Sentiment Analysis with Trump Impeachment 2019

Team 7:
Heng Fang
Harper He
Renjie Luo
Te Pi
Puzhen Qian

Table of Contents

Background

- August 12th, 2019 Whistleblower

- September 24th, 2019 House launched the impeachment inquiry

Project Motivation

- Opinion of the public matters
- Sentiment analysis
    - understand people’s feelings
    - public relation/customer service
    - artificial intelligence
- Reddit

Data Scraping

- What is data scraping
  - get data from the web

- Data Scraping Methods
  - Microsoft Excel with dynamic web queries
  - data scraping tool like PRAW

Data Preprocessing

```
#Reddit Parser
import praw

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

subreddit=reddit.subreddit('vipkid')
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))
```

Data Preprocessing

- Data preprocessing purpose
  - transform raw data to understandable format

- Data preprocessing methods
  - tokenization
  - stop words
  - normalization

Frequency Analysis

- Read the data

```
## Task 1 Tokenize
import nltk
from nltk import FreqDist
f=open('C:/Users/65730/TrumpImpeach_Reddit_Comments.txt',encoding = "utf-8")
raw = f.read()
```

- Word tokenize

```
#read file and tokenize
words=nltk.word_tokenize(raw)
print(len(words))
```

```
247944
```

- Remove non-alpha words

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

```
182074
```

Frequency Analysis

- Remove stop words

```
#remove stopwords and stemming based on Martin Porter's work
stopwords=nltk.corpus.stopwords.words('english')
stoppedtrump=[w for w in revisedwords if w not in stopwords]
porter=nltk.PorterStemmer()
stem=[porter.stem(t) for t in stoppedtrump]
len(stem)
```

```
94034
```

- Stem / Lematize

In [6]:
```
#Use NLTK FreqDist to Count Words
#(normalized by the length of documents)
#get top frequencies after stopwords but before stemming
trumpfdist = FreqDist(stoppedtrump)
trumpstoptopkeys=trumpfdist.most_common(50)
for item in trumpstoptopkeys:
    print(item[0],item[1]/len(stem))
```

```
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
```

```
#Skip stemming, get top frequencies after lemmatizing
#(normalized by the length of documents)
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))
```

```
trump 0.022034583235850863
election 0.00948593062083927
http 0.008996745857881192
like 0.006465746432141566
would 0.006455111980772912
republican 0.005583813801390986
get 0.00571070038496714
people 0.005614990322649255
one 0.00528532330220984
president 0.00527468787885233
impeachment 0.005019461046004637
```

Features - **Document_feature**

- Occurrence
- Most frequent 2,000 words

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	

Features - Document_feature

```
##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
```

```
0.727
```

Features - SL_feature

- Subjectivity Lexicon
- Count the positive and negative subjectivity words

	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

Count once

Count twice

Features - 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
```

Features - SL_feature

```
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
```

Features - **Negation_feature**

- Handle negation

it doesn't feel like one → it doesn't NOT_feel like one

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

Features - Negation_feature

```
# 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))
```

```
['there', 'is', 'a', 'difference', 'between', 'movies', 'with', 'the', 'courage', 'to', 'go', 'over', 'the', 'top', 'and', 'movies', 'that',
"don't", 'care', 'about', 'being', 'stupid']
['a', 'farce', 'of', 'a', 'parody', 'of', 'a', 'comedy', 'of', 'a', 'premise', ',', 'it', "isn't", 'a', 'comparison', 'to', 'reality', 'so',
'much', 'as', 'it', 'is', 'a', 'commentary', 'about', 'our', 'knowledge', 'of', 'films', '.']
['i', "didn't", 'laugh', '.', 'i', "didn't", 'smile', '.', 'i', 'survived', '.']
['i', "didn't", 'laugh', '.', 'i', "didn't", 'smile', '.', 'i', 'survived', '.']
['most', 'of', 'the', 'problems', 'with', 'the', 'film', "don't", 'derive', 'from', 'the', 'screenplay', ',', 'but', 'rather', 'the', 'mediocre', 'performances', 'by', 'most', 'of', 'the', 'actors', 'involved']
['the', 'lack', 'of', 'naturalness', 'makes', 'everything', 'seem', 'self-consciously', 'poetic', 'and', 'forced', '.', '.', '.', "it's",
'a', 'pity', 'that', "[nelson's]", 'achievement', "doesn't", 'match', 'his', 'ambition', '.']
0.762
```

Features - **newSL_feature**

- Remove stop words

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

Features - 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]
```

Features - newSL_feature

```
#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
```

```
0.749
```

Experiment description and evaluation

- Subjectivity count features

- Get the results

```
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)
```

```
pos precision: 0.773469387755102
pos recall: 0.7490118577075099
pos F-measure: 0.7610441767068272
```

```
#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))
```

```
2875
9202
```

Experiment description and evaluation

	pos precision	pos recall	pos F-measure	pos_list	neg_list
Documant_features	0.743215031	0.703557312	0.72284264	2761	9316
SL_features	0.776859504	0.743083004	0.75959596	2875	9202
New SL_features	0.750491159	0.754940711	0.75270936	2998	9079
Negation features	0.773469388	0.749011858	0.761044177	3341	8736

Related Work

- Al-Kabi, M., et al, Arabic/English Sentiment Analysis: An Empirical Study.

- Lubitz, M. Who drives the market? Sentiment analysis of financial news posted on Reddit and the Financial Times.

Conclusion and Discussion

- Reddit API helps to enhance the data collection steps
- Highest accuracy 0.762
- Two list of words of different features
    - Positive list SL_Features:2,875 words,New SL_Features:2998,Negation:3341
    - Negative list SL_Features:9,202 words,New SL_Features:9079,Negation:8736
- Most subreddits contain **negative words** toward the impeachment news
- Further explorations
    - Significance of social media interactions
    - What they tell us about the users behind the screens

Thank you

Present by Price Qian from Team 7