Note: My code runs on python 3.0

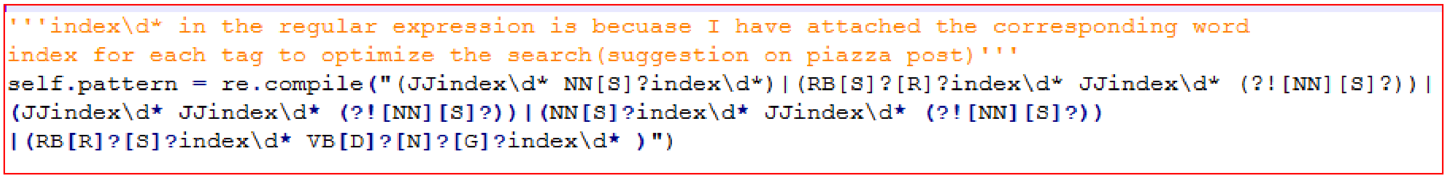
Command to run the code

Python SemanticLexiconInduction.py .\processed_docs

Regular Expressions and examples of Sentiment Phrases:

I have designed a single regular expression to search all the patterns, below is my regular expression

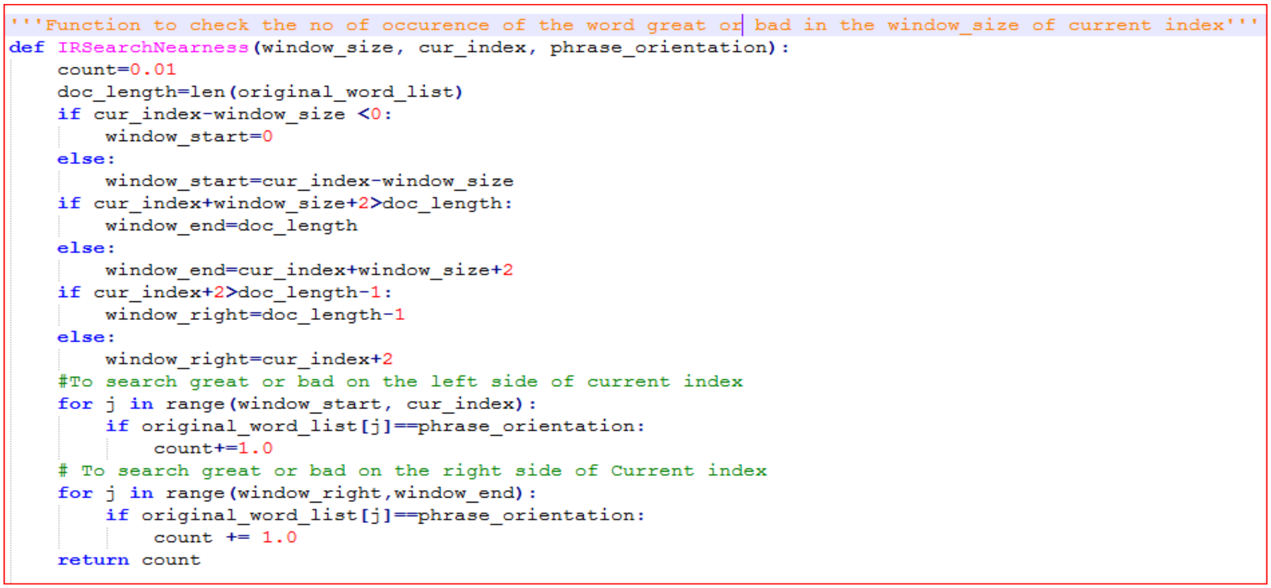
e.g. It will find the patterns like “JJindex3 NNS”, here index has no meaning other than to specify the location of the original word for the corresponfing POS_tag.



```
'''index\d* in the regular expression is becuase I have attached the corresponding word
index for each tag to optimize the search(suggestion on piazza post)'''
self.pattern = re.compile("(JJindex\d* NN[S]?index\d*)|(RB[S]?[R]?index\d* JJindex\d* (?![NN][S]?))|
(JJindex\d* JJindex\d* (?![NN][S]?))|(NN[S]?index\d* JJindex\d* (?![NN][S]?))
|(RB[R]?[S]?index\d* VB[D]?[N]?[G]?index\d* )")
```

code to conduct search including implementing the “NEAR” operator.

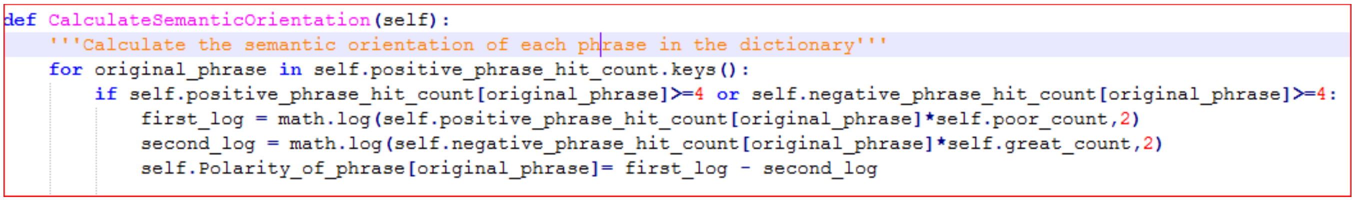
This function checks the no of occurrences of the word great or bad in the window_size of current index



```
'''Function to check the no of occurence of the word great or bad in the window_size of current index'''
def IRSearchNearness(window_size, cur_index, phrase_orientation):
    count=0.01
    doc_length=len(original_word_list)
    if cur_index-window_size <0:
        window_start=0
    else:
        window_start=cur_index-window_size
    if cur_index+window_size+2>doc_length:
        window_end=doc_length
    else:
        window_end=cur_index+window_size+2
    if cur_index+2>doc_length-1:
        window_right=doc_length-1
    else:
        window_right=cur_index+2
    #To search great or bad on the left side of current index
    for j in range(window_start, cur_index):
        if original_word_list[j]==phrase_orientation:
            count+=1.0
    # To search great or bad on the right side of Current index
    for j in range(window_right,window_end):
        if original_word_list[j]==phrase_orientation:
            count += 1.0
    return count
```

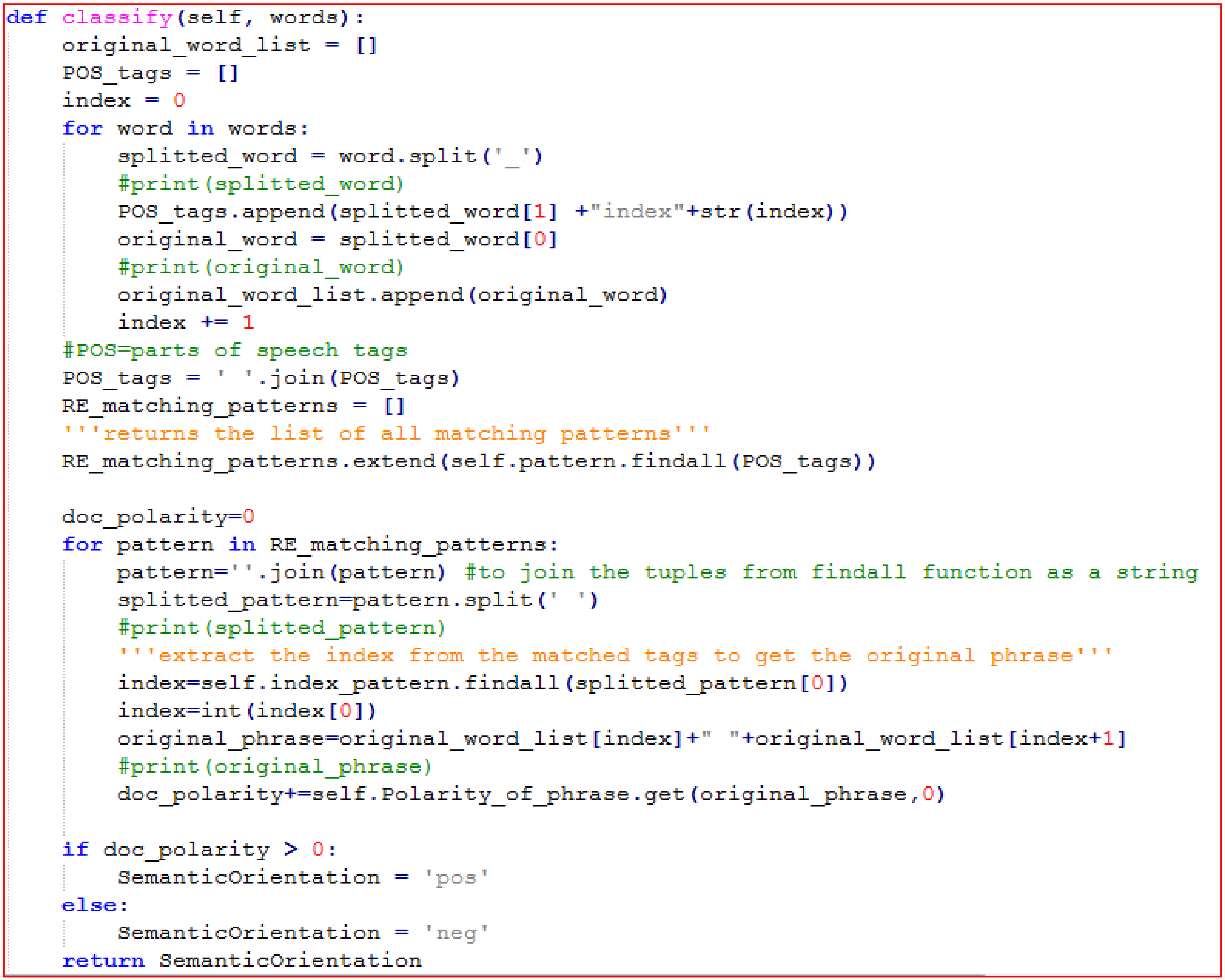
Code to check semantic orientation:

This function calculates the semantic orientation of each phrase in the dictionary after



```
def CalculateSemanticOrientation(self):
    '''Calculate the semantic orientation of each phrase in the dictionary'''
    for original_phrase in self.positive_phrase_hit_count.keys():
        if self.positive_phrase_hit_count[original_phrase]>=4 or self.negative_phrase_hit_count[original_phrase]>=4:
            first_log = math.log(self.positive_phrase_hit_count[original_phrase]*self.poor_count,2)
            second_log = math.log(self.negative_phrase_hit_count[original_phrase]*self.great_count,2)
            self.Polarity_of_phrase[original_phrase]= first_log - second_log
```

polarity score for each test review:

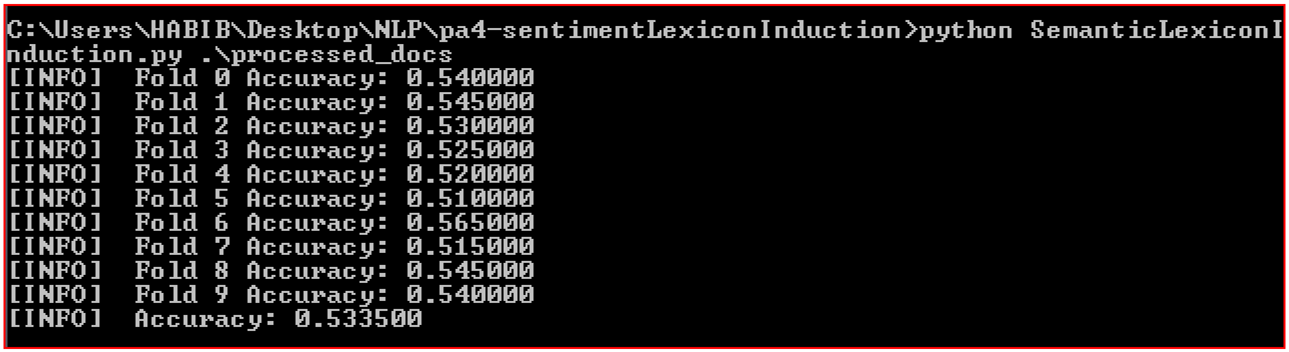


```
def classify(self, words):
    original_word_list = []
    POS_tags = []
    index = 0
    for word in words:
        splitted_word = word.split('_')
        #print(splitted_word)
        POS_tags.append(splitted_word[1] +"index"+str(index))
        original_word = splitted_word[0]
        #print(original_word)
        original_word_list.append(original_word)
        index += 1
    #POS=parts of speech tags
    POS_tags = ' '.join(POS_tags)
    RE_matching_patterns = []
    '''returns the list of all matching patterns'''
    RE_matching_patterns.extend(self.pattern.findall(POS_tags))

    doc_polarity=0
    for pattern in RE_matching_patterns:
        pattern=''.join(pattern) #to join the tuples from findall function as a string
        splitted_pattern=pattern.split(' ')
        #print(splitted_pattern)
        '''extract the index from the matched tags to get the original phrase'''
        index=self.index_pattern.findall(splitted_pattern[0])
        index=int(index[0])
        original_phrase=original_word_list[index]+" "+original_word_list[index+1]
        #print(original_phrase)
        doc_polarity+=self.Polarity_of_phrase.get(original_phrase,0)

    if doc_polarity > 0:
        SemanticOrientation = 'pos'
    else:
        SemanticOrientation = 'neg'
    return SemanticOrientation
```

Results:



```
C:\Users\HABIB\Desktop\NLP\pa4-sentimentLexiconInduction>python SemanticLexiconI
nduction.py .\processed_docs
[INFO]  Fold 0 Accuracy: 0.540000
[INFO]  Fold 1 Accuracy: 0.545000
[INFO]  Fold 2 Accuracy: 0.530000
[INFO]  Fold 3 Accuracy: 0.525000
[INFO]  Fold 4 Accuracy: 0.520000
[INFO]  Fold 5 Accuracy: 0.510000
[INFO]  Fold 6 Accuracy: 0.565000
[INFO]  Fold 7 Accuracy: 0.515000
[INFO]  Fold 8 Accuracy: 0.545000
[INFO]  Fold 9 Accuracy: 0.540000
[INFO]  Accuracy: 0.533500
```