CSCE 638: Natural Language Processing

Assignment 4 Report

Submitted By – Sameer Kumar Behera UIN: 526004296

System Requirements:

• Python must be installed
• Installation Link: https://www.python.org/downloads/
Note: Include the english.stop file from PA2 as a lot of the code has been built from existing code in PA2

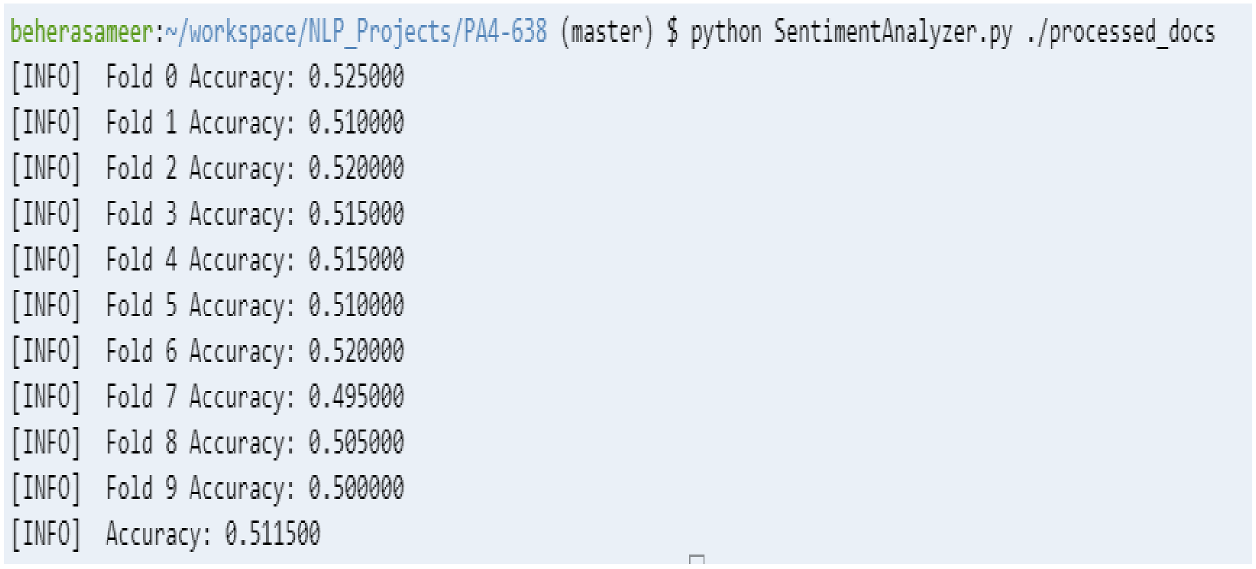
Compile and Run Method:

1. Open a Terminal
2. Go to the Project Folder, i.e. PA4-638
3. On the Terminal, Enter the Following Command:

python SentimentAnalyzer.py ./processed_docs

Result and Analysis:

10-Fold Cross-Validation Accuracy Results with Nearness Limit =10



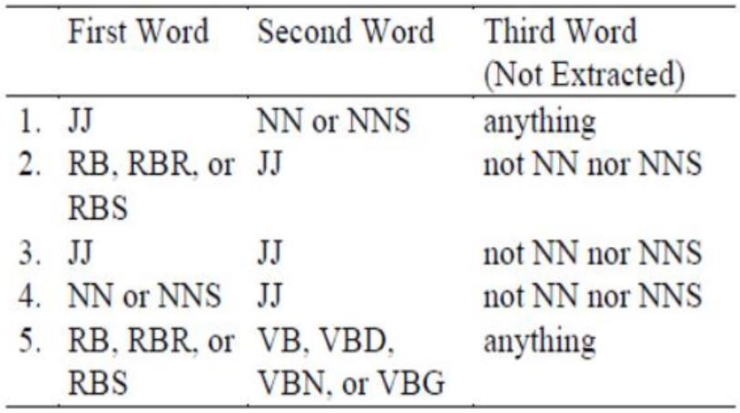
```
beherasameer:~/workspace/NLP_Projects/PA4-638 (master) $ python SentimentAnalyzer.py ./processed_docs
[INFO]  Fold 0 Accuracy: 0.525000
[INFO]  Fold 1 Accuracy: 0.510000
[INFO]  Fold 2 Accuracy: 0.520000
[INFO]  Fold 3 Accuracy: 0.515000
[INFO]  Fold 4 Accuracy: 0.515000
[INFO]  Fold 5 Accuracy: 0.510000
[INFO]  Fold 6 Accuracy: 0.520000
[INFO]  Fold 7 Accuracy: 0.495000
[INFO]  Fold 8 Accuracy: 0.505000
[INFO]  Fold 9 Accuracy: 0.500000
[INFO]  Accuracy: 0.511500
```

Overall Accuracy = 51.15%

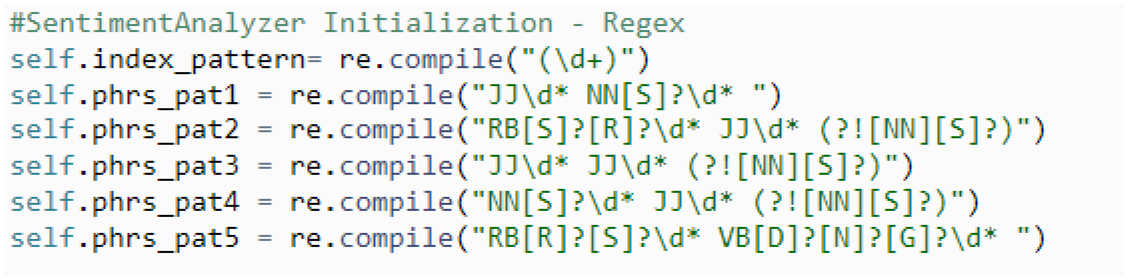
Implementation:

1.Regex

The Following Regular Expressions were used to extract two-word phrases:



	First Word	Second Word	Third Word (Not Extracted)
1.	JJ	NN or NNS	anything
2.	RB, RBR, or RBS	JJ	not NN nor NNS
3.	JJ	JJ	not NN nor NNS
4.	NN or NNS	JJ	not NN nor NNS
5.	RB, RBR, or RBS	VB, VBD, VBN, or VBG	anything



```
#SentimentAnalyzer Initialization - Regex
self.index_pattern= re.compile("(\d+)")
self.phrs_pat1 = re.compile("JJ\d* NN[S]?\d* ")
self.phrs_pat2 = re.compile("RB[S]?[R]?\d* JJ\d* (?![NN][S]?)")
self.phrs_pat3 = re.compile("JJ\d* JJ\d* (?![NN][S]?)")
self.phrs_pat4 = re.compile("NN[S]?\d* JJ\d* (?![NN][S]?)")
self.phrs_pat5 = re.compile("RB[R]?[S]?\d* VB[D]?[N]?[G]?\d* ")
```

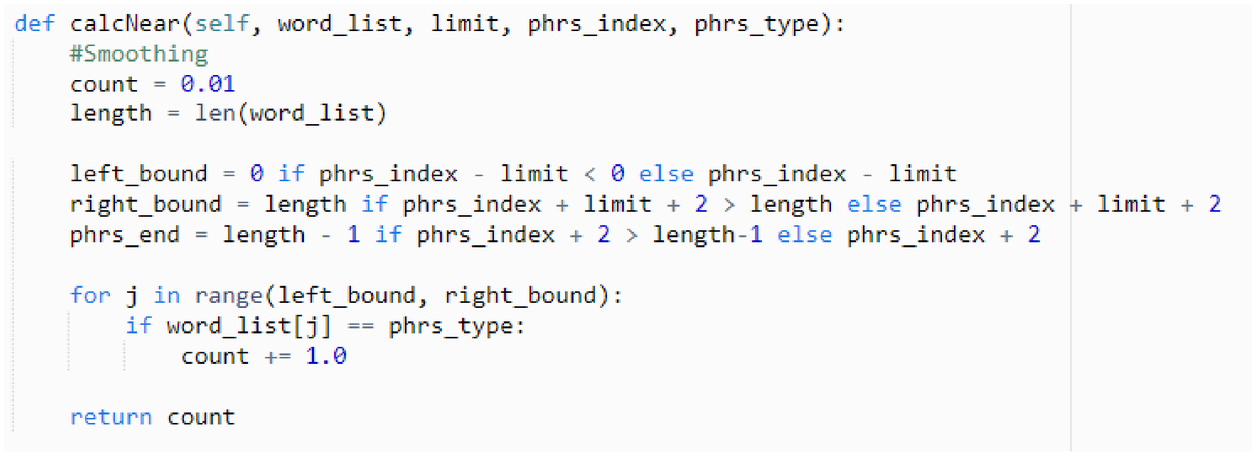
To make Execution faster, the tags were stored in a format “Tag”+<index>. The ‘\d+’ identifier in the above regular expression denotes the occurrence of corresponding index. Matches were like: “JJ21 NN22”, etc.

Examples of Sentiment Phrases Generated are:

bad films, good intentions, emotional closeness, good chance, slow parts, juvenile comedy, cheesy gore, special effects, awesome spectacle, beautiful cast, fast-paced action, pretty much, well sorry, brutally phony, fully aware, quite astounding, evil dead, automatically start, far surpassing, heavily brought, viciously decapitated, romantic comedy, flawed comedy, pesky ex-boyfriend, humorous cameo, fresh sort, romantic comedy, stubborn father, right time, usual flicks, beautiful actress, straightforward guy, romantic comedy, good laughs, etc.

2. NEAR Operator

For each phrase, a window limit of 10 nearby words that come before and after the phrase was used. Here, limit specifies the window length, phrs_type is (excellent/poor) and phrs_index the index of phrase. Also, to avoid division of zero cases, the count was initialized with 0.01.

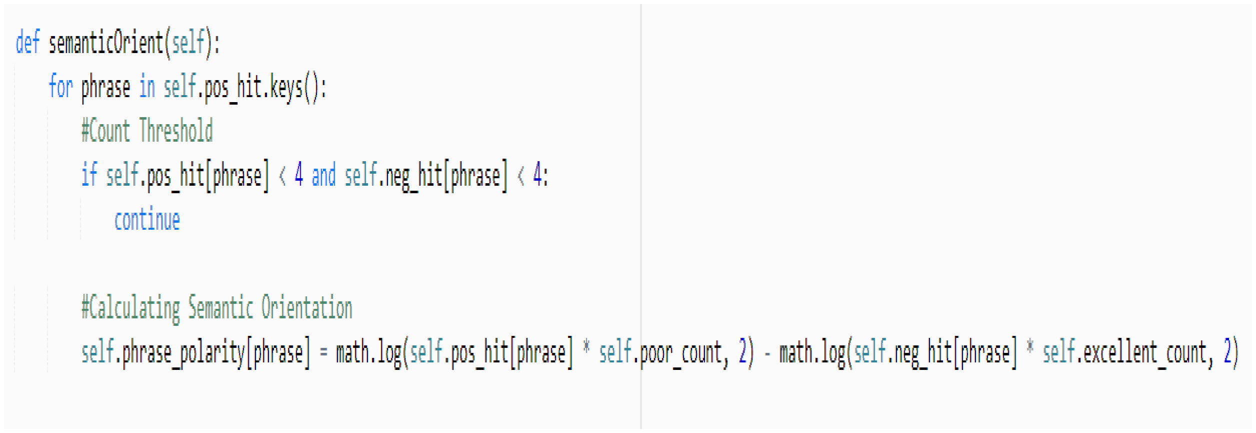


```
def calcNear(self, word_list, limit, phrs_index, phrs_type):
    #Smoothing
    count = 0.01
    length = len(word_list)

    left_bound = 0 if phrs_index - limit < 0 else phrs_index - limit
    right_bound = length if phrs_index + limit + 2 > length else phrs_index + limit + 2
    phrs_end = length - 1 if phrs_index + 2 > length-1 else phrs_index + 2

    for j in range(left_bound, right_bound):
        if word_list[j] == phrs_type:
            count += 1.0

    return count
```

3. Semantic Orientation

Semantic Orientation reflects whether the word is close to "excellent" keyword or to "poor" keyword.

$$\mathrm{SO}(\mathit{phrase}) = \log_2 \left[\frac{\mathrm{hits}(\mathit{phrase}\ \mathrm{NEAR}\ \text{“excellent”})\ \mathrm{hits}(\text{“poor”})}{\mathrm{hits}(\mathit{phrase}\ \mathrm{NEAR}\ \text{“poor”})\ \mathrm{hits}(\text{“excellent”})} \right]$$



```
def semanticOrient(self):
    for phrase in self.pos_hit.keys():
        #Count Threshold
        if self.pos_hit[phrase] < 4 and self.neg_hit[phrase] < 4:
            continue

        #Calculating Semantic Orientation
        self.phrase_polarity[phrase] = math.log(self.pos_hit[phrase] * self.poor_count, 2) - math.log(self.neg_hit[phrase] * self.excellent_count, 2)
```

4. Polarity Score

Using semantic orientation of individual phrases, polarity for a review is calculated. If polarity is greater than 0, then review is classified as positive, else negative.

```
def classify(self, words):
    pos_list = []
    word_list = []
    position = 0

    for word in words:
        splits = word.split('_')
        word_list.append(splits[0])
        pos_list.append(splits[1] + str(position))
        position += 1

    pos_str = ' '.join(pos_list)

    #For Extracting Patterns
    match_pattern = []
    match_pattern.extend(self.phrs_pat1.findall(pos_str))
    match_pattern.extend(self.phrs_pat2.findall(pos_str))
    match_pattern.extend(self.phrs_pat3.findall(pos_str))
    match_pattern.extend(self.phrs_pat4.findall(pos_str))
    match_pattern.extend(self.phrs_pat5.findall(pos_str))

    polarity = 0

    for match in match_pattern:
        pattern_parts = match.split(' ')
        index = self.index_pattern.findall(pattern_parts[0])
        phrase_index = int(index[0])
        phrase = word_list[phrase_index] + " " + word_list[phrase_index + 1]
        polarity += self.phrase_polarity.get(phrase, 0)

    prediction = 'pos' if polarity > 0 else 'neg'
    return prediction
```

Bugs or Limitations:

No such issues were found.