Sentiment Analysis on Trump Impeachment

Heng Fang, Harper He, Renjie Luo, Te Pi, Puzhen Qian

December 2019

Abstract

The news about the impeachment against the President of the United States has been occupying each television channel and newspaper for an extended period of time. While the power branches are performing their due process, it is very important to understand the feelings of the public. In this report, we perform sentiment analysis on user comments from Reddit, America's largest social news platform. We use a Python package named PRAW to scrape data from the website and do data pre-processing and frequency analysis on them. After training several models based on NLTK's polarity database, we apply them to classify the Reddit comments. From the length of the lists: 2875 of positive and 9202 of negative words, we conclude that most subreddits contain negative words toward the impeachment news. This result aligns well with a recent poll [1].

1 Introduction

In this section, we will first give a brief explanation on the event that we are interested in. This storytelling is followed by an elaboration of the techniques and the data source that we use for the project.

1.1 Background

On August 12th, 2019, Michael Atkinson, the Inspector General of the Intelligence Community, received a whistleblower complaint about a troubling phone call between US President Donald Trump and the President of Ukraine Volodymyr Zelensky. In this phone call, President Trump threatened to withhold military aid to Ukraine if Zelensky did not investigate the wrongdoing of Former Vice President Joe Biden and his son Hunter Biden in Ukraine.

The executive branch is accused of abusing its power in favor of a political campaign and House Democrats, as a result of that, launched an impeachment inquiry on September 24th. Following on that, on September 26th, the acting Director of the National Intelligence Office Joseph Maguire testified on Congress. House Democrats also subpoenaed Trump's personal lawyer Rudy Guiliani on September 30th and the White House on October 4th after the President openly solicited China's investigation into Joe Biden's business.

1.2 Motivation

While the executive and legislative branches are performing their due process, it is also important to have a comprehensive understanding of the opinions of the public. In this report, we will apply state-of-the-art natural language processing techniques to acquire deeper insights on how people feel about this ongoing event.

Sentiment analysis is a technique to understand people's feelings about a certain topic and has been widely adopted in the industry for better public relations and customer service. Companies leverage such technique to have meaningful interactions with customers. Artificial intelligence is used to identify positive, negative and neutral opinions from text and public relation professionals can make informed decisions based on these data.

Reddit is one of America's biggest social news platforms. It is aggregated with millions of reviews that are very suitable for NLP processing. This website is made of a variety of columns known as "subreddits" where topics of particular interests are discussed. At the time of writing, the top 5 subreddits are 1) r/HongKong, 2) r/rickandmorty, 3) r/relationship_advice, 4) r/pokemon, and 5 r/LivestreamFail.

Combining sentiment analysis and the data from Reddit, we aim to dive deeper into the public's reaction towards such a bombshell event. Section 2 talks about how we obtain the data and do preprocessing. We show the most frequent words under different circumstances in Section 3. In Section 4, we explain in detail the features we use for our classifier and how we implement them. Section 5 contains evaluation of the classifiers and the output when we apply the classifier to the data we get from Reddit. We also include the challenges we have while working on the project and some other discussion in the last few sections.

2 Data Scraping and Preprocessing

We decide to apply sentiment analysis and the first step is to get hold of the data. In this section, we will first talk about the method we used to scrap the data. Then we will aim at some basic preprocessing methods we are going to use the data for sentimental analysis.

2.1 Data Scraping

Data scraping, also known as web scraping, is the process of importing information from a website into a spreadsheet or local file saved on personal computer. It's one of the most efficient ways to get data from the web. For example, business intelligence analysts and price comparison sites constantly leverage such technique for their own purposes. In our report, we use data scraping to get data about a bombshell event from Reddit - the Trump impeachment, and analyze the public sentiment towards the event.

The methods for data scraping can be heterogeneous, such as using Microsoft Excel with a dynamic web query, and using an effective and dedicated data scraping tool. Here we use PRAW to scrape user comments and replies.

PRAW(Python Reddit API Wrapper) is a special package designed for subreddit parser to scrape all user comments and replies under the subreddit. The code for our Reddit parser is in Listing 1.

Listing 1: PRAW In Our Project

```
# Reddit Parser
import praw

#Reddit API log in
reddit=praw.Reddit(client_id='****OMITTED****',
                   client_secret='****OMITTED****',
                   user_agent='****OMITTED****'
                   )

subreddit=reddit.subreddit('Trump_impeachment')
hot_vipkid=subreddit.hot(limit=None)

# User loop to get comments under subreddits.
res=[]
for sub in hot_vipkid:
    res.append(sub.title)
    sub.comments.replace_more(limit=None)
    comments=sub.comments.list()
    for comment in comments:
        res.append(comment.body)
        if len(comment.replies)>0:
            for reply in comment.replies:
                res.append(reply.body)

# Output
with open("Reddit_Comments.txt", "w",encoding='utf-8') as output:
    output.write(str(res))
```

First, we import the PRAW package and log in the Reddit API. Then we get comments from subreddits in a for loop. After the for loop, we write the result to a text file.

2.2 Data Preprocessing

In the real world, data are generally incomplete, inconsistent, lacking in certain behaviors or trends, and likely to contain errors. Data preprocessing is a data mining technique that involves transforming raw data into an understandable format, which is a proven method of resolving those issues.

Preprocessing a text simply means bringing a text into a form that is predictable and analyzable for a task. Data preprocessing for sentiment analysis is a process to clean textual data and prepare the target variable. It includes tokenization, stopwords removal, normalization and so on. Tokenization is the process of converting text into tokens before transforming it into vectors, it also make it easier to filter out unnecessary tokens. Stop words are the most commonly occuring words which are not relevant in the context of the data and do not contribute any deeper meaning to the phrase. NLTK provide a stopword library used for this. Words which look different but have the same meaning need to be processed equally. Lemmatization and stemming are two usual technologies to ensure that these words are treated equally. Sometimes we may have noise in the text with raw format, such as HTML or XML wrappers. To deal with it, we usually use regular expressions to remove them from our text.

3 Top Words by Frequency

We started this analysis with lower casing and removing non-alpha words for normalization. In order to get meaningful top frequency words, we decided to remove the stop words. We also have chosen to normalize by the length of the document to gain extra meaningful insight about the data.

Also, we used stemmer to remove repeated words. After using the Porter stemmer, we found that the stemmer successfully removed suffix but failed to return the original stem. Therefore, we also used a different way - using NLTK lemmatizer instead to remove affixes since the Lancaster stemmer has proven to be more severe in removing word endings.

We show the implementation in Python below. First we read the file.

```
import nltk
from nltk import FreqDist
f=open('/Users/captainprice/Desktop/nlp/data.txt',encoding = "utf-8")
raw = f.read()
```

Then we use the NLTK library to tokenize the input.

```
words=nltk.word_tokenize(raw)
print(len(words))
```

The output from the last `print` function is 247944. We remove non-alpha words in the next step.

```
words=[w.lower()for w in words]
revisedwords=[w for w in words if w.isalpha()]
len(revisedwords)
```

The length of the `revisedwords` array is 182074. The stop words are removed after non-alpha words.

```
stopwords=nltk.corpus.stopwords.words('english')
stoppedtrump=[w for w in revisedwords if w not in stopwords]
```

Frequency Analysis After Stop Words Removal Since we are dealing with the event Trump Impeachment here, we name the sentence array as `stoppedtrump`. After removing the stop words, we calculate the most frequent words from `stoppedtrump`.

```
trumpfdist = FreqDist(stoppedtrump)
trumpstoptopkeys = trumpfdist.most_common(50)
for item in trumpstoptopkeys:
    print(item[0],item[1]/len(stoppedtrump))
```

The top 5 most frequent words are *trump*, *election*, *https*, *would*, and *like*. The complete list of the 50 words can be found in the appendix, Section A.

Frequency Analysis After Stemming Based on the `stoppedtrump` word list, we apply stemming using the code below.

```
porter = nltk.PorterStemmer()
trumpstem = [porter.stem(t) for t in stoppedtrump]

trumpfdist = FreqDist(trumpstem)
trumptopkeys = trumpfdist.most_common(50)
for item in trumptopkeys:
    print(item[0],item[1]/len(trumpstem))
```

The top 5 most frequent words are *trump*, *election*, *http*, *like*, *impeach*. The complete list of the 50 words can be found in the appendix, Section B.

Frequency Analysis After Lemmatizing Based on the `stoppedtrump` word list, we apply lemmatization using the code below.

```
wnl=nltk.WordNetLemmatizer()
trumpLemma=[wnl.lemmatize(t) for t in stoppedtrump]
trumpfdist=FreqDist(trumpLemma)
trumptopkeys = trumpfdist.most_common(50)
for item in trumptopkeys:
    print(item[0],item[1]/len(trumpLemma))
```

The top 5 most frequent words are *trump*, *election*, *http*, *like*, *would*. The complete list is omitted due to page restriction.

4 Features

In this section, we will talk about the features we use to build the classifiers. We use Document Feature, Subjectivity Lexicon, Negation Feature, and newSL to extract features.

4.1 Document Feature

In the sentiment analysis problem, we need to convert tokenized words into labels. For the document feature, we use bag-of-words model, which means we convert text into the matrix of occurrence of words. So, it concerns about whether a word occurred or not in the document. Considering the large number of words, we extract the most frequent 2,000 words as features for sentiment analysis.

Table 1 is an example showing how this feature works. The table contains two documents from our dataset. As you can see, each word is a column of the matrix and each document is a row in the matrix. The bag-of-words model will mark **1** if a certain word occurs in a certain document, otherwise, it marks **0**.

Doc 1. The rock is destined to be the 21st century's new "conan"
Doc 2. he is going to make a splash even greater than arnold schwarzenegger

	is	going	to	be	make	century	new	conan	splash	even	greater	than	...
Doc 1	1	0	1	1	0	1	1	1	0	0	0	0	
Doc 2	1	1	1	0	1	0	0	0	1	1	1	1	

Table 1: Document Feature

We started by loading the `sentence_polarity` corpus and creating a list of documents where each document represents a single sentence and its label. And we define the set of words that will be used for features. This set was limited to 2000 most frequent words. We lowercase the words but skip stemming or stopwords removed. Then we create the training and test sets, train a Naive Bayes classifier, and compute the accuracy. We ended up achieving 0.727 accuracy. The code for our document feature is in Listing 2.

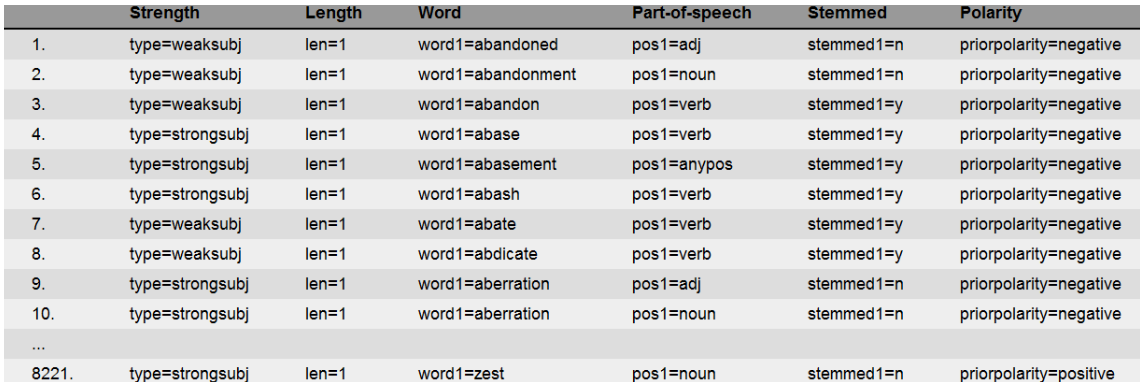
Listing 2: Document Feature

```python
# Sentiment Classification - Words as Features (BOW or unigram
   features)
# load the sentence_polarity corpus and create a list of documents
from nltk.corpus import sentence_polarity
import random
random.seed(30)
sentences=sentence_polarity.sents()
documents= [(sent,cat) for cat in sentence_polarity.categories()
         for sent in sentence_polarity.sents(categories=cat)]
random.shuffle(documents) #random shuffle documents

# define the set of words that will be used for features
# limit to the 2000 most frequent words
# lowercase words without stemming or removing stopwords
all_words_list=[word for (sent,cat) in documents for word in sent]
all_words = nltk.FreqDist(all_words_list)
word_items = all_words.most_common(2000)
word_features = [word for (word,freq) in word_items]

# define the unigram features for each document, using just the words
# The feature label will be ``contains(keyword)'' for each keyword
   (aka word) in the word_features set
# The value is whether the word is contained in the document
def document_features(document,word_features):
    document_words=set(document)
    features={}
    for word in word_features:
        features['contain({})'.format(word)] = (word in document_words)
    return features
```

```
# Define the feature sets for the documents.
featuresets=[(document_features(d,word_features),c) for (d,c) in
   documents]

# do a 90/10 split of our approximately 10,000 documents
word_train_set,word_test_set = featuresets[1000:],featuresets[:1000]
classifier=nltk.NaiveBayesClassifier.train(word_train_set)
print(nltk.classify.accuracy(classifier,word_test_set))#0.727
```

4.2 Subjectivity Lexicon Feature (SL_feature)

One early approach to sentiment analysis is using popular positive and negative words databases. In our analysis, we use the Subjectivity Lexicon which is maintained by Theresa Wilson, Janyce Wiebe, and Paul Hoffmann [6]. Figure 1 below shows what its structure is like.



	Strength	Length	Word	Part-of-speech	Stemmed	Polarity
1.	type=weaksubj	len=1	word1=abandoned	pos1=adj	stemmed1=n	priorpolarity=negative
2.	type=weaksubj	len=1	word1=abandonment	pos1=noun	stemmed1=n	priorpolarity=negative
3.	type=weaksubj	len=1	word1=abandon	pos1=verb	stemmed1=y	priorpolarity=negative
4.	type=strongsubj	len=1	word1=abase	pos1=verb	stemmed1=y	priorpolarity=negative
5.	type=strongsubj	len=1	word1=abasement	pos1=anypos	stemmed1=y	priorpolarity=negative
6.	type=strongsubj	len=1	word1=abash	pos1=verb	stemmed1=y	priorpolarity=negative
7.	type=weaksubj	len=1	word1=abate	pos1=verb	stemmed1=y	priorpolarity=negative
8.	type=weaksubj	len=1	word1=abdicate	pos1=verb	stemmed1=y	priorpolarity=negative
9.	type=strongsubj	len=1	word1=aberration	pos1=adj	stemmed1=n	priorpolarity=negative
10.	type=strongsubj	len=1	word1=aberration	pos1=noun	stemmed1=n	priorpolarity=negative
...						
8221.	type=strongsubj	len=1	word1=zest	pos1=noun	stemmed1=n	priorpolarity=positive

Figure 1: Subjectivity Lexicon

As we can see, the Subjectivity Lexicon is represented here as a dictionary, where each word has its strength, length, part-of-speech, stemmed or not and polarity.

Most of the time, subjectivity words from the subjectivity lexicon file are used as features themselves or in conjunction with other information, but we use it to create features by counting the positive and negative subjectivity words present in our document.

The `SL_feature` is developed from the bag-of-words feature above, where two features "positivecount" and "negativecount" are created to count all the positive and negative subjectivity words. Based on the strength of a word, it can be counted once ("weaksubj") or twice ("strongsubj").

We first read the subjectivity words from the subjectivity lexicon file created by Janyce Wiebe and her group at the University of Pittsburgh in the MPQA project and then defined Subjectivity Count features. These features contain counts of all the positive and negative subjectivity words, where each weakly subjective word is counted once and each strongly subjective word is counted twice. Then we used 90% Subjectivity Count feature sets documents as training set and 10% as testing set. After training with Naive Bayes classifier, we get the accuracy 0.762. The code for our `SL_feature` is in Listing 3.

Listing 3: SL Feature

```
# Sentiment Classification - Subjectivity Count features
# read in the subjectivity words from the subjectivity lexicon
SLpath = 'C:/Users/65730/subjclueslen1-HLTEMNLP05.tff'
def readSubjectivity(path):
    flexicon = open(path, 'r')
    # initialize an empty dictionary
    sldict = { }
    for line in flexicon:
        fields = line.split()    # default is to split on whitespace
        # split each field on the '=' and keep the second part as the
   value
        strength = fields[0].split("=")[1]
        word = fields[2].split("=")[1]
        posTag = fields[3].split("=")[1]
        stemmed = fields[4].split("=")[1]
        polarity = fields[5].split("=")[1]
        if (stemmed == 'y'):
            isStemmed = True
        else:
            isStemmed = False
        # put a dictionary entry with the word as the keyword  and a
   list of the other values
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict

# define features that include word counts of subjectivity words
# negative feature will have number of weakly negative words +
#    2 * number of strongly negative words
# positive feature has similar definition
#    not counting neutral words
SL = readSubjectivity(SLpath)
def SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    # count variables for the 4 classes of subjectivity
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
SL_featuresets = [(SL_features(d, word_features, SL), c) for (d, c) in
   documents]

# retrain the classifier using these features
SL_train_set, SL_test_set = SL_featuresets[1000:],
   SL_featuresets[:1000]
SL_classifier = nltk.NaiveBayesClassifier.train(SL_train_set)
nltk.classify.accuracy(SL_classifier, SL_test_set)#0.762
```

4.3 Negation Feature

Negation words can change the sentiment of a sentence from positive to negative or from negative to positive. So we would also like to handle them. First, we look for "prototype" negation word like "not", "no" and "never", then we add a "negated context" to the word next to the negation word to make sure the negation and degrees of it are kept. For example, this sentence from our dataset contains a negation word "doesn't": `''emerges as something rare, an issue movie that's so honest and keenly observed that it doesn't feel like one.''` To handle the negation, we add the word "NOT" to the word "feel" that is next to the negation word "doesn't". Table 2 shows how it works with bag-of-words models.

	it	doesn't	feel	NOT_feel	like	one
Before	1	1	1	0	1	1
After	1	0	0	1	1	1

Table 2: Negation Feature

After creating the negation features, we used same strategy - a 90/10 split for training and test dataset. After training with Naive Bayes classifier, the accuracy was 0.762.The code for our document feature is in Listing 4.

Listing 4: Negation Feature

```
# Negation words "not", "never" and "no"
# Not can appear in contractions of the form "doesn't"
for sent in list(sentences)[:50]:
   for word in sent:
     if (word.endswith("n't")):
        print(sent)

# this list of negation words includes some "approximate negators"
   like hardly and rarely
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing',
   'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom',
   'neither', 'nor']

# One strategy with negation words is to negate the word following the
   negation word,other strategies negate all words up to the next
   punctuation.Strategy is to go through the document words in order
   adding the word features,but if the word follows a negation words,
   change the feature to negated word.
# Start the feature set with all 2000 word features and 2000 Not word
   features set to false
def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or
   (word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in
   word_features)
        else:
            features['V_{}'.format(word)] = (word in word_features)
    return features

# define the feature sets
NOT_featuresets = [(NOT_features(d, word_features, negationwords), c)
   for (d, c) in documents]
not_train_set, not_test_set = NOT_featuresets[1000:],
   NOT_featuresets[:1000]
NOT_classifier  = nltk.NaiveBayesClassifier.train(not_train_set)
print(nltk.classify.accuracy(NOT_classifier, not_test_set))#0.762
```

4.4 New Subjectivity Lexicon Feature (newSL_feature)

To create `newSL_feature`, we remove stop words in word features using negation filter. Stop words are useless words in natural language processing, such as “the”, “a”, “an” etc. We use the list of stop words in NLTK to remove these words. Table 3 shows the difference before and after removing stop words from a sentence. After these preprocessing, we use the same method as the

SL_feature to build the model.

With stop words	**Without stop words**
he is going to make a splash	he going make splash

Table 3: Stop words removal

We run the `newSL_features` function with the new word features. However, the accuracy turned out to be only 0.749, which is lower than the previous SL_features result, 0.762. The code for our document feature is in Listing 5.

Listing 5: newSL Feature

```
# Remove stop words and re-run SL_features
# remove stop words in word features using negation filter
stopwords = nltk.corpus.stopwords.words('english')
newstopwords = [word for word in stopwords if word not in
   negationwords]
new_all_words_list=[word for word in all_words_list if word not in
   newstopwords]

# get new word features of length 2000 after the stopwords are removed
new_all_words = nltk.FreqDist(new_all_words_list)
new_word_items = new_all_words.most_common(2000)
new_word_features = [word for (word,count) in new_word_items]
# re run SL_features
def newSL_features(document, new_word_features, SL):
    document_words = set(document)
    features = {}
    for word in new_word_features:
        features['contains({})'.format(word)] = (word in
   document_words)
    # count variables for the 4 classes of subjectivity
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
newSL_featuresets = [(newSL_features(d, new_word_features, SL), c) for
   (d, c) in documents]
newSL_train_set, newSL_test_set =
   newSL_featuresets[1000:],newSL_featuresets[:1000]
newSLclassifier = nltk.NaiveBayesClassifier.train(newSL_train_set)
print(nltk.classify.accuracy(newSLclassifier, newSL_test_set)) # 0.749
```

5 Experiment Description and Evaluation

After training the models, we also calculate the precision, recall and F-measure(a measure of a test's accuracy and is defined as the weighted harmonic mean of the precision and recall of the test). The outcomes are showed below.

	pos precision	pos recall	pos F-measure
Document_features	0.743215031	0.703557312	0.72284264
SL_features	0.776859504	0.743083004	0.75959596
New SL_features	0.750491159	0.754940711	0.75270936
Negation features	0.773469388	0.749011858	0.761044177

Table 4: Feature Evaluation

The code for our evaluation is in Listing 6.

Listing 6: Evaluation

```
#Subjectivity Count features- SL_classifier precision, recall and
   F-measure
# evaluation measures showing performance of negation
from nltk.metrics import *
reflist = []
testlist = []
for (features, label) in not_test_set:
    reflist.append(label)
    testlist.append(NOT_classifier.classify(features))
# define and print confusion matrix
cm = ConfusionMatrix(reflist, testlist)
print(cm)
refpos = set()
refneg = set()
testpos = set()
testneg = set()
for i, label in enumerate(reflist):
    if label == 'pos': refpos.add(i)
    if label == 'neg': refneg.add(i)
for i, label in enumerate(testlist):
    if label == 'pos': testpos.add(i)
    if label == 'neg': testneg.add(i)
# compute precision, recall and F-measure for each label
def printmeasures(label, refset, testset):
    print(label, 'precision:', precision(refset, testset))
    print(label, 'recall:', recall(refset, testset))
    print(label, 'F-measure:', f_measure(refset, testset))
    printmeasures('pos', refpos, testpos)
```

Finally, we categorize Reddit comments with four different classifers `SL_Classifier`, `Classifier`, `NOT_classifier` and `newSL_classifier`. Then we get the predict results after applying classifier. We count the number of positive and negative words for each classifier. The results are in Table 5.

	pos_list	neg_list
Document_features	2761	9316
SL_features	2875	9202
New SL_features	2998	9079
Negation features	3341	8736

Table 5: Classification Results

The code for the classification is in Listing 7. Here we only show the `SL_classifier` because we calculate other classification results by substituting the classifier name.

Listing 7: Classification

```
# Predict reddit comments with SL_Classifier
# Get the predict results after applying classifier
pos_list=[]
neg_list=[]
for comments in words:
    if(SL_classifier.classify(SL_features(comments,
    word_features,SL))=='pos'):
       # use string.join() combine list
       pos_list.append(" ".join(comments))
    if(SL_classifier.classify(SL_features(comments,
    word_features,SL))=='neg'):
       neg_list.append(" ".join(comments))

print(len(pos_list))
print(len(neg_list))
```

6 Literature Review

There are a lot of researches on sentiment analysis and many of them use Reddit as the data source. [3] uses a small dataset consisting of 4050 Arabic and English reviews from sources like Reddit, Twitter, etc. Polarity is the major feature, which contains four types: positive, negative, neutral, and spam. In this paper, the researchers created three polarity dictionaries and conduct a comparison between two free online sentiment analysis tools, SocialMention and Twendz. They show that SocialMention is more accurate to identify the polarity of Arabic/English comments.

In his bachelor thesis [4], Michael Lubitz looked into financial news articles posted on Reddit. His method is based on dictionaries and he built one that contains positive and negative words. The results suggest that the predictive power of Reddit is slightly better than using standard news paper analysis. In this paper, Michael Lubitz achieved 56.49% of accuracy in predicting the future direction of the stock market.

Here we also include a review of some researches that use other data sources. *Sentiment Analysis of Twitter Data* [2] processes 11875 manually annotated tweets to generate prior polarity. In their study, a dictionary of about 8000 English language words assigns every word a pleasantness score between 1(Negative) and 3(Positive). They first normalize the scores by dividing each score by the scale(which is 3). Words with polarity less than 0.5 are considered as negative. Another set of words whose score are higher than 0.8 are considered as positive. The rest are neutral. They use previously proposed state-of-the-art uni-gram model as the baseline and report an overall gain of over 4% for two classification tasks: a binary, positive versus negative and a 3-way positive versus negative versus neutral.

In the paper *Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis* [7], a dataset named **MPQA Corpus** is used. This study features subjectivity clues which are words and phrases that may be used to express private states. They also gave the new words reliability tags, either `strongsubj` or `weaksubj`. The clues are tagged in the lexicon with their prior polarity. With this approach, they are able to automatically identify the contextual polarity for a large subset of sentiment expressions, achieving results that are significantly better than baseline.

Also taking data from Twitter, *Twitter as a Corpus for Sentiment Analysis and Opinion Mining* [5] values this platform because the tweets contain people's opinion on different aspects of life everyday. The features they use are subjective set, which is a mixture of the positive and the negative sets, and objective set, which contain neutral data. They observe that objective texts tend to contain more common and proper nouns, while authors of subjective texts use more often personal pronouns. The best performance of their method is achieved when using bi-grams.

7 Challenges and Discussion

We have encountered some challenges in this project. One of them is a lack of data. The Trump's impeachment happened in September 2019 and is still going on. There are large volume of comments online. But most machine learning models are data-hungry and their performance relies heavily on the size of training data available. We have collected over 12077 comments for our task. We can use some data augmentation techniques to increase the number of comments in our dataset.

Another challenge we are facing is the imbalance of our data. As we can see, the majority class of the comments are negative, which is in line with our observation and expectation. The imbalance of dataset may cause really high inaccuracy because the model spends most of its time on negative examples and not learn enough from positive ones. We understand there are many data manipulation methods to cope with this problem, but would like to use the real data because it represents true distribution of the data and the experiment outcome shows the model works well. To evaluate the performance of our models, we use not only accuracy, but also precision, recall and F-measure which are useful in cases where classes aren't evenly distributed.

The theme of this project involves Trump and the event of impeaching him. Although we guarantee the quality of data we collected in the scrapping step, our analysis cannot determine whether the sentiments of these comments are for the person "Trump" or for the movement of impeaching him. A comment may show that the commenter dislikes Trump but is in favour of the impeachment. So we can't deduce from our analysis whether most people are in favor of or opposite to the impeachment of Trump.

As you we see, the Reddit API makes it extremely easy to compile a lot of news data fairly quickly. It's definitely worth the time and effort to enhance the data collection steps since it's so simple to get thousands of rows of political headlines to use for further analysis and prediction.

8 Conclusion

We have performed sentiment analysis on Reddit comments on a bombshell event - Trump impeachment. We start by preprocessing and analyzing the word frequency. There are four major features that we use to train the Naive Bayes classifier, document feature, SL feature, negation feature, and new SL feature. In our experiment part, we tried these features we have learnt in class and the highest accuracy we get is the SL feature: 0.762. The experiment generates two list of words – positive and negative lists. From the length of the lists: 2875 of positive and 9202 of negative words, we conclude that most subreddits contain negative words toward the impeachment news. This result aligns well with a recent poll [1].

9 Future Work

In the future of using sentiment analysis, we are going to continue to dig deeper, far past the surface of the number of likes, comments and shares, and aim to reach, and truly understand, the significance of social media interactions and what they tell us about the consumers behind the screens.

We would like to apply some in-advance analysis, like conjunctive analysis which can very interesting. Our current models are able to cope with sentences with "prototype" negation word. However, there are so many syntactic, semantic and morphological devices expressing negation in English, which does make a promising path to pursue. We are going to dig deeper into negation.

References

[1] Poll finds negative views of democrats' and trump's handling of impeachment inquiry. *CBS NEWS* (Nov 2019).

[2] AGARWAL, A., XIE, B., VOVSHA, I., RAMBOW, O., AND PASSONNEAU, R. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Language in Social Media (LSM 2011)* (Portland, Oregon, June 2011), Association for Computational Linguistics, pp. 30–38.

[3] AL-KABI, M., AL-QUDAH, N. M., ALSMADI, I., DABOUR, M., AND WAHSHEH, H. Arabic/english sentiment analysis: an empirical study. In *The Fourth International Conference on Information and Communication Systems (ICICS 2013)* (2013), pp. 23–25.

[4] LUBITZ, M. *Who drives the market? Sentiment analysis of financial news posted on Reddit and the Financial Times*. PhD thesis, Bachelor's Thesis, Albert-Ludwigs-University, Department of Computer Science, 2017.

[5] PAK, A., AND PAROUBEK, P. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc* (2010), vol. 10, pp. 1320–1326.

[6] WILSON, T., WIEBE, J., AND HOFFMANN, P. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing* (2005).

[7] WILSON, T., WIEBE, J., AND HOFFMANN, P. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing* (Vancouver, British Columbia, Canada, Oct. 2005), Association for Computational Linguistics, pp. 347–354.

A Most Frequent Words After Stop Words Removal

```
    trump 0.021630474083842013
    election 0.009113724822936384
    https 0.008135355297020226
    would 0.006455111980772912
    like 0.006274326307505796
    people 0.005583086968543293
    president 0.005019461046004637
    impeachment 0.005008826594635983
    get 0.00489184762958079
    one 0.004774868664525597
    think 0.004455835123465980 5
    primary 0.004381393963885403
    even 0.003753961333134823 6
    deadline 0.003753961333134823 6
    house 0.003615713465342323
    know 0.00349873450028713
    republicans 0.003413658889337899
    time 0.003222387647021 29
    general 0.003179700959227513 5
    going 0.00311589425101559
    party 0.003052087542803667
    way 0.002913839675011166
    said 0.002871301869536550 6
    see 0.002743688453112704
    us 0.002743688453112704
    could 0.00273305400174405
    ukraine 0.002573537231214241 6
    right 0.002562902779845587 6
    registration 0.00254163387710828
    vote 0.002488461620265010 3
    still 0.002477827168896356 7
    republican 0.002424654912053087
    gop 0.002424654912053087
    say 0.00241402046068443 3
    white 0.002403386009315779 5
    make 0.002328944849735202 2
    go 0.00228640704426058 7
    never 0.00225450369015462 5
    really 0.00222260033604866 3
    country 0.002190696981942701 4
    want 0.002105621370993470 5
    news 0.002105621370993470 5
    much 0.002094986919624816 5
    april 0.002084352468256163
    shit 0.001978007954569624
    new 0.001978007954569624
    also 0.0019673735032009697
    take 0.001956739051832316
    office 0.001956739051832316
    good 0.0019248356977263543
```

B Most Frequent Words After Stemming

```
    trump 0.022045217687219516
    elect 0.010092094348852543
    http 0.008996745857881192
    like 0.007284599187527915
    impeach 0.007252695833421954
    get 0.006816683327307l445
    would 0.006455111980772912
    presid 0.005976561669183486
    republican 0.005838313801390986
    think 0.005646893676755216
    peopl 0.005614990322649255
    go 0.005402301295276177
    one 0.005285322330220984
    say 0.005115171108322522
    primari 0.004477104026203288
    know 0.00437075951251675
    make 0.004200608290618287
    vote 0.003956015909139248
    time 0.003839036944084054 5
    even 0.0037858646872407854
    deadlin 0.0037539613331348236
    hous 0.0036901546249229
    gener 0.003626347916710977
    want 0.0033392177297573216
    see 0.0033073143756513603
    need 0.003296679924282706
    parti 0.003296679924282706
    state 0.003115894251015 59
    way 0.0030414530914350127
    thing 0.003030818640066359
    call 0.002913839675011166
    said 0.0028713018695365506
    right 0.002743688453112704
    us 0.002743688453112704
    could 0.00273305400174405
    take 0.00273305400174405
    year 0.0026692472935321266
    support 0.002626709488057511
    ukrain 0.0025735372312142416
    registr 0.00254163387710828
    democrat 0.002520364974370972
    senat 0.002520364974370972
    still 0.0024778271688963567
    gop 0.0024778271688963567
    countri 0.002456558266159049
    white 0.002445923814790395
    use 0.0024352893634217413
    fuck 0.002414020460684433
    look 0.0024033860093157795
    tri 0.0023927515579471254
```