NLP Assignment 4: Sentiment Lexicon

Name: Vamsi Tallam UIN: 432001932

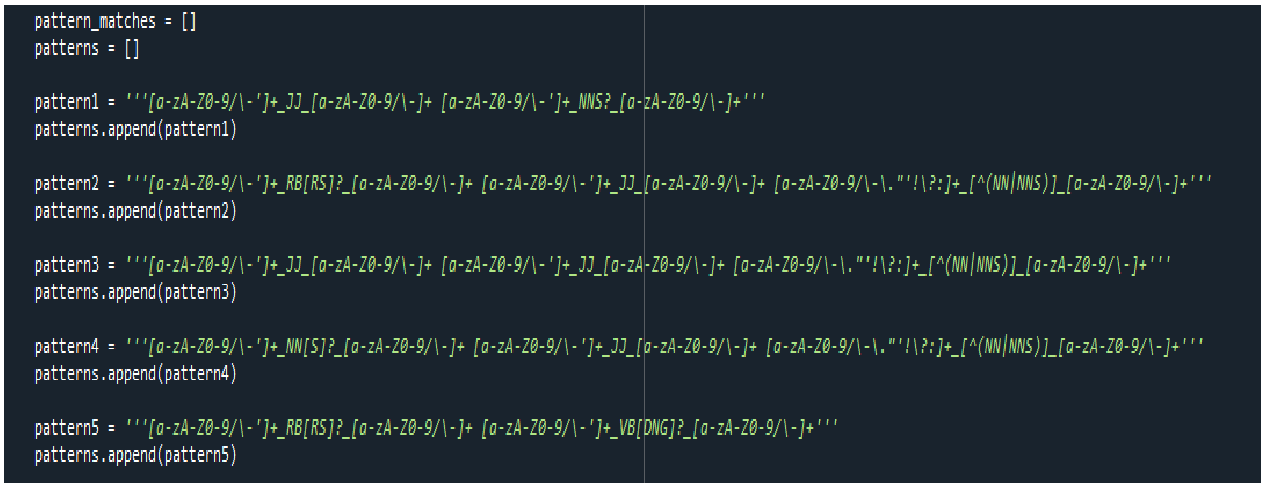
Task 1: design regular expressions to generate sentiment phrases as specified in the paper. Please show all your regular expressions and examples of sentiment phrases in your report.

How to run the code:

1. Download and extract the code from PA4_SentimentLexiconInduction.zip, unzip the folder imdb_tagged.zip it has processed_docs subfolder which has the data
2. This code is run end-to-end on a CPU hence did not work with google Collaboratory
3. The code is edited and run using spyder, to run the code:
   runfile('C:/Users/vamsi/OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-sentimentLexiconInduction/SentimentLexicon.py', args='C:/Users/vamsi/OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-sentimentLexiconInduction/imdb_tagged/processed_docs/', wdir='C:/Users/vamsi/OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-sentimentLexiconInduction')

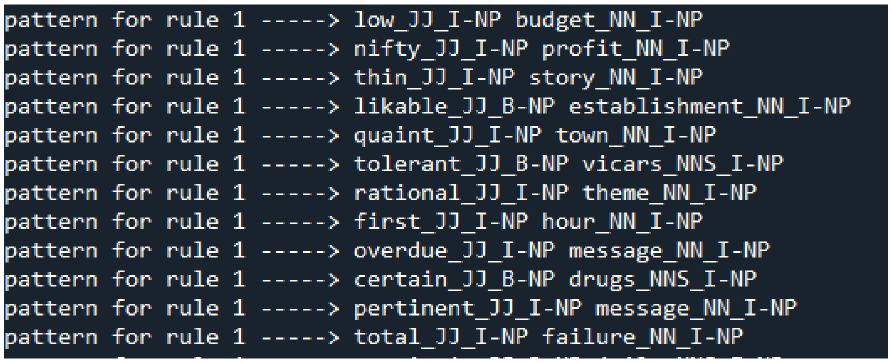
Regular expressions used to generate sentiment phrases:

The following is a snippet from generate_patterns function which has the regular expression patterns from the papers.

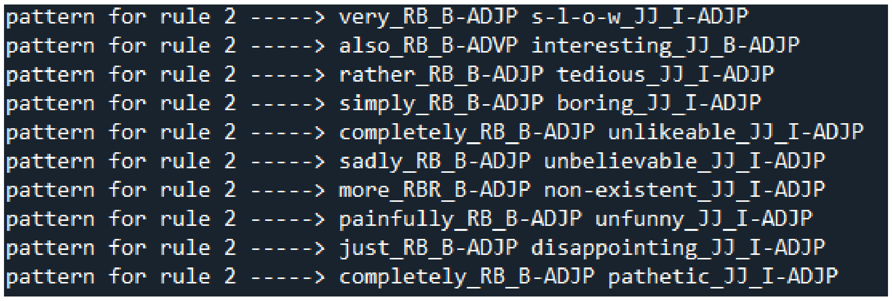


```
pattern_matches = []
patterns = []

pattern1 = '''[a-zA-Z0-9/\-']+_JJ_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-']+_NNS?_[a-zA-Z0-9/\-]+'''
patterns.append(pattern1)

pattern2 = '''[a-zA-Z0-9/\-']+_RB[RS]?_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-']+_JJ_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-\."'!\?:]+_[^(NN|NNS)]_[a-zA-Z0-9/\-]+'''
patterns.append(pattern2)

pattern3 = '''[a-zA-Z0-9/\-']+_JJ_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-']+_JJ_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-\."'!\?:]+_[^(NN|NNS)]_[a-zA-Z0-9/\-]+'''
patterns.append(pattern3)

pattern4 = '''[a-zA-Z0-9/\-']+_NN[S]?_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-']+_JJ_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-\."'!\?:]+_[^(NN|NNS)]_[a-zA-Z0-9/\-]+'''
patterns.append(pattern4)

pattern5 = '''[a-zA-Z0-9/\-']+_RB[RS]?_[a-zA-Z0-9/\-]+ [a-zA-Z0-9/\-']+_VB[DNG]?_[a-zA-Z0-9/\-]+'''
patterns.append(pattern5)
```

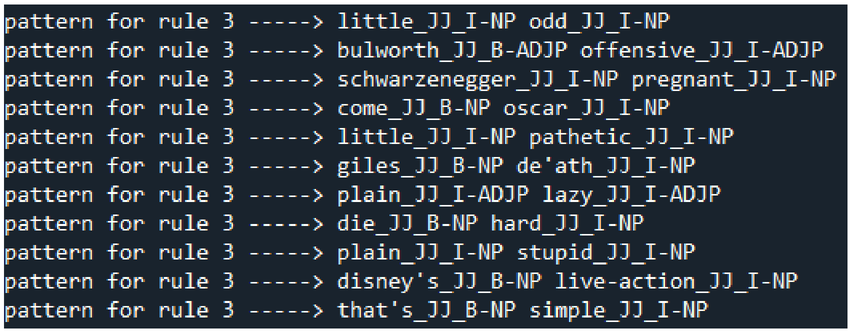
Examples of sentiment phrases are the following:



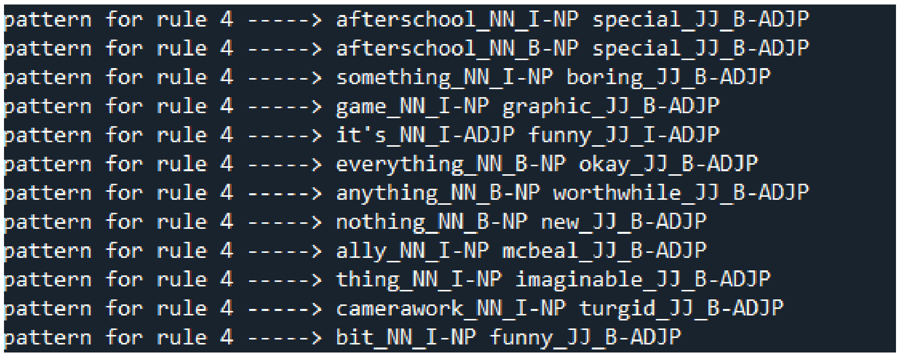
```
pattern for rule 1 -----> low_JJ_I-NP budget_NN_I-NP
pattern for rule 1 -----> nifty_JJ_I-NP profit_NN_I-NP
pattern for rule 1 -----> thin_JJ_I-NP story_NN_I-NP
pattern for rule 1 -----> likable_JJ_B-NP establishment_NN_I-NP
pattern for rule 1 -----> quaint_JJ_I-NP town_NN_I-NP
pattern for rule 1 -----> tolerant_JJ_B-NP vicars_NNS_I-NP
pattern for rule 1 -----> rational_JJ_I-NP theme_NN_I-NP
pattern for rule 1 -----> first_JJ_I-NP hour_NN_I-NP
pattern for rule 1 -----> overdue_JJ_I-NP message_NN_I-NP
pattern for rule 1 -----> certain_JJ_B-NP drugs_NNS_I-NP
pattern for rule 1 -----> pertinent_JJ_I-NP message_NN_I-NP
pattern for rule 1 -----> total_JJ_I-NP failure_NN_I-NP
```



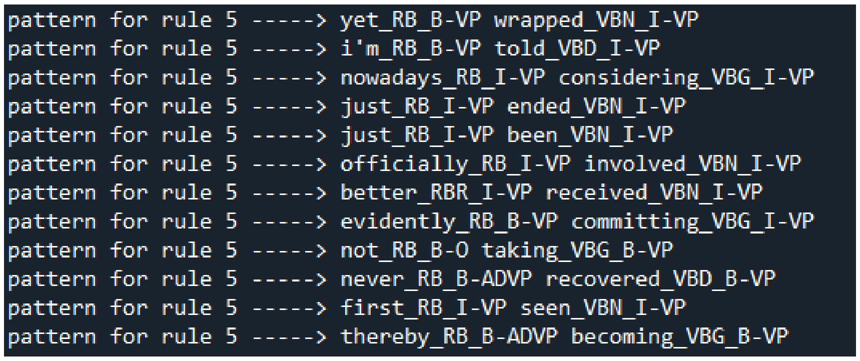
```
pattern for rule 2 -----> very_RB_B-ADJP s-l-o-w_JJ_I-ADJP
pattern for rule 2 -----> also_RB_B-ADVP interesting_JJ_B-ADJP
pattern for rule 2 -----> rather_RB_B-ADJP tedious_JJ_I-ADJP
pattern for rule 2 -----> simply_RB_B-ADJP boring_JJ_I-ADJP
pattern for rule 2 -----> completely_RB_B-ADJP unlikeable_JJ_I-ADJP
pattern for rule 2 -----> sadly_RB_B-ADJP unbelievable_JJ_I-ADJP
pattern for rule 2 -----> more_RBR_B-ADJP non-existent_JJ_I-ADJP
pattern for rule 2 -----> painfully_RB_B-ADJP unfunny_JJ_I-ADJP
pattern for rule 2 -----> just_RB_B-ADJP disappointing_JJ_I-ADJP
pattern for rule 2 -----> completely_RB_B-ADJP pathetic_JJ_I-ADJP
```



```
pattern for rule 3 -----> little_JJ_I-NP odd_JJ_I-NP
pattern for rule 3 -----> bulworth_JJ_B-ADJP offensive_JJ_I-ADJP
pattern for rule 3 -----> schwarzenegger_JJ_I-NP pregnant_JJ_I-NP
pattern for rule 3 -----> come_JJ_B-NP oscar_JJ_I-NP
pattern for rule 3 -----> little_JJ_I-NP pathetic_JJ_I-NP
pattern for rule 3 -----> giles_JJ_B-NP de'ath_JJ_I-NP
pattern for rule 3 -----> plain_JJ_I-ADJP lazy_JJ_I-ADJP
pattern for rule 3 -----> die_JJ_B-NP hard_JJ_I-NP
pattern for rule 3 -----> plain_JJ_I-NP stupid_JJ_I-NP
pattern for rule 3 -----> disney's_JJ_B-NP live-action_JJ_I-NP
pattern for rule 3 -----> that's_JJ_B-NP simple_JJ_I-NP
```

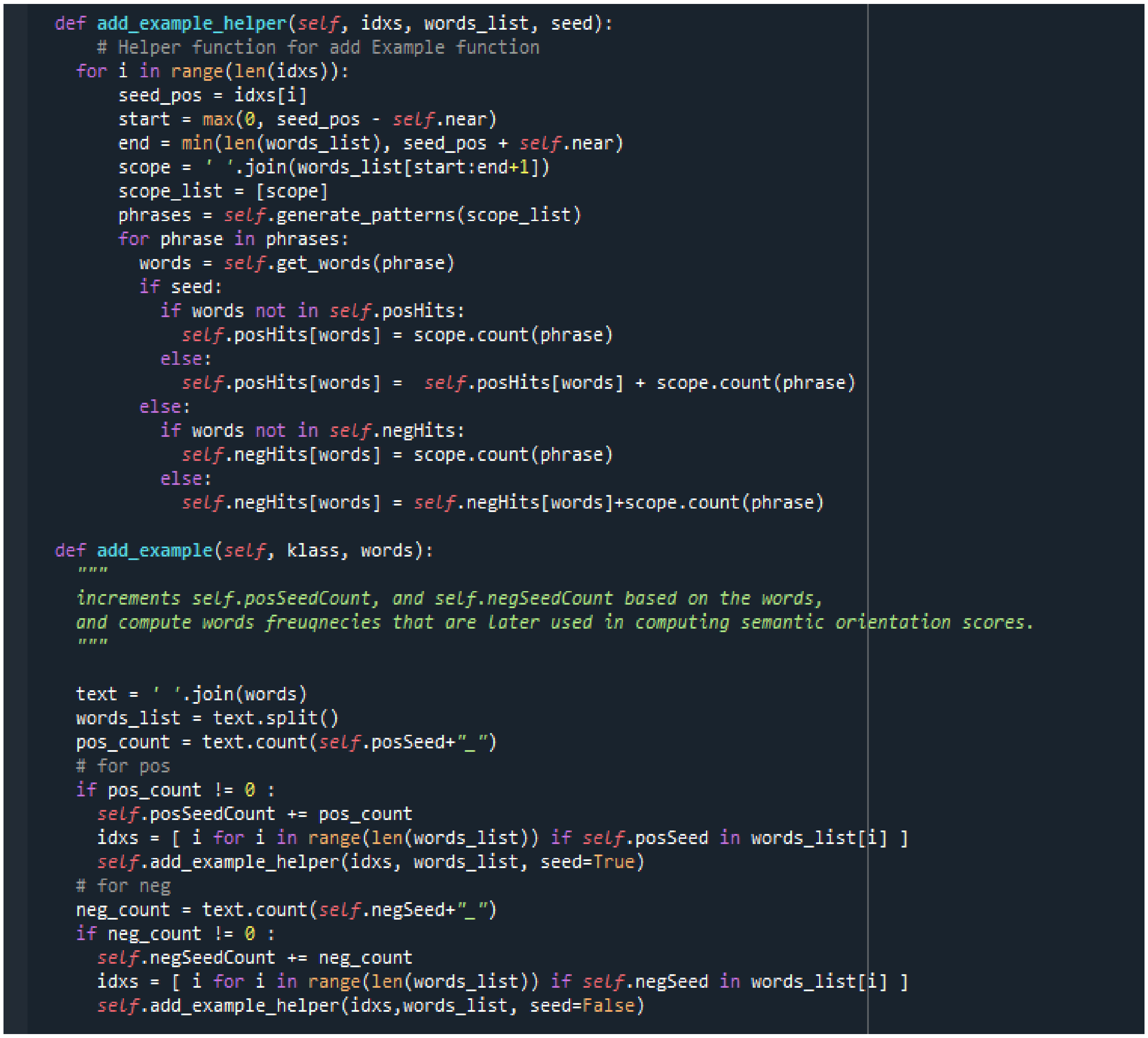


```
pattern for rule 4 -----> afterschool_NN_I-NP special_JJ_B-ADJP
pattern for rule 4 -----> afterschool_NN_B-NP special_JJ_B-ADJP
pattern for rule 4 -----> something_NN_I-NP boring_JJ_B-ADJP
pattern for rule 4 -----> game_NN_I-NP graphic_JJ_B-ADJP
pattern for rule 4 -----> it's_NN_I-ADJP funny_JJ_I-ADJP
pattern for rule 4 -----> everything_NN_B-NP okay_JJ_B-ADJP
pattern for rule 4 -----> anything_NN_B-NP worthwhile_JJ_B-ADJP
pattern for rule 4 -----> nothing_NN_B-NP new_JJ_B-ADJP
pattern for rule 4 -----> ally_NN_I-NP mcbeal_JJ_B-ADJP
pattern for rule 4 -----> thing_NN_I-NP imaginable_JJ_B-ADJP
pattern for rule 4 -----> camerawork_NN_I-NP turgid_JJ_B-ADJP
pattern for rule 4 -----> bit_NN_I-NP funny_JJ_B-ADJP
```



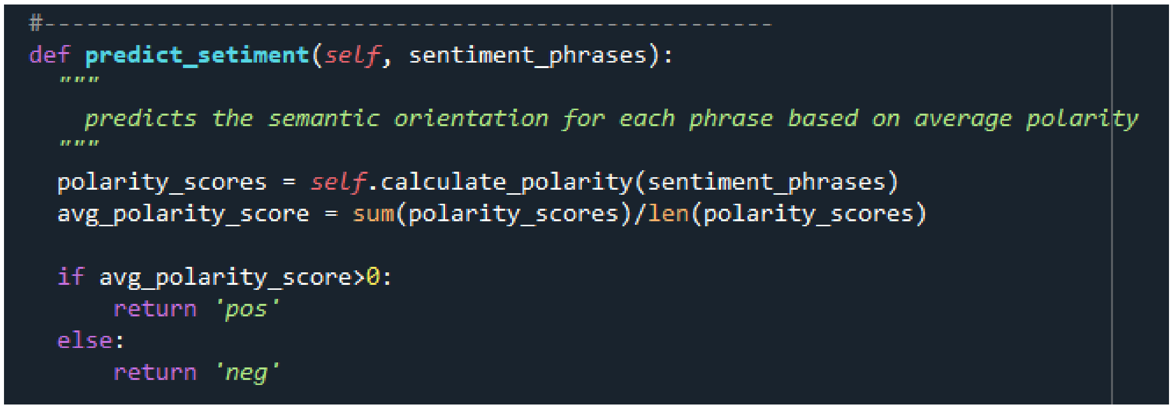
```
pattern for rule 5 -----> yet_RB_B-VP wrapped_VBN_I-VP
pattern for rule 5 -----> i'm_RB_B-VP told_VBD_I-VP
pattern for rule 5 -----> nowadays_RB_I-VP considering_VBG_I-VP
pattern for rule 5 -----> just_RB_I-VP ended_VBN_I-VP
pattern for rule 5 -----> just_RB_I-VP been_VBN_I-VP
pattern for rule 5 -----> officially_RB_I-VP involved_VBN_I-VP
pattern for rule 5 -----> better_RBR_I-VP received_VBN_I-VP
pattern for rule 5 -----> evidently_RB_B-VP committing_VBG_I-VP
pattern for rule 5 -----> not_RB_B-O taking_VBG_B-VP
pattern for rule 5 -----> never_RB_B-ADVP recovered_VBD_B-VP
pattern for rule 5 -----> first_RB_I-VP seen_VBN_I-VP
pattern for rule 5 -----> thereby_RB_B-ADVP becoming_VBG_B-VP
```

Task 2: write code to conduct search including implementing the “NEAR" operator. Please paste the relevant part of code in your report.



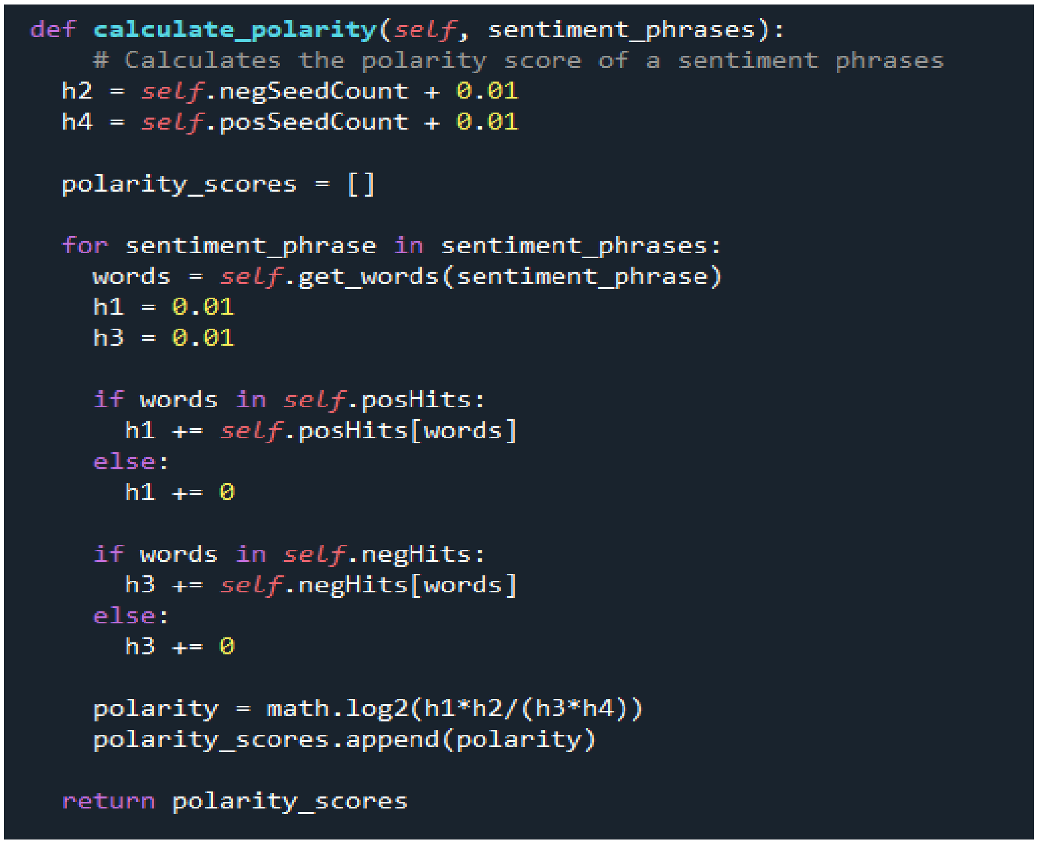
```python
def add_example_helper(self, idxs, words_list, seed):
    # Helper function for add Example function
  for i in range(len(idxs)):
      seed_pos = idxs[i]
      start = max(0, seed_pos - self.near)
      end = min(len(words_list), seed_pos + self.near)
      scope = ' '.join(words_list[start:end+1])
      scope_list = [scope]
      phrases = self.generate_patterns(scope_list)
      for phrase in phrases:
        words = self.get_words(phrase)
        if seed:
          if words not in self.posHits:
            self.posHits[words] = scope.count(phrase)
          else:
            self.posHits[words] =  self.posHits[words] + scope.count(phrase)
        else:
          if words not in self.negHits:
            self.negHits[words] = scope.count(phrase)
          else:
            self.negHits[words] = self.negHits[words]+scope.count(phrase)

def add_example(self, klass, words):
  """
  increments self.posSeedCount, and self.negSeedCount based on the words,
  and compute words freuqnecies that are later used in computing semantic orientation scores.
  """

  text = ' '.join(words)
  words_list = text.split()
  pos_count = text.count(self.posSeed+"_")
  # for pos
  if pos_count != 0 :
    self.posSeedCount += pos_count
    idxs = [ i for i in range(len(words_list)) if self.posSeed in words_list[i] ]
    self.add_example_helper(idxs, words_list, seed=True)
  # for neg
  neg_count = text.count(self.negSeed+"_")
  if neg_count != 0 :
    self.negSeedCount += neg_count
    idxs = [ i for i in range(len(words_list)) if self.negSeed in words_list[i] ]
    self.add_example_helper(idxs,words_list, seed=False)
```

Task 3: calculate the semantic orientation for each sentiment phrase. Please paste the relevant part of code in your report.



```
#----------------------------------------------------
def predict_setiment(self, sentiment_phrases):
  """
    predicts the semantic orientation for each phrase based on average polarity
  """
  polarity_scores = self.calculate_polarity(sentiment_phrases)
  avg_polarity_score = sum(polarity_scores)/len(polarity_scores)

  if avg_polarity_score>0:
      return 'pos'
  else:
      return 'neg'
```

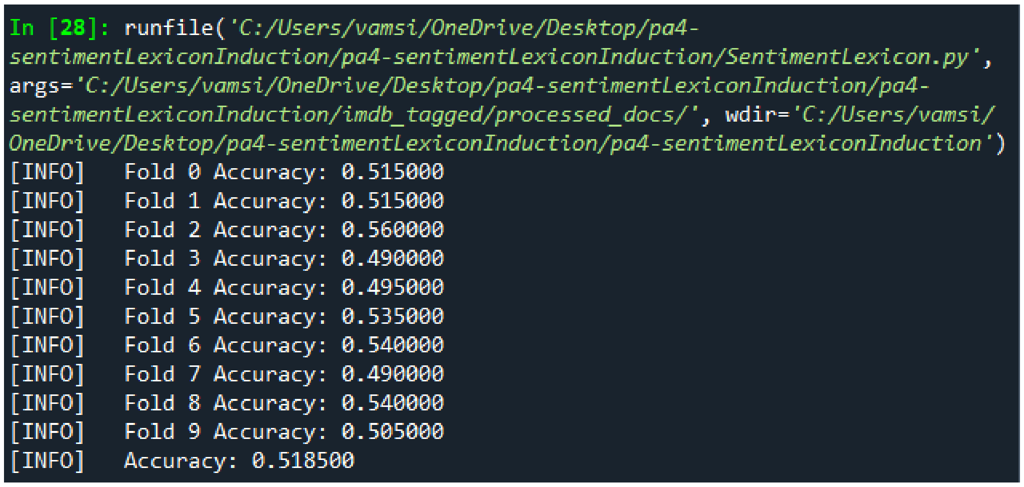
Task 4: calculate the polarity score for each test review. Please paste the relevant part of code in your report.



```
def calculate_polarity(self, sentiment_phrases):
    # Calculates the polarity score of a sentiment phrases
  h2 = self.negSeedCount + 0.01
  h4 = self.posSeedCount + 0.01

  polarity_scores = []

  for sentiment_phrase in sentiment_phrases:
    words = self.get_words(sentiment_phrase)
    h1 = 0.01
    h3 = 0.01

    if words in self.posHits:
      h1 += self.posHits[words]
    else:
      h1 += 0

    if words in self.negHits:
      h3 += self.negHits[words]
    else:
      h3 += 0

    polarity = math.log2(h1*h2/(h3*h4))
    polarity_scores.append(polarity)

  return polarity_scores
```

Results:

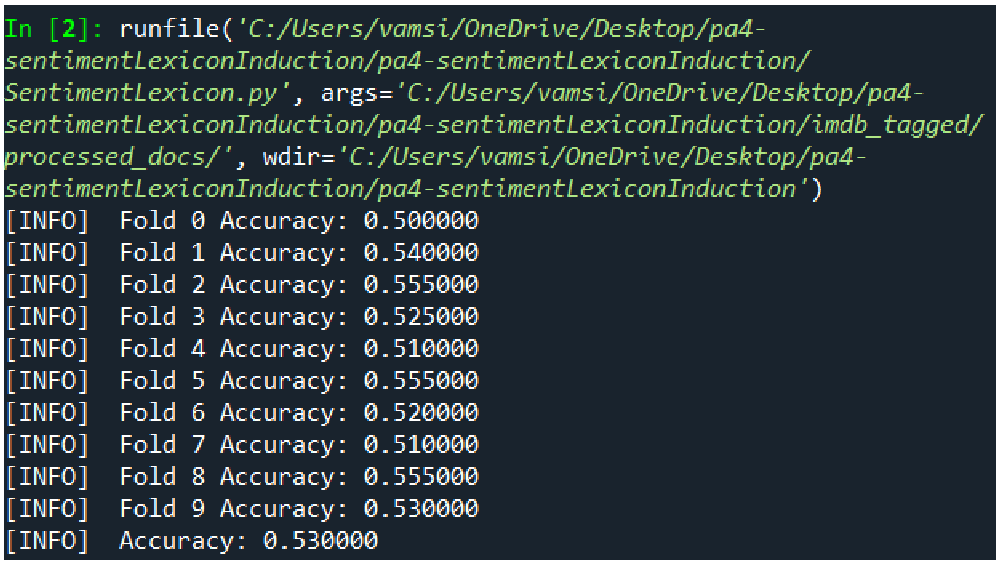
- The code leverage on programming assignment 2 for cross-validation and other key tasks like data loading, splitting data, and others are clearly highlighted in the code.
- The following are the results for when seed words “excellent” and “poor” are used and the near range of 10 is considered.



```
In [28]: runfile('C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction/SentimentLexicon.py',
args='C:/Users/vamsi/OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-
sentimentLexiconInduction/imdb_tagged/processed_docs/', wdir='C:/Users/vamsi/
OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-sentimentLexiconInduction')
[INFO]   Fold 0 Accuracy: 0.515000
[INFO]   Fold 1 Accuracy: 0.515000
[INFO]   Fold 2 Accuracy: 0.560000
[INFO]   Fold 3 Accuracy: 0.490000
[INFO]   Fold 4 Accuracy: 0.495000
[INFO]   Fold 5 Accuracy: 0.535000
[INFO]   Fold 6 Accuracy: 0.540000
[INFO]   Fold 7 Accuracy: 0.490000
[INFO]   Fold 8 Accuracy: 0.540000
[INFO]   Fold 9 Accuracy: 0.505000
[INFO]   Accuracy: 0.518500
```

near range=10, seed words – “excellent” and “poor”

- Increasing the near range to 20 has increased the accuracy to 53%

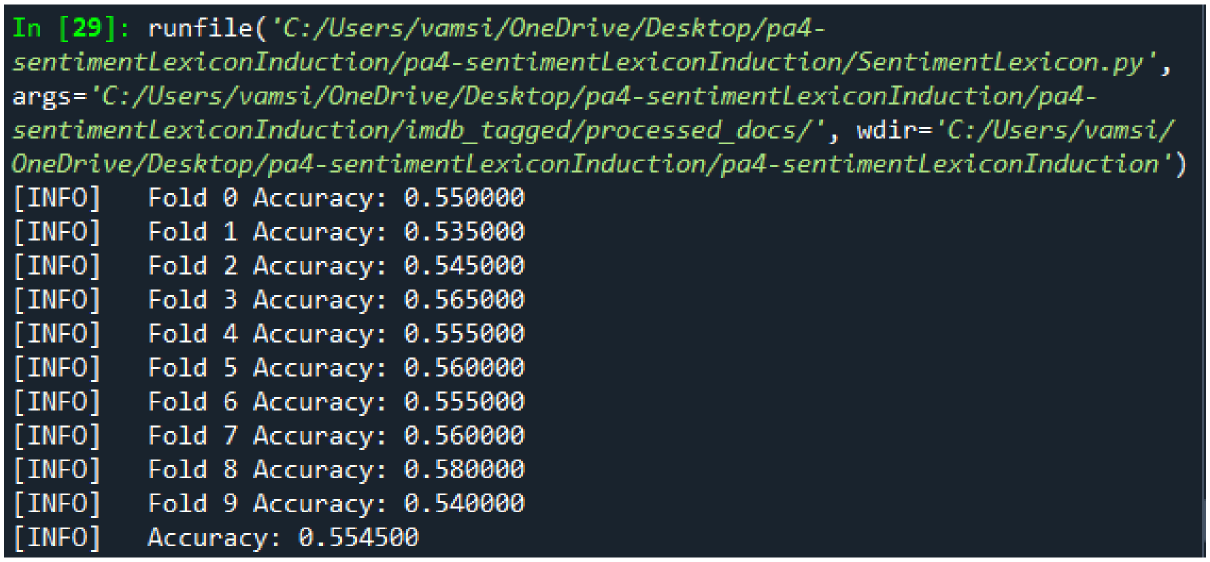


```
In [2]: runfile('C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction/
SentimentLexicon.py', args='C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction/imdb_tagged/
processed_docs/', wdir='C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction')
[INFO]  Fold 0 Accuracy: 0.500000
[INFO]  Fold 1 Accuracy: 0.540000
[INFO]  Fold 2 Accuracy: 0.555000
[INFO]  Fold 3 Accuracy: 0.525000
[INFO]  Fold 4 Accuracy: 0.510000
[INFO]  Fold 5 Accuracy: 0.555000
[INFO]  Fold 6 Accuracy: 0.520000
[INFO]  Fold 7 Accuracy: 0.510000
[INFO]  Fold 8 Accuracy: 0.555000
[INFO]  Fold 9 Accuracy: 0.530000
[INFO]  Accuracy: 0.530000
```

near range=20, seed words – “excellent” and “poor”

Extra Points1: Can you try to replace either or both of the two given seed words “excellent" and “poor", or/and add additional seed words, and see if the updated sentiment lexicon can help to improve the performance of sentiment analysis?

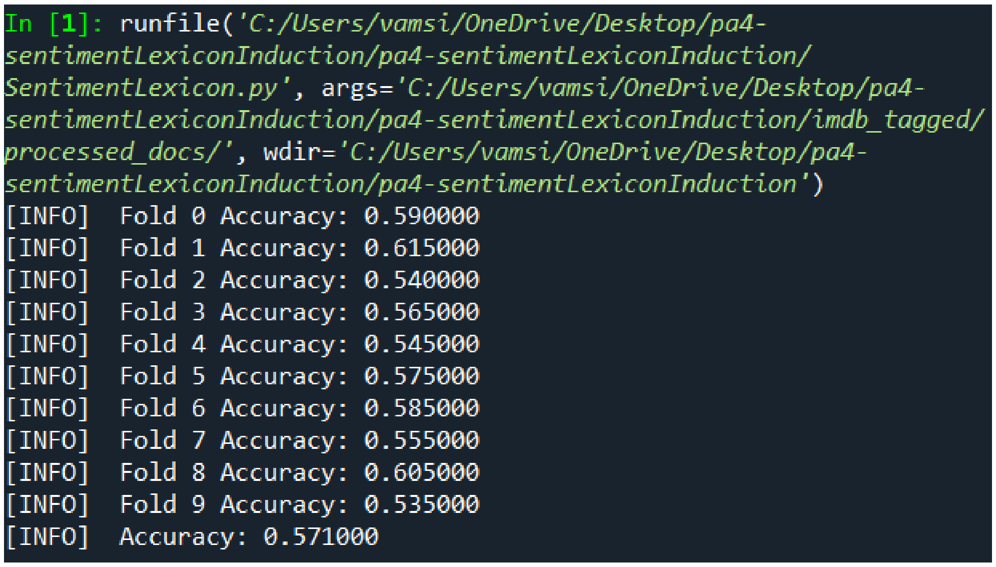
- To tackle the bonus question, I tried various seed words, the following is the result we see when the seed words are “great” and “bad”
- We can see a clear improvement in accuracy compared to 51.85% when the seed words are “excellent” and “poor”



```
In [29]: runfile('C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction/SentimentLexicon.py',
args='C:/Users/vamsi/OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-
sentimentLexiconInduction/imdb_tagged/processed_docs/', wdir='C:/Users/vamsi/
OneDrive/Desktop/pa4-sentimentLexiconInduction/pa4-sentimentLexiconInduction')
[INFO]   Fold 0 Accuracy: 0.550000
[INFO]   Fold 1 Accuracy: 0.535000
[INFO]   Fold 2 Accuracy: 0.545000
[INFO]   Fold 3 Accuracy: 0.565000
[INFO]   Fold 4 Accuracy: 0.555000
[INFO]   Fold 5 Accuracy: 0.560000
[INFO]   Fold 6 Accuracy: 0.555000
[INFO]   Fold 7 Accuracy: 0.560000
[INFO]   Fold 8 Accuracy: 0.580000
[INFO]   Fold 9 Accuracy: 0.540000
[INFO]   Accuracy: 0.554500
```

near range=10, seed words – “great” and “bad”

- I also tried increasing the near range to 20, the following are the results:



```
In [1]: runfile('C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction/
SentimentLexicon.py', args='C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction/imdb_tagged/
processed_docs/', wdir='C:/Users/vamsi/OneDrive/Desktop/pa4-
sentimentLexiconInduction/pa4-sentimentLexiconInduction')
[INFO]  Fold 0 Accuracy: 0.590000
[INFO]  Fold 1 Accuracy: 0.615000
[INFO]  Fold 2 Accuracy: 0.540000
[INFO]  Fold 3 Accuracy: 0.565000
[INFO]  Fold 4 Accuracy: 0.545000
[INFO]  Fold 5 Accuracy: 0.575000
[INFO]  Fold 6 Accuracy: 0.585000
[INFO]  Fold 7 Accuracy: 0.555000
[INFO]  Fold 8 Accuracy: 0.605000
[INFO]  Fold 9 Accuracy: 0.535000
[INFO]  Accuracy: 0.571000
```

near range=10, seed words – “excellent” and “poor”

- Increasing the near range to 20 has increased the accuracy to 57%

Analysis:

- Based on the Turney’s paper, I designed 5 regular expressions for the five patterns mentioned. Then the semantic orientation of each phrase is predicted based on the average polarity score.
- For seed words “excellent” and “poor”, and the near range = 10, we achieved an accuracy of 51.85%. Increasing the near range to 20 has increased the accuracy to 53%.
- As part of extra credit1, the tried various seed words and observed an increase in accuracy. When using the seed words “great” and “bad”, and the near range = 10, we achieved an accuracy of 55.45%. when the near range is increased to 20 we see an accuracy of 57.1%.

Known issues and Limitations:

- As observed from the above results, the performance of the model is heavily dependent on the choice of seed words. Hence, identifying the best seed words could be challenging on large data.
- The patterns described in the paper are simple patterns and we might need more complex patterns in real life to make the model robust.
- Increasing near range has increased accuracy so identifying the range can pose a challenge.