**NLP Homework 3**

**Sentiment Analysis**

Student: Sharat Sripada (vssripad@syr.edu)

### 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:

**Preparing the Data Visualization Classification Analytics**

Plot top-50 adj, adv, verb and noun from pos/neg sentences

Tokenizing the corpora for sentences/words

Extract features using unigram/bi-gram word freq.

Plot counts of Pos/Neg/Neutral sentiments

Use Naïve Bayes classification to determine model accuracy

Using *textblob* to attach polarity to each sentence

Translate sentiment to Pos/ Neg/Neutral

Create a data-frame containing sent. & sentiment

To begin with, we will continue to extract the corpora as before by indexing books from the Gutenberg corpora:

Graphical user interface, text, application, email

Description automatically generated

### 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:

Graphical user interface, text

Description automatically generated

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

Text

Description automatically generated

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(polarity=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:

Graphical user interface, text, application

Description automatically generated

Output:

['Hence: home you idle Creatures, get you home:\nIs this a Holiday?', Sentiment(polarity=0.0, subjectivity=0.0), **'Neutral'**] *<- Notice the tag Nuetral which is extracted from 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.

Graphical user interface, text, application

Description automatically generated

Snapshot of the data-frame:

Graphical user interface

Description automatically generated with medium confidence

### 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:

Graphical user interface, text

Description automatically generated

Correspondingly, this is the bar-plot visualization:

Chart, bar chart

Description automatically generated Chart, bar chart

Description automatically generated

#### 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:

Text

Description automatically generated

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

### 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:

Chart, diagram

Description automatically generated

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:

Text

Description automatically generated

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.

Graphical user interface, text, application

Description automatically generated

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.

Graphical user interface, text, application

Description automatically generated

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'**)

1. 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.

Graphical user interface, text, application, email

Description automatically generated

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.

Graphical user interface, text

Description automatically generated

Example of the list comprehension caeser\_featureset2(trimmed for brevity):

({"contains(('I', 'haue'))": False, "contains(('I', 'know'))": False, "contains(('Cassi', 'I'))": False, "contains(('I', 'shall'))": False, "contains(('Bru', 'I'))": False, "contains(('Mark', 'Antony'))": False, "contains(('let', 'vs'))": False, "contains(('Marke', 'Antony'))": False, "contains(('And', 'I'))": False, "contains(('Lord', 'Bru'))": False, "contains(('I', 'feare'))": False, "contains(('Caesar', 'Caes'))": False, "contains(('Enter', 'Brutus'))": False, "contains(('For', 'I'))": False, "contains(('Noble', '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 influence 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():*

Graphical user interface, text

Description automatically generated

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.

### 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:

Text

Description automatically generated with medium confidence

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:

Text

Description automatically generated

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

Text

Description automatically generated with medium confidence

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

1. 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  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 | 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  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 | 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  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 | 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  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 | 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  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 | 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  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 | 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  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 | 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  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', 'gentle', 'further', 'wrong', 'Honourable', 'haue', 'ready', 'strong', 'worthy', 'mighty', 'seene', 'common', 'huge', 'happy'}

* Negative sentiment adjectives:

print(set(adj\_neg) - set(h\_adj\_neg))

{'better', 'safe', 'least', 'vile', 'bloody', 'sorry', 'feeble', 'constant', 'gentle', 'ordinary', 'pitty', 'terrible', 'secret', 'many', 'certaine', 'euery', 'deceiu', 'Honourable', 'worst', 'stubborne', 'angry', 'senslesse', 'borne', 'much', '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', 'Fathers', 'King', 'Ophelia', 'youth', 'loue', 'matter', 'Hor', 'Hamlet', 'Queene', 'Sonne', 'head', 'vp', 'Heauen', 'purpose', 'life', 'Mother', 'eyes', 'Denmarke', 'world', 'Father', 'Nature', 'Giue', 'Fortinbras', 'Rosin', 'Ham', 'Ile'}

*Caesar mostly certainly had references to Rome while Hamlet shows references to Denmark, possibly therefore 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', 'grow', '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 differences here. Nothing apparent.*

### 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:

A picture containing table

Description automatically generated

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:

Table

Description automatically generated

Table

Description automatically generated