cd into the python folder and run using following command:

- cd turney
- cd python
- python turney.py ..

Results and Analysis:

[INFO] Fold 0 Accuracy: 0.550000
[INFO] Fold 1 Accuracy: 0.560000
[INFO] Fold 2 Accuracy: 0.550000
[INFO] Fold 3 Accuracy: 0.505000
[INFO] Fold 4 Accuracy: 0.560000
[INFO] Fold 5 Accuracy: 0.550000
[INFO] Fold 6 Accuracy: 0.565000
[INFO] Fold 7 Accuracy: 0.540000
[INFO] Fold 8 Accuracy: 0.515000
[INFO] Fold 9 Accuracy: 0.545000
[INFO] Accuracy: 0.544000

Running the POS tagger:

Code to run the tagger in the report is posTag.py in the python folder. It runs the following code and creates a tagged version of the files in neg_tagged and pos_tagged folders.

```
import os
import subprocess
dirs_neg = os.listdir("/home/ubuntu/workspace/turney/data/imdb1/neg/")
dirs_pos = os.listdir("/home/ubuntu/workspace/turney/data/imdb1/pos/")

for filename in dirs_neg:
        os.system("./tagchunk.i686 -predict . w-5
/home/ubuntu/workspace/turney/data/imdb1/neg/"+filename+"
/home/ubuntu/workspace/pos-tagger/resources > /home/ubuntu/workspace/turney/neg_tagged/"
+ filename)


for filename in dirs_pos:
        os.system("./tagchunk.i686 -predict . w-5
/home/ubuntu/workspace/turney/data/imdb1/pos/"+filename+"
```

/home/ubuntu/workspace/pos-tagger/resources > /home/ubuntu/workspace/turney/pos_tagged/" + filename)

Regular Expressions:

reg_exp = "((\w+)_JJ_\S+ (\w+)_(NN|NNS)_\S+)|((\w+)_(RB|RBR|RBS)_\S+ (\w+)_(JJ)_\S+ (\w+)_(?!NN|NNS).*_\S+)|((\w+)_(JJ)_\S+ (\w+)_(JJ)_\S+ (\w+)_(?!NN|NNS).*_\S+)|((\w+)_(NN|NNS)_\S+ (\w+)_(JJ)_\S+ (\w+)_(?!NN|NNS).*_\S+)|((\w+)_(RB|RBR|RBS)_\S+ (\w+)_(VB|VBD|VBG|VBS)_\S+)"

Examples of sentiment phrases:

('severe_JJ_I-NP', 'tragedy_NN_I-NP'): 1, ('ralph_JJ_B-NP', 'fiennes_NNS_I-NP'): 1, ('eventually_RB_I-VP', 'etch_VB_I-VP'): 1, ('subtley_JJ_B-NP', 'great_JJ_I-NP'): 1, ('film_NN_I-NP', 'worthy_JJ_B-ADJP'): 2, ('very_RB_B-ADJP', 'good_JJ_I-ADJP'): 1

Search implementing the “NEAR” operator:

I created phrases which matched the regular expression created above and looked for the word “great” and “poor” within a distance of 10 words from these phrases to the left and right. If “great” was found within that distance, I updated the count of these phrases near “great” by one. Similarly it was done for “poor”.

Code:

```
reg_exp = "((\w+)_JJ_\S+ (\w+)_(NN|NNS)_\S+)|((\w+)_(RB|RBR|RBS)_\S+ (\w+)_(NNR)_\S+ (\w+)_(?!NN|NNS).*_\S+)|((\w+)_(JJ)_\S+ (\w+)_(JJ)_\S+ (\w+)_(?!NN|NNS).*_\S+)|((\w+)_(NN|NNS)_\S+ (\w+)_(JJ)_\S+ (\w+)_(?!NN|NNS).*_\S+)|((\w+)_(RB|RBR|RBS)_\S+ (\w+)_(VB|VBD|VBG|VBS)_\S+)"
for i in range(len(words) - 2):
  if "great" in words[i]:
    self.great_count += 1
  if "poor" in words[i]:
    self.poor_count += 1
```

```
  string = " ".join([words[i], words[i+1], words[i+2]])
  m = re.match(reg_exp, string)
  length = len(words)
  if m:
    self.phrase_count += 1
    if i < 10:
      list_words = words[0:i+10]
    elif len(words) - 10 < i < len(words):
      list_words = words[i-10:length]
    else:
      list_words = words[i-10:i+10]
    for word in list_words:
      if "great" in word:
        self.indexes["great"][(words[i], words[i+1])] += 1
      if "poor" in word:
        self.indexes["poor"][(words[i], words[i+1])] += 1
```

Semantic Orientation Calculation:

```
semantic_orientation = 0.0
count = 0
#regular expression for identifying phrases
reg_exp = "((\w+)_JJ_\S+ (\w+)_(NN|NNS)_\S+)|((\w+)_(RB|RBR|RBS)_\S+ (\w+)_(NNR)_\S+
(\w+)_(?!NN|NNS).*_\S+)|((\w+)_(JJ)_\S+ (\w+)_(JJ)_\S+
(\w+)_(?!NN|NNS).*_\S+)|((\w+)_(NN|NNS)_\S+ (\w+)_(JJ)_\S+
(\w+)_(?!NN|NNS).*_\S+)|((\w+)_(RB|RBR|RBS)_\S+ (\w+)_(VB|VBD|VBG|VBS)_\S+)"

    # calculating semantic orientation. Adding 0.01 for each hit near great and near poor.
    for i in range(len(words)-2):
      string = " ".join([words[i], words[i+1], words[i+2]])
      m = re.match(reg_exp, string)
      if m:
        if self.indexes["great"][(words[i], words[i+1])] == 0 and self.indexes["poor"][(words[i],
words[i+1])] == 0:
          continue
        semantic_orientation += math.log(float((self.indexes["great"][(words[i], words[i+1])] + 0.01)
* self.poor_count)/float((self.indexes["poor"][(words[i], words[i+1])] + 0.01) * self.great_count))
        count += 1
    if count == 0:
      avg_semantic_orientation = 0.0
```

```
    else:
      avg_semantic_orientation = semantic_orientation/count
```

Polarity Score:

```
if avg_semantic_orientation > 0.0:
  return 'pos'
else:
  return 'neg'
```