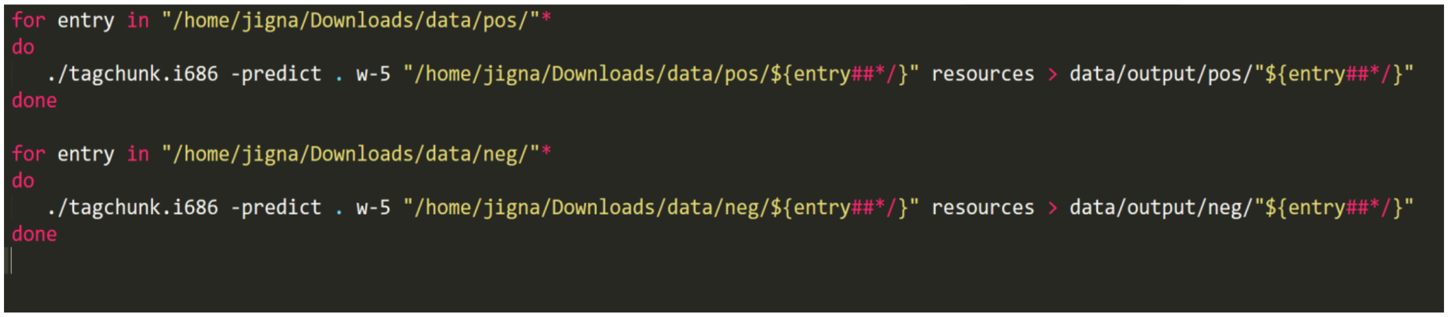
NLP Assignment #4 Report

Submitted by:
Jigna Reshamwala [425008356]

This report contains details about NLP Assignment#4.

Commands to run POS tagger:

Following is the run.sh script for getting the tagged output files.



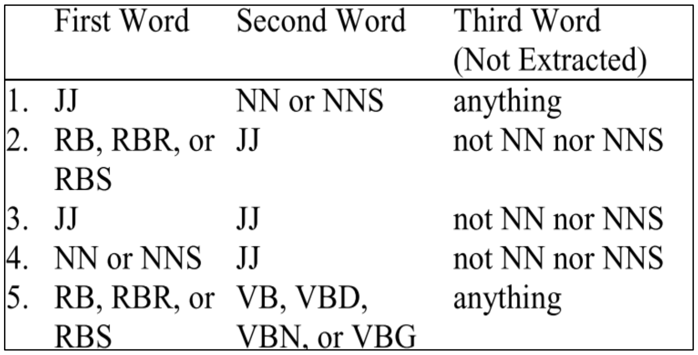
```
for entry in "/home/jigna/Downloads/data/pos/"*
do
   ./tagchunk.i686 -predict . w-5 "/home/jigna/Downloads/data/pos/${entry##*/}" resources > data/output/pos/"${entry##*/}"
done

for entry in "/home/jigna/Downloads/data/neg/"*
do
   ./tagchunk.i686 -predict . w-5 "/home/jigna/Downloads/data/neg/${entry##*/}" resources > data/output/neg/"${entry##*/}"
done
```

Regular Expressions and examples of Sentiment Phrases:

- I parsed the tagged data files and made the string of the POS tags and their index as follows:

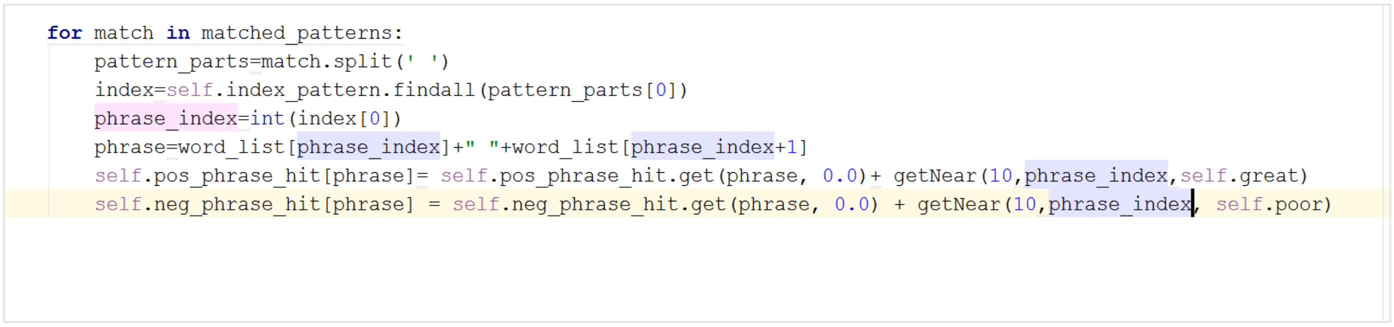
  Eg. NNS0 NN1 .2 NNS3 ,4 CD5 IN6 DT7 NNS8 IN9 DT10 NNS11 CC12 NN13 IN14 NN15 NN16 ,17 VBZ18 TO19 DT20 JJ21 NN22 IN23 DT24 NN25 ,26 IN27 RB28 JJ29 NN30 .31
- Then I used following regex patterns to extract the required POS patterns:



	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

Regex Patterns
`"JJ\d* NN[S]?\d* "`
`"RB[S]?[R]?\d* JJ\d* (?![NN][S]?)"`
`"JJ\d* JJ\d* (?![NN][S]?)"`
`"NN[S]?\d* JJ\d* (?![NN][S]?)"`
`"RB[R]?[S]?\d* VB[D]?[N]?[G]?\d* "`

- Then for each matched pattern I extracted the index of the first tag:
  Eg: match: JJ47 NN48
  Extracted 47 and searched for “great” and “poor” in the proximity of word at the index 47 in the original text using “NEAR” operator:

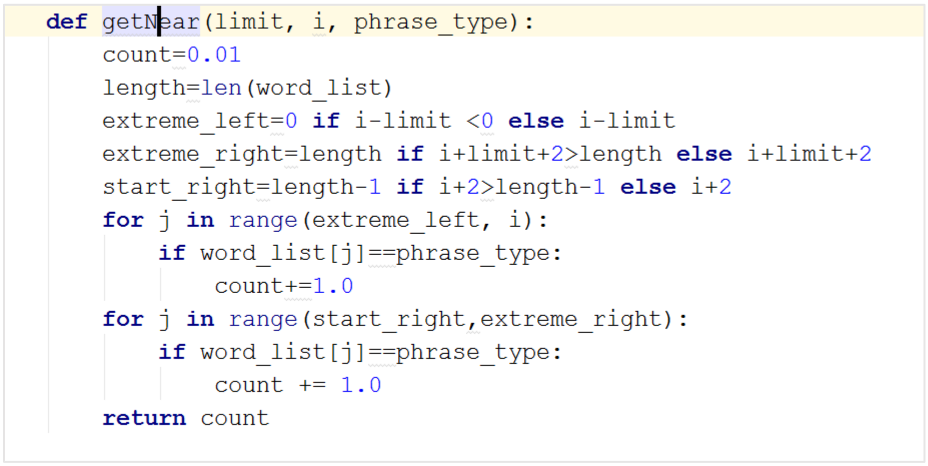


```
for match in matched_patterns:
    pattern_parts=match.split(' ')
    index=self.index_pattern.findall(pattern_parts[0])
    phrase_index=int(index[0])
    phrase=word_list[phrase_index]+" "+word_list[phrase_index+1]
    self.pos_phrase_hit[phrase]= self.pos_phrase_hit.get(phrase, 0.0)+ getNear(10,phrase_index,self.great)
    self.neg_phrase_hit[phrase] = self.neg_phrase_hit.get(phrase, 0.0) + getNear(10,phrase_index, self.poor)
```

<u>Code to conduct search and implementing the “NEAR” operator:</u>

getNear() method takes three parameters:

- limit: Number of words before anf after the phrase to be matched. Eg: 10
- i: index of the phrase around which we look for the matching of of “phrase_type”
- phrase_type: string to match “poor” or “great”

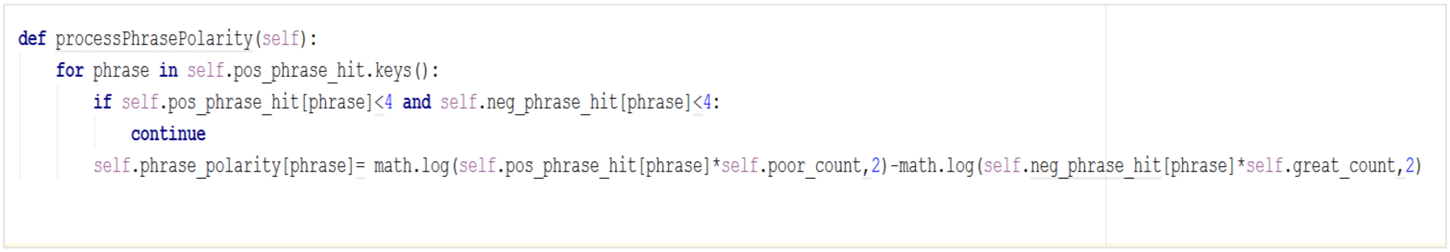


```
def getNear(limit, i, phrase_type):
    count=0.01
    length=len(word_list)
    extreme_left=0 if i-limit <0 else i-limit
    extreme_right=length if i+limit+2>length else i+limit+2
    start_right=length-1 if i+2>length-1 else i+2
    for j in range(extreme_left, i):
        if word_list[j]==phrase_type:
            count+=1.0
    for j in range(start_right,extreme_right):
        if word_list[j]==phrase_type:
            count += 1.0
    return count
```

<u>Code to check semantic orientation for each sentiment phrase:</u>

The processPhrasePolarity() methods processes the phrase semantic orientation based on following formula:

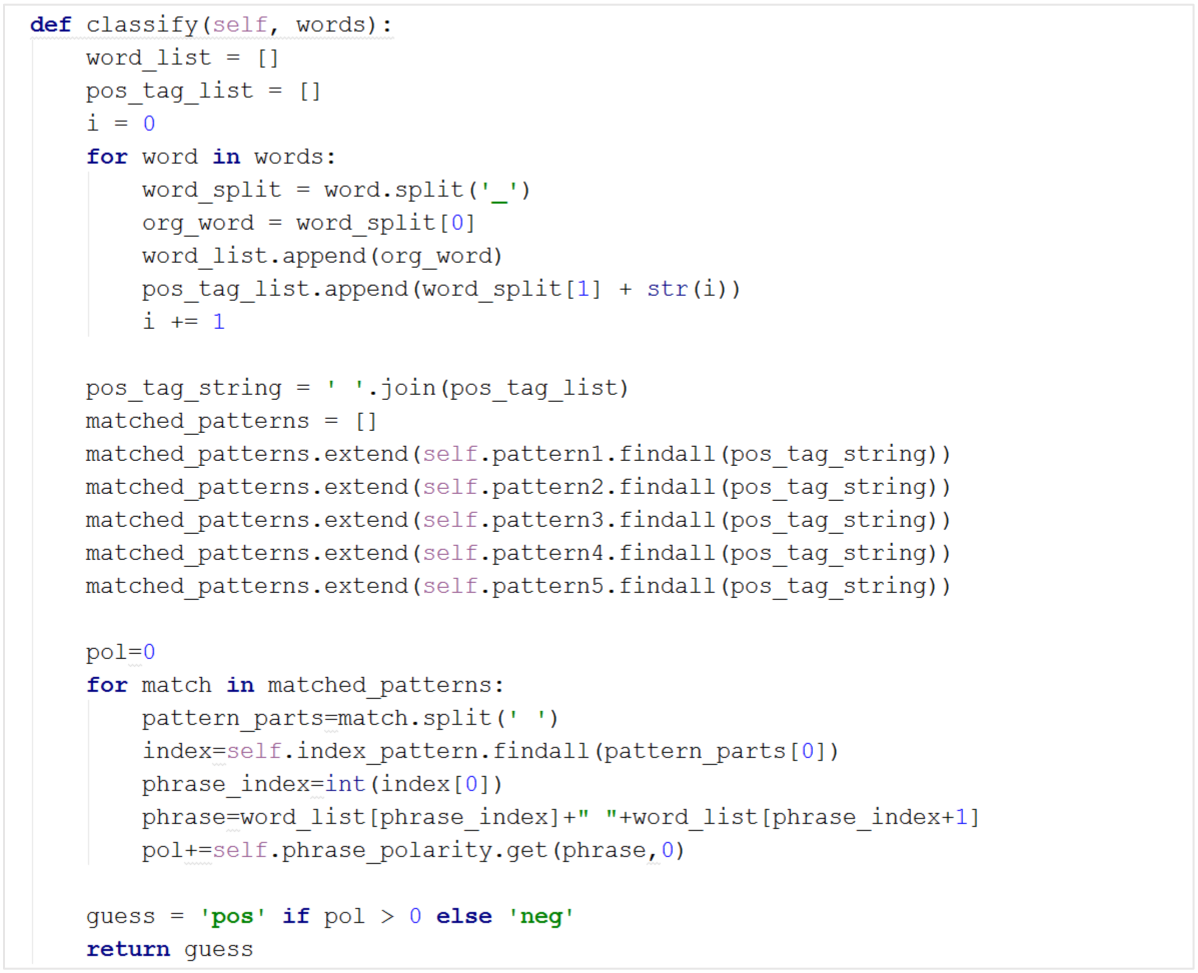
$$\mathrm{SO}(\mathit{phrase}) = \log_2\left[\frac{\text{hits}(\mathit{phrase}\ \text{NEAR “excellent”})\ \text{hits(“poor”)}}{\text{hits}(\mathit{phrase}\ \text{NEAR “poor”})\ \text{hits(“excellent”)}}\right]$$



```
def processPhrasePolarity(self):
    for phrase in self.pos_phrase_hit.keys():
        if self.pos_phrase_hit[phrase]<4 and self.neg_phrase_hit[phrase]<4:
            continue
        self.phrase_polarity[phrase]= math.log(self.pos_phrase_hit[phrase]*self.poor_count,2)-math.log(self.neg_phrase_hit[phrase]*self.great_count,2)
```

<u>Code to Calculate the polarity score for each test review:</u>

classify() method classifies the test review by finding the sum of the SO of the phrases in the test review. Just like in the training the phrases are found using regular expressions over POS tags string. And then the SO of the phrases already calculated are added to find the final polarity score.



```python
def classify(self, words):
    word_list = []
    pos_tag_list = []
    i = 0
    for word in words:
        word_split = word.split('_')
        org_word = word_split[0]
        word_list.append(org_word)
        pos_tag_list.append(word_split[1] + str(i))
        i += 1

    pos_tag_string = ' '.join(pos_tag_list)
    matched_patterns = []
    matched_patterns.extend(self.pattern1.findall(pos_tag_string))
    matched_patterns.extend(self.pattern2.findall(pos_tag_string))
    matched_patterns.extend(self.pattern3.findall(pos_tag_string))
    matched_patterns.extend(self.pattern4.findall(pos_tag_string))
    matched_patterns.extend(self.pattern5.findall(pos_tag_string))

    pol=0
    for match in matched_patterns:
        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]
        pol+=self.phrase_polarity.get(phrase,0)

    guess = 'pos' if pol > 0 else 'neg'
    return guess
```

1. *<u>How to compile and run the code</u>*

My code is tested for Python 2.7.12. It might give errors with Python 3.

To compile and run the code following are the two methods:

<u>Using IDE</u>

- Import the project in any Python IDE

1. For 10 cross validation:
    - Edit Run Configurations and set script parameters for SetimentAnalyzer.py to: ..\tagged_data
2. For Testing on a test set:
    - Edit Run Configurations and set script parameters for SetimentAnalyzer.py to: ..\tagged_data ..\tagged_data_test

- Run the script to get the output

<u>From Terminal</u>

- Open Terminal
- Go the folder NLP_HW4/python
- Type in the following command:

1. For 10-cross validation:

```
python SetimentAnalyzer..py ..\tagged_data
```

2. For testing

```
python SetimentAnalyzer..py ..\tagged_data ..\tagged_data_test
```

2. ***<u>Results and Analysis</u>***

Nearness limit: 10, Phrase Polarity Count Threshold: 4

```
[INFO]      Fold 0 Accuracy: 0.510000
[INFO]      Fold 1 Accuracy: 0.540000
[INFO]      Fold 2 Accuracy: 0.535000
[INFO]      Fold 3 Accuracy: 0.515000
[INFO]      Fold 4 Accuracy: 0.505000
[INFO]      Fold 5 Accuracy: 0.500000
[INFO]      Fold 6 Accuracy: 0.555000
[INFO]      Fold 7 Accuracy: 0.530000
[INFO]      Fold 8 Accuracy: 0.525000
[INFO]      Fold 9 Accuracy: 0.515000
[INFO]      Accuracy: 0.523000
```

Nearness limit: 15, Phrase Polarity Count Threshold: 4

```
[INFO]      Fold 0 Accuracy: 0.540000
[INFO]      Fold 1 Accuracy: 0.535000
[INFO]      Fold 2 Accuracy: 0.500000
[INFO]      Fold 3 Accuracy: 0.535000
[INFO]      Fold 4 Accuracy: 0.525000
[INFO]      Fold 5 Accuracy: 0.510000
[INFO]      Fold 6 Accuracy: 0.570000
[INFO]      Fold 7 Accuracy: 0.510000
[INFO]      Fold 8 Accuracy: 0.530000
[INFO]      Fold 9 Accuracy: 0.560000
```

```
    [INFO]      Accuracy: 0.531500
```

3. ***Errors and Bugs***

There are no known bugs.