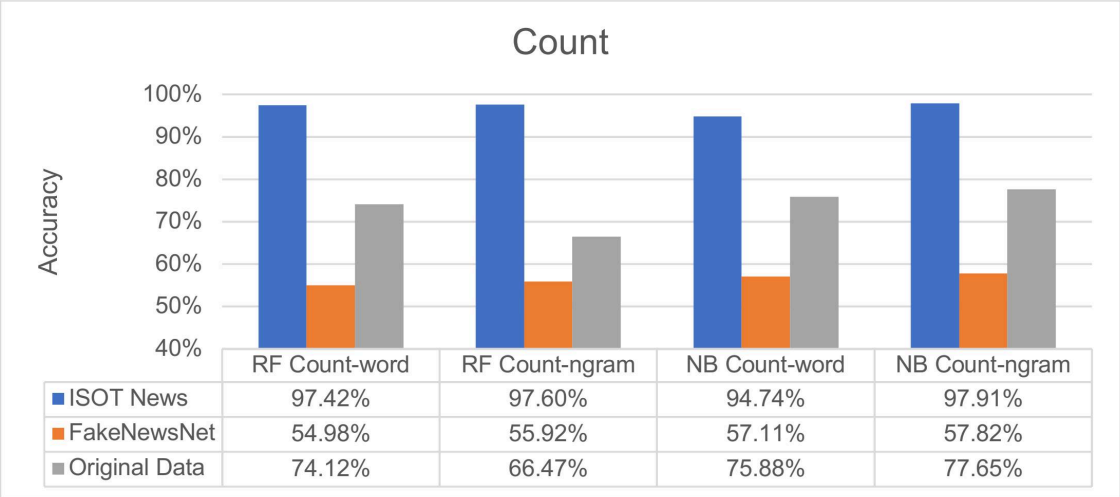
Chapter 3

Results

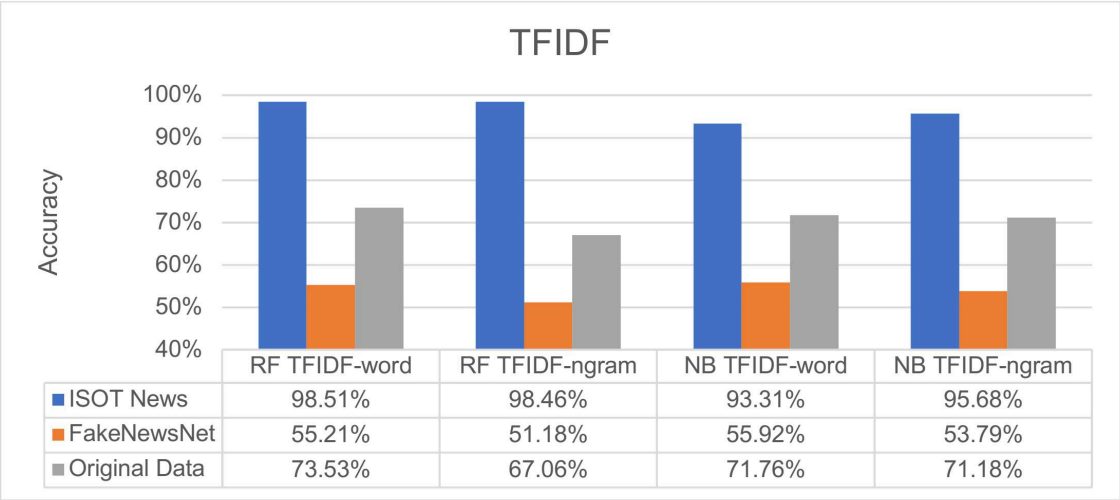
Using two different models, each extracted feature was tested. The models used were Random Forest (RF) and Naïve Bayes (NB). There is some difference between the two classifiers. There is a much larger difference between datasets. The following is a detailed discussion of each set of features. We will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

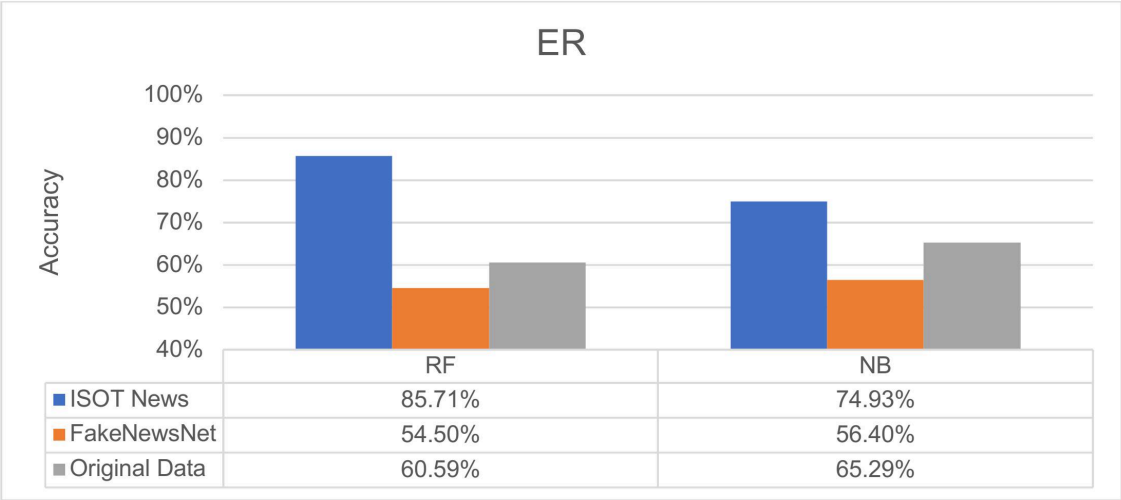
Count-word and Count-ngram: First, most notable the ISOT testing data is getting way higher accuracy results than either the Original dataset or the FakeNewsNet dataset. After the ISOT, the Original dataset is getting the next highest accuracy rates. This suggests that the Original dataset is closer 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

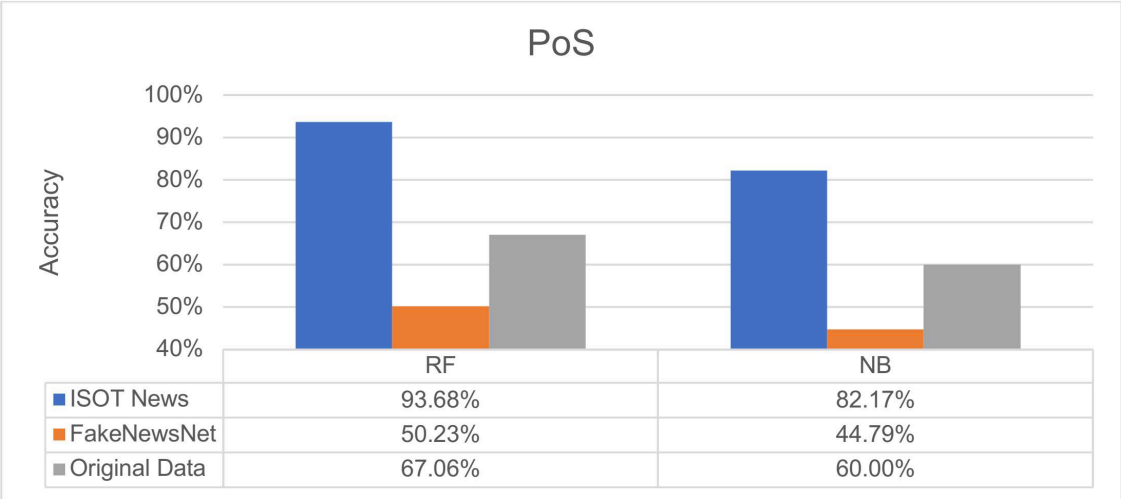
TFIDF-word and TFIDF-ngram: As seen in Figure 3, the ISOT testing data has the highest accuracies again. The random forest classifiers get better results with the ISOT dataset than the Naive Bayes. However, the NB does generalize better to the Original dataset and the FakeNewsNet dataset. TFIDF-word 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.

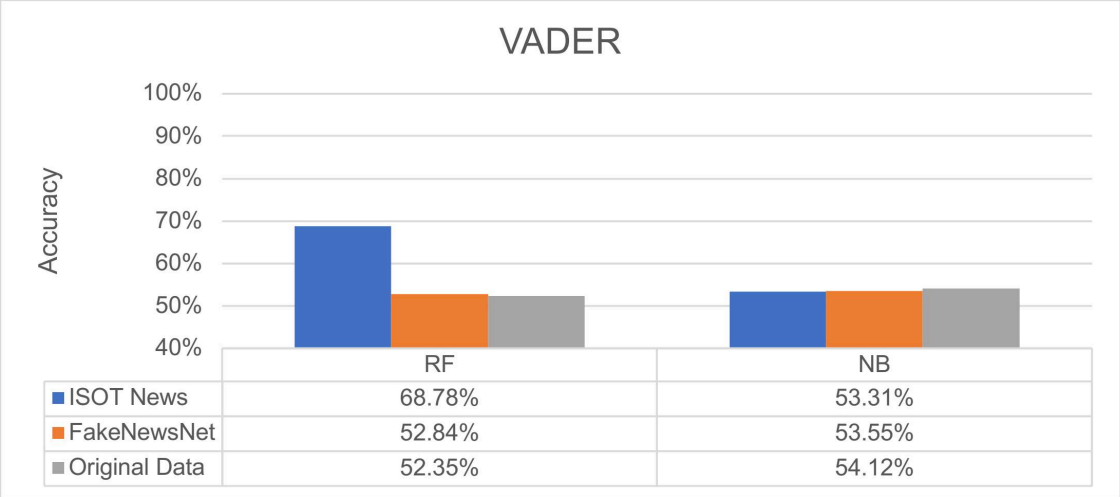


	RF	NB
ISOT News	93.68%	82.17%
FakeNewsNet	50.23%	44.79%
Original Data	67.06%	60.00%

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

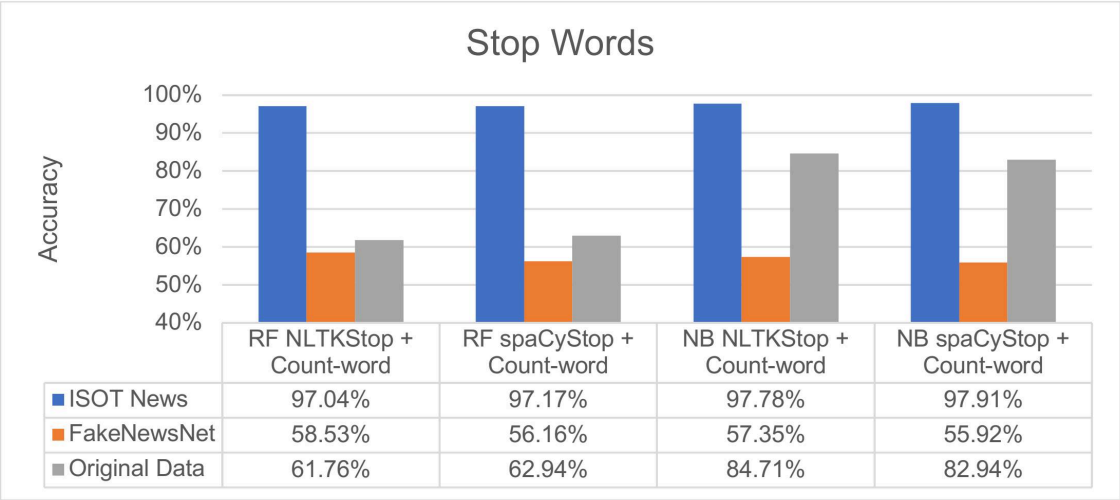
random guess. With an accuracy as low as 44.70%, it can be concluded that PoS by itself is definitely not a good feature for *Fake News* classification. However, there is a chance that when combined with another feature, PoS might be a good feature.



	RF	NB
ISOT News	68.78%	53.31%
FakeNewsNet	52.84%	53.55%
Original Data	52.35%	54.12%

Figure 6: VADER Accuracies

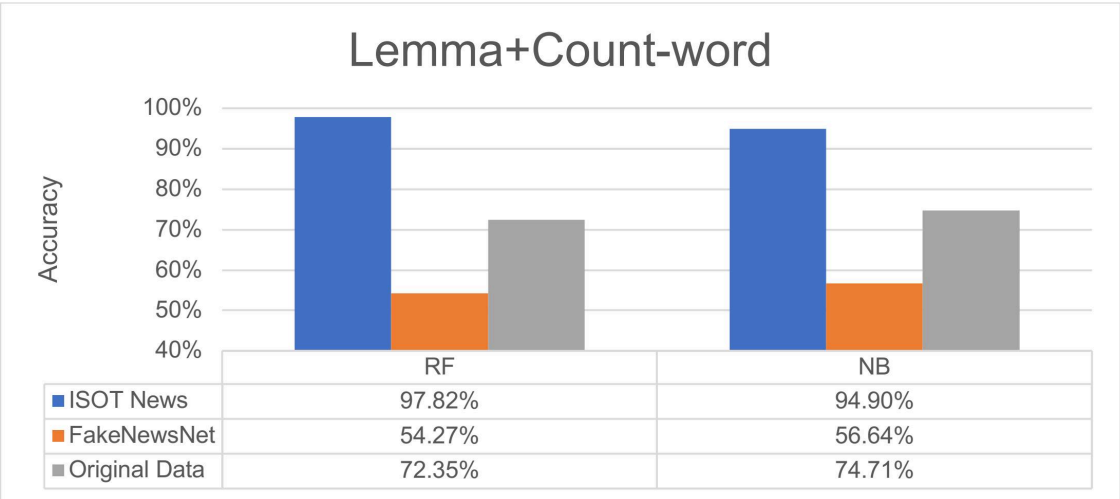
VADER: As Figure 6 shows, the VADER feature is very detrimental to the accuracy rates. While this is not enough to conclude that VADER will not be helpful when combined with other features, it does suggest that VADER alone is not very helpful for classifying *Fake News*. Although, PoS has an instance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



	RF NLTKStop + Count-word	RF spaCyStop + Count-word	NB NLTKStop + Count-word	NB spaCyStop + Count-word
ISOT News	97.04%	97.17%	97.78%	97.91%
FakeNewsNet	58.53%	56.16%	57.35%	55.92%
Original Data	61.76%	62.94%	84.71%	82.94%

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.



	RF	NB
ISOT News	97.82%	94.90%
FakeNewsNet	54.27%	56.64%
Original Data	72.35%	74.71%

Figure 8: Lemma Accuracies

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.
```

```
160.        basic_tests(train_data, train_labels,       # Data for training classifier
161.                    validate_data, validate_labels, # Test data & labels for ISOT
162.                    FNN_data, FNN_labels,           # Test data & labels for FNN
163.                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
164.
165.
166.        print("============================")
167.        print("==    TFIDF_Ngram Only    ==")
168.        print("============================")
169.
170.        FNN_data, OriNews_data = get_TFIDF_NGram3(total_raw_data, FNN_raw_data, OriNews_
    raw_data)
171.        train_data, validate_data = get_TFIDF_NGram3(total_raw_data, raw_train_data, raw
    _validate_data)
172.
173.        basic_tests(train_data, train_labels,       # Data for training classifier
174.                    validate_data, validate_labels, # Test data & labels for ISOT
175.                    FNN_data, FNN_labels,           # Test data & labels for FNN
176.                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
177.
178.
179.        print("============================")
180.        print("==       VADER Only       ==")
181.        print("============================")
182.
183.        FNN_data = make_VADER_score_non_neg(get_VADER_score(FNN_raw_data))
184.        OriNews_data = make_VADER_score_non_neg(get_VADER_score(OriNews_raw_data))
185.        train_data = make_VADER_score_non_neg(get_VADER_score(raw_train_data))
186.        validate_data = make_VADER_score_non_neg(get_VADER_score(raw_validate_data))
187.
188.        basic_tests(train_data, train_labels,       # Data for training classifier
189.                    validate_data, validate_labels, # Test data & labels for ISOT
190.                    FNN_data, FNN_labels,           # Test data & labels for FNN
191.                    OriNews_data, OriNews_labels)   # Test data & labels for OriNews
```

RemoveReuters.py:

Used to "clean" the ISOT dataset.

```
1.  import csv
2.  import re
3.
4.
5.  with open("News_dataset/TrueClean.csv", mode='w') as write_file:
6.      writer = csv.writer(write_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MIN
    IMAL)
7.      with open("News_dataset/True.csv") as read_file:
8.          csv_reader = csv.reader(read_file, delimiter=',')
9.          for row in csv_reader:
10.             writer.writerow([row[0], re.sub(r'\w*\s*\(Reuters\) - ',"",row[1],count=1)
    , row[2], row[3]])
```