REPORT - (CSCE-689, Programming Assignment #4 Sentiment Lexicon Induction)

Name – Shiva Kumar Pentyala
UIN - 127003995

1. Compile and Execution:
   - Developed with python 2.7
   - The original training data and results are in the zip folder, along with the source code.
     Steps –
     1. Unzip the file.
     2. cd into that folder
     3. python python/SentimentAnalyzer.py --tagged=data/tagged --untagged=data/untagged
     4. Output will get printed on the terminal

2. Results and Analysis Results:

```
[INFO] Fold 0 Accuracy: 0.480000
[INFO] Fold 1 Accuracy: 0.505000
[INFO] Fold 2 Accuracy: 0.520000
[INFO] Fold 3 Accuracy: 0.520000
[INFO] Fold 4 Accuracy: 0.550000
[INFO] Fold 5 Accuracy: 0.535000
[INFO] Fold 6 Accuracy: 0.525000
[INFO] Fold 7 Accuracy: 0.505000
[INFO] Fold 8 Accuracy: 0.535000
[INFO] Fold 9 Accuracy: 0.550000
[INFO] Accuracy: 0.522500
```

3. Problems and Limitations:

- The distribution of “great” and “poor” is very skewed in training data. Occurrence of “great” is 4 times that of “poor”. This leads to very small values of semantic orientation and results in ‘neg’ classification most of the times.
- Overlapping Regexes are not accounted.
- The training data is very small. Hence for many phrases near operator gives 0 in the results.
- Takes a bit more time to run on my machine. This may be because of calculating semantic orientation of all possible phrases in training phrase and this happens 10 times because of cross validation.

Steps:

1. Regular expressions to generate sentiment phrases <-> Examples

```
([\S]+)_JJ_[A-Z-]+ ([\S]+)_NNS?_[A-Z-]+
   Teen_JJ_I-NP couples_NNS_I-NP
([\S]+)_RB.?_[A-Z-]+ ([\S]+)_JJ_[A-Z-]+ ([\S]+)_((?!NN)..?.?)_[A-Z-]+
   pretty_RB_I-NP decent_JJ_I-NP teen_JJ_I-NP
([\S]+)_JJ_[A-Z-]+ ([\S]+)_JJ_[A-Z-]+ ([\S]+)_((?!NN)..?.?)_[A-Z-]+
   decent_JJ_I-NP teen_JJ_I-NP mind-fuck_JJ_I-NP
([\S]+)_NN.?_[A-Z-]+ ([\S]+)_JJ_[A-Z-]+ ([\S]+)_((?!NN)..?.?)_[A-Z-]+
   SOMETHING_NN_B-NP other_JJ_B-ADJP than_IN_B-PP
([\S]+)_RB.?_[A-Z-]+ ([\S]+)_VB.?_[A-Z-]+
   even_RB_I-ADVP harden_VBD_B-VP
```

2. 'NEAR' operator

```
def findPhrasePovertyAndGreatness(self, phrase, wordList):
        poor = 0
        great = 0
        for i in xrange(len(wordList) - 1):
            new_phrase = wordList[i] + " " + wordList[i + 1]
            if new_phrase == phrase:
                k = i + 11
                j = i + 2
                while j < len(wordList) and j <= k:
                    if wordList[j] == 'poor':
                        poor += 1
                    elif wordList[j] == 'great':
                        great += 1
                    j += 1
                k = i - 10
                j = i - 1
                while j >= 0 and j >= k:
                    if wordList[j] == 'poor':
                        poor += 1
                    elif wordList[j] == 'great':
                        great += 1
                    j -= 1
        return poor, great
```

3. Semantic Orientation of each sentiment phrase

```
                great_count = self.greatDict[phrase]
                poor_count = self.poorDict[phrase]
                great = self.great
                poor = self.poor

                if great_count < 4 and poor_count < 4:
                    continue
                num = (great_count + 0.01) * (poor + 0.01)
                den = (poor_count + 0.01) * (great + 0.01)
                so = math.log(float(num) / den, 2)
```

4. Polarity Score

```
def classify(self, example):
    count = 0
    total_so = 0.0
    for regex in self.reg_exps:
        matches = re.findall(regex, example.taggedData)
        for m in matches:
            phrase = '%s %s' %(m[0], m[1])
            great_count = self.greatDict[phrase]
            poor_count = self.poorDict[phrase]
            great = self.great
            poor = self.poor

            if great_count < 4 and poor_count < 4:
                continue
            num = (great_count + 0.01) * (poor + 0.01)
            den = (poor_count + 0.01) * (great + 0.01)
            so = math.log(float(num) / den, 2)
            total_so += so
            count += 1
    if count == 0:
        avg_so = 0.0
    else:
        avg_so = total_so / count
    if avg_so > 0:
        return 'pos'
    else:
        return 'neg'
```

THANK YOU