CSCE-638, Programming Assignment #4
Rohan Chaudhury
UIN: 432001358

How to run the code files:

I have used **Python 3.9.13** to code the solution. The code files (including the extra part) are attached in the submission along with this pdf file. The code file names are:

A. **PA4_code.py** - Implementation of the code for this assignment and extra credit 1 question. We can run the code by executing the following command in the directory containing the code:
    a. **python PA4_code.py <data_dir>**

where,
data_dir is the directory containing the processed docs

Example usage:
python PA4_code.py processed_docs/

B. **Embedding_Bias.py.py** - Implementation of the extra credit 2 question. It is in the **PA4_Extra** folder. We can run the code by executing the following command in the directory containing the code and the **w2v_gnews_small.txt** file:

    b. **python Embedding_Bias.py**

OUTPUT AND ANSWERS:

Q1: (2 Point!) 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.

Following are the regular expressions that I have used:

```
    self.pos_pattern=["^(JJ)__((NN)|(NNS))__(.)+",
                      "^((RB)|(RBR)|(RBS))__(JJ)__(?!(NN)|(NNS))",
                      "^(JJ)__(JJ)__(?!(NN)|(NNS))",
                      "^((NN)|(NNS))__(JJ)__(?!(NN)|(NNS))",
                      "^((RB)|(RBR)|(RBS))__((VB)|(VBD)|(VBN)|(VBG))__(.)+"]
```

I have used a separator "__" to append the words and then used regex to match them.

Examples of sentiment phrases:

"powerful government" - of type JJ NN NNS
"later approached" - of type RB VBD NNS

“real life” - of type JJ NN .
“many years” - of type JJ NNS DT
“commercial hit” - of type JJ NN IN
"**live-action-disney flick**" of type - JJ NN IN
"**such plot**" of type - JJ NN NNS
"**otherwise standard**" of type - JJ JJ JJ
"**villainous love**" of type - JJ NN NN
"**gold-digging sharon**" of type - JJ NN NN
"**elaine hendrix** " of type - JJ NN ,
"**deliciously tormented**" of type - RB VBN IN
"**olsen twins**" of type - JJ NNS)
"**smart ones**" of type - JJ NNS ,
"silly ones" of type - JJ NNS ,
"other way" of type - JJ NN IN
"certain scenes" of type - JJ NNS VBP
"past dilemnas" of type - JJ NNS CC
"then fought" of type - RB VBD IN
"old flames" of type - JJ NNS VBG
"more cinematic" of type - RBR JJ CC
"mere reminscing" of type - JJ NNS ,
"howard hawks/cary" of type - JJ JJ JJ
"grant films" of type - JJ NNS IN
"as much" of type - RB JJ IN
"mere cutedom" of type - JJ NNS ,
"not gritty" of type - RB JJ ,
"more stylized" of type - RBR JJ CC
"as suitable" of type - RB JJ IN
"young audiences" of type - JJ NNS .
"never be" of type - RB VB RB
"as great" of type - RB JJ CC
"original film" of type - JJ NN ,
"same effects" of type - JJ NNS IN
"petty annoyances" of type - JJ NNS .
"star neil" of type - NN JJ JJ
"patrick harris" of type - JJ NN IN
"military intelligence" of type - JJ NN VBZ
"guilty pleasures" of type - JJ NNS IN

Q2: (3 Points!) write code to conduct search including implementing the “NEAR" operator. Please paste the relevant part of code in your report.

Following is the code for the NEAR Operator:

```
def NEAR_operator(self, index, key, words_split, pos_words, neg_words):
  for j in range(index-self.distance, index+self.distance+1):
    for pos in pos_words:
```

```
        if j >= 0 and j < len(words_split) and words_split[j][0].lower() ==
pos:
            self.phrases_excellent[key]+=1
      for neg in neg_words:
        if j >= 0 and j < len(words_split) and words_split[j][0].lower() ==
neg:
            self.phrases_poor[key]+=1
```

self.distance is set to 10.

It is used in this function to find semantic phrases:

```
  def find_semantic_phrases(self,words,pos_words,neg_words):

    words_split=[]
    for i in range(len(words)):
        word_split=words[i].split('_')
        words_split.append(word_split)
        for pos in pos_words:
            if word_split[0].lower()==pos.lower():
              self.excellent_count+=1
        for neg in neg_words:
            if word_split[0].lower()==neg.lower():
              self.poor_count+=1


    for i in range(len(words_split)-2):
        pos_tag = words_split[i][1]+ "__"+words_split[i+1][1] +"__"+
words_split[i+2][1]
        for pattern in self.pos_pattern:
            if re.match(pattern,pos_tag):
                key=words_split[i][0]+"__"+words_split[i+1][0]
                # print (key.replace("__", " ")," of type ",
pos_tag.replace("__", " "))
                if key not in self.phrases_excellent:
                  self.phrases_excellent[key]=0.01
                  self.phrases_poor[key]=0.01
                self.NEAR_operator(i, key, words_split, pos_words, neg_words)
                break
```

which is finally called in the add_example function:

```
  def addExample(self, klass, words):

    pos_words=["excellent"]
    neg_words=["poor"]
```

```
    self.find_semantic_phrases(words, pos_words, neg_words)
```

Q3: (3 Points!) calculate the semantic orientation for each sentiment phrase. Please paste the relevant part of code in your report.

Following is the code to find the semantic orientation of each sentiment phrase:

```
  def calculate_semantic_orientation(self, phrase):
    so=0
    if phrase in self.phrases_excellent:
      so+=np.log2((self.phrases_excellent[phrase]*self.poor_count))
      so-=np.log2((self.phrases_poor[phrase]*self.excellent_count))
    return so
```

Q4: (2 Points!) calculate the polarity score for each test review. Please paste the relevant part of code in your report.

Following is the code to calculate the polarity score of each test review and to classify them as positive or negative:

```
  def classify(self, words):
    # """ TODO
    #   'words' is a list of words to classify. Return 'pos' or 'neg'
classification.
    # """
    polarity_score=0
    words_split=[]
    for i in range(len(words)):
        words_split.append(words[i].split('_'))
    for i in range(len(words_split)-2):
        pos_tag = words_split[i][1]+ "__"+words_split[i+1][1] +"__"+
words_split[i+2][1]
        for pattern in self.pos_pattern:
            if re.match(pattern,pos_tag):
                phrase=words_split[i][0]+"__"+words_split[i+1][0]
                polarity_score+=self.calculate_semantic_orientation(phrase)
                break
    # print(polarity_score)
    if polarity_score>0:
      return 'pos'
    else:
      return 'neg'
```

Command to run the code: **python PA4_code.py processed_docs/**

Output:
[INFO] Fold 0 Accuracy: 0.530000
[INFO] Fold 1 Accuracy: 0.550000
[INFO] Fold 2 Accuracy: 0.570000
[INFO] Fold 3 Accuracy: 0.495000
[INFO] Fold 4 Accuracy: 0.530000
[INFO] Fold 5 Accuracy: 0.565000
[INFO] Fold 6 Accuracy: 0.515000
[INFO] Fold 7 Accuracy: 0.530000
[INFO] Fold 8 Accuracy: 0.510000
[INFO] Fold 9 Accuracy: 0.555000
[INFO] Accuracy: **0.535000**

Extra Credit I: (2 Points!)

I have used 2 additional seed words for both positive sentiments and for negative sentiments. Following are the seed words that I used:

```
pos_words=["excellent","best","positive"]
neg_words=["poor","bad","negative"]
```

This is added in the add_example function of the code.

With this, the accuracy rose to **59.85%** which is **6.35%** greater than the previously obtained accuracy of **53.5%**

Command to run the code: **python PA4_code.py processed_docs/**

Output:
[INFO] Fold 0 Accuracy: 0.560000
[INFO] Fold 1 Accuracy: 0.535000
[INFO] Fold 2 Accuracy: 0.610000
[INFO] Fold 3 Accuracy: 0.630000
[INFO] Fold 4 Accuracy: 0.560000
[INFO] Fold 5 Accuracy: 0.605000
[INFO] Fold 6 Accuracy: 0.645000
[INFO] Fold 7 Accuracy: 0.635000
[INFO] Fold 8 Accuracy: 0.575000
[INFO] Fold 9 Accuracy: 0.630000
[INFO] Accuracy: 0.598500

Extra Credit II: (3 Points!) (quanitfying gender biases in w2vNEWS embeddings)

(1 Point!) Task 1:

Implemented the code to find the g vector and the direct bias for the different occupations. The obtained g vector is a **numpy.ndarray of shape (1, 300)**

Code to find the gender direction:

```
def genderDicrection(wordpairs,words, vecs):
    # implement your code here
    word_embeddings={k:v for k,v in zip(words,vecs)}
    g_vecs=[]
    for i in range(len(wordpairs)):
        mid_vec=(word_embeddings[wordpairs[i][0]]+word_embeddings[wordpairs[i]
[1]])/2
        # vec_diff=word_embeddings[wordpairs[i][0]]-
word_embeddings[wordpairs[i][1]]
        vec_diff1=word_embeddings[wordpairs[i][0]]-mid_vec
        vec_diff2=word_embeddings[wordpairs[i][1]]-mid_vec
        # normalized_vec_diff=vec_diff/np.linalg.norm(vec_diff) if
np.linalg.norm(vec_diff)!=0 else vec_diff
        normalized_vec_diff1=vec_diff1/np.linalg.norm(vec_diff1) if
np.linalg.norm(vec_diff1)!=0 else vec_diff1
        normalized_vec_diff2=vec_diff2/np.linalg.norm(vec_diff2) if
np.linalg.norm(vec_diff2)!=0 else vec_diff2
        # g_vecs.append(normalized_vec_diff)
        g_vecs.append(normalized_vec_diff1)
        g_vecs.append(normalized_vec_diff2)

    pca = PCA(n_components=1)
    pca.fit_transform(g_vecs)
    return pca.components_
```

Command to run the code: **python Embedding_Bias.py**

(2 Point!) Task 2:
Calculated the direct bias for various occupations. Following is the output of the code:

5 occupations with lowest direct bias:
[('learner', array([-0.00094443])), ('naturalist', array([-0.00115699])), ('warden', array([-0.00117513])), ('poet', array([0.00123705])), ('sheriff_deputy', array([0.00131487]))]

5 occupations with highest direct bias:
[('businesswoman', array([0.3975372])), ('actress', array([0.38665033])), ('housewife', array([0.37260025])), ('saleswoman', array([0.34853349])), ('beautician', array([0.34244732]))]

5 occupations with highest direct bias towards man:

[('maestro', array([-0.25159415])), ('businessman', array([-0.23306245])), ('protege', array([-0.2190738])), ('sportsman', array([-0.21117968])), ('statesman', array([-0.20967881]))]

5 occupations with highest direct bias towards woman:
[('businesswoman', array([0.3975372])), ('actress', array([0.38665033])), ('housewife', array([0.37260025])), ('saleswoman', array([0.34853349])), ('beautician', array([0.34244732]))]

So, the two occupation words that have the highest direct bias values are: **'businesswoman', 'actress'**
And, the two occupation words that have the lowest direct bias values are: **'learner', 'naturalist'**

Code to find the direct bias is as follows:

```
def directBias(g, occupations, words, vecs):
    # implement your code here
    dict_occupation={}
    for occupation in occupations:
        occupation_vec=vecs[words.index(occupation)]
        dict_occupation[occupation]=np.dot(occupation_vec,g.T)/(np.linalg.norm
(occupation_vec)*np.linalg.norm(g))

    return dict_occupation
```

and to print the different bias is as follows:

```
g=genderDicrection(wordpairs,words, vecs)
# print(g)
dval=directBias(g, occupations, words, vecs)
dval_sorted=sorted(dval.items(), key=lambda x: abs(x[1]), reverse=False)
print("5 occupations with lowest direct bias:", dval_sorted[:5])
print("\n")
dval_sorted_reverse=sorted(dval.items(), key=lambda x: abs(x[1]),
reverse=True)
print("5 occupations with highest direct bias:", dval_sorted_reverse[:5])
print("\n")
dval_sorted_reverse_man=sorted(dval.items(), key=lambda x: x[1],
reverse=False)
print("5 occupations with higest direct bias towards man:",
dval_sorted_reverse_man[:5])
print("\n")
dval_sorted_reverse_woman=sorted(dval.items(), key=lambda x: x[1],
reverse=True)
```

```
print("5 occupations with highest direct bias towards woman:", dval_sorted_reverse_woman[:5])
print("\n")
```

Limitations:

There are no limitations of the code as such except the fact that the accuracy obtained from the classifier is less than 60%. More training data can fix the issue as descried in the assignment question.