NLP Homework 2

Sentiment Analysis: Exploratory Analysis

Student: Sharat Sripada (vssripad@syr.edu)

1. Introduction

In this homework we will perform exploratory and sentiment analysis on the same corpora of Shakespearean books used in HW-1.

Below is an excerpt showing indexing books from the Gutenberg corpora:

```
# HW-2
# Exploratory exercise for sentiment analysis
# Finding adverb and adjective phrases, and computing basic statistics

# importing required nltk libraries
import nltk
from nltk import sent_tokenize

# In HW-1, I used books written by Shakespeare - Caesar and Hamlet. We will continue exploratory
# sentiment analysis on the same books
nltk.corpus.gutenberg.fileids()

# Get Shakespeare books in the Gutenberh corpus
shakespeare_books = [book for book in nltk.corpus.gutenberg.fileids( ) \
                     if 'shakespeare' in book]

# Book-1: Caesar (Genre: Tragedy)
caesar = nltk.corpus.gutenberg.raw(shakespeare_books[0])

# Book-2: Hamlet (Genre: Tragedy/Comedy)
hamlet = nltk.corpus.gutenberg.raw(shakespeare_books[1])

print(caesar[:50])
print(hamlet[:50])
```

```
[The Tragedie of Julius Caesar by William Shakespe
[The Tragedie of Hamlet by William Shakespeare 159
```

2. Processing the corpus

The HW will adopt the existing code from *HMW 2-Code Ideas.ipynb* to the above corpora and provide a detailed code walk through and explanation of all steps used to answer the following questions:

- List top-50 adjective/adverb phrases (by frequency)
- List top-50 adjective/adverb words
- List top-50 nouns or verbs
- Average length of sentence

First, we will split the two corpora into sentences using *nltk.sent_tokenize()* before iterating and word tokenizing sentences using *nltk.word_tokenize()*

```
# Apply the word tokenizer to each sentence
token_caesar = [nltk.word_tokenize(sent) for sent in caesar_split]
print(token_caesar[:2])

# the output is a list of strings that contains the sentences
print('Type %s' %(type(token_caesar)))
print('Caesar tokens: %d' %(len(token_caesar)))

# Repeat the same for hamlet text
token_hamlet = [nltk.word_tokenize(sent) for sent in hamlet_split]
print(token_hamlet[:2])
print('Hamlet tokens: %d' %(len(token_hamlet)))
```

Next, we will use the Stanford POS tagger to tag tokens:

```
## POS Tagging, to retrieve adjective (JJs) and adverb (RBs) tags

# use the Stanford POS tagger to POS tag tokens of each sentence
# this is the default tagger in nltk
caesar_tagged = [nltk.pos_tag(tokens) for tokens in token_caesar]
print(caesar_tagged[:2])

hamlet_tagged = [nltk.pos_tag(tokens) for tokens in token_hamlet]
print(hamlet_tagged[:2])
```

Output:

```
[[('[', 'IN'), ('The', 'DT'), ('Tragedie', 'NNP'), ('of', 'IN'), ('Julius', 'NNP'), ('
Caesar', 'NNP'), ('by', 'IN'), ('William', 'NNP'), ('Shakespeare', 'NNP'), ('1599', 'C
D'), (']', 'NNP'), ('Actus', 'NNP'), ('Primus', 'NNP'), ('.', '.')], [('Scoena', 'NNP'
), ('Prima', 'NNP'), ('.', '.')]]
[[('[', 'IN'), ('The', 'DT'), ('Tragedie', 'NNP'), ('of', 'IN'), ('Hamlet', 'NNP'), ('
by', 'IN'), ('William', 'NNP'), ('Shakespeare', 'NNP'), ('1599', 'CD'), (']', 'NNP'),
('Actus', 'NNP'), ('Primus', 'NNP'), ('.', '.')], [('Scoena', 'NNP'), ('Prima', 'NNP')
, ('.', '.')]]
```

2.1 List the top 50 adjective/adverb phrases (by frequency)

For this purpose, we will use the technique of chunking. NLTK offers two utilities, *RegexpParser()*, to define or update custom grammar rules and *parse()* to chunk and parse through the sentence.

Since the above was going to be used across corpora and different POS tags I decided to write it as a python function as seen below:

```
def chunking(grammar, taggedtext, metadata):
    # Extract the metadata
    label = metadata['label']
    desc = metadata['desc']
    text = metadata['text']

    # Second step: import the nltk parser to process each sentence
    chunk_parser = nltk.RegexpParser(grammar)

    _tags = []
    for sent in taggedtext:
        if len(sent) > 0:
            tree = chunk_parser.parse(sent)
            for subtree in tree.subtrees():
                if subtree.label() == label:
                    _tags.append(subtree)

    # Visualizing the actual adj/adv phrase
    _phrases = []
    for sent in _tags:
        temp = ''
        for w, t in sent:
            temp += w+ ' '
        _phrases.append(temp)

    print('First 10 %s phrases (%s): %s' %(desc, text, _phrases[:10]))

    # Following our NLTK textbook, chapter 1 on Language Processing (https://www.nltk.org/book/ch01.html)
    # FREQUENCY DISTRIBUTIONS
    # Top 50 adjective phrases
    _freq = nltk.FreqDist(_phrases)

    print('Top %s phrases by frequency (%s): ' %(desc, text))
    for word, freq in _freq.most_common(50):
        print(word, freq)


    #print the list of our sentences:
    print('Length of %s phrase sentences (%s): %d' %(desc, text, len(_tags)))

    return _tags
```

In this function, we would:

- Define a parser based on custom grammar which is one of the args
- Iterate through the sentences in the corpora and use the parser
- Walk the subtrees and match for labels like ADJPH, ADVPH etc
- And last, walk *_tags* list and collect phrases to run through the *most_common()* function to mine for frequencies

Below is an excerpt of code showing an instance of calling or invoking the *chunking()* function for the adjective phrase:

```
grammar_adjph = "ADJPH: {<RB.?>+<JJ.?>}"
# This regex reads as: "find groups ("< >") of RBs (adverbs) together with groups of JJs (adjectives), with grou
# RBs with any ending (the "." is a placeholder or wildcard for the "R" and the "S" at the end of RBR and RBS,
# while "?" indicates "optional character" so RB can be found alone as well). Same regex operators apply to JJs.

caesar_adjph_tags = chunking(grammar_adjph, caesar_tagged, {'text':'Caesar', 'label':'ADJPH', 'desc':'adjective'
print('---------')
hamlet_adjph_tags = chunking(grammar_adjph, hamlet_tagged, {'text':'Hamlet', 'label':'ADJPH', 'desc':'adjective'
```

Output:

```
First 10 adjective phrases (Caesar): ['Truly sir ', 'Truly sir ', 'then senslesse ', '
there haue ', 'most exalted ', 'not mou ', 'yet againe ', 'once againe ', 'too stubbor
ne ', 'too strange ']

Top adjective phrases by frequency (Caesar):
so much  7
too much  4
so great  3
then thy  3
so good  3
Truly sir  2
so many  2
most Noble  2
more worthy  2
not backe  2
not meete  2

Length of adjective phrase sentences (Caesar): 143

First 10 adjective phrases (Hamlet): ['now strook ', 'once againe ', 'So frown ', 'mos
t obseruant ', 'most emulate ', 'So hallow ', 'so gracious ', 'So haue ', 'so farre ',
'not fayl ']

Top adjective phrases by frequency (Hamlet):
so farre  5
too much  3
as much  3
too blame  3
So much  2
not thy  2
not fit  2
so much  2
most excellent  2

Length of adjective phrase sentences (Hamlet): 237
```

Next, we will call the *chunking()* function for Adverb phrases:

```
# Now we look for "adverb phrases" or chunks that have 2 consecutive adverbs ('RB')
# First step: writing a grammar that defines POS rules of the adverb phrase the chunk
# we name this grammar "ADVPH" ("ADVerb PHrase")
grammar_advph = "ADVPH: {<RB>+<RB>}"

caesar_advph_tags = chunking(grammar_advph, caesar_tagged, {'text':'Caesar', 'label':'ADVPH', 'desc':'adverb'})
print('---------')
hamlet_advph_tags = chunking(grammar_advph, hamlet_tagged, {'text':'Hamlet', 'label':'ADVPH', 'desc':'adverb'})
```

Output:

```
First 10 adverb phrases (Caesar): ['art not ', 'So well ', 'heere so long ', 'as well
', 'as well ', 'so indeed ', 'till now ', 'not so ', 'So soone ', 'not then ']
Top adverb phrases by frequency (Caesar):
not so  9
not well  4
as well  3
heere so  3
So well  2
art not  1
heere so long  1
so indeed  1
till now  1
So soone  1
not then  1

Length of adverb phrase sentences (Caesar): 74

First 10 adverb phrases (Hamlet): ['spoke too ', 'Thus twice before ', 'doth well ', '
Thus much ', 'Not so ', 'too too ', 'not so much ', 'too roughly ', 'not well ', 'else
neere ']
Top adverb phrases by frequency (Hamlet):
not so  5
not well  4
thee well  2
very well  2
So much  2
too much  2
so well  2
spoke too  1
Thus twice before  1
doth well  1


Length of adverb phrase sentences (Hamlet): 84
```

2.2 List the top-50 adjective/adverb words

For this exercise, we would define a function *top_tokens()* taking POS tags as one of the arguments (type list).

```
# Top 50 tokens (grouped by Adjective, Adverb or Nouns)

def top_tokens(taggedtext, pos_list):
    _tokens = []
    for sentence in taggedtext:
        for word, pos in sentence:
            if pos in pos_list:
                if len(word)>1:
                    _tokens.append(word)
    freq_pos = nltk.FreqDist(_tokens)

    for word, freq in freq_pos.most_common(50):
        print(word,freq)

# Top 50 adjective tokens
print('Top 50 Adjective tokens:')
top_tokens(caesar_tagged, ['JJ', 'JJR', 'JJS'])
top_tokens(hamlet_tagged, ['JJ', 'JJR', 'JJS'])
```

Within the function, we would:

- Iterate the POS tag tokens and find a match in *pos_list*
- If there's a match, we append it to a list *_tokens* and then run an *nltk.FreqDist()* utility and use the *most_common()* words to mine for frequency

In the excerpt above, we see the usage for Adjective tokens and the corresponding output below.

Output:

```
Adjectives (Caesar):
good 48
thy 41
Noble 32
great 25
thou 24
such 23
much 22
true 18
Good 18
many 15
dead 14

Adjectives (Hamlet):
good 76
thy 54
more 37
such 34
most 30
much 25
dead 25
true 21
Good 21
thou 20
```

Next, using the same *top_tokens()* call, we mine for top-50 adverbs:

```
# Top 50 adverb tokens
print('Top 50 Adverb tokens:')
top_tokens(caesar_tagged, ['RB', 'RBR', 'RBS'])
top_tokens(hamlet_tagged, ['RB', 'RBR', 'RBS'])
```

Output:

```
Adverbs (Caesar):
not 255
so 103
then 79
well 40
now 39
too 30
Then 28
yet 27
heere 26
So 24
more 24
Now 24
there 15
once 13
thus 13

Adverbs (Hamlet):
not 313
so 139
then 75
now 68
well 53
too 50
more 46
very 44
most 35
So 33
Then 33
thus 33
Now 24
yet 23
```

And finally, nouns and verbs using the same custom function:

```
## TO DO / YOUR TURN NOW!
## NOUN EXTRACTION
## VERB EXTRACTION
## REMEMBER TO CHECK THE PENN POS TAGS LIST: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_
## TO FIND ALL TAGS

print('Top 50 Noun tokens:')
top_tokens(caesar_tagged, ['NN', 'NNS', 'NNP', 'NNPS']) #Noun, Noun-plural, Noun-Proper, Noun-Proper-plural
top_tokens(hamlet_tagged, ['NN', 'NNS', 'NNP', 'NNPS'])


print('\nTop 50 Verb tokens:')
top_tokens(caesar_tagged, ['VB', 'VBD', 'VBG', 'VBP', 'VBZ']) # Verb, Verb-past-tense, Verb-present participle,
                                                              # Verb-past participle, singular present (non-3rd)
                                                              # singular present (3rd)
top_tokens(hamlet_tagged, ['VB', 'VBD', 'VBG', 'VBP', 'VBZ'])
```

Output:

```
Nouns (Caesar):
Caesar 187
Brutus 160
Bru 152
```

```
Cassi 107
Cassius 85
Antony 75
Enter 58
men 57
man 54
thou 49
Ant 48
Lord 44


Nouns (Hamlet):
Ham 334
Lord 210
King 170
Hamlet 98
Hor 95
Enter 80
Qu 62
Laer 59
Ile 58
Ophe 55
Pol 48


Verbs (Caesar):
is 247
be 132
do 107
haue 100
are 96
was 64
know 63
did 61
am 52
let 41


Verbs (Hamlet):
is 349
be 175
haue 129
are 111
do 79
was 79
know 66
's 65
let 59
come 50
```

2.3 Average length of sentences

The average length of sentences can be computed by summing the length of all sentences and dividing it by the number of sentences in *caesar_split* or *hamlet_split* (which is essentially output of nltk.sent_tokenize function).

Below is a code excerpt:

```
# Following our NLTK textbook, Writing Structural Programs chapter
# section on Procedural vs Declarative style (http://www.nltk.org/book_1ed/ch04.html)

## CORPUS STATISTICS--SENTENCES LENGTH

# Calculating the average length of sentences in the entire corpus
# from http://www.nltk.org/book_1ed/ch04.html
caesar_total_corpus = sum(len(sent) for sent in caesar_split)
print('Average len of sentence (Caesar): %s' %(caesar_total_corpus / len(caesar_split)))

hamlet_total_corpus = sum(len(sent) for sent in hamlet_split)
print('Average len of sentence (Caesar): %s' %(hamlet_total_corpus / len(hamlet_split)))
```

Output:

```
Average len of sentence (Caesar): 69.16394472361809
Average len of sentence (Hamlet): 67.78853503184713
```

3. Conclusion

As we make further in-roads into the corporal statistics using NLP let us review data analyzed in this homework for both the Caesar and Hamlet Shakespearean texts.

Here's a summary:

Statistics	Caesar corpus	Hamlet corpus	Comment
Length of adjective phrases in corpus	143	237	As much as 1.65x more adjective phrases used in the Hamlet corpus
Length of adverb phrases in corpus	74	84	Marginal difference in the number of adverb phrases
Average length of sentence	69	67	Almost same length of sentences

As seen through each of the sections in this homework for adjective phrases, adverb phrases, adjectives, adverbs, nouns, or verbs very subtle differences are seen. The language or style adopted for each of these Shakespearean plays for most part, seem comparable.