29 November 2019

CSCE-638
Programming Assignment #4
Report

Table of Contents

1 Introduction

The program performs Sentimental Analysis by inducing a sentiment phrasal lexicon. The approach to do the same is followed from the paper:
Turney, Peter D. 2002. "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews." Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL).

1.1 Contents of Submission

The deliverables of this assignment has been tabulated as below:

430000753/	Contains the source python file pa4.py and the dataset imdb_tagged directory
RizuJain_430000753_Report.pdf	Assignment Report (this)

2 Steps of Compilation and Execution

Platform Used: Python (Version 3)
Host PC: Windows PC

Steps for the compilation and execution of the python program

1. Unzip the 430000753.zip
2. Go to ~/430000753/python directory using the commands and run the following commands to execute the program pa4.py
    - `$ cd ~/430000753`
    - `$ python.exe .\pa4.py .\imdb_tagged\processed_docs`
3. To execute the program with the bonus optimisation, append a "-m" at the end of the program:
    - `python.exe .\pa4.py .\imdb_tagged\processed_docs -m`

3 Implementation

3.1 Regular Expressions

Five regular expressions were designed to match the sentiment phrases based upon the rules given in the paper [1]:

	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

Figure 1: [Paper] Patterns of tags for extracting two-word

The following table lists the five regular expressions used with their related examples of sentiment phrases:

	Regex	Sentiment Phrase
1	(\w*'*\w*_JJ\w*-\w*?)\s(\w*_NN[S]?\w*-*\w+)\s(\w*)	graphic_JJ_B-NP novel_NN_I-NP; violent_JJ_I-NP street_NN_I-NP crime_NN_I;
2	(\w*'*\w*_RB[RS]?_\w*-\w*?)\s(\w*_JJ\w*-*\w+)\s(\w+_(?!NN[S]?).\w+-\w*)	it's_RB_B-O much_RB_I-NP more_JJR_I-NP; it_PRP_B-NPfunny_JJ_B-ADJP to_TO_B-VP;
3	(\w*'*\w*_JJ_\w*-\w*?)\s(\w*_JJ\w*-*\w+)\s(\w+_(?!NN[S]?).\w+-\w*)	unfortunate_JJ_I-NP named_JJ_I-NP mary_JJ_I-NP; little_JJ_I-NP nervous_JJ_B-ADJP about_IN_B-PP;
4	(\w*'*\w*_NN[S]?_\w*-\w*?)\s(\w*_JJ\w*-*\w+)\s(\w+_(?!NN[S]?).\w+-\w*)	something_NN_B-NP other_JJ_B-ADVP than_IN_B-PP;
5	(\w*'*\w*_RB[RS]?\w*-\w*?)\s(\w*_VB[DNG]?\w*-*\w+)\s(\w*)	also_RB_B-ADVP explains_VBZ_B-VP why_WRB_B; always_RB_B-ADVP played_VBD_B-VP;

3.2 Search with Near Operator

The following code snippet shows the implementation that conducts search including the Near Operator.

```
for idx,word in enumerate(just_words):
     if word in pos_words:
       self.hits_excellent += 1
       min_id = max(0,idx-10)
       max_id = min(idx+10+1,len(just_words))

       for neighbor_id in range(min_id,max_id-1):
          self.near_excellent[(just_words[neighbor_id],just_words[neighbor_id+1])]
          += 1

     elif word in neg_words:
       self.hits_poor += 1
       min_id = max(0,idx-10)
       max_id = min(idx+10+1,len(just_words))

       for neighbor_id in range(min_id,max_id-1):
          self.near_poor[(just_words[neighbor_id],just_words[neighbor_id+1])] += 1
```

- For each word in our training corpus, we check if it is one of our positive or negative word.
- If it is, then we increment the respective count.
- Also, for the 'NEAR' operator, we take all the phrases within 10 words from our keyword, and increment their seen count in a dictionary.
- When we are looking for hits during classification, we will simply take the count of that phrase from the respective dictionaries.

3.3 Semantic Orientation

The following code snippet shows the calculation of semantic orientation:

```
for phrase in phrases:
  phrase_hits_excellent = self.near_excellent[phrase]
  phrase_hits_poor = self.near_poor[phrase]

  if phrase_hits_poor < 1 and phrase_hits_excellent < 1:
    # skip to the next phrase as mentioned in the paper
    continue

  phrase_hits_excellent += 0.01
  phrase_hits_poor += 0.01

  numerator = phrase_hits_excellent * self.hits_poor
  denominator = phrase_hits_poor * self.hits_excellent

  phrase_SO = math.log(numerator/denominator)
```

- For calculating the semantic orientation, we make use of the hit count that we have generated during the training phase.
- Hit count for positive keyword is in near_excellent and hti count for negative keyword is in near_poor.
- If the hit counts for both is 0, I skip the phrase. Also, 0.01 is added to avoid division by zero.

3.4 Polarity Score

For each test review, the polarity score is calculated. The following code snippet shows the same:

```
def classify(self, phrases):

   avg_SO = 0

   for phrase in phrases:
     phrase_hits_excellent = self.near_excellent[phrase]
     phrase_hits_poor = self.near_poor[phrase]

     if phrase_hits_poor < 1 and phrase_hits_excellent < 1:
       # skip to the next phrase as mentioned in the paper
       continue

     phrase_hits_excellent += 0.01
     phrase_hits_poor += 0.01

     numerator = phrase_hits_excellent * self.hits_poor
     denominator = phrase_hits_poor * self.hits_excellent

     phrase_SO = math.log(numerator/denominator)
     avg_SO += phrase_SO

   ans = ('neg','pos')[avg_SO>=0]
   return ans
```

- Polarity of a review is the sum of the semantic orientation of it's relevant phrases.
- We calculated the semantic orientation in previous step and it is stored in phrase_SO.
- We keep on adding phrase_SO to get the avg_SO which is our polarity for this review.
- Based on the polarity we predict the review as positive or negative.
- If the polarity is zero, which will happen if we have no seen phrases, we predict the sentence as positive.

3.5 Bonus Section

As we are using a small training dataset instead of a search engine, we are less likely to find phrases in the context of our keyword. It was observed that the hits are very low and in many cases none.

- A workaround to this could be using more positive and negative keywords so that we have more hits.
- To verify this, I added the following keywords to the positive and negative classes each:
- Positive keywords = “excellent”,”good”,”best”
- Negative keywords = “poor”,”bad”,”worst”
- With this change, I see an improvement of 5% in the accuracy. Please find below the output logs for the same.

4 Results and Analysis

Upon the execution of the program the following logs can be obtained on the console:

```
PS D:\430000753> python.exe .\pa4.py .\imdb_tagged\processed_docs
cv done
[INFO]  Fold 0 Accuracy: 0.535000
[INFO]  Fold 1 Accuracy: 0.490000
[INFO]  Fold 2 Accuracy: 0.550000
[INFO]  Fold 3 Accuracy: 0.545000
[INFO]  Fold 4 Accuracy: 0.495000
[INFO]  Fold 5 Accuracy: 0.540000
[INFO]  Fold 6 Accuracy: 0.520000
[INFO]  Fold 7 Accuracy: 0.535000
[INFO]  Fold 8 Accuracy: 0.515000
[INFO]  Fold 9 Accuracy: 0.490000
[INFO]  Accuracy: 0.521500
```

BONUS LOGS:

```
PS D:\430000753> python.exe .\pa4.py .\imdb_tagged\processed_docs
-m
cv done
[INFO]  Fold 0 Accuracy: 0.575000
[INFO]  Fold 1 Accuracy: 0.520000
[INFO]  Fold 2 Accuracy: 0.595000
[INFO]  Fold 3 Accuracy: 0.615000
[INFO]  Fold 4 Accuracy: 0.565000
[INFO]  Fold 5 Accuracy: 0.560000
[INFO]  Fold 6 Accuracy: 0.545000
[INFO]  Fold 7 Accuracy: 0.550000
[INFO]  Fold 8 Accuracy: 0.610000
[INFO]  Fold 9 Accuracy: 0.555000
[INFO]  Accuracy: 0.569000
```

5 References

1. Turney, Peter D. 2002. “Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews.” Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL).
2. CSCE 638, Fall 2019, Programming Assignment #2