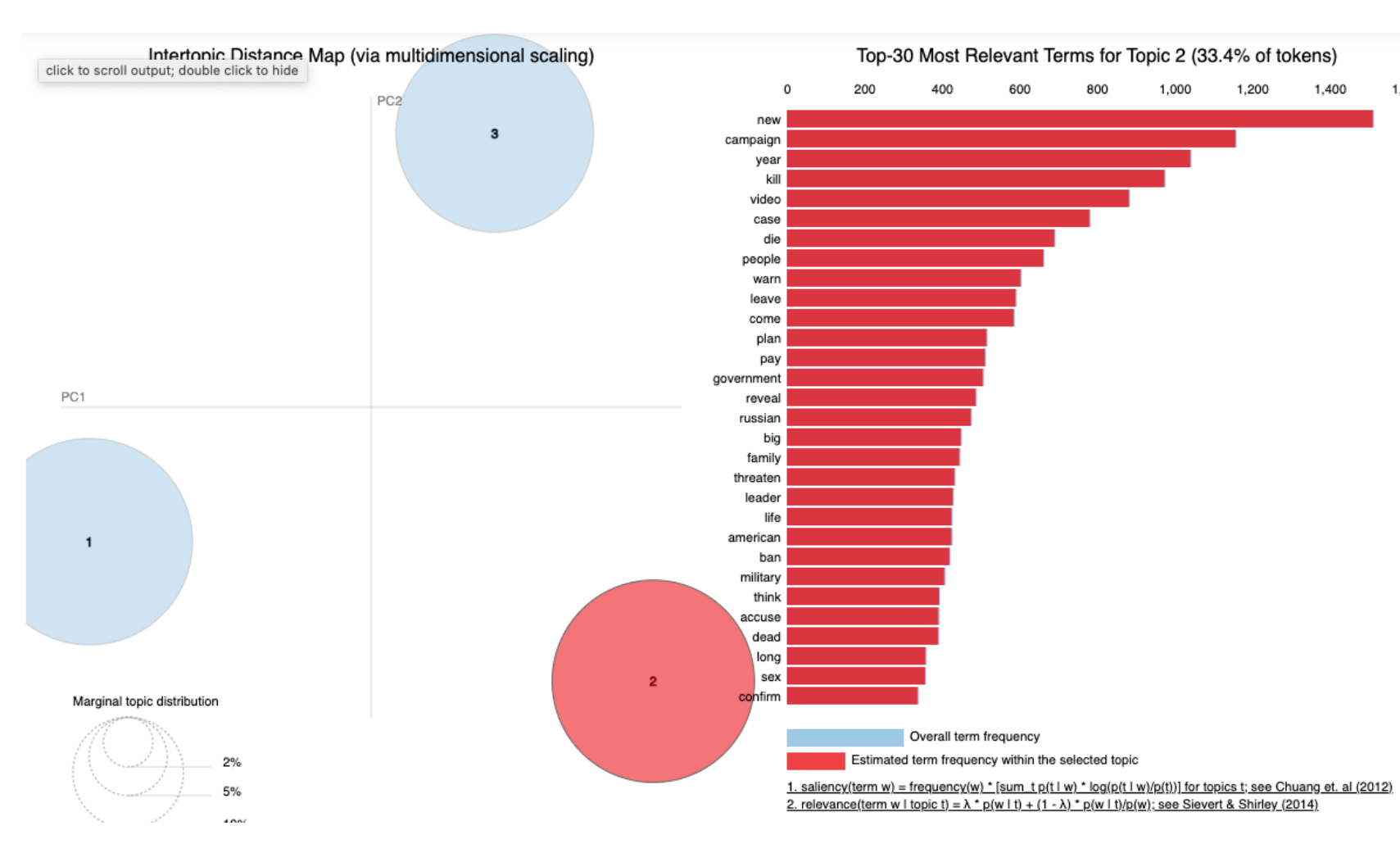


While the 20 components gives us a good coherence score, there are some issues using a value this highrelative to its context. Visually, you can see there is a lot of overlap in roughly 17 topics, due to them having similar terms. This is something that should be avoided in topic modeling, due to the goal of achievinig dissimarity between topics while keeping similarity within topics high. Keeping that in consideration, and after some fine tuning through observing intertopic distance, I eventually landed on a far smaller number of topics: 3, for reasons that should be clear in the intertopic distance maps below:









After looking at the terms in the topics, it seems like they fall into three categories:

Topic 1: Potentially related to protests and social justic issues
Topic 2: Potentially related to the reports on Russian interference with the 2016 election
Topic 3: Potentially related to the reports on Hillary Clinton's email scandal

After mapping these back to the original dataset, it's clear there is no pattern between which topics belong in fake and real news. This makes sense due to the fact that fake news publications would wnat to align with real news topics in hopes of obscuring the perception of them as fake news.

LSI

Latent Semantic Analysis follows a similar logic to LDA, but also utilizes Truncated Singular Value Decomposition to find a lower-rank approximation of the document term matrix, similar to how I explored reducing dimensions above. There wasn't much more to glean from these topics as opposed to LDA, it seems that there were very similar topics found with a similar distribution of terms. Intertopic Visualization wasn't available for the Latent Semantic Analysis, but below are the top 30 words of each topic for comparison.

Topic: 1
Words: ['trump', 'breitbart', 'election', 'new', 'video', 'win', 'say', 'vote', 'break', 'campaign', 'medium', 'supporter', 'attack', 'voter', 'get', 'comment', 'woman', 'anti', 'report', 'call', 'day', 'black', 'protest', 'time', 'people', 'poll', 'go', 'take', 'order', 'plan']

Topic: 2
Words: ['breitbart', 'trump', 'new', 'comment', 'say', 'man', 'medium', 'child', 'election', 'woman', 'year', 'life', 'government', 'world', 'video', 'police', 'kill', 'good', 'report', 'people', 'attack', 'go', 'earth', 'process', 'exclusive', 'student', 'transform', 'brick', 'president', 'healing']

Topic: 3
Words: ['election', 'breitbart', 'new', 'rig', 'comment', 'video', 'day', 'poll', 'say', 'steal', 'people', 'fake', 'expose', 'expectation', 'break', 'win', 'trump', 'stop', 'close', 'market', 'american', 'create', 'campaign', 'time', 'email', 'free', 'show', 'race', 'vote', 'panic']

Results and Evaluation

While a surface level analysis makes the complexity-accuracy tradeoff of stemming to lemmatization seems minimal, There are many other downstream issues that can arise in an analysis/model from utilizing stemmed words instead of their base lemma. So while there's a high accuracy on a model run on stemmed text, there is a reduction in granularity that I wish I could've delved into further if I had more time. Dimensionality reduction is likely the most effective method to retain as much of the signal within the text as possible, while maintaining feasible computatinoal requirements. My biggest takeaway from the entire project is that Natural Language Processing is still in its relatively nascent stages. It was very interesting to see how many "eye-tests" and heuristics are still used in the field. While that is exciting to have a little more subjectivity/ in this field as an analyst, it leaves open a lot of possibilities for new metrics and algorithms to be developed to fine tune decision making (e.g. topics utilized, number of components in Truncated SVD) I wish there were more context on the dates of the fake news, so I could gain more insight into why there were 3 clear topics at the time. If this text was collected in a more condensed timeframe, that would make more sense, given how quickly the news cycles run, even during an election year. Topic modeling also appears to make

References

1. Gavish, Matan, and David L. Donoho. "The optimal hard threshold for singular values is." IEEE Transactions on Information Theory 60.8 (2014): 5040-5053.
2. Portilla, Jose, Natural Language Processing with Python. Udemy. https://www.udemy.com/course/nlp-natural-language-processing-with-python/
3. Data Mining: Practical Machine Learning Tools and Techniques, 4th edition, 2016.
4. Beginners Guide To Truncated SVD For Dimensionality Reduction, https://analyticsindiamag.com/beginners-guide-to-truncated-svd-for-dimensionality-reduction/
5. Logistic regression python solvers' definitions, https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers-definitions
6. Quick Introduction to Bag-of-Words (BoW) and TF-IDF for Creating Features from Text, https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/