Spring 2019

Using data science to detect fake news

Eliza Shoemaker

Using Data Science to Detect Fake News

An Honors College Project Presented to

the Faculty of the Undergraduate

College of Intergated Science and Enginerring

James Madison University

by Eliza Shoemaker

Accepted by the faculty of the Department of Computer Science, James Madison University, in partial fulfillment of the requirements for the Honors College.

FACULTY COMMITTEE:

Project Advisor: Ramon A Mata-Toledo, Ph.D.
Professor, Computer Science Department

Reader: Nathan R Sprague, Ph.D.
Associate Professor, Computer Science Department

Reader: Sharon A Cote, Ph.D.
Associate Professor, English Department

HONORS COLLEGE APPROVAL:

Bradley R. Newcomer, Ph.D.,
Dean, Honors College

PUBLIC PRESENTATION

This work is accepted for presentation, in part or in full, at Computer Science Research Seminar on April 12th, 2019.

Table of Contents

List of Figures

Abstract

The purpose of this thesis is to assist in automating the detection of *Fake News* by identifying which features are more useful for different classifiers. The effectiveness of different extracted features for *Fake News* detection are going to be examined. When classifying text with machine learning algorithms features have to be extracted from the articles for the classifiers to be trained on. In this thesis, several different features are extracted: word counts, ngram counts, term frequency-inverse document frequency, sentiment analysis, lemmatization, and named entity recognition to train the classifiers. Two classifiers are used, a Random Forest classifier and a Naïve Bayes classifier. Training on different features combined with different machine learning algorithms yields different accuracies. By testing the different features on different classifiers, it can be determined which features are the best for *Fake News* detection. Classifying news articles as either *Fake News* or as not *Fake News* is explored using three datasets, which in total contains over 40,000 articles. One of the datasets is used to partly to train the classifiers and partly to test the classifiers. The remaining two datasets are used purely for testing the classifiers. All the code used in conjunction with thesis can be found in Appendix B.

Acknowledgements

I would like to thank my advisor, Dr. Ramon A. Mata-Toledo, for his priceless help and advice. I would also like to thank my readers Dr. Nathan Sprague and Dr. Sharon Cote for their insights and suggestions. I would also like to thank the Computer Science department of James Madison University for allowing me this opportunity.

Chapter 1

Introduction

The purpose of this thesis is to assist in automating the detection of *Fake News* by identifying which features are more useful for different classifiers. The effectiveness of different extracted features for *Fake News* detection are going to be examined. When classifying text with machine learning algorithms features have to be extracted from the articles for the classifiers to be trained on. In this thesis, several different features are extracted: word counts, ngram counts, term frequency-inverse document frequency, sentiment analysis, lemmatization, and named entity recognition. Two classifiers are used, a Random Forest classifier and a Naïve Bayes classifier. Training on different features combined with different machine learning algorithms yields different accuracies. By testing the different features on different classifiers, it can be determined which features are the best for *Fake News* detection. Classifying news articles as either *Fake News* or as not *Fake News* is explored using three datasets, which in total contains over 40,000 articles. One of the datasets is used to partly to train the classifiers and partly to test the classifiers. The remaining two datasets are used purely for testing the classifiers. All the code used in conjunction with thesis can be found in Appendix B.

The term *Fake News* has many definitions, for this paper we will be using Axel Galfert's [1].

> "Fake news is the deliberate presentation of (typically) false or misleading claims as *news*, where the claims are misleading *by design.*"

Although some form of *Fake News* has been around for many years, it is now mainstream and is widely considered to be a major issue [1]. The 2016 presidential election and Brexit are clear

examples of the relevance of *Fake News* in modern society [1], [2]. With the nature of the Internet as it is, anybody can spread untrue and biased information. It is virtually impossible to prevent *Fake News* from being created. Therefore, the next best thing is to find a way to identify and differentiate *Fake News* from *real* news. One of the ways to determine validity is to fact check, but this is time consuming and requires skills that are not shared by everyone. The next best thing is to automate the detection of *Fake News* by using the methods and techniques of Data Science.

Data Science is an interdisciplinary field that tries to find patterns in data that, in this case, may enable society to differentiate between *Fake News* and *real* news [3]. In addition, coupled with algorithms and large sets of data, Data Science can give the necessary insight to help find patterns within the data that would otherwise take a long time to discover or never be discovered at all. Artificial Intelligence (AI) and machine learning in particular, may be used to detect patterns that may characterize *Fake News* when the human eye cannot see it clearly.

Only in the past few decades, there has been an effort to use AI to detect types of deception. Additionally, in the past few years there has been an effort to use AI to detect *Fake News*. The majority of the research has not focused on full news articles, but on short statements. Most of the research that has been done is on small pieces of text that may vary in length; a sentence to a few sentences generally derived directly from tweets or text messages. One of the more predominant datasets, LIAR, is derived from politifact's database of statements. The LIAR dataset has 6 levels of truth values, and includes author data [4]. Datasets that use full sized articles are not as prevalent [5]. This is because it is much easier to label a single statement rather than a full-length article.

There are a few datasets that contain full length articles such as FakeNewsNet which is a small dataset with supplementary data. This data was collected from articles posted to twitter and contains data such as the profile of the user who posted the article and other social media context [6]. Another dataset, called BS DETECTOR, lists websites and their labels; the labels include, among others, fake, conspiracy, and bias [7]. Only URL, no articles, are provided and many of these sites are no longer operational or even available. Therefore, gathering articles from this dataset's sources is complicated and sometimes impossible. Lastly, there is the ISOT Fake News Dataset which contains over 40,000 articles and is far larger than all other datasets readily available [8], [9]. However, all articles in this dataset that are labeled "true" are from Reuters. These "true" articles skew the data because machine learning algorithms may detect the style of Reuters authors or editors and "learn" to label news as "not fake" if it fits this pattern.

In the next section I will discuss of previous research into *Fake News* detection. Afterword, we will examine the datasets used for both training and testing. Then we will go over feature extraction, and the different methods of feature extraction will be discussed. Then a basic introduction into the classifiers that are used is presented. Next, we will examine and discuss the results from the trained classifiers. Finally, we will explore future research.

Literature Review

Researchers have tried a few different classifiers for detecting *Fake News* some of the are: Convolutional Neural Networks (CNN), and Long Short Term Memory units (LSTM) [10]–[12]. CNNs tend to be fairly effective, despite being designed for machine vision applications. However, in some cases LSTMs outperform CNNs. LSTMs are able to "forget" certain details and focus on, or "remember," more relevant details. Hence, they work well with large bodies of text data [12].

Rubin et al created a classifier that that achieved 90% precision and distinguished between "legitimate news" and "satire news" [13]. They focused on "satire news" because it is deceptive, but it does not intend to deceive as *Fake News* does. Satire is meant to be noticeably fake as compared to *Fake News* which is meant to deceive. Rubin et al choose to focus on satire rather than *Fake News* because it is simpler to detect satire than *Fake News*. Rubin et al used a small dataset with only 290 training articles and 90 test articles. As a classifier they use a Support Vector Machine which is well suited for binary classification but not for multiclass classification.

Chapter 2

Objective

The goal of this research is to find the patterns that correlate with a piece of news which are potentially fake. Obviously, in any classification analysis there must to be human intervention at some time. Although, it may not be possible to achieve 100 percent accuracy, finding the commonalities of *Fake News* would be a step forward. For this purpose, the plan is to initially collect a large amount of data already known and verified as *Fake News* and try to train a model that will associate a piece of news with the probability of it being *Fake News*.

To classify news articles the raw text data needs to be turned into something more useful. This is called feature extraction. Feature extraction can take many forms: word counts, n-gram counts, punctuation usage, sentiment analysis, and many others. The extracted features can then be used to classify the article that the features came from. Different features may give different results depending on the underlaying patterns in the data. By testing different classifiers with different features one can determine patterns in the data. By determining the best features for classifying *Fake News* the potential for automated *Fake News* detection can be increased.

Methods

Datasets

To find patterns in *Fake News*, first news needs to be collected and labeled. Both *Fake News* and legitimate news needs to be represented in roughly equal amounts. This is to avoid the frequency of *Fake News* in the dataset being used as a determining factor in classifying. Having good data is essential producing valid results. Good data in this context is data that is representative of the real world and is generalizable.

The dataset used to train the classifiers is the ISOT Fake News Dataset, the largest available dataset of full length Fake News articles [8], [9]. The ISOT dataset contains 21,417 articles labeled Real and 23,481 that are labeled Fake, totaling 44,898. FakeNewsNet is another data set containing full length articles, however there are only 422 labeled articles in it [6]. And lastly there is a set of 180 articles, 90 Fake and 90 Real, collected by the author which, will be referred to as the Original Data. These two additional datasets will be used to test the accuracy of the trained classifiers.

Each model will initially be trained with 80% of the ISOT data. The remaining 20% of the ISOT data will be used to test the accuracy of the trained classifiers. As mentioned, FakeNewsNet and the Original Data will be used for testing as well. The reasoning behind using these additional tests is to make sure we are detecting *Fake News* and not some other pattern of the ISOT dataset, such as a style of a particular news organization.

Each article labeled as Real in the ISOT dataset was collected from Reuters; all articles their started with the word “Reuters”. This pattern could easily be picked up by humans and machines alike. To avoid this issue the beginning “Reuters” phrase was removed from each article.

Features

To find patterns, several different features should be tested. Features are numeric values that describe the text. Examples of these numeric values are word count or the number of times a particular punctuation mark is used. Some features will be more helpful than others, for instance the number of verbs is more likely to be useful compared to the number of times a particular word is used, such as ‘kitten’. The goal is to find the features that are most helpful in detection of *Fake News*. Next, each extracted feature will be discussed in detail.

Word counts are among the most easily obtained features that can be extracted from raw text. It is simply a count of all the terms in a body of text. Word counts are also called a ‘bag of words’, however, to keep names descriptive, we shall call this type of feature a count. To get the word count in texts, scikit-learn’s CountVectorizer is used; the CountVectorizer tokenizes the data and then counts each term [14]. The data can be tokenized by word or by n-gram. N-grams are series of *n* items, such as words or characters. In this thesis n-grams refers to groupings of two and three characters. For instance, the n-grams of the word ‘feature’ would be as follows: ‘fe’, ‘ea’, ‘at’, ‘tu’, ‘ur’, ‘re’, ‘fea’, ‘eat’, ‘atu’, ‘tur’, and ‘ure’. These features will be referred to as count-word and count-ngram respectively.

Term frequency-inverse document frequency, or TF-IDF, is calculated as follows: term frequency times the inverse document frequency. Where term frequency is the number of times a term is in a document divided by the number of terms in a document. The inverse document frequency is the logarithm of the number of text (or articles) in the collection divided by the number of texts or articles where the term appears. Below is the equation for TF-IDF:

$$\text{TF-IDF} = \frac{\text{number of term occurrences}}{\text{terms in text}} \times \log \frac{\text{number of texts in collection}}{\text{number of texts where term occurs}}$$

TF-IDF is a way to rank the importance of a term within a text with respect to all the texts in the collection. It ranks common words as less important (smaller numeric value) and less used words as more important (large numeric values). The implementation used in the software produced in conjunction with this thesis is part of sklearn which is included in the scikit-learn extraction module [14]. The terms can either be on a word or n-gram level; these features will be referred to as TFIDF-word and TFIDF-ngram respectively.

Fake News often uses people's emotions and preconceptions to manipulate the readers [15]. Although sentiment analysis is considered to be separate from *Fake News* detection, sentiment analysis could improve *Fake News* detection. To explore this using data science, the sentiment of an article needs to be articulated. To achieve this a sentiment analyzer is required and several are available. VADER (Valence Aware Dictionary and sEntiment Reasoner) is one of those tools [16]. VADER is publicly available and performs better than other benchmark sentiment analysis tools such as LIWC, GI, WordNet, and SentiWordNet. This feature will be referred to as VADER.

VADER gives four numbers: a score of how negative the tone of a piece of text is, how positive the text's tone is, how neutral it is, and how 'compound' it is or how mixed it is between the other values. The values it gives range from -1 to 1. Due to the fact that some classifiers are not able to use negative numbers, each VADER score will have 1 added to it, making it range vary from 0 to 2. Shifting VADER score's range in this way does not affect the meaning of the score.

Stop words are common words that are taken out of a text to improve accuracy in some data science applications. By removing stop words from a body of text we can focus better into words which make the text distinct. There are a number of ready-made lists of stop words, however, not all lists are good for all applications [17]. For instance, a word that is common and useless in one

context could be important in another. Two English ready-made lists are NLTK's stop word list and spaCy's stop word list. These will be referred to as NLTKStop and spaCyStop respectively.

Part of speech tagging, PoS, tagging is the process of labeling what part of speech a word is, based on the word and the surrounding words. Sentences are formed by using different PoS, sentences can be analyzed by looking at the patterns formed by combining the PoS. Exploring these patterns, where they occur, could provide valuable insight. NLTK provides PoS tagging capabilities [18]. The NLTK tagging includes different tags for different tenses. For instance, a past tense verb is not the same as a verb in the present tense. This feature will help the machine learning algorithms to take into account if an author is writing in present, future, or any other tense. This feature will be referred to as PoS.

Lemmatization is the process of getting the root from a word. For example, cats would be cat and feet would be foot. Computers do not understand that feet and foot are closely related and therefore cannot take such things into account. However, by lemmatizing the text we can turn all the forms of a word into the root word, allowing the classifying algorithms to focus on the root words. NLTK provides lemmatization. A wrapper function, written by Ken Tsuji, was used in the software produced in conjunction with this thesis[19]. Although by lemmatizing a word the tense of the word is lost, this should not be a problem because the close relationship between different tenses of a word is being revealed. This feature will be referred to as lemma.

Named entity recognition is the process of identifying persons, organizations, and other named entities. This is important for algorithms as they do not process the meaning of words. By labeling words as 'person' or 'organizations' algorithms can pick on patterns involving these entities that would otherwise be obstructed. For this thesis, spaCy's named entity recognition was used. This feature will be referred to as ER.

Classifiers

As previously mentioned, the extracted features were used to train classifiers. The classifiers used are now discussed. Naïve Bayes, NB, is a type of classifier that takes each feature and treats it as unrelated to any other feature. It then calculates the probability that the particular feature belongs to a classification. It does that for each feature and then aggerates each individual probability to calculate the final classification. For example, with a count-word it would calculate the probability that the count of the first word would belong to *Fake News* as opposed to not. This process will continue for every word and these probabilities a final decision would be made.

Before describing the next classifier, we will consider decision trees. A decision tree classifier takes the values of the features and splits them into two groups such that each group is as close as possible to only having a single classification. This is repeated until each group consists of a single classification. See Figure 1, for a visual example of a decision tree. The main issue with decision trees is that they do not generalize very well. They tend to fit the training data so well that the general patterns in the data are over looked.

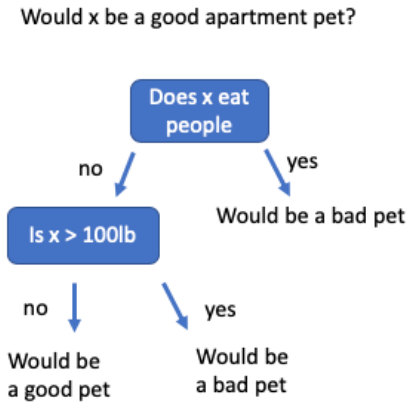


Figure 1: Decision Tree

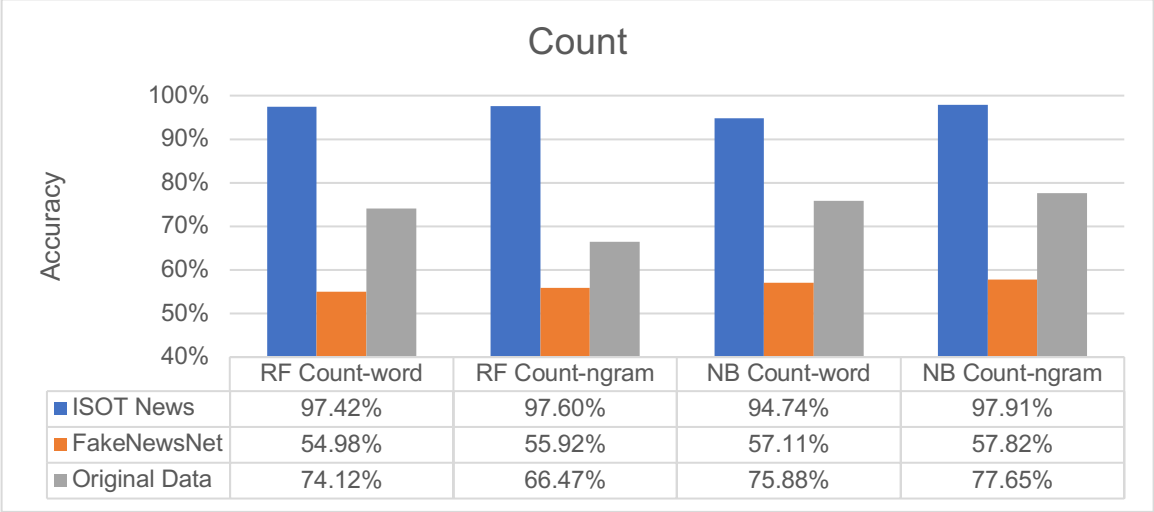
This is where the next classifier comes in. Random Forests are a type of classifier built out of a collection of decision trees. But, instead of each decision tree training on all of the data, each decision tree gets a random subset of the data to train on. Making each decision tree in the forest unique. When classifying, each decision tree in the forest gives its own classification, then whichever classification gets the most votes of the decision trees wins.

All of the code written for this thesis is provided in Appendix B. *DataFunctions.py* contains function for reading datasets from files, splitting training and testing data, training classifiers, testing classifiers, and printing results. *FeatureExtraction.py* contains functions for feature extraction. *FeatureTest.py* uses the function from *FeatureExtraction.py* and *DataFunctions.py* to test the different features. *RemoveReuters.py* simply contains the code used to remove the Reuters headers from the news articles.

Chapter 3

Results

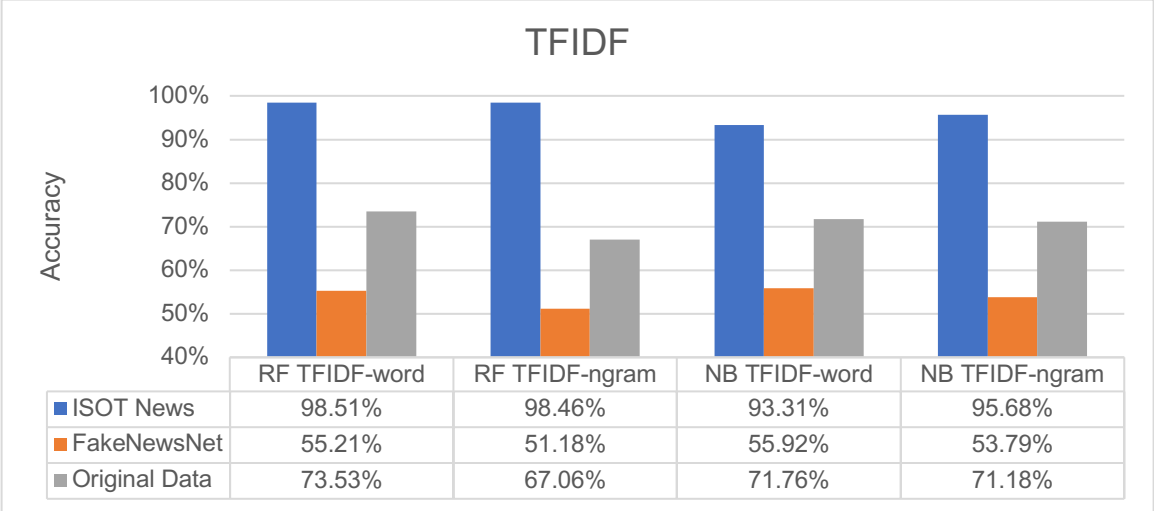
Using two different models, each extracted feature was tested. The models used were Random Forest (RF) and Naïve Bayes (NB). There is some difference between the two classifiers. There is a much larger difference between datasets. The following is a detailed discussion of each set of features. We will compare features and classifiers by their accuracy, which is the percentage of correct classification made by the classifier



	RF Count-word	RF Count-ngram	NB Count-word	NB Count-ngram
ISOT News	97.42%	97.60%	94.74%	97.91%
FakeNewsNet	54.98%	55.92%	57.11%	57.82%
Original Data	74.12%	66.47%	75.88%	77.65%

Figure 2: Count Accuracies

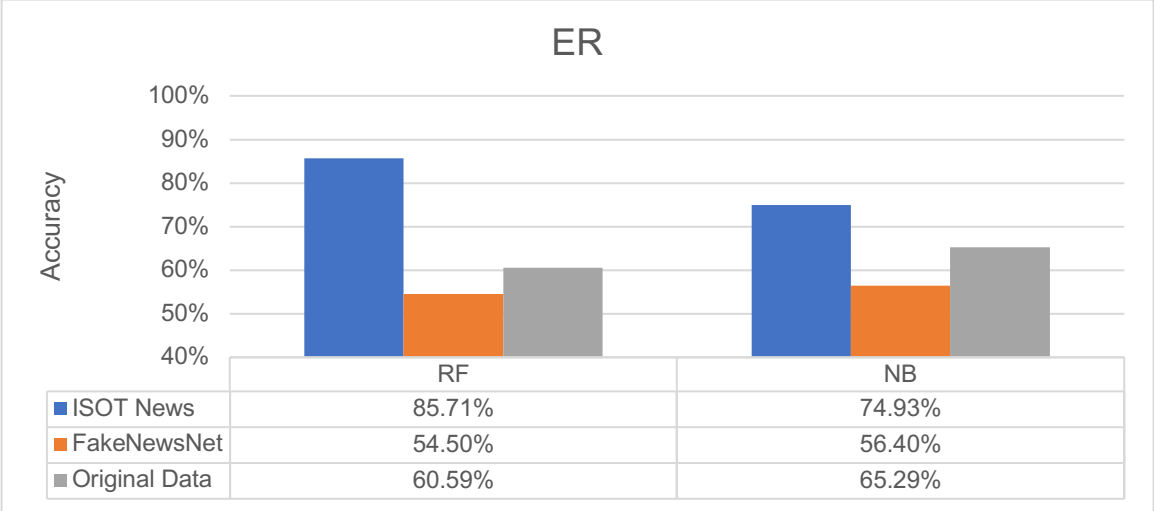
Count-word and Count-ngram: First, most notable the ISOT testing data is getting way higher accuracy results than either the Original dataset or the FakeNewsNet dataset. After the ISOT, the Original dataset is getting the next highest accuracy rates. This suggests that the Original dataset is closer in makeup to the ISOT dataset than the FakeNewsNet is. Next the data shows that the NB classifier generalizes better than the RF classifier. The NB classifier gets better accuracy rates with count-ngram. The RF has no clear winner between count-word and count-ngram.



	RF TFIDF-word	RF TFIDF-ngram	NB TFIDF-word	NB TFIDF-ngram
ISOT News	98.51%	98.46%	93.31%	95.68%
FakeNewsNet	55.21%	51.18%	55.92%	53.79%
Original Data	73.53%	67.06%	71.76%	71.18%

Figure 3: TFIDF Accuracies

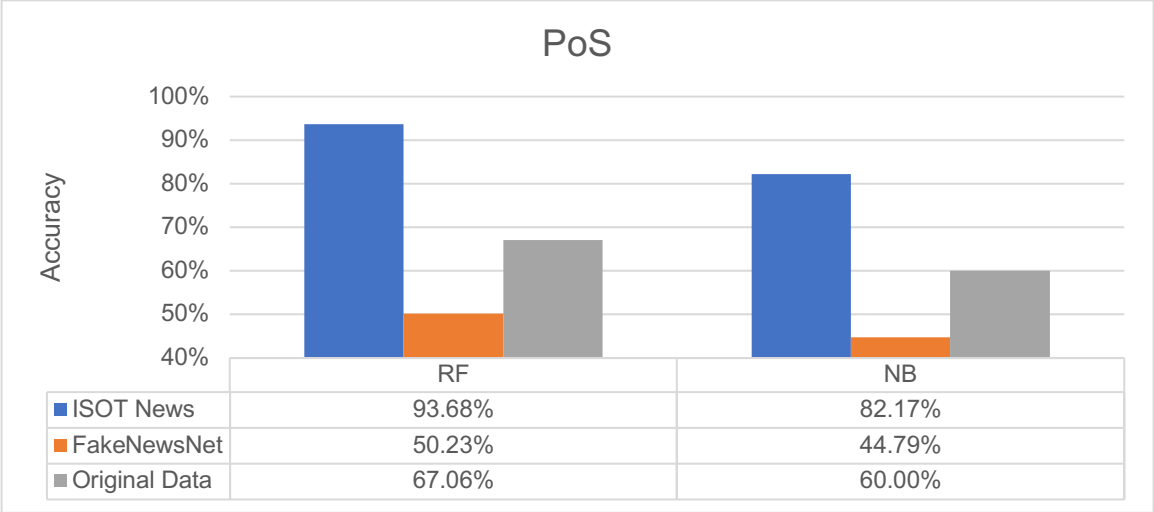
TFIDF-word and TFIDF-ngram: As seen in Figure 3, the ISOT testing data has the highest accuracies again. The random forest classifiers get better results with the ISOT dataset than the Naive Bayes. However, the NB does generalize better to the Original dataset and the FakeNewsNet dataset. TFIDF-word is getting better accuracy rates over TFIDF-ngram. In the case of the RF's classification of the Original dataset, the TFIDF-word is getting 6.47% more accuracy. Again, the Original dataset is being classified better than the FakeNewsNet dataset. Between TFIDF and Count, the Count-ngram is getting the best accuracy results.



	RF	NB
ISOT News	85.71%	74.93%
FakeNewsNet	54.50%	56.40%
Original Data	60.59%	65.29%

Figure 4: ER Accuracies

ER: Still, ISOT is doing best and NB generalizes better. Compared to the previous features, ER is not as good of a feature by itself. However, it cannot be concluded that ER is not a good feature. More testing with ER combined with other features should be done before disregarding ER as a feature for *Fake News* detection.



	RF	NB
ISOT News	93.68%	82.17%
FakeNewsNet	50.23%	44.79%
Original Data	67.06%	60.00%

Figure 5: PoS Accuracies

PoS: Here we see for the first time that NB is not generalizing better than the RF. Also, there is an accuracy below 50%, which shows that by using this feature to classify is no better than a

random guess. With an accuracy as low as 44.70%, it can be concluded that PoS by itself is definitely not a good feature for *Fake News* classification. However, there is a chance that when combined with another feature, PoS might be a good feature.

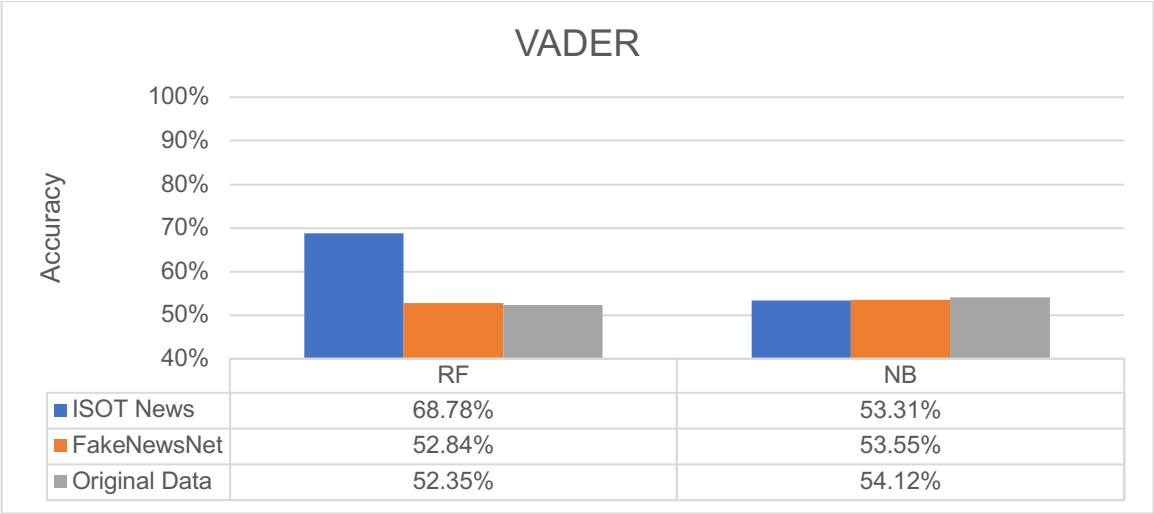


Figure 6: VADER Accuracies

VADER: As Figure 6 shows, the VADER feature is very detrimental to the accuracy rates. While this is not enough to conclude that VADER will not be helpful when combined with other features, it does suggest that VADER alone is not very helpful for classifying *Fake News*. Although, PoS has an instance of lower accuracy, VADER is lower overall and therefore is a worse feature.

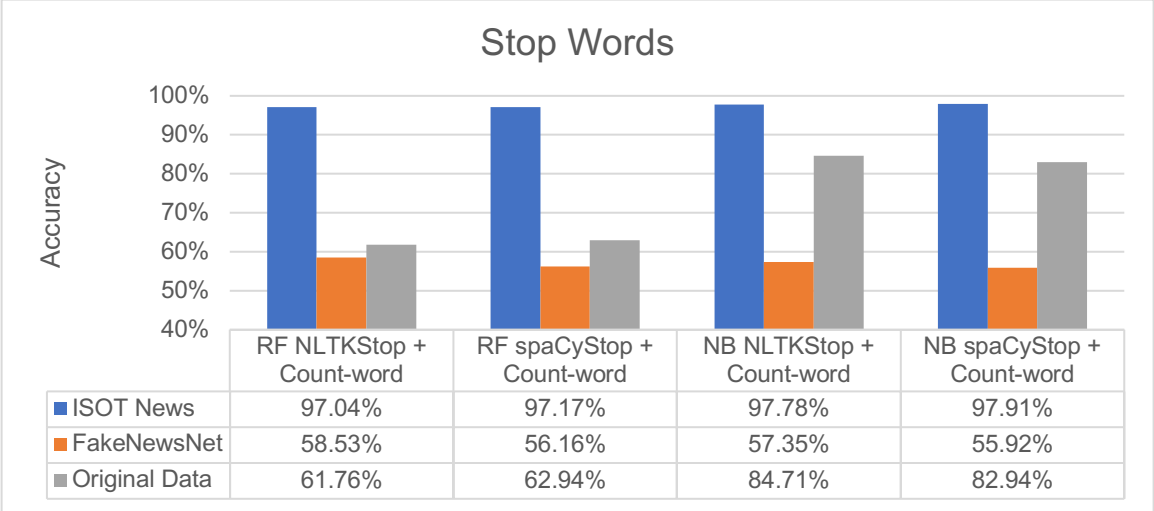


Figure 7: Stop Word Accuracies

Stop Word: Once more, ISOT is dominating the accuracy rates and the Original dataset is in second place. Figure 7 shows that NB generalizes much better than the RF classifier. Although close, the NLTK list of stop words is superior to the spaCy list for *Fake New* detection. From the results, we can see that Original dataset benefits greatly from NLTKStop and spaCyStop compared to Count-word. Additionally, FakeNewsNet also benefits from NLTKStop and spaCyStop, just not as much.

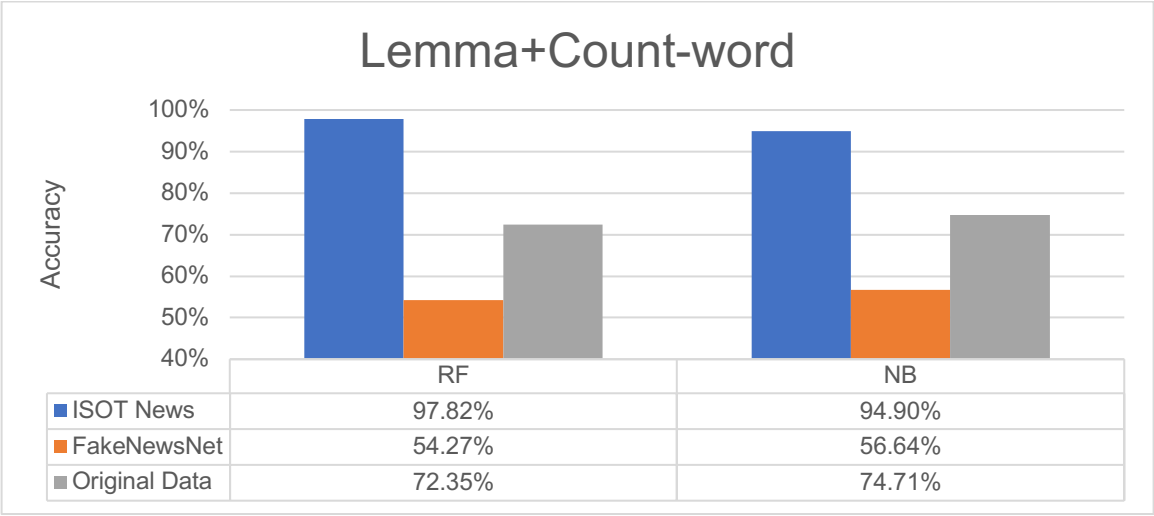


Figure 8: Lemma Accuracies

Lemma: Once again, ISOT accuracies are the highest, with Original coming in second. The NB classifier is still generalizing better than the RF classifier. The results from lemma are better than some of the other features. However, lemma with a Count-word is not as accurate as a Count-word. Suggesting that the different forms of a word are helpful to the classifier.

From the results a few more general conclusions can be made. The most notable is that the accuracy on the ISOT test data is much higher than the accuracies of the other datasets. From this, it can be concluded that there is a pattern in the ISOT dataset that is being picked up by the two classifiers. However, it appears that these patterns do not generalize well to the other available datasets. The patterns that the classifiers are picking up on could be a pattern found in Reuters articles, or could be another pattern that exists mainly in the ISOT dataset Such as article topic, or political leaning. All of this suggests that ISOT is not a good dataset to train with.
Next, it can be seen that the Original dataset is classifying with better accuracy than the FakeNewsNet dataset. The Original dataset does not contain articles from Reuters hence, this does not explain the jump in accuracy. Therefore, it is possible that the *Fake News* within the ISOT and Original dataset are closer in underlaying structure.
For accuracy rates, it can be concluded that Counts and TFIDF generalize better than ER, PoS, and VADER. More tests should be done with ER, PoS, and VADER features before any of them are discarded for providing lower accuracy rates. They still may be a benefit to accuracy rates when combined with other features, despite not doing well by themselves.
Complete tables of accuracy rates for both classifiers can be found in appendix A.

Conclusions

With the nature of the Internet as it is, *Fake News* is easily created and distributed. Fact checking is tedious and time consuming, so automating *Fake News* detection is critical. Thus *Fake News* classifiers should be created. However, a classifier does not come out of thin air, it must be trained on already existing data. The quality and quantity of the data is important. Three datasets were used for the research in this thesis. ISOT, a huge dataset of over 40,000 articles. FakeNewsNet is another, much smaller dataset containing 422 articles. Lastly, the Original dataset, containing 180 articles, that was gathered specifically for this research. However, a classifier cannot read, so it must have features extracted for the articles. A feature is a numeric value extracted from the article. Such as a word count, or a count of parts of speech, or more complicated features. Such as a count of the named entities, like businesses or organizations. However, which features work best?

Two different classifiers, Random Forests and Naive Bayes, were trained on the 80% of the ISOT dataset reserved for testing using each of the ten different features: Count-word, Count-ngram, TFIDF-word, TFIDF-ngram, PoS, ER, Lemma, VADER, NLTKStop, and spaCyStop. Then each classifier was tested on the remaining 20% from ISOT, all of FakeNewsNet, and all of the Original dataset. The accuracy results where then examined and conclusions were drawn.

The ISOT dataset did not generalize well to the other two datasets used for testing. Making the test results for the 20% testing portion of ISOT get way higher results than the other two datasets. This could be found the fact that ISOT got all of its *real* news from Reuters and the classifiers ended up being a Reuters vs not-Reuters classifier.

Next it was discovered that Count/TFIDF are better standalone features than PoS, ER, and VADER. However, these features still have potential to be used in conjunction with other

features. Although Lemma was one of the better features, it was outperformed by Count-word, suggesting that some of the removed data was improving the classification.

Future Work

A different dataset should be used to train classifiers to verify the result obtained with ISOT. The size of ISOT makes it a valuable dataset, however, it is probably best as a testing dataset than a training dataset.

Using combinations of the features should be explored. For instance, combining ER with lemma. Even more testing with VADER scores could be beneficial.

The best accuracy rate on the Original data set was achieved with a Naïve Bayes classifier with a word count feature after NLTK stop words were removed. These results should be explored more, for example which words when removed provide the greatest increase in accuracy. Additionally, it should be look into if the removal of any the NLTK stop words actually harm the overall accuracy of the classification.

One thing that has not been tried is differentiating and classifying *real* news, satire, and *Fake News*. This would be valuable because satire is a type of deceptive news that isn't *Fake News*. Hence, we should avoid labeling it as such.

Appendix A

Naive Bayes Classification Results

Naive Bayes – Classification accuracies.			
	ISOT News	FakeNewsNet	Original Data
TFIDF-word	93.31%	55.92%	71.76%
TFIDF-ngram	95.68%	53.79%	71.18%
ER	74.93%	56.40%	65.29%
Count-word	94.74%	57.11%	75.88%
Count-ngram	97.91%	57.82%	77.65%
PoS	82.17%	44.79%	60.00%
VADER	53.31%	53.55%	54.12%
NLTKStop+Count-word	97.78%	57.35%	84.71%
spaCyStop+Count-word	97.91%	55.92%	82.94%
lemmat+Count-word	94.90%	56.64%	74.71%

Random Forest Classification Results

Random Forest – Classification accuracies.			
	ISOT News	FakeNewsNet	Original Data
TFIDF-word	98.51%	55.21%	73.53%
TFIDF-ngram	98.46%	51.18%	67.06%
ER	85.71%	54.50%	60.59%
Count-word	97.42%	54.98%	74.12%
Count-ngram	97.60%	55.92%	66.47%
PoS	93.68%	50.23%	67.06%
VADER	68.78%	52.84%	52.35%
NLTKStop+Count-word	97.04%	58.53%	61.76%
spaCyStop+Count-word	97.17%	56.16%	62.94%
Lemmat+Count-word	97.82%	54.27%	72.35%

Appendix B

This code is also available at: https://github.com/Rugdumph/FakeNewsDetection

DataFunctions.py

Wrappers for training classifiers, testing classifiers, reading in dataset, and printing results.

```
1.  from sklearn.ensemble import RandomForestClassifier
2.  from sklearn.model_selection import train_test_split
3.  from sklearn.naive_bayes import MultinomialNB
4.  from FeatureExtraction import *
5.
6.  import json
7.  import numpy as np
8.  import csv
9.
10. def split_data(data,labels):
11.     return train_test_split(data, labels, test_size=0.2, random_state=42,
12.         shuffle="true")
13.
14.
15. def train_NB(train_data, train_labels):
16.     return MultinomialNB().fit(train_data, train_labels)
17.
18.
19. def train_random_foest(train_data, train_labels, est):
20.     return RandomForestClassifier(n_estimators=est).fit(train_data,
21.         train_labels)
22.
23.
24. def test_classifier(clf, validate_data, validate_labels, str):
25.     predicted = clf.predict(validate_data)
26.     print(str)
27.     print(np.mean(predicted == validate_labels))
28.
29. def get_News_dataset():
30.     ml_data = list()
31.     ml_labels = list()
32.     with open("News_dataset/Fake.csv") as csv_file:
33.         csv_reader = csv.reader(csv_file, delimiter=',')
34.         for row in csv_reader:
35.             ml_data.append(row[1])
36.             ml_labels.append(0)
37.     with open("News_dataset/CleanTrue.csv") as csv_file:
38.         csv_reader = csv.reader(csv_file, delimiter=',')
39.         for row in csv_reader:
40.             ml_data.append(row[1])
41.             ml_labels.append(1)
42.     return ml_data, ml_labels
43.
44. def get_FNN():
45.     ml_data = list()
46.     ml_labels = list()
47.     # open News.txt
48.     with open("FakeNewsNet/News.txt") as f:
49.         # for each line in News.txt
```

```
        for line in f:
            # read in the data (ie filename)

            # create openable file name
            json_filename = "FakeNewsNet/"+line.rstrip()+"-Webpage.json"

            # open file and read everything
            with open(json_filename, encoding='utf-8') as data_file:
                data = json.loads(data_file.read())

                # create data array
                ml_data.append(data['text'])

                # create label array
                if "Real" in json_filename:
                    ml_labels.append(1)
                else:
                    ml_labels.append(0)
    return ml_data, ml_labels

def get_OriNews():
    ml_data = list()
    ml_labels = list()
    with open("MyNews/researcharticles.csv") as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            filename = "MyNews/" + row[0]
            if row[3] == "Not-Real-Other":
                with open(filename, encoding='utf-8') as data_file:
                    ml_data.append(data_file.read())
                    ml_labels.append(0)
            elif row[3] == "Real":
                with open(filename, encoding='utf-8') as data_file:
                    ml_data.append(data_file.read())
                    ml_labels.append(1)
    return ml_data, ml_labels
```

FeatureExtraction.py

The methods I used to extract features.

```
1.  import spacy
2.  from collections import Counter
3.  from nltk import pos_tag
4.  from nltk.data import load
5.  from nltk.tokenize import word_tokenize
6.  from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
7.  from TagLemmatize import *
8.  from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
9.  from nltk.corpus import stopwords
10. import en_core_web_sm
11. from nltk.tokenize.treebank import TreebankWordDetokenizer as Detok
12.
13. # count-word feature extaction
14. def get_CountVector3(all_data, train_data, test_data):
15.     count_vect = CountVectorizer()
16.     count_vect = count_vect.fit(all_data)
17.     x_train_data =  count_vect.transform(train_data)
18.     x_test_data =  count_vect.transform(test_data)
19.     return x_train_data, x_test_data
20.
21. def get_CountVector1(all_data):
22.     count_vect = CountVectorizer()
23.     count_vect = count_vect.fit(all_data)
24.     return count_vect.transform(all_data)
25.
26. def remove_NLTK_stop3(all_data, train_data, test_data):
27.     sw = stopwords.words('english')
28.     deto = Detok()
29.
30.     all_cleaned = list()
31.     train_cleaned = list()
32.     test_cleaned = list()
33.
34.     for article in all_data:
35.         word_tokens = word_tokenize(article)
36.         all_cleaned.append(deto.detokenize(
37.             [w for w in word_tokens if not w in sw]))
38.
39.     for article in train_data:
40.         word_tokens = word_tokenize(article)
41.         train_cleaned.append(deto.detokenize(
42.             [w for w in word_tokens if not w in sw]))
43.
44.     for article in test_data:
45.         word_tokens = word_tokenize(article)
46.         test_cleaned.append(deto.detokenize(
47.             [w for w in word_tokens if not w in sw]))
48.
49.     return all_cleaned, train_cleaned, test_cleaned
50.
51.
52. def remove_spaCy_stop3(all_data, train_data, test_data):
53.     spacy_nlp = spacy.load('en')
54.     sw = spacy.lang.en.stop_words.STOP_WORDS
55.     deto = Detok()
```

```
56.
57.     all_cleaned = list()
58.     train_cleaned = list()
59.     test_cleaned = list()
60.
61.     for article in all_data:
62.         word_tokens = word_tokenize(article)
63.         all_cleaned.append(deto.detokenize(
64.             [w for w in word_tokens if not w in sw]))
65.
66.     for article in train_data:
67.         word_tokens = word_tokenize(article)
68.         train_cleaned.append(deto.detokenize(
69.             [w for w in word_tokens if not w in sw]))
70.
71.     for article in test_data:
72.         word_tokens = word_tokenize(article)
73.         test_cleaned.append(deto.detokenize(
74.             [w for w in word_tokens if not w in sw]))
75.
76.     return all_cleaned, train_cleaned, test_cleaned
77.
78. def remove_spaCy_stop1(all_data):
79.     spacy_nlp = spacy.load('en')
80.     sw = spacy.lang.en.stop_words.STOP_WORDS
81.     deto = Detok()
82.
83.     all_cleaned = list()
84.
85.     for article in all_data:
86.         word_tokens = word_tokenize(article)
87.         all_cleaned.append(deto.detokenize(
88.             [w for w in word_tokens if not w in sw]))
89.
90.     return all_cleaned
91.
92. def remove_NLTK_stop1(all_data):
93.     sw = stopwords.words('english')
94.     deto = Detok()
95.
96.     all_cleaned = list()
97.
98.     for article in all_data:
99.         word_tokens = word_tokenize(article)
100.                all_cleaned.append(deto.detokenize(
101.                    [w for w in word_tokens if not w in sw]))
102.
103.           return all_cleaned
104.
105.
106.       def get_CountVector_NLTK_Stop3(all_data, train_data, test_data):
107.           sw = stopwords.words('english')
108.           count_vect = CountVectorizer(stop_words=sw)
109.           count_vect = count_vect.fit(all_data)
110.           x_train_data =  count_vect.transform(train_data)
111.           x_test_data =  count_vect.transform(test_data)
112.           return x_train_data, x_test_data
113.
114.       def get_CountVector_spaCy_Stop3(all_data, train_data, test_data):
115.           spacy_nlp = spacy.load('en')
116.           spacy_stopwords = spacy.lang.en.stop_words.STOP_WORDS
```

```
            count_vect = CountVectorizer(stop_words=spacy_stopwords)
            count_vect = count_vect.fit(all_data)
            x_train_data =  count_vect.transform(train_data)
            x_test_data =  count_vect.transform(test_data)
            return x_train_data, x_test_data


        # count-ngram feature extaction
        def get_CountVector_Ngram3(all_data, train_data, test_data):
            count_vect = CountVectorizer(ngram_range=(2,3))
            count_vect = count_vect.fit(all_data)
            x_train_data =  count_vect.transform(train_data)
            x_test_data =  count_vect.transform(test_data)
            return x_train_data, x_test_data


        def get_CountVector_Ngram1(all_data):
            count_vect = CountVectorizer(ngram_range=(2,3))
            count_vect = count_vect.fit(all_data)
            return count_vect.transform(all_data)


        # TFIDF-word feature extraction
        def get_TFIDF_Word3(all_data, train_data, test_data):
            tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
                                         max_features=5000)
            tfidf_vect.fit(all_data)
            x_train_data =  tfidf_vect.transform(train_data)
            x_test_data =  tfidf_vect.transform(test_data)
            return x_train_data, x_test_data

        def get_TFIDF_Word1(all_data):
            tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
                                         max_features=5000)
            tfidf_vect.fit(all_data)
            return tfidf_vect.transform(all_data)


        # TFIDF-ngram feature extraction
        def get_TFIDF_NGram3(all_data, train_data, test_data):
            tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
                                         ngram_range=(2,3), max_features=5000)
            tfidf_vect.fit(all_data)
            x_train_data =  tfidf_vect.transform(train_data)
            x_test_data =  tfidf_vect.transform(test_data)
            return x_train_data, x_test_data


        # TFIDF-ngram feature extraction
        def get_TFIDF_NGram1(all_data):
            tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
                                         ngram_range=(2,3), max_features=5000)
            tfidf_vect.fit(all_data)
            return tfidf_vect.transform(all_data)


        # VADER feature extraction
        def get_VADER_score(data_list):
            analyser = SentimentIntensityAnalyzer()
            ret_list = list()
            for data in data_list:
```

```
178.                 ret_list.append(list(analyser.polarity_scores(data).values()))
179.             return ret_list
180.
181.         def make_VADER_score_non_neg(article_list):
182.             ret_list = list()
183.             for article_vals in article_list:
184.                 ret_list.append([x+1 for x in article_vals])
185.             return ret_list
186.
187.         def tag_and_lem_list(data_list):
188.             ret_list = []
189.             for d in data_list:
190.                 ret_list.append(tag_and_lem(d))
191.             return ret_list
192.
193.         def get_PoS(all_data):
194.             # Turn all_data into PoS
195.             all_pos = list()
196.             for article in all_data:
197.                 all_pos.append(pos_tag(word_tokenize(article)))
198.
199.             # Create a counter for all_pos
200.             all_pos_counter = list()
201.             for article in all_pos:
202.                 all_pos_counter.append(Counter( tag for word,  tag in article))
203.
204.             all_pos_count = list()
205.
206.             tagdict = load('help/tagsets/upenn_tagset.pickle')
207.             # Count up each PoS and giving a value of 0 to those that do not occur
208.             for counter in all_pos_counter:
209.                 temp = list()
210.                 for key in tagdict:
211.                     temp.append(counter[key])
212.                 all_pos_count.append(temp)
213.
214.             return all_pos_count
215.
216.         def get_ER(all_data):
217.             named_entity_list = ("PERSON", "NORP", "FAC", "ORG", "GPE", "LOC",
218.                                  "PRODUCT", "EVENT","WORK_OF_ART", "LAW", "LANGUAGE",
219.                                  "DATE", "TIME", "PERCENT", "MONEY", "QUANTITY",
220.                                  "ORDINAL", "CARDINAL")
221.             nlp = en_core_web_sm.load()
222.
223.             all_list = list()
224.
225.             # get entites
226.             for article in all_data:
227.                 nlpa = nlp(article)
228.                 all_list.append(Counter([(X.label_) for X in nlpa.ents]))
229.
230.             all_list_counts = list()
231.
232.             for counter in all_list:
233.                 temp = list()
234.                 for entity in named_entity_list:
235.                     temp.append(counter[entity])
236.                 all_list_counts.append(temp)
237.
238.             return all_list_counts
```

FeatureTest.py

FeatureTest.py: The code for training and testing I used for my testing and analysis.

```
#!/usr/bin/env python3
from FeatureExtraction import *
from DataFunctions import *

def basic_tests(train_data, train_labels,       # Data for training classifier
                validate_data, validate_labels, # Test data & labels for ISOT
                FNN_data, FNN_labels,           # Test data & labels for FNN
                OriNews_data, OriNews_labels):  # Test data & labels for OriNews

    clf = train_random_foest(train_data, train_labels, 50)
    test_classifier(clf, validate_data, validate_labels, "RF: validate_data")
    test_classifier(clf, FNN_data, FNN_labels, "RF: FNN_data")
    test_classifier(clf, OriNews_data, OriNews_labels, "RF: OriNews_data")

    clf = train_NB(train_data, train_labels)
    test_classifier(clf, validate_data, validate_labels, "NB: validate_data")
    test_classifier(clf, FNN_data, FNN_labels, "NB: FNN_data")
    test_classifier(clf, OriNews_data, OriNews_labels, "NB: OriNews_data")

    clf = train_SVC(train_data, train_labels)
    test_classifier(clf, validate_data, validate_labels, "RF: validate_data")
    test_classifier(clf, FNN_data, FNN_labels, "RF: FNN_data")
    test_classifier(clf, OriNews_data, OriNews_labels, "RF: OriNews_data")


raw_data, labels = get_News_dataset()
FNN_raw_data, FNN_labels = get_FNN()
OriNews_raw_data, OriNews_labels = get_OriNews()

total_raw_data = raw_data+FNN_raw_data+OriNews_raw_data
raw_train_data, raw_validate_data, train_labels, validate_labels = split_data(raw_data,
 labels)


print("======================")
print("== Count_Ngram Only ==")
print("======================")

FNN_data, OriNews_data = get_CountVector_Ngram3(total_raw_data, FNN_raw_data, OriNews_r
aw_data)
train_data, validate_data = get_CountVector_Ngram3(total_raw_data, raw_train_data, raw_
validate_data)

basic_tests(train_data, train_labels,       # Data for training classifier
            validate_data, validate_labels, # Test data & labels for ISOT
            FNN_data, FNN_labels,           # Test data & labels for FNN
            OriNews_data, OriNews_labels)   # Test data & labels for OriNews


print("======================")
print("== Count_Word  Only ==")
print("======================")

```

```
FNN_data, OriNews_data = get_CountVector3(total_raw_data, FNN_raw_data, OriNews_raw_data)
train_data, validate_data = get_CountVector3(total_raw_data, raw_train_data, raw_validate_data)

basic_tests(train_data, train_labels,       # Data for training classifier
            validate_data, validate_labels, # Test data & labels for ISOT
            FNN_data, FNN_labels,           # Test data & labels for FNN
            OriNews_data, OriNews_labels)   # Test data & labels for OriNews


print("======================")
print("==      ER Only     ==")
print("======================")

FNN_data = get_ER(FNN_raw_data)
OriNews_data = get_ER(OriNews_raw_data)
train_data = get_ER(raw_train_data)
validate_data = get_ER(raw_validate_data)

basic_tests(train_data, train_labels,       # Data for training classifier
            validate_data, validate_labels, # Test data & labels for ISOT
            FNN_data, FNN_labels,           # Test data & labels for FNN
            OriNews_data, OriNews_labels)   # Test data & labels for OriNews

print("======================")
print("==  Lemma + Count   ==")
print("======================")

FNN_data1 = tag_and_lem_list(FNN_raw_data)
OriNews_data1 = tag_and_lem_list(OriNews_raw_data)

raw_train_data, raw_validate_data, train_labels, validate_labels = split_data(raw_data, labels)
train_data1 = tag_and_lem_list(raw_train_data)
validate_data1 = tag_and_lem_list(raw_validate_data)

lemma_total_train = validate_data1+train_data1+FNN_data1+OriNews_data1

train_data, validate_data = get_CountVector3(lemma_total_train, train_data1, validate_data1)
FNN_data, OriNews_data = get_CountVector3(lemma_total_train, FNN_data1, OriNews_data1)


basic_tests(train_data, train_labels,       # Data for training classifier
            validate_data, validate_labels, # Test data & labels for ISOT
            FNN_data, FNN_labels,           # Test data & labels for FNN
            OriNews_data, OriNews_labels)   # Test data & labels for OriNews



print("=========================")
print("== NLTK removed + Count ==")
print("=========================")

      raw_data_stop = remove_NLTK_stop1(raw_data)
      FNN_raw_data_stop = remove_NLTK_stop1(FNN_raw_data)
      OriNews_raw_data_stop = remove_NLTK_stop1(OriNews_raw_data)

      raw_train_data_stop, raw_validate_data_stop, train_labels, validate_labels = split_data(raw_data_stop, labels)
```

```
106.
107.
108.        total_stop_data = raw_data_stop+FNN_raw_data_stop+OriNews_raw_data_stop
109.
110.        FNN_data, OriNews_data = get_CountVector_Ngram3(total_stop_data, FNN_raw_data_st
    op, OriNews_raw_data_stop)
111.        train_data, validate_data = get_CountVector_Ngram3(total_stop_data, raw_train_da
    ta_stop, raw_validate_data_stop)
112.
113.        basic_tests(train_data, train_labels,       # Data for training classifier
114.                    validate_data, validate_labels, # Test data & labels for ISOT
115.                    FNN_data, FNN_labels,           # Test data & labels for FNN
116.                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
117.
118.
119.        print("============================")
120.        print("== spaCy removed + Count ==")
121.        print("============================")
122.
123.        raw_data_stop = remove_spaCy_stop1(raw_data)
124.        FNN_raw_data_stop = remove_spaCy_stop1(FNN_raw_data)
125.        OriNews_raw_data_stop = remove_spaCy_stop1(OriNews_raw_data)
126.
127.        raw_train_data_stop, raw_validate_data_stop, train_labels, validate_labels = spl
    it_data(raw_data_stop, labels)
128.
129.        FNN_data, OriNews_data = get_CountVector_Ngram3(total_stop_data, FNN_raw_data_st
    op, OriNews_raw_data_stop)
130.        train_data, validate_data = get_CountVector_Ngram3(total_stop_data, raw_train_da
    ta_stop, raw_validate_data_stop)
131.
132.        basic_tests(train_data, train_labels,       # Data for training classifier
133.                    validate_data, validate_labels, # Test data & labels for ISOT
134.                    FNN_data, FNN_labels,           # Test data & labels for FNN
135.                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
136.
137.
138.        print("============================")
139.        print("==        PoS Only       ==")
140.        print("============================")
141.
142.        FNN_data = get_PoS(FNN_raw_data)
143.        OriNews_data = get_PoS(OriNews_raw_data)
144.        train_data = get_PoS(raw_train_data)
145.        validate_data = get_PoS(raw_validate_data)
146.
147.        basic_tests(train_data, train_labels,       # Data for training classifier
148.                    validate_data, validate_labels, # Test data & labels for ISOT
149.                    FNN_data, FNN_labels,           # Test data & labels for FNN
150.                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
151.
152.
153.        print("============================")
154.        print("==    TFIDF_Word Only    ==")
155.        print("============================")
156.
157.        FNN_data, OriNews_data = get_TFIDF_Word3(total_raw_data, FNN_raw_data, OriNews_r
    aw_data)
158.        train_data, validate_data = get_TFIDF_Word3(total_raw_data, raw_train_data, raw_
    validate_data)
159.
```

```
        basic_tests(train_data, train_labels,       # Data for training classifier
                    validate_data, validate_labels, # Test data & labels for ISOT
                    FNN_data, FNN_labels,           # Test data & labels for FNN
                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews


        print("============================")
        print("==    TFIDF_Ngram Only    ==")
        print("============================")

        FNN_data, OriNews_data = get_TFIDF_NGram3(total_raw_data, FNN_raw_data, OriNews_
    raw_data)
        train_data, validate_data = get_TFIDF_NGram3(total_raw_data, raw_train_data, raw
    _validate_data)

        basic_tests(train_data, train_labels,       # Data for training classifier
                    validate_data, validate_labels, # Test data & labels for ISOT
                    FNN_data, FNN_labels,           # Test data & labels for FNN
                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews


        print("============================")
        print("==       VADER Only       ==")
        print("============================")

        FNN_data = make_VADER_score_non_neg(get_VADER_score(FNN_raw_data))
        OriNews_data = make_VADER_score_non_neg(get_VADER_score(OriNews_raw_data))
        train_data = make_VADER_score_non_neg(get_VADER_score(raw_train_data))
        validate_data = make_VADER_score_non_neg(get_VADER_score(raw_validate_data))

        basic_tests(train_data, train_labels,       # Data for training classifier
                    validate_data, validate_labels, # Test data & labels for ISOT
                    FNN_data, FNN_labels,           # Test data & labels for FNN
                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
```

RemoveReuters.py:

Used to "clean" the ISOT dataset.

```
import csv
import re


with open("News_dataset/TrueClean.csv", mode='w') as write_file:
    writer = csv.writer(write_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MIN
IMAL)
    with open("News_dataset/True.csv") as read_file:
        csv_reader = csv.reader(read_file, delimiter=',')
        for row in csv_reader:
            writer.writerow([row[0], re.sub(r'\w*\s*\(Reuters\) - ',"",row[1],count=1)
, row[2], row[3]])
```

Bibliography

[1] A. Gelfert, “Fake news: A definition,” *Informal Log.*, vol. 38, no. 1, pp. 84–117, 2018.

[2] K. Rogers and J. Bromwich, “The Hoaxes, Fake News and Misinformation We Saw on Election Day,” *NY Times*, New York, 08-Nov-2016.

[3] I. Witten, E. Frank, and M. Hall, *Data Mining 4th Edition*. 2016.

[4] W. Y. Wang, “‘Liar, Liar Pants on Fire’: A New Benchmark Dataset for Fake News Detection,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 422–426.

[5] D. Byrd, “The science of fake news gets a boost,” 2018. [Online]. Available: http://earthsky.org/human-world/fake-news-mar-2018-article-science-calling-for-studies.

[6] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu, “FakeNewsNet: A Data Repository with News Content, Social Context and Dynamic Information for Studying Fake News on Social Media,” 2018.

[7] “BS DETECTOR.” [Online]. Available: https://github.com/selfagency/bs-detector.

[8] H. Ahmed, I. Traore, and S. Saad, “Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques,” in *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, 2017, pp. 127–138.

[9] H. Ahmed, I. Traore, and S. Saad, “Detecting opinion spams and fake news using text classification,” *Secur. Priv.*, vol. 1, no. 1, p. e9, 2018.

[10] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1746–1751.

[11] H. Rashkin, E. Choi, J. Y. Jang, S. Volkova, and Y. Choi, “Truth of Varying Shades:

Analyzing Language in Fake News and Political Fact-Checking," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2931–2937.

[12] Y. Long, Q. Lu, R. Xiang, M. Li, and C.-R. Huang, "Fake News Detection Through Multi-Perspective Speaker Profiles," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2017, pp. 252–256.

[13] V. Rubin, N. Conroy, Y. Chen, and S. Cornwell, "Fake News or Truth? Using Satirical Cues to Detect Potentially Misleading News," in *Proceedings of the Second Workshop on Computational Approaches to Deception Detection*, 2016, pp. 7–17.

[14] "4.2. Feature extraction — scikit-learn 0.20.3 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html.

[15] V. Bakir and A. McStay, "Fake News and The Economy of Emotions," *Digit. Journal.*, vol. 6, no. 2, pp. 154–175, 2017.

[16] C. J. C. J. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," in *ICWSM*, 2014, no. May, pp. 216–225.

[17] J. Nothman, H. Qin, and R. Y. Symerio, "Stop Word Lists in Free Open-source Software Packages," *Proc. ofWorkshop NLP Open Source Softw.*, pp. 7–12, 2018.

[18] "Categorizing and Tagging Words." [Online]. Available: https://www.nltk.org/book/ch05.html.

[19] K. Tsuji, "tag-lemmatize." [Online]. Available: https://github.com/KT12/tag-lemmatize.