NLP Homework 3

Sentiment Analysis

Student: Sharat Sripada (vssripad@syr.edu)

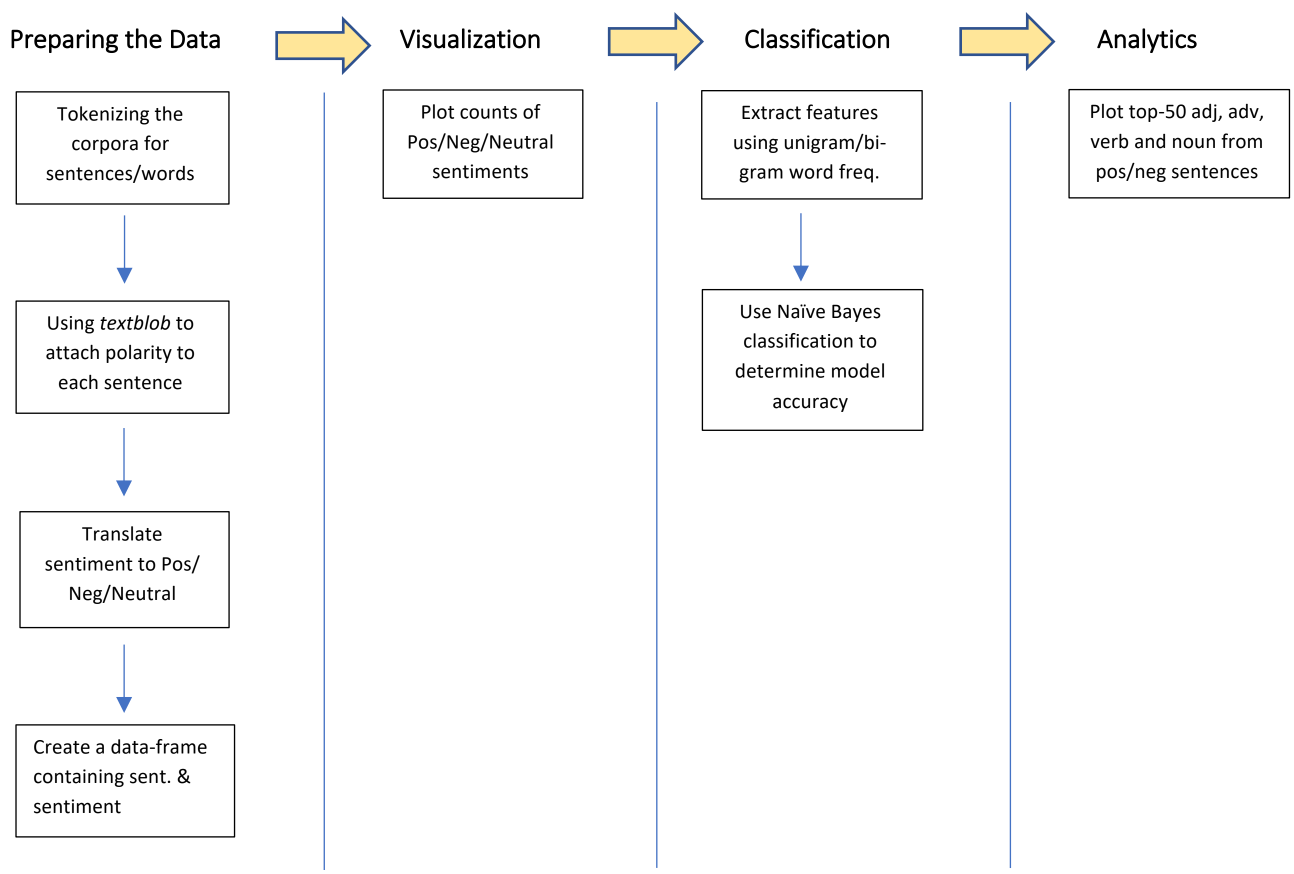
1. Introduction

In this week's homework we will perform sentiment analysis on the same corpora of Shakespearean books used in HW-1 and HW-2 namely, Julius Caesar and Hamlet. While the goal was to use knowledge and methodologies related to NLP gathered in IST-664 to bring out any subtle differences in language or style, we have not found any significant deviation thus far (apart from Hamlet just being a longer play and hence a larger corpus).

That is likely to continue with exploration into sentiment analysis as well, since the genre of both plays is tragic. Here are the high-level tasks:

- Sentiment analysis at sentence level
- Build a classifier to classify sentiment polarity of a sentence (as Positive, Negative or Neutral)
- Write to a csv file content of title, author, country, #pos-sent, #neg-sent, #nuetral-sent – This may not be particularly applicable for this case-study (although we will demonstrate the ability to write csv)
- Analyze all the positive sentences to identify top-50 adjective, adverb, noun, or verb phrases and do the same for negative sentences as well.

To better visualize all the tasks or assignments, here is a high-level summary what we are seeking to achieve through this report:



To begin with, we will continue to extract the corpora as before by 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. Preparing the data

Since the corpus mostly comprises contents from the book, the requirement was to first tokenize it into sentences and add a sentiment which could then be used to train and test models. For this purpose, we use a library called *TextBlob* which offers a *sentiment* functionality as below:

```
print (blob)
blob.sentiment
>> Analytics Vidhya is a great platform to learn data science.
Sentiment(polarity=0.8, subjectivity=0.75)
```

The code below shows how TextBlob was adopted for this study:

```
# Get polarity/subjectivity for each sent in both corpora
caesar_sentiment = [[sent, TextBlob(sent).sentiment] for sent in caesar_sentences[2:]]
print(caesar_sentiment[1:10])

hamlet_sentiment = [[sent, TextBlob(sent).sentiment] for sent in hamlet_sentences[2:]]
print(hamlet_sentiment[1:10])
```

Notice that, we walk sentences (tokenized by function *sent_tokenize()*) in both corpora and create a list within a list comprehension to encompass a sentiment polarity or subjectivity tag alongside each sentence. For example:

```
['Hence: home you idle Creatures, get you home:\nIs this a Holiday?', Sentiment(polari
ty=0.0, subjectivity=0.0)]
```

Further, we can also specifically access the polarity or subjectivity of each sentence using an object or instance attribute as shown below. This would later help us translate a polarity range (-1, 1) to Negative (range (-1, <0), or to Neutral (0) or Positive (0, 1):

```
sent = caesar_sentiment[2][0]
polarity = caesar_sentiment[2][1].polarity
subjectivity = caesar_sentiment[2][1].subjectivity
print('Sentence %s has polarity %s and subjectivity %s' %(sent, polarity,
subjectivity))
```

Output showing sentence and importantly its polarity and subjectivity can be extracted:

```
Sentence Hence: home you idle Creatures, get you home:Is this a Holiday? has polarity
0.0 and subjectivity 0.0
```

The translate function, its usage and output:

```python
# Next we will translate the polarity and subjectivity index to the tags we want to see.
import copy

def translate_sentiment(sent_blob):
    '''
    Given a sentence, derive its sentiment
    '''
    if sent_blob[1].polarity < 0:
        sent_blob.append('Negative')
    elif sent_blob[1].polarity == 0:
        sent_blob.append('Neutral')
    elif sent_blob[1].polarity > 0:
        sent_blob.append('Positive')
    return sent_blob

print(translate_sentiment(caesar_sentiment[2]))
print(translate_sentiment(hamlet_sentiment[2]))
```

Output:

```
['Hence: home you idle Creatures, get you home:\nIs this a Holiday?', Sentiment(polari
ty=0.0, subjectivity=0.0), 'Neutral'] <- Notice the tag Nuetral which is extracted fro
m the polarity index
```

Finally, we summarize this data into a Pandas data-frame comprising column-names – Sentence, TextBlob-sentiment(raw) and Sentiment. All further study would be based on this data-frame.

```python
# Let's make a data-frame comprising all the above data
import pandas as pd

def create_df(textblob_raw):
    df = pd.DataFrame(columns=['Sentence', 'TextBlob-sentiment(raw)', 'Sentiment'])
    for blob in textblob_raw:
        new_blob = translate_sentiment(blob)
        row = {'Sentence': new_blob[0],
               'TextBlob-sentiment(raw)': new_blob[1],
               'Sentiment': new_blob[2]
        }
        df = df.append(row, ignore_index=True)
    return df

```

Snapshot of the data-frame:

```
In [7]:
df_hamlet = create_df(hamlet_sentiment)
df_hamlet.head()
```

Out[7]:

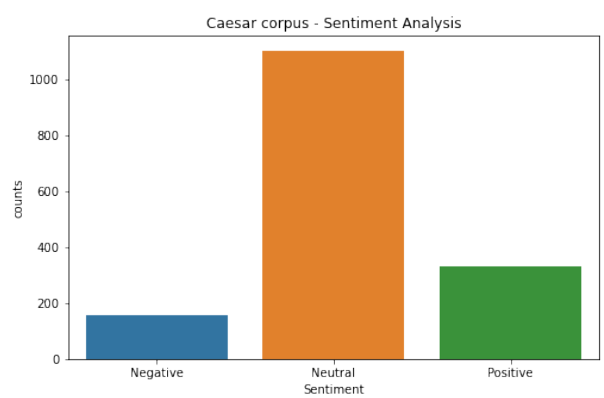
	Sentence	TextBlob-sentiment(raw)	Sentiment
0	Enter Barnardo and Francisco two Centinels.	(0.0, 0.0)	Neutral
1	Barnardo.	(0.0, 0.0)	Neutral
2	Who's there?	(0.0, 0.0)	Neutral
3	Fran.	(0.0, 0.0)	Neutral
4	Nay answer me: Stand & vnfold\nyour selfe\n\n ...	(0.0, 0.0)	Neutral

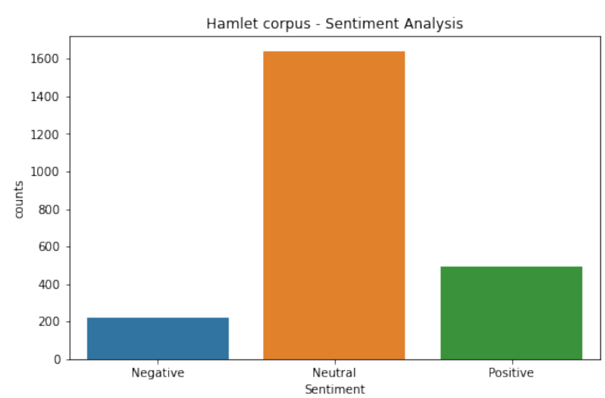
3. Visualization

The translation function based on sentiment polarity, helps us now group and count occurrences of each type viz. Positive, Negative or Neutral. Here is the code that groups the data and plots a bar graph using matplotlib and sns libraries:

```python
# Make a bar-plot of sentiment-counts across both the corpora

import matplotlib.pyplot as plt
import seaborn as sns

def plot_bar_graph(corpus_name, label_groups):
    plt.figure(figsize=(8,5))
    ax = sns.barplot(x="Sentiment", y="counts", data=label_groups)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
    ax.set_title(label="%s" %corpus_name)
    plt.show()


caesar_labels = df_caesar.groupby('Sentiment').size().reset_index(name='counts')
plot_bar_graph('Caesar corpus - Sentiment Analysis', caesar_labels)

hamlet_labels = df_hamlet.groupby('Sentiment').size().reset_index(name='counts')
plot_bar_graph('Hamlet corpus - Sentiment Analysis', hamlet_labels)
```

Correspondingly, this is the bar-plot visualization:





Interpreting the charts

We know by now Hamlet is comparatively a longer play or book and would therefore understandably comprise a larger corpus of words.

It is however interesting to see from the plots that although the genre is tragic, there is still largely a positive polarity to sentence sentiment (two times that of negative sentence polarity). This goes to show Shakespeare wrote with a subtle undertone of humor or sarcasm. Perhaps, examining words in greater detail (and their POS references) at a later point would make this evident.

NOTE

The plots are based on raw text without necessarily filtering for stop words. Neither have we used techniques like regular expressions to clean up for sentence anomalies. Presumably employing such techniques could spill sentiment from Neutral to Positive or Negative; but may not drastically alter the ratio since we have sufficient data when making the inference.

Further, a summary was written to csv (utilizing the *to_csv()* function) with the following code:

```
# Also, write a csv with this data
df_summary = pd.DataFrame(columns=['Book', 'Negative', 'Neutral', 'Positive'])

df_summary = df_summary.append({'Book': 'Caesar',
                                'Negative': df_caesar_sentiment.loc[0][1],
                                'Neutral': df_caesar_sentiment.loc[1][1],
                                'Positive': df_caesar_sentiment.loc[2][1]}, ignore_index=True)

df_summary = df_summary.append({'Book': 'Hamlet',
                                'Negative': df_hamlet_sentiment.loc[0][1],
                                'Neutral': df_hamlet_sentiment.loc[1][1],
                                'Positive': df_hamlet_sentiment.loc[2][1]}, ignore_index=True)

df_summary.head()
df_summary.to_csv('new_data.csv', index=False)
```

Output:
Shows the contents of the csv file with counts of positive, negative and neutral sentences from both corpora:

```
# $ less new_data.csv
# Book,Negative,Neutral,Positive
# Caesar,157,1101,332
# Hamlet,222,1640,491
```

4. Classification Task

With the data collection sufficiently complete that is, every sentence in both corpora have a corresponding sentiment tag, we can now build a model and make predictions.

For this exercise, we will use the Naïve Bayes classification model and use K-fold (k=5) cross-validation method rather than a flat 80-20 split.

Why is k-fold cross-validation method?

Cross-validation is a powerful tool that helps us better use our data (spread) and therefore provides useful information about the performance of the algorithms we may choose.

Here's a good visualization of how cross-validation works:

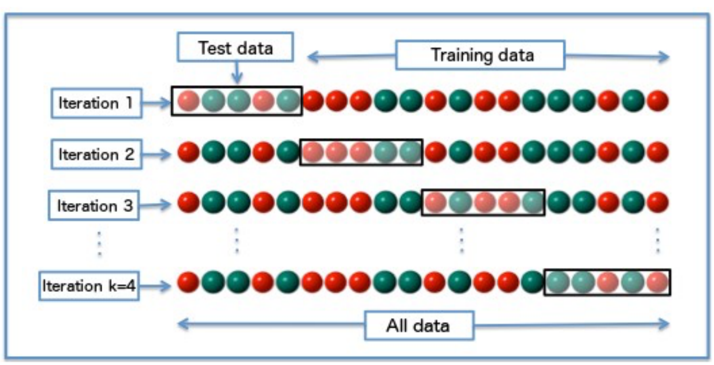


Diagram of k-fold cross-validation with k=4.

Data is split more than once (controlled by folds or commonly known as variable k), than in the classic 1-split 80-20, 90-10 or 70-30 representing train and validation/test data.

Each split or fold is a good representation of the whole data and therefore helps train or develop comprehensive and robust models. Accuracy or performance is subsequently derived by averaging the results over several iterations as shown below:

```
1  # Using k-folds = 5 we will fairly randomize the train & test data
2
3  import numpy as np
4  from sklearn.model_selection import KFold
5
6  def ml_nb(featuresets):
7      kf = KFold(n_splits = 5)
8      sum = 0
9
10     for train, test in kf.split(featuresets):
11         train_data = np.array(featuresets)[train]
12         test_data = np.array(featuresets)[test]
13         classifier = nltk.NaiveBayesClassifier.train(train_data)
14         sum += nltk.classify.accuracy(classifier, test_data)
15
16
17
18     #storing the score in a variable
19     acc1 = sum/5
20
21     return  classifier, acc1
22
23 _, caesar_acc = ml_nb(caesar_featuresets)
24 _, hamlet_acc = ml_nb(hamlet_featuresets)
25
26 print('Accuracy for unigram feature-sets on Caesar corpus %s' %caesar_acc)
27 print('Accuracy for unigram feature-sets on Hamlet corpus %s' %hamlet_acc)
```

See rows 14-19 above on how accuracy is summed over each k-fold iteration and then averaged to provide an overall accuracy.

Building models

While we chose a classification algorithm, Naïve Bayes to solve our problem here and make a prediction of sentence sentiment we take two fundamental NLP approaches:

1. Uni-gram feature-set

   In this approach, we derive a feature set of single words ordered by frequency and pick the top-2000 to find a match on each sentence.

   Note here, we use a regex to find only words (re.findall(r'\w+')) and eliminate any alpha-numeric words or words that may comprise punctuation or new-line characters.

```
# Defining set of words that will be used for features

# We'll find the 2000 most common words and used them as an important feature of the whole corpus
def unigram_freq(docs):
    all_words = []
    # Write a regex to pull only the word portion & leave
    # out any punctuation marks etc.
    for (sentence, category) in docs:
        for word in sentence.split():
            try:
                all_words.append(re.findall(r'\w+', word)[0])
            except IndexError:
                pass
    top_words = nltk.FreqDist(all_words)
    most_common_words = top_words.most_common(2000)
    word_features = [word for (word,count) in most_common_words]
    return all_words, word_features
```

Once the word features are identified, *document_features()* is called to iterate over the corpus and create a dictionary that would comprise 'contains(<word>): <Boolean>' where Boolean True/False would be dependent on a word match.

```python
1 # now we will use that list of most frequent words in the entire corpus
2 # to iterate over each sentence and check if any of those words are present
3 # in that way, we will see if this unigram corpus feature is present on that particular sentence
4 # using Boolean logic that matches values and returns 'True' or 'False'
5 # we do this by defining a Python "function," i.e.a piece of code writen to be reused
6 def document_features(document, word_features):
7     document_words = set(document)
8     #we open a Pytnon dictionary instead of a list
9     features = {}
10     for word in word_features:
11         #checking if the word from word_features matches a word in the document
12         features['contains({})'.format(word)] = (word in document_words)
13     return features
14
15 caesar_featuresets = [(document_features(d, caesar_word_features), c) for (d, c) in caesar_docs]
16 hamlet_featuresets = [(document_features(d, hamlet_word_features), c) for (d, c) in hamlet_docs]
17
18
19 print("Length of feature set for Caesar = %d" %len(caesar_featuresets))
20 print("Length of feature set for Hamlet = %d" %len(hamlet_featuresets))
```

```
Length of feature set for Caesar = 1590
Length of feature set for Hamlet = 2353
```

The list comprehensions on row 15-16 are organized as a tuple at each index with a dictionary (in the format above) and a sentence sentiment alongside.

Example of the list-comprehension *caesar_featuresets* at index 0 (trimmed for brevity):

```
({'contains(I)': False, 'contains(the)': False, 'contains(and)': False, 'contains(to)': False, 'contains(you)': False, 'contains(of)': False, 'contains(not)': False, 'contains(a)': True, 'contains(is)': False, 'contains(And)': False, 'contains(in)': False, 'contains(that)': False, 'contains(my)': False, 'contains(Caesar)': False, 'contains(me)': False, 'contains(it)': False, 'contains(him)': False, 'contains(Brutus)': False, 'contains(Bru)': False, 'contains(his)': False, 'contains(this)': False, 'contains(your).. False}, 'Neutral')
```

2. Bi-gram feature-set

   In this approach, we will group two-words at a time using the NLTK collocations.BigramAssocMeasures() and score frequencies using the score_ngrams() functionality.

```python
# Since the accuracy is low, let's also try this with a
# bi-gram featureset

# Re-using the code to clean-up
from nltk.collocations import *
import re

def bigram_freq(all_words):
    #data cleaning and preprocessing
    stopwords = nltk.corpus.stopwords.words('english')

    #creating bigrams features for the corpus and applying cleaning steps
    bigram_measures = nltk.collocations.BigramAssocMeasures()
    finder = BigramCollocationFinder.from_words(all_words)
    finder.apply_word_filter(lambda w: w in stopwords)
    scored = finder.score_ngrams(bigram_measures.raw_freq)

    #extracting clean bigrams (no frequency information)
    bigram_features = [bigram for (bigram, count) in scored[:2000]]

    return bigram_features
```

Correspondingly, the *bi_document_features()* function would walk the word pairs (collocations) in each email, and attempt to find a match with the bi-gram features. Like the unigram document features functionality, the function would return a dictionary comprising ‘contains(<word-pairs>): <Boolean>` which would be then placed alongside a category spam or ham at each node or index in the list.

```
def bi_document_features(document, bigram_features):
    document_words = list(nltk.bigrams(document))
    features = {}
    for word in bigram_features:
        #boolean logic will return 'True' if there is a match, or 'False' if not
        features['contains({})'.format(word)] = (word in document_words)
    return features

# applying the function to our documents
caesar_featuresets2 = [(bi_document_features(d, caesar_word_bi_features), c) for (d, c) in caesar_docs]
hamlet_featuresets2 = [(bi_document_features(d, hamlet_word_bi_features), c) for (d, c) in hamlet_docs]
```

Example of the list comprehension caeser_featureset2(trimmed for brevity):

```
({"contains(('I', 'haue'))": False, "contains(('I', 'know'))": False, "contains(('Cass
i', 'I'))": False, "contains(('I', 'shall'))": False, "contains(('Bru', 'I'))": False,
"contains(('Mark', 'Antony'))": False, "contains(('let', 'vs'))": False, "contains(('M
arke', 'Antony'))": False, "contains(('And', 'I'))": False, "contains(('Lord', 'Bru'))
": False, "contains(('I', 'feare'))": False, "contains(('Caesar', 'Caes'))": False, "c
ontains(('Enter', 'Brutus'))": False, "contains(('For', 'I'))": False, "contains(('Nob
le', 'Brutus'))": False, "contains(('Bru', 'O'))": False, "contains(('I', 'may'))":..
'Neutral')
```

NOTE:
Notice the difference in dictionary keys at index 0 for the unigram and bi-gram feature sets. This will largely infl uence the models we build hence.

The modeling itself was written in a general manner so it can return a classifier and accuracy score given a feature-set, and is represented by the function *ml_nb():*

```
1  # Using k-folds = 5 we will fairly randomize the train & test data
2  
3  import numpy as np
4  from sklearn.model_selection import KFold
5  
6  def ml_nb(featuresets):
7      kf = KFold(n_splits = 5)
8      sum = 0
9  
10     for train, test in kf.split(featuresets):
11         train_data = np.array(featuresets)[train]
12         test_data = np.array(featuresets)[test]
13         classifier = nltk.NaiveBayesClassifier.train(train_data)
14         sum += nltk.classify.accuracy(classifier, test_data)
15 
16 
17 
18     #storing the score in a variable
19     acc1 = sum/5
20 
21     return  classifier, acc1
```

The data is split as train and test before running the NLTK NaiveBayesClassifier() on train-data (seen on row-13). The prediction or classification and accuracy is recorded and averaged over the k-folds.

Results

Uni-gram vs bi-gram results are tabulated here:

Naïve Bayes classifier k-fold = 5	Accuracy - Caesar corpus	Accuracy - Hamlet corpus
Uni-gram frequency	67.0%	69.3%
Bi-gram frequency	69.2%	69.7%

The results show a slightly improved accuracy when using the bi-gram frequency sets. Specifically, in the case of the Caesar corpus we see a 2.2% improvement while a marginal improvement for the Hamlet corpus.

5. Analytics

Finally, we would like to look at the point-of-speech tags and how they vary in sentences conveying positive or negative polarity.

The code is mostly a re-use from HW-2 written generally like this:

```
# Utilizing code from HW-2, we will now call the same functions for

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)
```

Essentially, given a POS tagged sentence and pre-defined POS tags the function *top_tokens()* would iterate each sentence and word, sort the words based on frequency and print the top FIFTY.

The Stanford POS tagger *nltk.pos_tag()* will help with tagging and we create list comprehensions comprising tokenized words and the corresponding POS tag:

```
# Walk the positive/negative sentences and word tokenize it, before running
# it through the pos_tag()
hamlet_tokens_pos = [nltk.word_tokenize(sent) for sent in hamlet_sent_pos]
hamlet_tags_pos = [nltk.pos_tag(token) for token in hamlet_tokens_pos]

hamlet_tokens_neg = [nltk.word_tokenize(sent) for sent in hamlet_sent_neg]
hamlet_tags_neg = [nltk.pos_tag(token) for token in hamlet_tokens_neg]
```

We then call the *top_tokens()* function for each POS we are interested in viz. Adjectives, Adverbs, Nouns or verbs:

```
# Stats for Hamlet data:
# Top 50 adjective tokens
print('Top 50 Adjective tokens (positive):')
h_adj_pos = top_tokens(hamlet_tags_pos, ['JJ', 'JJR', 'JJS'])

print('Top 50 Adjective tokens (negative):')
h_adj_neg = top_tokens(hamlet_tags_neg, ['JJ', 'JJR', 'JJS'])

# Top 50 adverb tokens
print('Top 50 Adverb tokens (positive):')
h_adv_pos = top_tokens(hamlet_tags_pos, ['RB', 'RBR', 'RBS'])

print('Top 50 Adverb tokens (negative):')
h_adv_neg = top_tokens(hamlet_tags_neg, ['RB', 'RBR', 'RBS'])

print('Top 50 Noun tokens: (positive)')
h_noun_pos = top_tokens(hamlet_tags_pos, ['NN', 'NNS', 'NNP', 'NNPS']) #Noun, Noun-plural, Noun-Proper, Noun-Pro
print('Top 50 Noun tokens: (negative)')
h_noun_neg = top_tokens(hamlet_tags_neg, ['NN', 'NNS', 'NNP', 'NNPS'])


print('\nTop 50 Verb tokens: (positive)')
h_verb_pos = top_tokens(hamlet_tags_pos, ['VB', 'VBD', 'VBG', 'VBP', 'VBZ']) # Verb, Verb-past-tense, Verb-prese
                                                                              # Verb-past participle, singular present (non-3rd)
                                                                              # singular present (3rd)
print('Top 50 Verb tokens: (negative)')
h_verb_neg = top_tokens(hamlet_tags_neg, ['VB', 'VBD', 'VBG', 'VBP', 'VBZ'])
```

Following is a summary of the words for different POS tags across the corpora:

1. Caesar corpus (top-50 words)

Adjectives – Positive sentences	Adjectives – Negative sentences	Adverbs – Positive sentences	Adverbs – Negative sentences	Verbs – Positive sentences	Verbs – Negative sentences	Nouns – Positive sentences	Nouns – Negative sentences
good 46	such 11	not 85	not 52	is 87	is 45	Caesar	Caesar
Noble 31	dead 11	so 46	then 18	be 64	do 24	78	30
great 24	bad 10	then 32	so 16	haue 47	haue 24	Brutus	Cassius
much 20	other 10	more 16	now 11	are 38	be 23	69	21
Good 18	thy 7	now 13	well 10	do 38	are 16	Bru 36	men 20
many 14	common 6	well 12	too 9	was 28	did 15	Antony	Bru 19
true 13	dangerous	yet 11	Then 9	am 23	was 14	29	Brutus
thy 11	6	Now 11	more 8	know 23	know 12	men 26	16
best 10	hard 5	too 11	yet 6	let 19	tell 9	Cassi 2	Cassi 1
thou 10	strange 5	heere 11	heere 4	did 19	were 8	6	3
full 10	angry 5	very 10	So 3	come 15	wrong 8	Cassius	Caesars
strong 10	true 5	most 9	long 3	say 13	Let 6	24	10
such 9	bloody 5	So 9	once 3	were 13	come 6	man 24	man 10
sure 9	wrong 5	directly	Now 3	make 13	see 6	selfe 1	Rome 9
worthy 8	sad 4	8	rather 3	go 11	let 5	8	vs 9
better 8	little 4	there 8	yong 3	tell 11	beare 5	day 17	selfe 8
Most 8	Honourabl	Then 7	indeed 2	see 10	say 5	Friends	Antony
most 8	e 4	as 6	else 2	had 10	am 5	16	8
high 6	thou 3	thus 6	Well 2	feare 9	go 5	death 1	thou 6
old 5	euery 3	first 6	as 2	finde 9	loues 4	6	things
more 5	ordinary	still 5	alone 2	take 9	hold 4	vs 15	6
mine 5	2	else 5	still 2	stand 9	loue 4	Gods 14	day 6
gentle 5	late 2	onely 4	thus 2	beare 8	had 4	Caesars	Romans
thee 5	gentle 2	once 4	Indeed 2	heare 7	doth 4	14	6
glad 4	borne 2	much 4	much 2	fell 7	looke 4	Which 1	'd 6
vs 4	feeble 2	wisely 4	wherefore	thee 7	hath 4	3	time 5
first 4	sleepe 2	Therefore	1	Let 7	thee 4	Caes 13	word 5
welcome 4	much 2	3	till 1	put 6	take 4	thou 12	Alas 5
wrong 4	Such 2	together	doth 1	till 6	giue 4	Rome 12	hands 5
haue 3	third 2	3	therefore	giue 6	vse 3	night 1	thing 5
further 3	most 2	Yet 3	1	thou 6	perceiue	2	Cas 5
other 3		yee 3	very 1	hath 6	3	hand 11	Lord 5

new 3 huge 3 giuen 3 right 3 common 3 seene 3 strange 3 free 3 enough 3 dead 3 happy 3 mighty 3 ready 3 Ambitious 3 Honourable 3 meete 3 sir 2 narrow 2	terrible 2 secret 2 impossible 2 better 2 good 2 constant 2 least 2 pitty 2 cold 2 more 2 certaine 2 poor 2 sorry 2 vile 2 worst 2 ill 2 safe 1 senslesse 1 many 1 seruile 1 stubborne 1 deceiu 1	hence 3 Not 3 along 3 better 3 away 3 Truly 2 perhaps 2 nod 2 downe 2 sayes 2 sicke 2 dye 2 Thus 2 fast 2 smile 2 best 2 straight 2 truly 2 seene 2 softly 2 partly 2	almost 1 laugh 1 away 1 Not 1 surly 1 already 1 Nobly 1 Quite 1 hard 1 further 1 impatiently 1 no 1 litter 1 forth 1 deere 1 marke 1 merry 1 instantly 1 hence 1 fast 1	speake 6 made 5 looke 5 loue 5 Is 5 keepe 5 'd 5 thinke 5 comes 5 get 4 saw 4 doth 4 run 4 went 4 lay 4 vs 4 being 4 appeare 4	make 3 follow 3 heare 3 doe 3 thinke 3 bring 3 speake 3 feare 3 keepe 2 euery 2 thou 2 shake 2 loose 2 write 2 get 2 thinkes 2 put 2 fell 2 went 2	Ant 11 hath 10 'd 10 Caska 10 blood 10 Haue 9 Octauius 9 Will 8 time 8 morrow 8 Lord 8 heart 8 Octa 8 Did 7 things 7 Friend 7 Cask 7 Noble 7 Thy 7 cause 7 thee 7 Mark 7 Come 7 Honor 6 Spirit 6 words 6 Roman 6 minde 6 euer 6	Sir 4 hearts 4 Will 4 hand 4 World 4 Caska 4 Cask 4 fire 4 Roman 4 blood 4 Caius 4 death 4 night 4 Which 4 bloody 4 thee 4 Octauius 4 Trade 3 Thou 3 Be 3 eyes 3 shew 3 Friend 3 cold 3 hee 3 Did 3

2. Hamlet corpus (top-50 words)

Adjectives – Positive sentences	Adjectives – Negative sentences	Adverbs – Positive sentences	Adverbs – Negative sentences	Verbs – Positive sentences	Verbs – Negative sentences	Nouns – Positive sentences	Nouns – Negative sentences
good 74 more 27 most 26 much 23 Good 21 true 18 thy 18 great 18 such 17 sweet 14 many 14 old 13 Most 12 full 9 other 9 deere 9 excellent 9	dead 20 thy 12 mad 11 more 10 Other 10 such 7 thou 6 little 6 owne 6 other 6 long 6 dangerous 6 common 5 late 5 wrong 5 strange 4 second 4	not 117 so 56 then 37 more 33 most 31 very 30 now 25 well 22 too 21 Then 17 much 16 heere 13 So 13 thus 13 Now 12 yet 12 as 10 once 8	not 59 so 26 then 15 more 13 very 11 too 11 well 11 thus 10 now 8 So 5 long 5 Then 5 n't 5 away 5 once 4 most 4 indeed 4 Thus 3	is 129 be 72 haue 55 are 50 was 33 know 30 am 28 do 27 did 27 let 25 make 22 had 17 see 16 doe 16 come 15 tell 15 's 15 Let 14	is 75 be 40 haue 16 are 15 was 13 's 11 say 10 had 9 do 9 let 8 make 8 am 7 did 7 go 7 speake 6 know 6 come 6 tell 6	Ham 60 Lord 59 King 46 Hamlet 29 Father 23 Ile 21 time 21 death 20 Sir 19 vs 17 Hor 17 man 17 Queene 16	Ham 23 King 21 Hamlet 12 Lord 11 selfe 9 'd 8 Laer 8 Alas 8 Qu 8 hath 7 Hor 7 death 7 day 7 againe 7 time 7 Sir 7

Noble 9 first 9 welcome 8 young 8 best 8 free 8 fine 8 owne 7 whole 7 better 7 last 6 hot 6 thou 6 thee 6 mine 6 right 6 particular 5 strange 5 soft 5 dead 5 selfe 5 honest 5 vs 5 second 5 same 4 gracious 4 farre 4 late 4 glad 4 oft 4 sure 4 fit 4 Such 4	false 4 bad 4 ill 4 guilty 3 most 3 old 3 hard 3 wide 3 double 3 true 3 horrible 3 desperate 3 tame 3 farre 3 cold 3 absurd 2 flat 2 pale 2 dull 2 madnesse 2 fit 2 mine 2 Noble 2 wondrous 2 violent 2 proper 2 hid 2 loose 2 tedious 2 capeable 2 heere 2 good 2 rude 2	there 6 first 6 indeed 6 better 6 freely 5 Not 5 else 5 therefore 5 neuer 5 still 5 further 4 away 4 enough 4 doth 3 truly 3 goodly 3 Very 3 poore 3 alone 3 together 3 right 3 here 3 perhaps 3 quite 3 twice 2 nightly 2 n't 2 long 2 vs 2 Together 2 Thus 2 willingly 2	as 3 ere 3 there 3 Now 3 Long 2 neere 2 heartily 2 rather 2 yong 2 no 2 heard 2 humbly 2 neyther 2 hence 2 'Twere 2 strangely 2 sicke 1 twice 1 before 1 vnmanly 1 still 1 meerely 1 Indeed 1 coldly 1 together 1 exactly 1 stately 1 Almost 1 alone 1 here 1 thou 1 shrewdly 1	giue 14 speake 13 hold 13 put 12 say 11 were 11 hath 10 take 10 Is 9 set 9 said 9 finde 8 thinke 8 please 8 comes 7 pray 7 shew 7 selfe 7 heare 7 go 7 goes 6 're 6 Take 6 lost 5 thy 5 th 5 stand 5 liue 5 keepe 5 vse 5 loue 5 does 5	were 5 does 5 comes 5 Let 5 liue 4 hath 4 thou 4 doe 4 're 4 finde 4 looke 4 makes 4 thinke 4 take 4 call 4 thy 3 being 3 grow 3 selfe 3 draw 3 downe 3 end 3 seeke 3 made 3 fell 3 beare 3 put 3 wrong 2 shewes 2 Take 2 followed 2 thine 2	Horatio 15 life 15 Nature 15 loue 15 thing 14 Mother 14 Qu 14 thou 13 Laer 13 night 12 world 12 heart 12 Heauen 12 Rosin 12 Laertes 11 Ophelia 11 Sonne 11 Fathers 11 Giue 10 'd 10 thee 10 Pol 10 eyes 9 day 9 Which 9 head 9 vp 9 purpose 9 youth 9 blood 9 matter 9 mine 8 downe 8 Fortinbras 8 hand 8 hath 8 Denmarke 8	Heauen 6 body 6 Father 6 head 6 Loue 6 Come 6 Pol 6 Ile 6 haue 6 Mother 6 'T 5 poore 5 winde 5 meanes 5 man 5 Queene 5 No 5 thee 5 vp 5 nothing 5 vs 5 heart 4 th 4 't 4 fault 4 Horatio 4 night 4 nature 4 Soule 4 Polon 4 Be 4 hast 4 loue 4 cause 4

Comparing results between POS tags across corpora

Since it is hard to compare the large set of words in the tables via visual inspection, we use Python *sets* to bring out any differences. The syntax is *set1 minus set2 and* would show all elements present in set1 but not set2.

NOTE: The prepend *h_<var-name>* is reflective of set comprising words from the Hamlet corpus

There are subtle differences in the words used between the corpora:

- Positive sentiment adjectives:
  print(set(adj_pos) - set(h_adj_pos))
  ```
  {'giuen', 'new', 'enough', 'narrow', 'sir', 'high', 'Ambitious', 'meete', 'gent
  le', 'further', 'wrong', 'Honourable', 'haue', 'ready', 'strong', 'worthy', 'mi
  ghty', 'seene', 'common', 'huge', 'happy'}
  ```

- Negative sentiment adjectives:
  print(set(adj_neg) - set(h_adj_neg))
  ```
  {'better', 'safe', 'least', 'vile', 'bloody', 'sorry', 'feeble', 'constant', 'g
  entle', 'ordinary', 'pitty', 'terrible', 'secret', 'many', 'certaine', 'euery',
  'deceiu', 'Honourable', 'worst', 'stubborne', 'angry', 'senslesse', 'borne', 'm
  uch', 'sad', 'Such', 'impossible', 'seruile', 'sleepe', 'poor', 'third'}
  ```

  Notice adjectives like bloody, angry, sad etc. a little more prevalent in the Caesar corpus.

- Nouns would be interesting to compare. We would expect a few different names of protagonists, antagonists, or places as seen below:
  print(set(noun_pos) - set(h_noun_pos))
  ```
  {'Mark', 'Caes', 'Cask', 'Antony', 'morrow', 'Come', 'Caesar', 'Cassi', 'Ant',
  'minde', 'Gods', 'Will', 'Cassius', 'men', 'cause', 'Haue', 'Honor', 'Caska', '
  Octauius', 'Spirit', 'Roman', 'euer', 'Did', 'Brutus', 'Noble', 'Octa', 'things
  ', 'Friends', 'words', 'Rome', 'Thy', 'selfe', 'Bru', 'Caesars', 'Friend'}
  ```

  print(set(h_noun_pos) - set(noun_pos))
  ```
  {'mine', 'Horatio', 'Laertes', 'Qu', 'thing', 'Sir', 'Laer', 'Pol', 'downe', 'F
  athers', 'King', 'Ophelia', 'youth', 'loue', 'matter', 'Hor', 'Hamlet', 'Queene
  ', 'Sonne', 'head', 'vp', 'Heauen', 'purpose', 'life', 'Mother', 'eyes', 'Denma
  rke', 'world', 'Father', 'Nature', 'Giue', 'Fortinbras', 'Rosin', 'Ham', 'Ile'}
  ```

  Caesar mostly certainly had references to Rome while Hamlet shows references to Denmark, possibly t herefore bracing some of that nativity in the plot.

  Words like 'Sir' standout in the Hamlet corpus whereas 'Noble' or 'Honor' are prevalent in Caesar.

- Finally, examining verbs - comparing words with negative sentiment:
  print(set(h_verb_neg) - set(ver_neg))
  ```
  {'thy', 'comes', 'made', 'being', 'end', 'downe', 'thine', 'seeke', 'finde', 'g
  row', 'shewes', 'followed', 'makes', 'Take', 'liue', 'does', 'selfe', 'draw', '
  call'}
  ```

  print(set(ver_neg) - set(h_verb_neg))
  ```
  {'keepe', 'see', 'went', 'shake', 'bring', 'thee', 'giue', 'vse', 'g et', 'hold
  ', 'perceiue', 'follow', 'feare', 'loose', 'heare', 'thinkes', 'loues', 'euery'
  , 'doth', 'loue', 'write'}
  ```

To corroborate any verbs with adjectives like bloody, angry etc. it seemed worthwhile explore any differ ences here. Nothing apparent.

6. Conclusion

The case-study at the outset was to analyze two popular Shakespearean plays Julius Caesar and Hamlet. The goal was to be objective and let *Natural Language Processing* analyze corpora and bring out any nuances in style of language, grammar, or sentiment; albeit when the plays were popularly known to be of the same genre.

Turns out through work presented across the past few weeks (HW-1 and HW-2 included), we did not find anything glaringly different and summarize the findings as follows:

- Shakespeare seems to have largely used similar constructs on language, grammar, or sentiment.
- Sentiment was analyzed to be largely neutral but subtly inclining towards a positive polarity and that is particularly interesting given the tragic genre of both plays. It also indicates his style of having an undercurrent of humor or sarcasm to every situation that made his plays popular and entertaining (to both watch or read).
- Finally, we did find Hamlet to be a larger corpus which is in line with any research one would do on the internet.

Appendix

Extras - Demonstrate ability to write classification results to a csv file

The csv will encompass data with the following headings:

Sentence, pos-sent, neg-sent, neutral-sent

A function *create_predictions_csv()* was written which given a book-name, associated classifier, sentences, and a feature-set and would correspondingly run the classifier and capture predictions.

The sentence along with a counter to indicate if it was a pos-sent, neg-sent or neutral-sent would be appended to a data-frame df_predict which is eventually written to a .csv called predictions.csv

NOTE:
Since results earlier showed better accuracy with bigrams at ~70%, we will use that classifier here.

Here is the code:

```
1  # Write classifier predictions to a csv
2  # Form a test_data with bottom 20% of sentences
3
4  def create_predictions_csv(book_name, classifer, book_sentences, bi_features):
5      test_len = int(0.2 * len(book_sentences))
6      sentences = book_sentences[-test_len:]
7
8      df_predict = pd.DataFrame(columns=['Sentence', 'Book', 'pos-sent', 'neg-sent', 'neutral-sent'])
9
10     for sents in sentences:
11         senti = caesar_classifier2.classify(bi_document_features(nltk.word_tokenize(sents), bi_features))
12         #adding items to the counter as they are classified
13         if senti.lower() == 'positive':
14             df_predict = df_predict.append({'Sentence': sents,
15                                             'Book': book_name,
16                                             'pos-sent': 1,
17                                             'neg-sent': 0,
18                                             'neutral-sent': 0}, ignore_index=True)
19
20         elif senti.lower() == 'negative':
21             df_predict = df_predict.append({'Sentence': sents,
22                                             'Book': book_name,
23                                             'pos-sent': 0,
24                                             'neg-sent': 1,
25                                             'neutral-sent': 0}, ignore_index=True)
26
27         else:
28             df_predict = df_predict.append({'Sentence': sents,
29                                             'Book': book_name,
30                                             'pos-sent': 0,
31                                             'neg-sent': 0,
32                                             'neutral-sent': 1}, ignore_index=True)
33
34     df_predict.to_csv('predictions.csv', index=False, mode='a')
35     print(df_predict.head())
36
37 print('----Caesar predictions ----')
38 create_predictions_csv('Caesar', caesar_classifier2, caesar_sentences,
39                        caesar_word_bi_features)
40 print('----Hamlet predictions ----')
41 create_predictions_csv('Hamlet', hamlet_classifier2, hamlet_sentences,
42                        hamlet_word_bi_features)
```

See rows 13-32 where we check for a predicted value and compare it with positive, negative, or neutral. Correspondingly we make a row of data and append to the data-frame. As demonstrated earlier we then use the <dataframe>.to_csv() functionality to write the results.

And correspondingly, here are snapshots of the csv comprising data for both corpora:

	A	B	C	D	E
1	Sentence	Book	pos-sent	neg-sent	neutral-sent
2	Layest thou thy Leaden Mace vpon my Boy,	Caesar	0	1	0
3	Gentle knaue good night:	Caesar	0	1	0
4	Let me see, let me see; is not the Leafe turn'd downe	Caesar	0	1	0
5	Heere it is I thinke.	Caesar	0	1	0
6	Enter the Ghost of Caesar.	Caesar	0	0	1
7	How ill this Taper burnes.	Caesar	0	0	1
8	Ha!	Caesar	0	0	1
9	Who comes heere?	Caesar	0	1	0
10	I thinke it is the weakenesse of mine eyes	Caesar	0	1	0
11	It comes vpon me: Art thou any thing?	Caesar	0	1	0
12	Art thou some God, some Angell, or some Diuell,	Caesar	0	1	0
13	Speake to me, what thou art	Caesar	0	1	0
14	Thy euill Spirit Brutus?	Caesar	0	0	1
15	Bru.	Caesar	0	0	1
16	Why com'st thou?	Caesar	0	0	1
17	Ghost.	Caesar	0	0	1
18	To tell thee thou shalt see me at Philippi	Caesar	0	1	0
19	Well: then I shall see thee againe?	Caesar	0	1	0
20	Ghost.	Caesar	0	0	1
21	I, at Philippi	Caesar	0	1	0

321	So that with ease,	Hamlet	0	0	1
322	I will doo't.	Hamlet	0	0	1
323	And for that purpose Ile annoint my Sword:	Hamlet	0	1	0
324	Let's further thinke of this,	Hamlet	0	0	1
325	Enter Queene.	Hamlet	0	0	1
326	Queen.	Hamlet	0	0	1
327	One woe doth tread vpon anothers heele,	Hamlet	0	0	1
328	Drown'd!	Hamlet	0	0	1
329	O where?	Hamlet	0	0	1
330	Queen.	Hamlet	0	0	1
331	There is a Willow growes aslant a Brooke,	Hamlet	0	0	1
332	Alas then, is she drown'd?	Hamlet	0	0	1
333	Queen.	Hamlet	0	0	1
334	Drown'd, drown'd	Hamlet	0	0	1
335	Too much of water hast thou poore Ophelia,	Hamlet	0	1	0