IST-736 Text Mining **Dr. Gates**
Final Project **Arthur Fu, Brian Hogan, Olga Perera, Jeff Victor**

"To Mine, or Not to Mine"
Text Analysis of the Works of William Shakespeare

Introduction

William Shakespeare was a playwright and poet who lived from 1564-1616. He has been described as "the greatest writer in the English language and the world's greatest dramatist." His 39 plays include over 835,997 words and have been translated into every current major language.

What can we learn about his writing style using traditional machine learning tools? Did he use different word choices for his comedies than for his tragedies? Did his characters use different words because of differences in their upbringing or social class? This research attempts to answer questions similar to these.

Data and Methods

Data

Theatrical scripts have a well-known format: one or more acts, and scenes within acts. Each scene consists of a sequence of speeches ("lines") delivered by the actors playing the characters in that show.

Analysis of scripts can use any of those elements as a document: one line, all of the lines in a scene, or all of the lines in the script. Various other subsets are possible, including all of the lines in a script delivered by one character, all of the lines in a script delivered by women, and many others. The meaning of "document" depends on the analysis being performed.

Once that decision has been made, the meaning of labels is obvious. For example, if sentiment analysis is being performed to contrast different plays, a document is an entire play.

Data Acquisition and Cleaning

Access to modern scripts is limited by copyright laws and the copyright asserted by playwright and publisher. Scripts written long ago are free of copyright, simplifying access and release of contents of the script. Because of its age, one large corpus of scripts that is free of copyright is William Shakespeare's plays.

Many web sites contain Shakespeare's plays. One of the well-known repositories is at the Massachusetts Institute of Technology (MIT)[1]. They adhered strictly to one format for the web pages, greatly simplifying the process of parsing the web pages that contain the scripts.

Prior to the analysis described in this paper, the 37 scripts of Shakespeare's plays were downloaded. A Python program was written to parse the downloaded HTML code. This program has flexible output choices, creating either one document per script, or creating a CSV file including one line of text per speech, labeled with the character that speaks that line.

The following HTML code shows part of the structure used by MIT:

```
<A NAME=speech1><b>BERNARDO</b></a>
<blockquote>
<A NAME=1.1.1>Who's there?</A><br>
</blockquote>
```

Parsing this was further simplified by the use of the Python HTML parser named *html.parser*[2]. This parser requires the re-definition of methods that will perform actions when certain HTML tags are interpreted.

The speeches were further cleaned by removing punctuation and converting uppercase letters to lowercase.

When this program was used to label each line of dialogue, the output produced after parsing the HTML code shown above is:

```
BERNARDO,whos there
```

The clustering methods described below used unlabeled text. Other methods used labeled text.

Data Exploration

Descriptive analysis provides the first glimpse into Shakespeare's text data. The following table summarizes some basic statistics of the corpus.

[1] The website address is mit.shakespeare.edu

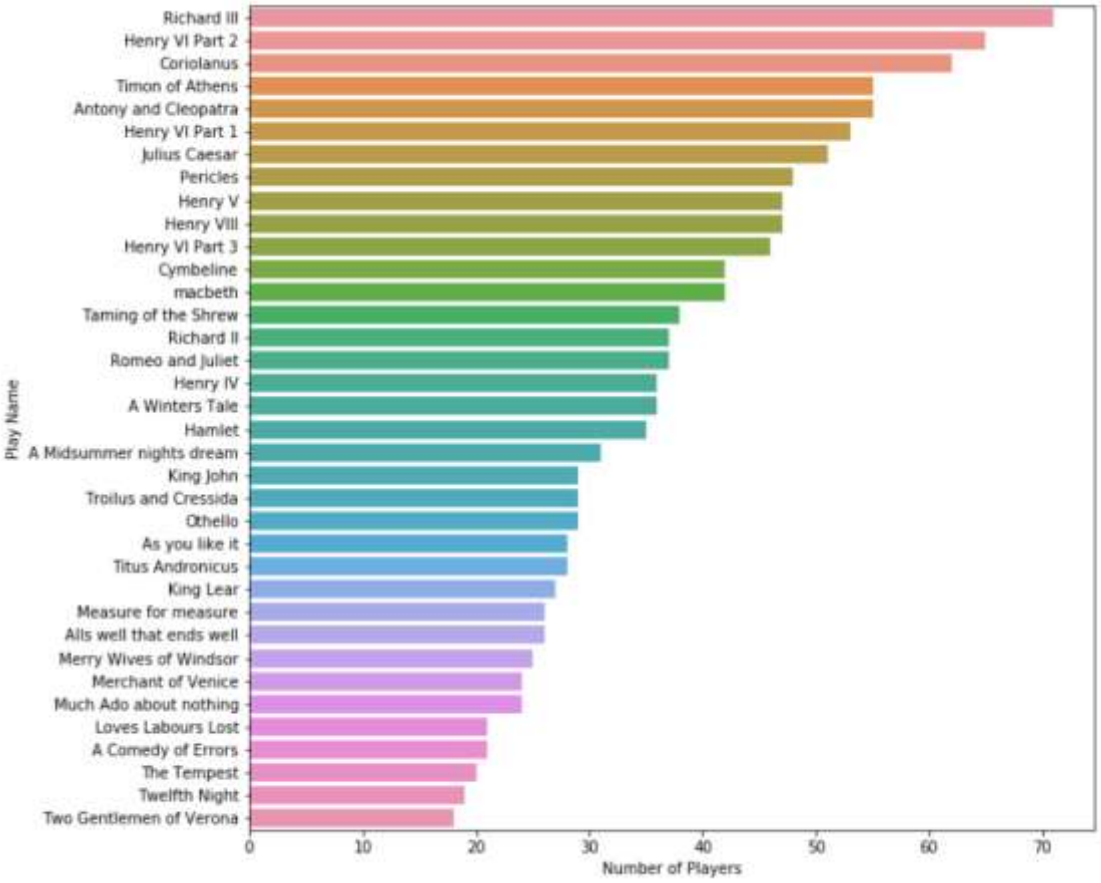
[2] This HTML parser is documented at http://docs.python.org/3/library/html.parser.html

Table 1. Basic statistics of the corpus

	Acts	Scenes	Characters	Lines	Words
All's Well That Ends Well	5	23	25	910	22496
As You Like It	5	22	27	783	21074
The Comedy of Errors	5	11	20	587	14495
Cymbeline	5	27	41	742	23494
Love's Labours Lost	5	9	21	1025	21056
Measure for Measure	5	17	26	871	21136
The Merchant of Venice	5	20	23	612	20853
The Merry Wives of Windsor	5	23	24	992	21042
A Midsummer Night's Dream	5	9	30	469	15961
Much Ado About Nothing	5	17	23	954	20758
Pericles, Prince of Tyre	5	21	47	589	17322
Taming of the Shrew	6	14	37	854	20400
The Tempest	5	9	19	620	15847
Troilus and Cressida	5	24	28	1108	25069
Twelfth Night	5	18	18	904	19293
Two Gentlemen of Verona	5	20	17	839	16828
Winter's Tale	5	15	35	710	24336
Henry IV, part 1	5	18	36	739	23831
Henry IV, part 2	5	19	47	796	25411
Henry V	5	23	46	693	25454
Henry VI, part 1	5	27	51	599	20512
Henry VI, part 2	5	24	65	724	24419
Henry VI, part 3	5	28	46	766	23286
Henry VIII	5	17	46	1358	22954
King John	5	16	28	651	20306
Richard II	5	19	36	513	21315
Richard III	5	25	70	990	27706
Antony and Cleopatra	5	40	54	1119	23637
Coriolanus	5	29	61	1043	26455
Hamlet	5	20	34	1103	29456
Julius Caesar	5	18	51	744	19060
King Lear	5	26	26	1041	25230
Macbeth	5	28	41	607	16296
Othello	5	15	28	1151	25695
Romeo and Juliet	5	24	35	716	21614
Timon of Athens	5	17	54	709	17283
Titus Andronicus	5	14	27	537	19704

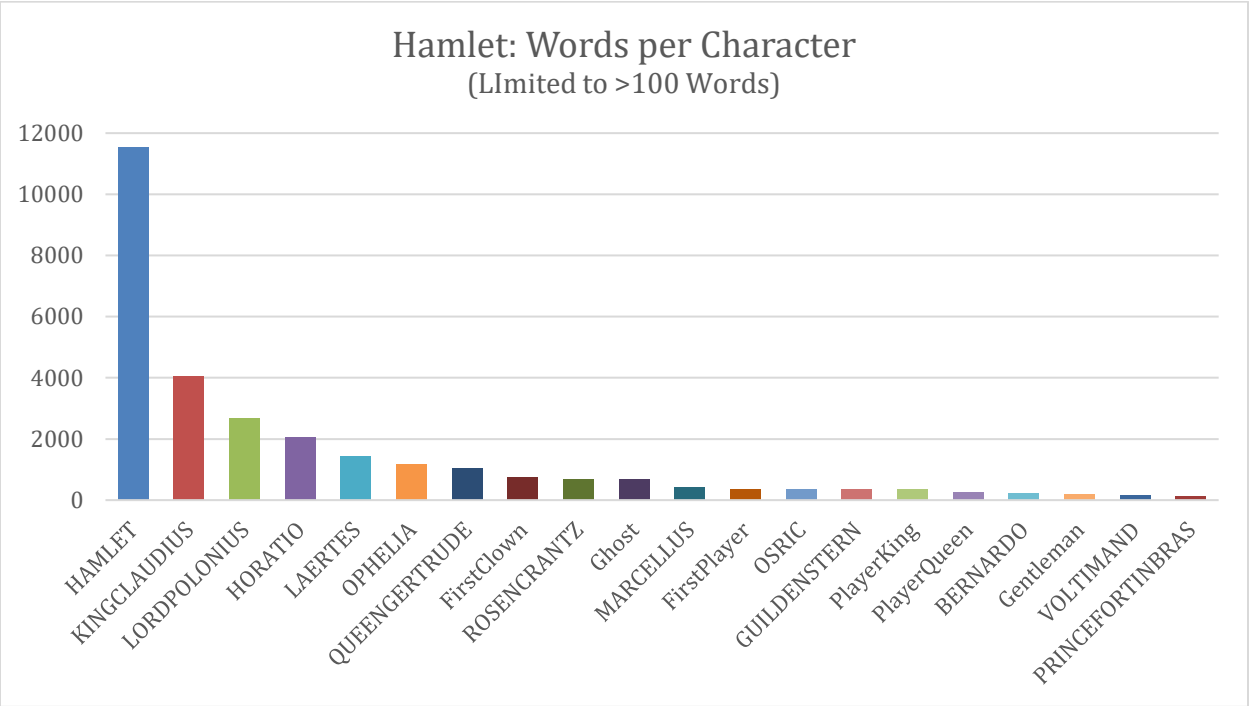
Below is a frequency distribution for number of players by each Shakespeare play. The number of players greatly varies from play to play with highest value in Richard III and lowest value in Two Gentleman of Verona.

Figure 1. Number of Players by Play



It seemed reasonable that the main characters of a play should be obvious form the amount of words that they deliver. Testing this highlighted the title character, but otherwise demonstrated the absence of a clear distinction between “main characters” and “other characters.”

Figure 2. Hamlet: Words per Character



Further, a word cloud shows the frequencies of the most common words in the text. “A word cloud is a popular visualization of words typically associated with Internet keywords and text data. They are most commonly used to highlight popular or trending terms based on frequency of use and prominence.”[3] Word frequency distribution and word clouds are the first steps to describing the text data.

Figure 3. Shakespeare Plays Word Cloud

[3] https://www.peachbeltconference.org/information/thePBCis/wordcloud

Method Overview and Options

Several commonly used analytical methods were used, including k-means clustering,

Clustering Analysis

Clustering analysis is a type of unsupervised learning that works by dividing data points into several groups with similar characteristics. Clustering analysis creates a group of objects based on similarity.

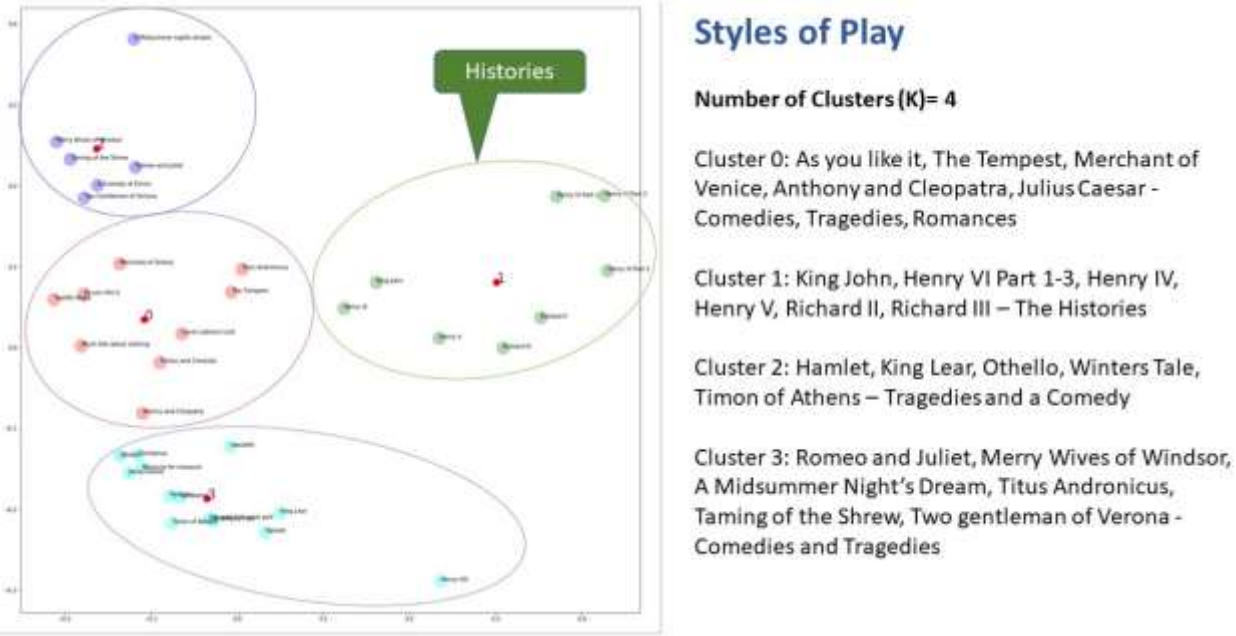
Popular clustering methods:

1. Partitioning method – partition objects into *k* number of clusters, where the distance is a major parameter. The key component is to be able to determine the number of clusters.
2. Hierarchical method – clusters are formed in a tree type of structure based of hierarchy, when new clusters are formed based on previous clusters. Flexible selection of numbers of clusters.

There are no good criteria or rules for the clustering analysis, and it depends on the data and analysis objectives. The *k*-means partitioning method groups data points into *k* clusters. Each cluster is represented by its centroid, which corresponds to the mean of the positions of data points assigned to each cluster. The number of clusters must be determined prior to the analysis and must be specified as a parameter.

Clustering analysis is applied in order to split Shakespeare's works into similar groups of plays. Traditionally Shakespeare's plays are divided into three, sometimes four categories: the comedies, the histories, the tragedies, and the romances. The Comedies have common elements as they usually involve lovers and story lines with positive endings. On the other hand, the Tragedies lead to ultimate destruction and downfall. The Tragedies and Comedies "have more similarities than differences and what they have in common is a recognizability that comes from their all being the work of the same writer whose genius makes him the best plot constructor, character-maker, story-teller and poet of his time." Shakespeare wrote ten history plays with central theme of gain or loss of power. Shakespeare put effort in describing what makes a good, wise, and successful ruler. The Romances are also called tragicomedies with a focus on both, and mostly refer to the writer's later works.

Figure 4. K-Means clustering visualization



K-means clustering model was able to clearly distinguish between The Histories (Cluster 1) and the rest of the plays (Clusters 0, 2, 3). However, there is no clear similarity was observed between tragedies, romances, and comedies.

Topic Modeling Analysis

Apart from clustering, topic modeling is another way to analyze a text corpus. A "topic" consists of a cluster of words that frequently occur together. Using contextual clues, topic models can connect words with similar meanings and distinguish between uses of words with multiple meanings. In this study, an LDA topic modeling algorithm was also used to distinguish comedy and tragedy.

Vectorization for the Comedy and Tragedy Works

In this task, the corpus used was the play mixture with both comedy and tragedy. First, *CountVectorizer* was imported from *sklearn.feature_extraction.text*. , then the parameters of the CountVectorizer were tuned and a series were created. For instances, with/without stop words, using unigram/ngram, and using binary or count frequency. The vector and the parameters used were shown in the upper panel of Figure 5, and part of the clean data frame which was ready for the future analysis, was shown in the lower panel of Figure 5.

Figure 5 The vector parameters and part of the clean data frame

```
vect = CountVectorizer(input="filename",
                       analyzer='word',
                       stop_words='english',
                       binary=False,
#                      the minimum document frequencies words/n-grams
#                      must have to be used as features.
#                       min_df=0.1, max_df=0.7,
                       max_features=1000,
#                       token_pattern='(?u)[a-zA-Z]+',
                      token_pattern='[a-zA-Z0-9]{3,}',
#                       tokenizer=LemmaTokenizer(),
                      #strip_accents = 'unicode',
                      ngram_range=(1,2),
                       lowercase=True
                       )
```

	aaron	achilles	act	act scene	action	...	yes	yield	yonder	young	youth
All's Well That Ends Well-C	0	0	10	4	2	...	8	1	2	24	12
Antony and Cleopatra-T	0	0	12	4	5	...	7	6	2	3	5
As You Like It-C	0	0	5	4	1	...	11	2	4	29	29
Comedy of Errors-C	0	0	5	4	0	...	3	2	3	0	1
Coriolanus-T	0	0	9	4	13	...	7	8	2	6	2
Cymbeline-C	0	0	13	4	8	...	6	9	0	11	15
Hamlet-T	0	0	19	4	10	...	5	2	2	17	16
Julius Caesar-T	0	0	7	4	2	...	6	3	3	9	0
King Lear-T	0	0	10	4	2	...	11	3	0	10	2
Love's Labour's Lost-C	0	1	5	4	2	...	7	3	1	5	6
Macbeth-T	0	0	9	4	1	...	2	4	0	13	1
Measure for Measure-C	0	0	11	4	5	...	13	6	2	6	9
Merchant of Venice-C	0	0	8	4	0	...	8	4	2	24	8
Midsummer Night's Dream-C	0	0	5	4	2	...	2	3	4	7	7
Much Ado About Nothing-C	0	0	5	4	2	...	6	2	2	11	7
Othello-T	0	0	16	4	7	...	8	1	1	10	5

Latent Dirichlet Allocation (LDA) Model

LDA is a widely used model for topic modeling. It is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. In this project, the *sklearn.decomposition* package was used to perform LDA. In the model, two topics were assumed to exist, and the number of iterations was limited to 10.

Decision Tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences. It is one way to display an algorithm that only contains conditional control statements. In this study, the decision tree algorithm was also used to distinguish comedy and tragedy.

The same corpus and vector in previous LDA model were used. First, the cleaned data matrix and the labels were split into training and valuation data set and labels, with the split point equaling 0.4. Then, the decision tree model, which is imported from *sklearn* package as well, was fit with training set and labels. The accuracy score and confusion table were then generated.

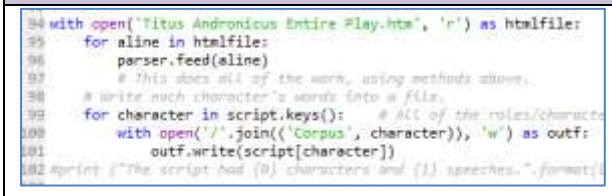
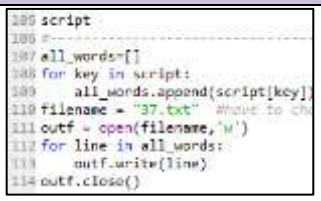
Much Ado About Stopwords and Document Term Matrix (DTM) Models

In the literary domain words both large and small have been found to have an important influence on machine learning (ML) prediction algorithms such as naïve Bayes (NB) and support vector machines (SVM). As data scientist continue to enhance their skill sets in text mining it can be commonplace to remove stopwords in a blanket fashion as tools, such as Python, are able to quickly read, transpose to data frames, and perform prediction models. According to Dr. Bei Yu (2008), sentiment classification with literature texts, such as early American novel writer Emily Dickenson or mid-century playwright Willian Shakespeare, should pay special attention to English language grammar structures as "a common word in one collection might not be common in another one" (p. 330).

This research knowledge led to the creation of two main modeling environments. The first focuses on an in-depth analysis of inclusion or exclusion of stop-words in Shakespeare's play "Much Ado About Nothing." The second model broadened the landscape by building a document term matrix (DTM) based on all plays and combining all character words to facilitate support vector machine (SVM) and decision tree (DT) prediction environments. The DTM environment didn't focus on stopword discrimination, as significant effort was focused on properly building training and testing data sets, hyperplane visualization, Ward's cosine linkage visualization, and successfully visualizing DT results.

As shown in Figure 6, two primary data mining methods were used to vectorize the data: building a character corpus by sentence where labels were character and sentiment were their play lines and building a document term matrix of all play words per play where the labels where the type of play. Below, on the left-hand the team's html parsing procedure building a character by sentence corpus. The right-hand procedure modified the *character+sentence* HTML parsing by concatenating each Python dictionary ***key***, i.e. sentence, combining it into a single text file building a document term matrix (DTM). The process read each play one by one as challenging to read.

Figure 6.

Corpus Creation Overview & Raw Corpus Size	
Character by Sentence Data Pull	Document Term Creation per Play Approach
"Much Ado About Nothing" Corpus	Shakespearean Corpus
55 characters by 6024 Words	37 Plays by 29,028 Words (vectorized)
	

```
In [422]: bbex_dh.shape
Out[422]: (55, 6024)
```

```
In [401]: corpusDF0.shape
Out[401]: (37, 29028)
```

Figure 7 illustrates corpus labeling for a detailed assessment of stopwords in a single play ("Much Ado About Nothing") was formed by ***grouping*** cast members into families called: main character, character family, and ensemble cast groupings. Nominal labels were used for the 55 cast members in "corpus_much_ado_play." Interestingly, when a k-means analysis was run to assess cluster distinctions for "corpus_much_ado_play" a cluster for "hamlet" emerged! This was a result of the author getting familiar with the *html.parser* code and some hamlet data was inadvertently stored in a corpus. In terms of the DTM corpus, labels for comedy, history, and tragedy were created for the 37 total plays with nominal values 1, 2 and 3.

Figure 7.

Corpus Labeling

Corpus Much Ado Play – 55 Characters

Label	Grouping	# Players
0	Ensemble Cast	39
1	Hamlet (mistake!)	1
2	Benedict – main character	1
3	Main character's family	6
4	Significant 2nd level cast	8

Corpus DTM

Play	Label	Count
comedy	1	17
history	2	10
tragedy	3	10
	Grand Total	37

Technical challenges resulted in labeling being updated manually by creating a list and applying to a label variable in the corpus_DTM. It should have been possible to associate values from the data frame but when CountVectorizer "filename" the dataframe index positions were not sequential. Sorting perhaps could have taken care of it but this was a quick fix to get correct DF.

Figure 8.

```
mylabels = []  #hand jam
mylabels= [1,1,1,1,1,1,1,1,1,2,2,1,2,2,2,2,2,2,2,2,3,3,1,3,3,3,3,3,3,3,1,1,1,1,1,1]
labels = pd.DataFrame(mylabels)
```

Model 1: Much Ado About Stopwords with a k-means Analysis

Referring to Figure 9, sentences were grouped by actor for 55 total actors. The right-hand side shows a numeric label for each role in the play, associated with all of the text spoken by that character. Data tables were created with and without stopwords; their removal resulted in a decrease of 2.8% or 168 stopwords from a corpus size of 6024 words. This led to a broader investigation of stopwords in Shakespearean works. From a direct inspection of 5 random pieces

of work the average stopwords didn't exceed 3%. This may speak to this work being a play versus a novel or literature but resonated with Bei Yu's research findings (Yu, 2008).

Figure 9.

Corpus Creation

Figure 9

The k-means clustering method was used due to the algorithm's ability to cluster data and attempt to visualize the inclusion or exclusion of stopwords. Experiments were performed varying clusters with five clusters delivering the best results.

In Figure 10, Corpus+Much+Ado shows a total of 282 words that were classified as NLTK "English" stopwords. However, fewer than 37 words had fewer than 3 characters. Their removal resulted in a testing corpus for normalization of 5742 words versus 6024 total words.

Figure 10.

Stop Words for Much Ado About Nothing Corpus

```
In [495]: print(stops)
['act', 'add', 'ado', 'age', 'ago', 'ah', 'aid', 'aim', 'air', 'all', 'am', 'an', 'and', 'ant',
'any', 'ape', 'apt', 'are', 'arm', 'art', 'as', 'ash', 'ass', 'at', 'ate', 'awe', 'axe', 'ay',
'aye', 'bad', 'ban', 'bar', 'bat', 'be', 'bed', 'beg', 'bet', 'bid', 'bow', 'box', 'boy', 'bud',
'but', 'buy', 'buz', 'by', 'byr', 'ca', 'cap', 'cat', 'cog', 'cow', 'coy', 'coz', 'cry', 'cue',
'cup', 'cut', 'daw', 'day', 'den', 'dew', 'did', 'die', 'dig', 'dip', 'do', 'dog', 'don', 'dot',
'dry', 'due', 'dug', 'dye', 'ear', 'eat', 'een', 'eer', 'ee', 'end', 'are', 'err', 'et', 'eve',
'eye', 'far', 'fat', 'fay', 'fed', 'fee', 'few', 'fie', 'fit', 'fly', 'foe', 'foh', 'for', 'fox',
'gae', 'get', 'gib', 'gis', 'go', 'god', 'got', 'gum', 'ha', 'had', 'hap', 'has', 'hat', 'he',
'hem', 'her', 'hes', 'hey', 'hi', 'hic', 'hid', 'hie', 'hip', 'his', 'hit', 'ho', 'hot', 'how',
'hue', 'hum', 'ica', 'if', 'ift', 'ild', 'ill', 'im', 'in', 'ink', 'ins', 'int', 'ic', 'ist', 'it'
'its', 'ive', 'jaw', 'jew', 'jig', 'jot', 'joy', 'key', 'kin', 'la', 'lap', 'law', 'lay', 'led',
'leg', 'let', 'lie', 'lo', 'lot', 'low', 'lug', 'mad', 'man', 'may', 'me', 'meg', 'men', 'met',
'mew', 'moe', 'mum', 'my', 'nay', 'net', 'new', 'no', 'non', 'nor', 'not', 'now', 'nut', 'oak',
'odd', 'oer', 'of', 'off', 'oft', 'old', 'on', 'one', 'ont', 'ope', 'or', 'orb', 'ore', 'our',
'out', 'owe', 'owl', 'own', 'pah', 'pat', 'pay', 'pen', 'pit', 'ply', 'pox', 'put', 'rat', 'raw',
'red', 'rid', 'rob', 'rod', 'rot', 'row', 'rub', 'rue', 'run', 'sad', 'sat', 'saw', 'say', 'se',
'sea', 'see', 'set', 'sex', 'she', 'sin', 'sir', 'sit', 'six', 'so', 'see', 'spy', 'sty', 'sum',
'sun', 'tax', 'ten', 'the', 'thy', 'tip', 'tis', 'to', 'toe', 'too', 'top', 'tot', 'toy', 'try',
'two', 'up', 'us', 'use', 'vex', 'vow', 'wag', 'war', 'was', 'wee', 'way', 'we', 'wed', 'who',
'why', 'wi', 'win', 'wit', 'woe', 'won', 'woo', 'wot', 'yaw', 'ye', 'yea', 'yes', 'yet', 'yon',
'you']
```

Stemming was tested and had a potential reduction of less than 5742 words. Referring to Figure 11, the word "breed" would have helped reduced the hyperplane but there were programmatic issues working correctly so this work was not performed. The results will reveal this was not an issue given literary corpus' nature and feature-weights generated in the model.

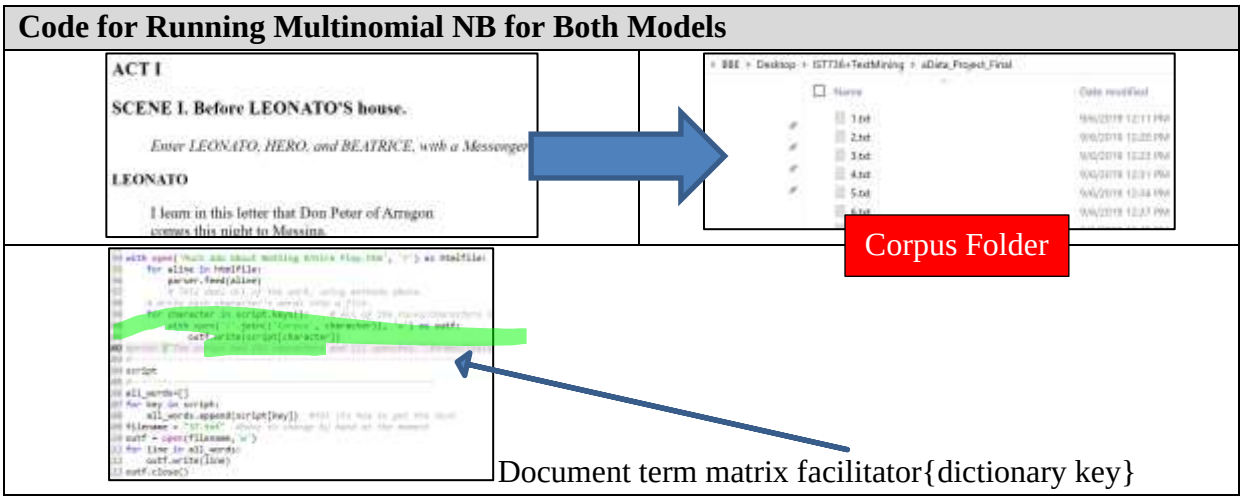
Figure 11.

```
'breed',
'breeding',
'breeds',
'breeder',
```

Model 2: Using SVM for Sentiment Prediction with a Shakespearean DTM

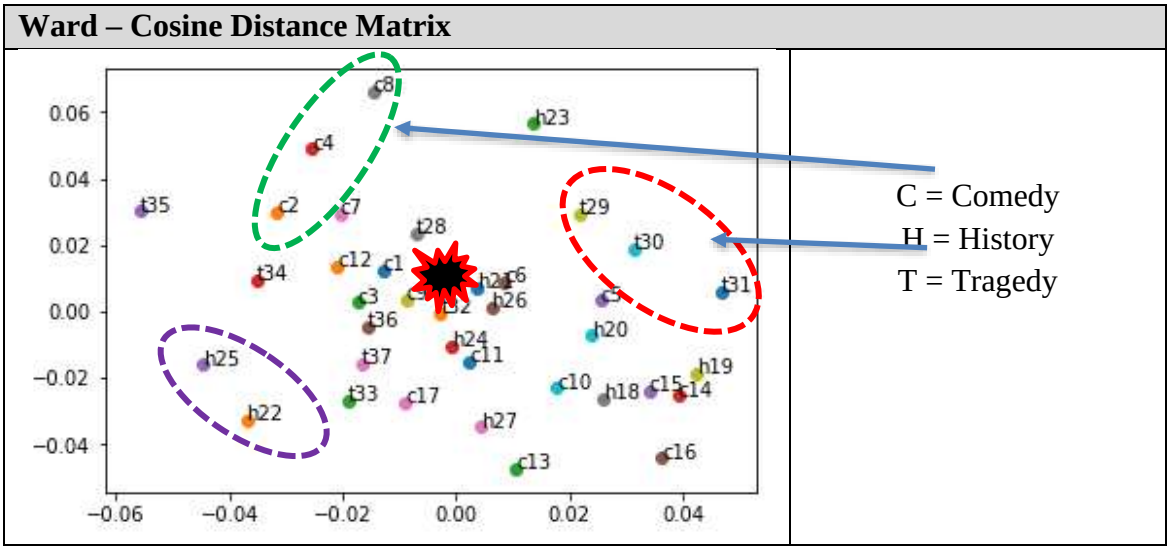
Once it was understood from one play that stopwords should not be removed, focus was shifted to expanding prediction options by building a Shakespearean document term matrix (DTM). Referring to Figure 12, all 37 plays were compiled into a single corpus for input into the model via the *sklearn.feature_extraction.text.CountVectorizer*. This was facilitated by creating a Python list for each play concatenating every character's play line dialogue. Import and data transformation methods were assisted with Dr. Gates coding strategies (Gates, 2019).

Figure 12.



Ward's clustering (Gates, 2019) was used to build a high-level understanding of the DTM's distance among the 37 plays. Using the red star in Figure 13 as a center point, it is interesting to see the proximity within the green, purple, and red circles representing comedy, history, and tragedy plays respectively. However, many other documents are mixed suggesting more complex hyperplane prediction methods, such as SVM, may be useful in predicting sentiment amongst play types.

Figure 13.



Referring to Figure 14, it is also interesting to have a primary grouping among four plays (purple dotted circle) of which three are from the different categories. Ward's cosine distance linkage matrix seems more appropriate for this data as an early indicator of predictors. This high-level overview provided an early indication of model feature weights that informed subsequent prediction model efforts. Referring to Figure 15 , Dr. Gates Python code for linkage matrix (Gates, 2019).

Figure 14.

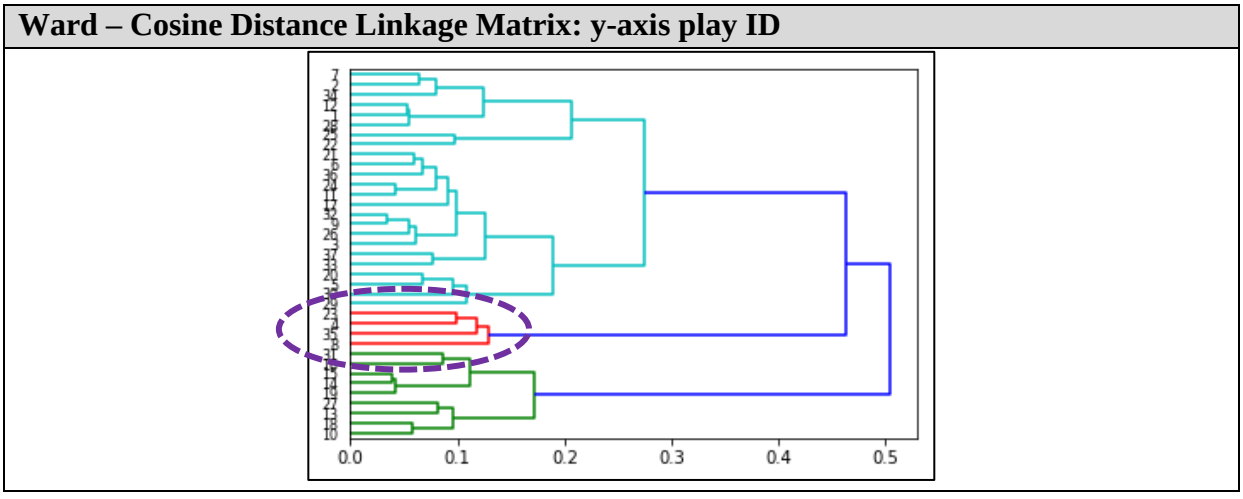


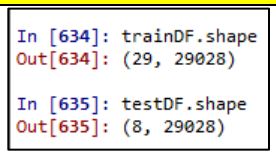
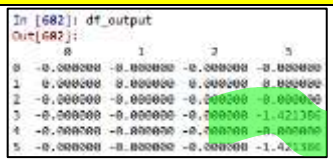
Figure 15.

Ward Linkage Code- Python – Dr. Gates

```
dtm.shape  #any vectorized data cube
dist = euclidean_distances(dtm)
print(np.round(dist,0))  #dist betweeen emall and prices is 3846
cosdist = 1 - cosine_similarity(dtm)
print(np.round(cosdist,3))
cosdist.shape
mds = MDS(n_components = 2, dissimilarity = "precomputed", random_state=1)
pos = mds.fit_transform(cosdist)
names = ['c1','c2','c3','c4','c5','c6','c7','c8','c9','c10','c11','c12','c1
xs, ys = (pos[:,0],pos[:,1])
for x, y, name in zip (xs, ys, names):
    plt.scatter(x,y)
    plt.text(x,y,name)
plt.show()
```

Referring to Figure 16, an SVM testing environment was created with the DTM using an 80/20 percent split to create training and testing dataframes. This was accomplished by taking the labeled, vectorized, TF-IDF normalized DTM and randomizing which plays went into a testing DTM dataframe. Of the 37 original plays, 5 comedies, 2 histories, and 2 tragedies were chosen for predicting sentiment with an SVM algorithm using a cross-fold validation factor of 10. The creating of this training and test environment also facilitated a decision tree analysis.

Figure 16.

Training & Test Dataframe	Normalized for SVM
	

Model 3: Applying Decision Tree to DTM

Referring to Figure 17, a decision tree environment was organized to help compare the ability of a DT to predict play sentiment versus SVM. It was initially thought that a narrow and deep tree was appropriate, but a wide and shallow tree provided better results.

Figure 17.

DTM Decision Tree Environment

```
1599 import numpy as np
1600 import pandas as pd
1601 from sklearn.cross_validation import train_test_split
1602 from sklearn.tree import DecisionTreeClassifier
1603 from sklearn.metrics import accuracy_score
1604 from sklearn import tree
1605
1606 clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,
1607                                    max_depth=3, min_samples_leaf=5)
1608 clf_gini.fit(X_train, y_train)
1609 clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,
1610 max_depth=3, min_samples_leaf=5)
1611 clf_entropy.fit(X_train, y_train)
1612 y_pred = clf_gini.predict(X_test)
1613 y_pred
1614 print "Accuracy is ", accuracy_score(y_test,y_pred)*100
```

Character Distinction using Naïve Bayes

A good author makes the characters in a story feel authentic in many ways. One method is tailoring a character's word choices. People make different word choices based on their conversational histories, built up through their lives. Authenticity can be improved by consistently choosing, for each character, from a distinct vocabulary.

For this paper, the phrase "character distinction" was created. It represents the distinctness of the vocabulary of a particular character, compared to the other characters in the story. Ideally, in addition to individual character distinction, characters with a similar background should have a similar vocabulary, one that is distinct from characters with a different background.

William Shakespeare is widely regarded as one the finest playwrights using the English language. Did he use character distinction? An answer to this question was sought using naive Bayes classifiers, assuming that word choice or term frequency would reveal the answer.

Because a naive Bayes classifier is a binary classifier, it cannot be used to distinguish between N characters in a play. Instead, pairwise classification for one character against other characters can be performed and combined. The results for each character can be compared to determine which, if any, of the characters exhibit character distinction.

Specifically, for each character, a naive Bayes classifier ingested labeled data to model word choice or word frequency. The model was tested with unlabeled data to quantify character distinction. In other words, the ability of a model to predict the character associated with a speech was used as a proxy for character distinction.

This part of the research used as input one file with one speech per line, labeled with the name of one of the two characters being compared. For each run, two characters were chosen: the

character being evaluated and another prominent character. At this point, additional data cleaning was performed: punctuation was removed, as were words with fewer than three letters. If stopword removal was chosen for the run, those words removed before further processing was performed.

Two naive Bayes models were generated for each run, and the performance of each model was tested. One model used a Multinomial Naive Bayes (MNB) classifier, the other used a Bernoulli Naive Bayes (BNB) classifier. For the MNB models, an sklearn TFIDF vectorizer was used to create a document term matrix. An *sklearn* boolean vectorizer was used for the BNB model. This means that word frequency is important to MNB, but BNB only considers whether a character ever used a particular word in dialogue. Ten-fold cross validation was used to generate consistent results.

Results

Clustering

K-Means clustering is a great option for finding similarities between Shakespeare's plays by genre. The number of clusters can be set to the number of play styles and the k-means method can be used to cluster plays by genre: the comedies, the histories, the tragedies, and the romances.

K-means clustering analysis was performed on all plays with only standard stop words removed during text pre-processing. The number of clusters was set to four as there are four known styles of play written by Shakespeare. As a result, the cluster #1 contained The Histories, while clusters #0, #2, #3 contained mixed genres. Such results point out to use of distinctive words in history plays, while tragedies, comedies, and romances did not produce any meaningful similarity between words.

The k-means clustering model was able to clearly distinguish between The Histories (Cluster 1) and the rest of the plays (Clusters 0, 2, 3). However, there is no clear similarity was observed between tragedies, romances, and comedies.

Topic Modeling

Two topics were generated via LDA models, and the top 100 high frequency words from both topic clusters were shown in Figure 3B. It is obvious the content in topic 1-word cloud is different from topic 2, however it is not easy to tell which one belongs to comedies and which one belongs to tragedies. In other words, this LDA model did not distinguish the comedies and tragedies very well. To further understand the model, the prediction results were listed along with the play's original category and summarized in the table shown in Figure 19A. The

mismatched predictions were labeled in red. The confusion table and the accuracy score were also shown in Figure 19B and 19C. A total of 11 out of 26 plays were mislabeled, and the accuracy score is only 57.7%.

Thus, according to the result we have generated so far, meaningful topics have not been clustered, suggesting LDA model is not a favorable model for distinguishing comedy plays from tragedies. Other models should be attempted.

In the decision tree models, the prediction accuracy score is only 0.6, which is a bit higher than the LDA model, but still not good enough to distinguish the two topics. As there are only 15 plays in the training set and 11 plays in the test set, the small size of the dataset may cause the lower prediction accuracy. Increasing the data size may increase the prediction accuracy, but this may not apply to the current study. As there are only 26 plays altogether and the number is fixed. Thus, we may conclude that the either LDA or decision tree may not able to distinguish the tragedy and comedy.

Figure 18 The corpus list and the word cloud for the two topics

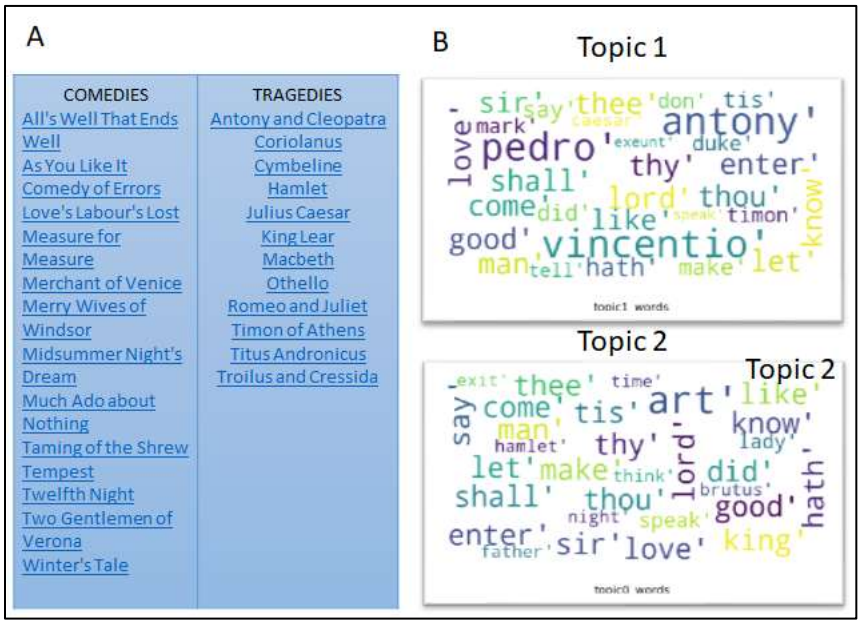
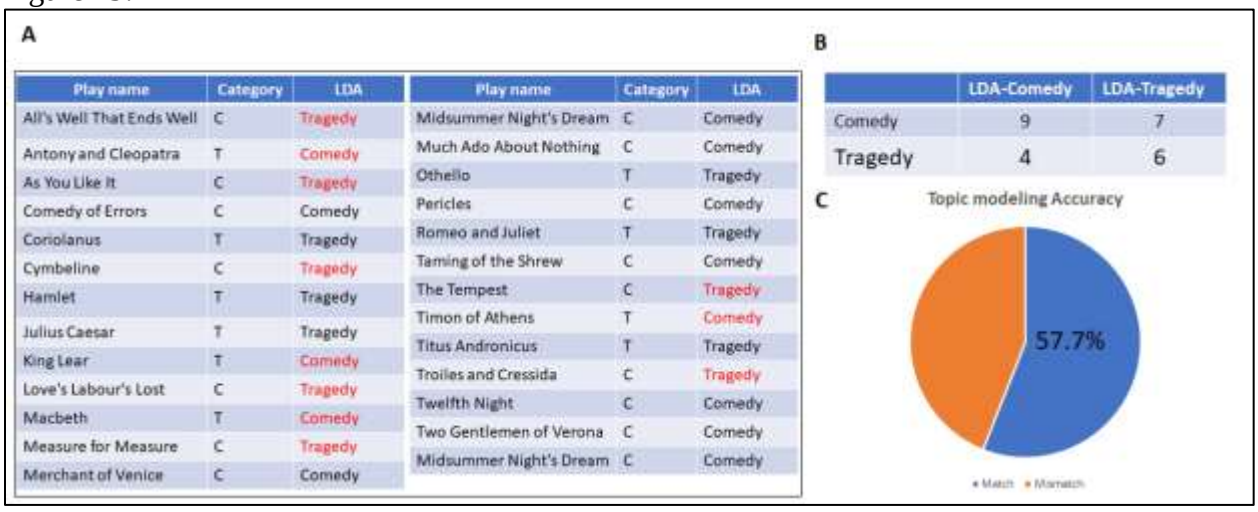


Figure 19.



Play name	Category	LDA
All's Well That Ends Well	C	Tragedy
Antony and Cleopatra	T	Comedy
As You Like It	C	Tragedy
Comedy of Errors	C	Comedy
Coriolanus	T	Tragedy
Cymbeline	C	Tragedy
Hamlet	T	Tragedy
Julius Caesar	T	Tragedy
King Lear	T	Comedy
Love's Labour's Lost	C	Tragedy
Macbeth	T	Comedy
Measure for Measure	C	Tragedy
Merchant of Venice	C	Comedy

Play name	Category	LDA
Midsummer Night's Dream	C	Comedy
Much Ado About Nothing	C	Comedy
Othello	T	Tragedy
Pericles	C	Comedy
Romeo and Juliet	T	Tragedy
Taming of the Shrew	C	Comedy
The Tempest	C	Tragedy
Timon of Athens	T	Comedy
Titus Andronicus	T	Tragedy
Troiles and Cressida	C	Tragedy
Twelfth Night	C	Comedy
Two Gentlemen of Verona	C	Comedy
Midsummer Night's Dream	C	Comedy

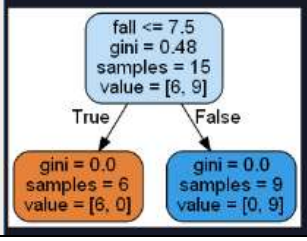
	LDA-Comedy	LDA-Tragedy
Comedy	9	7
Tragedy	4	6

Figure 19a.

Decision Tree Analysis of Comedy and Tragedy

```
#_____________

import pydotplus
from IPython.display import Image

vocabulary= feature_names
dot_data = tree.export_graphviz(clf,
                                feature_names=vocabulary,
                                out_file=None,
                                filled=True,
                                rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

```

```
In [31]: from IPython.display import Image

In [32]: Image(graph.create_png())
Out[32]:
```



Much Ado About Stopwords and Document Term Matrix (DTM) Models

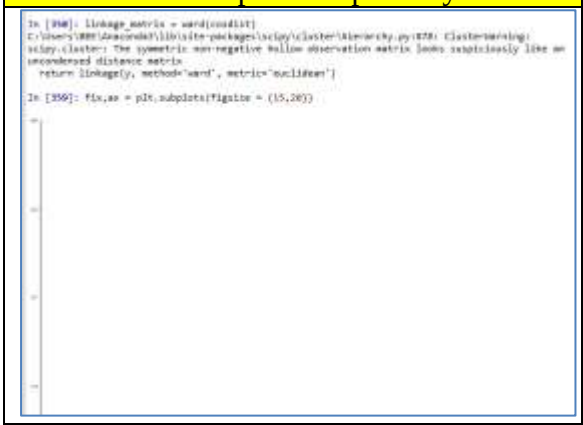
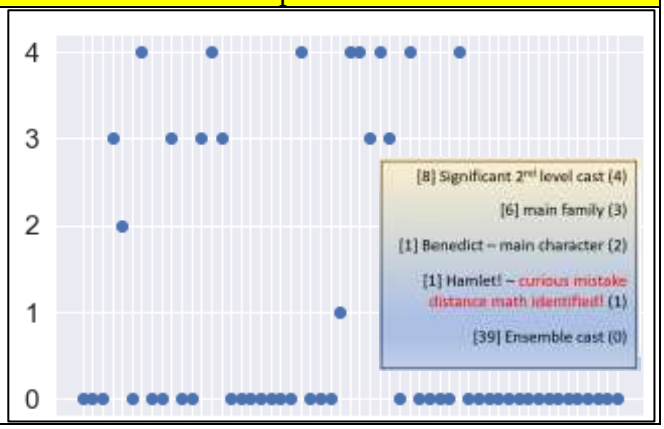
Results Model 1: Much Ado About Stopwords with a k-means Analysis

The data frame was exported as a CSV file. On the right-hand side of Figure 20 are the resulting clusters with no stopwords as desired graphics in Python were not achieved. Characters were analyzed and found five main groups were formed ranging from the play's main character through secondary and ensemble cast. Interestingly, the original data import included some lines

from Hamlet by pure coincidence. It was fascinating to see that Hamlet's word choices resulted in its own cluster so this mistake in the corpus was left for illustration purposes. K-means was run with and without stopword removal and player clustering was identical. Referring to Figure 17 cluster graphing in Python was challenging so a Pandas dataframe was exported to Excel for quick scatterplot graphing.

Figure 20.

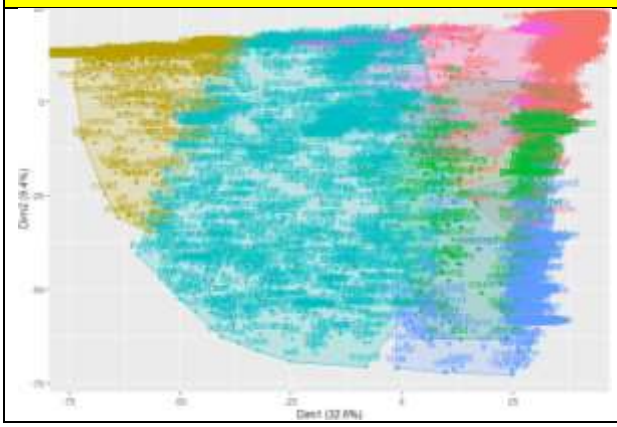
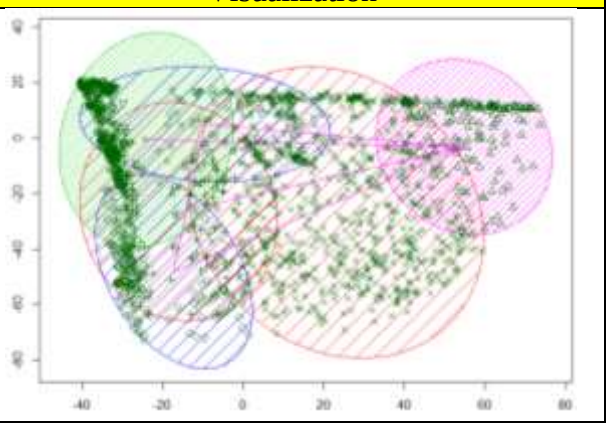
Initial Clustering Results from k-means

Failed Attempt to Graph in Python	Clustered Data Exported to Excel & Tabled
	

Based on the research from Dr. Yu, the author was intent on building two- and three-dimensional visualizations to understand the influence of stopword feature-weights on prediction outcomes. The author was not able successfully build visuals in Python and explored k-means graphing in an **R** language program. Figure 21 visualizes character clusters with lots of word data "noise." A hyperplane would have been better but was not able to be built.

Figure 21.

k-means Prediction Outcome Graphing in R as Could not Generate in Python

6 Cluster of Cast Word Dialogue	Alternative Tidyverse R k-means Visualization
	

Referring to Figure 22, k-means predicted the same grouping with or without stopwords even though different cluster IDs were generated. There was no difference with or without stopword inclusion. This speaks to the importance of investigating common and function words depending upon genre of source word information. It is interesting that k-means provided different outcome labels for predicted cast members. There were cross – referenced in Excel as was not able to complete all the Panda data frame coding correctly.

Figure 22.

k-means Predicted Same Grouping with and without Stopwords

All	ANTONIO	BALTHASAI	BERNARDO	Boy	Captain	CONRADE	Danes	FirstAmbas	FirstClown	FirstPlayer	FirstPriest	FirstSailor	FirstWatch	FRANCISCO	Gentlem
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
All	ANTONIO	BALTHASAI	BERNARDO	Boy	Captain	CONRADE	Danes	FirstAmbas	FirstClown	FirstPlayer	FirstPriest	FirstSailor	FirstWatch	FRANCISCO	Gentlem
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Ghost	GUILDENST	Lord	LUCIANUS	MARCELLU	MARGARET	Messenger	OSRIC	PlayerKing	PlayerQuee	PRINCEFOR	Prologue	QUEENGER	REYNALDO	ROSENCRA	SecondCl
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
Ghost	GUILDENST	Lord	LUCIANUS	MARCELLU	MARGARET	Messenger	OSRIC	PlayerKing	PlayerQuee	PRINCEFOR	Prologue	QUEENGER	REYNALDO	ROSENCRA	SecondCl
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

SecondWat	Servant	Sexton	URSULA	VERGES	VOLTIMAN	Watchman
1	1	1	1	1	1	1
SecondWat	Servant	Sexton	URSULA	VERGES	VOLTIMAN	Watchman
0	0	0	0	0	0	0

HAMLET	BENEDICK	BEATRICE	CLAUDIO	DOGBERRY	DONPEDRO	KINGCLAUI	LEONATO	BORACHIO	DONJOHN	FRIARFRAN	HERO	HORATIO	LAERTES	LORDPOLO	OPHELIA
2	4	0	0	0	0	0	0	3	3	3	3	3	3	3	
HAMLET	BENEDICK	BEATRICE	CLAUDIO	DOGBERRY	DONPEDRO	KINGCLAUI	LEONATO	BORACHIO	DONJOHN	FRIARFRAN	HERO	HORATIO	LAERTES	LORDPOLO	OPHELIA
1	2	3	3	3	3	3	3	4	4	4	4	4	4	4	

Results Model 2: Entire Literary Corpus as a DTM with SVM for Sentiment Prediction

Referring to Figure 23, a 62.5% accuracy score was achieved with the inclusion of stopwords in an SVM model. Three of eight plays were predicted incorrectly with one play wrong for each category. 3000 iterations were required to achieve model convergence. Three variations of SVM models were run altering cross fold validation, changing feature "penalty" to "I1" and switching random states. In all cases the prediction results were the same

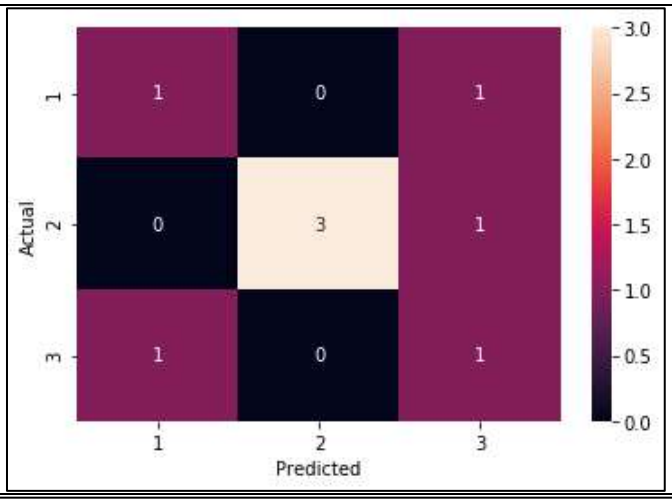
Figure 23.

DTM - SVM Model Outcomes

```
In [716]: svm_model.fit(trainDFn, mytrainlabels)
Out[716]:
LinearSVC(C=10, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=3000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)

In [717]: print("SVM prediction:\n", svm_model.predict(testDFn))
SVM prediction:
 [2 2 3 2 1 3 1 3]
```

Every category had 1 play predicted wrong

```
In [765]: print(df)
   y_actual  y_predict  play_id
0         2          2       21
1         2          2       22
2         2          3       23
3         2          2       24
4         3          1       32
5         3          3       33
6         1          1        8
7         1          3        9
```



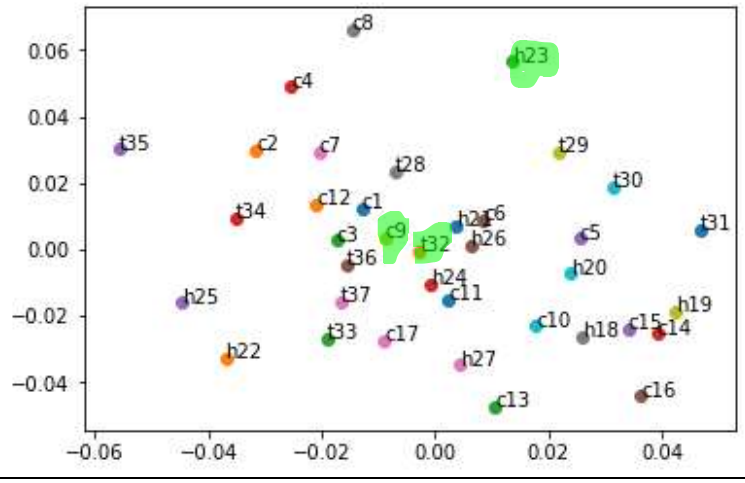

```
In [755]: accuracy_score(df['y_actual'], df['y_predict'])
Out[755]: 0.625
```

Referring to Figure 24, interestingly initial cosine distance of History play #23, “Henry VI, part 2” had an early indication of a difference in language. Further research analysis of specific features in this play would probably helped assess key feature differences as the play was not able to be classified by SVM correctly.

Figure 24.

DTM - SVM Model Outcomes

```
In [765]: print(df)
   y_actual  y_predict  play_id
0         2          2       21
1         2          2       22
2         2          3       23
3         2          2       24
4         3          1       32
5         3          3       33
6         1          1        8
7         1          3        9
```

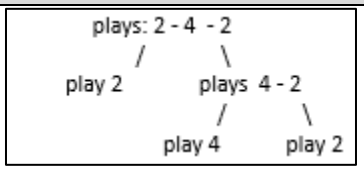


Results Model 3: Corpus DTM and Decision Tree

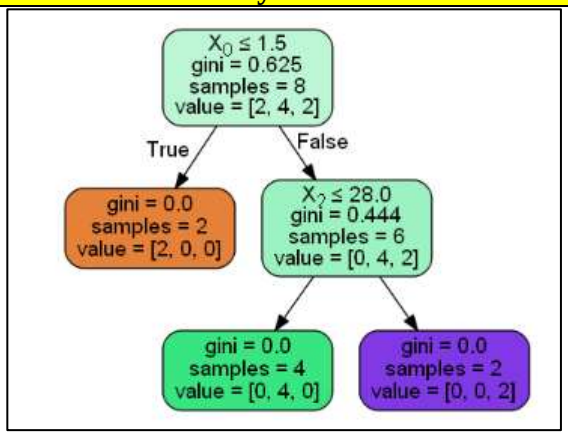
Decision tree analysis was focused heavily on the comedy plays as primary sentiment predictors. This outcome was completely unexpected and opposed results from the SVM. Referring to Figure 25, DT visualization was achieved with the package ***pydotplus***. Three variations of the DT models were run, 2 with gini and one with entropy. Max depth and min_leaf_samples were changed, and the outcomes were all the same.

Figure 25.

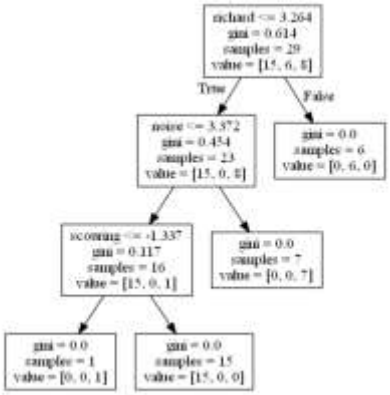
DTM - Decision Tree Prediction Outcomes



Play Level



Decision Tree with Words as Features



```
1056 import pydotplus as pydot
1057 d9f=pd.DataFrame(trainDFn, mytrainlabels) #your training and labels
1058 dtree=DecisionTreeClassifier()
1059 dtree.fit(d9f,y)
1060 #-----------------------------graphing code
1061 dot_data = StringIO()
1062 export_graphviz(dtree, out_file=dot_data,
1063                 filled=True, rounded=True,
1064                 special_characters=True)
1065 #'-this does not always building in python
1066     #graph = pydotplus.graph_from_dot_data(dot_data.getvalue()) #this now wor
1067     #Image(graph.create_png())
1068 #-----------------------------------------------workaround
1069 #this code creates a .dot file
1070     ########################################################################
1071 tree.export_graphviz(dtree, out_file=dot_data, feature_names=DT_vocab)
1072 graph = pydot.graph_from_dot_data(dot_data.getvalue())
1073 Image(graph.create_png())
1074 tree.export_graphviz(dtree, out_file='treepic.dot', feature_names=DT_vocab)
```

```
Select Anaconda Prompt

(base) C:\Users\BBE>conda install -c conda-forge pydotplus
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done
```

Character Distinction using Naïve Bayes

As described above, character distinction was quantified as the ability to predict a character's lines. The averaged results were tabulated, as shown in Table 2:

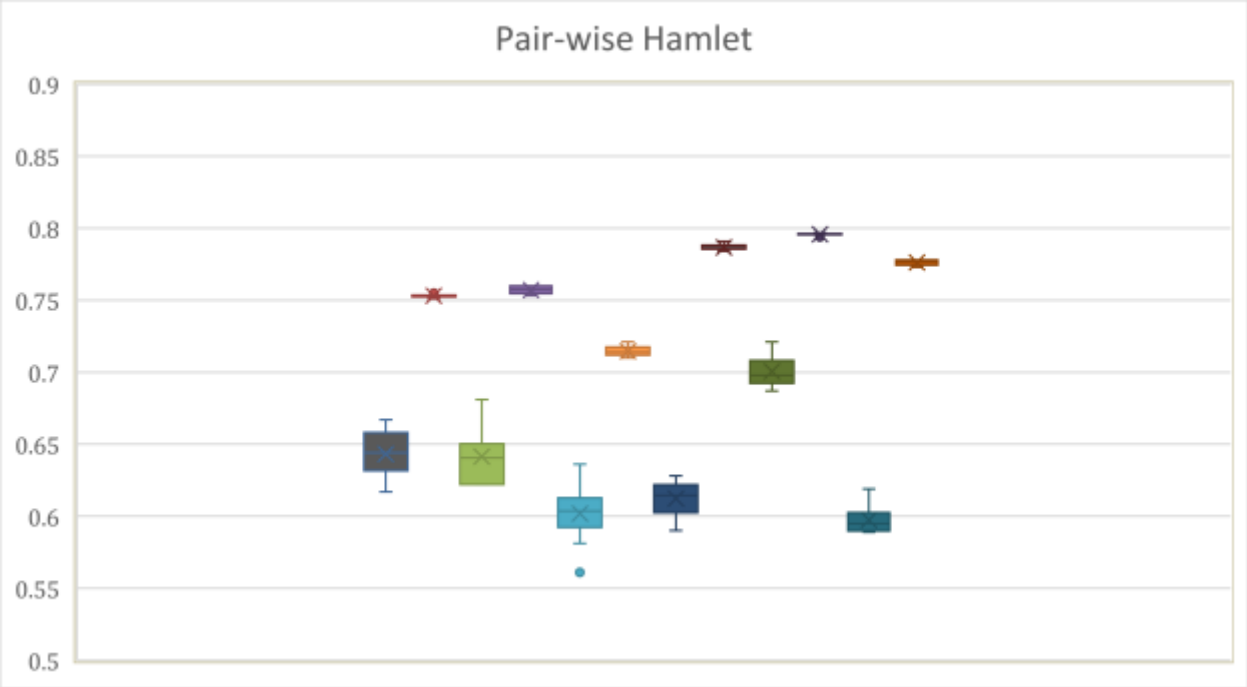
Table 2

MNB	Hamlet	King Claudius	Lord Polonius	Horatio	Laertes	Ophelia	Queen Gertrude		Average
Hamlet		0.643	0.641	0.602	0.612	0.701	0.597		0.633
King Claudius	0.643		0.653	0.704	0.629	0.704	0.588		0.654
Lord Polonius	0.641	0.653		0.567	0.590	0.588	0.613		0.609
Horatio	0.602	0.704	0.567		0.635	0.633	0.620		0.627
Laertes	0.612	0.629	0.590	0.635		0.633	0.501		0.600
Ophelia	0.701	0.704	0.588	0.633	0.633		0.615		0.646
Queen Gertrude	0.597	0.588	0.613	0.620	0.501	0.615			0.589

BNB	Hamlet	King Claudius	Lord Polonius	Horatio	Laertes	Ophelia	Queen Gertrude		Average
Hamlet		0.753	0.757	0.715	0.787	0.796	0.776		0.764
King Claudius	0.753		0.656	0.640	0.611	0.739	0.582		0.664
Lord Polonius	0.757	0.656		0.640	0.662	0.517	0.606		0.640
Horatio	0.715	0.640	0.640		0.660	0.683	0.639		0.663
Laertes	0.787	0.611	0.662	0.660		0.618	0.581		0.653
Ophelia	0.796	0.739	0.517	0.683	0.618		0.658		0.669
Queen Gertrude	0.776	0.582	0.606	0.639	0.581	0.658			0.640

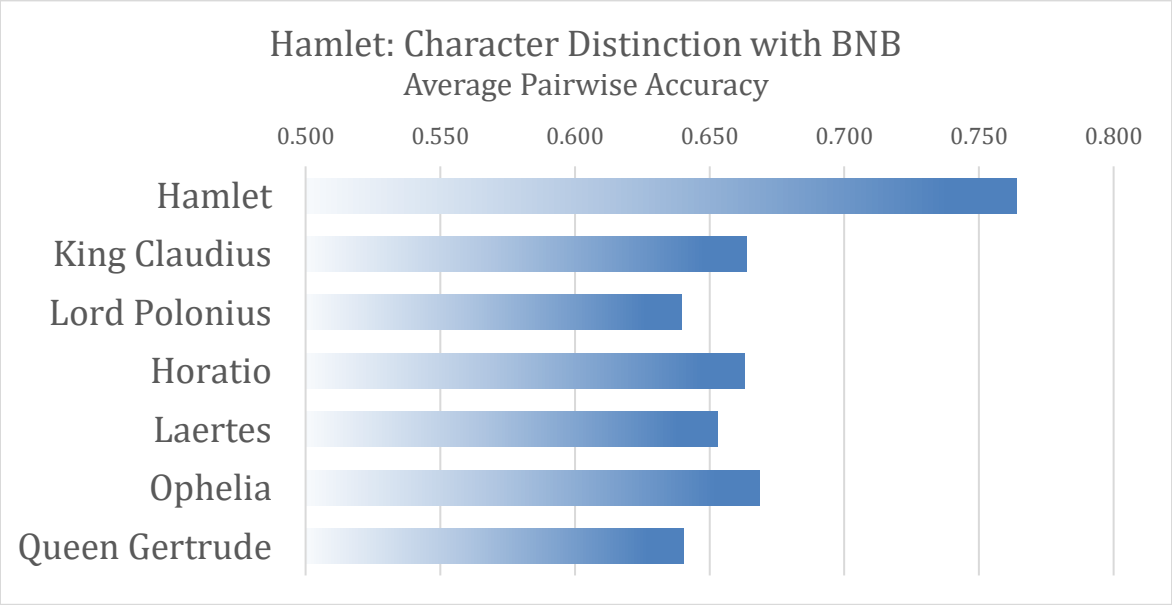
The MNB model did not show much distinction. However, the BNB model showed a clear distinction for the main character, Hamlet. Figure 26 shows performance results for both MNB and BNB models. The upper six boxes show the BNB results, and the lower six boxes show MNB results. The plot indicates the superior performance of BNB for this situation, and also shows that the BNB model produced more consistent results.

Figure 26.



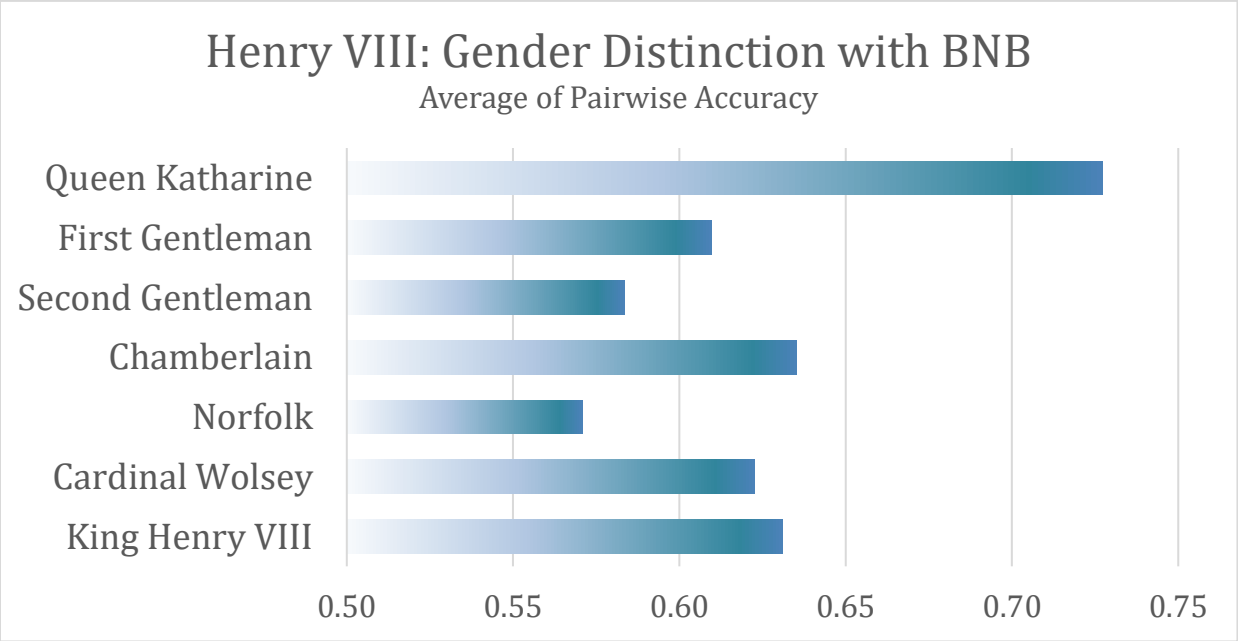
The other characters did not show much distinction. This is easier to see in Figure 27:

Figure 27.



A similar process was used for the play Henry VIII. Instead of showing a distinct vocabulary for the main character, it showed that the one female lead character used different words, as shown in Figure 28. This gender distinction seems appropriate.

Figure 28.



Conclusions

Authors who write dialogue do so with more than spelling, grammar and plot in mind. They aim to convey emotion, display human characteristics, and to distinguish between characters – all with vocabulary. In some cases, such as the Shakespeare corpus studied for this research, the text is easy to gather.

Software tools exist to perform simple analysis, demonstrating quantities of plays, characters, and, in the case of Shakespeare's plays, a total of almost 900,000 words. More sophisticated tools were used for more complex analysis.

Multiple tools demonstrated that Shakespeare wrote somewhat differently for his Histories than for his Comedies and Tragedies. Somewhat surprisingly, the language used for Comedies and for Tragedies was not significantly different. Apparently, mood is more subtle than word choice.

If Shakespeare attempted to distinguish between his individual characters with their vocabularies, he did so rarely. In some cases, the word choices of a lead character were different from the other characters, but they were not different from each other. In other cases, the men

used different sets of words than the women, but it was not possible to distinguish characters within each group.

Techniques that ignored the most common words did not influence the ability to distinguish between genre, topics, or characters.

By and large, it seems that Shakespeare's characters spoke with one voice.

References

"Shakespeare's Play Types"
https://www.nosweatshakespeare.com/shakespeares-plays/play-types/

"Word Cloud Definition"
https://www.peachbeltconference.org/information/thePBCis/wordcloud

Gates, A., 2019. Python and Text Mining Lecture Notes. Syracuse University, IST736.

Yu, B., (2008). An evaluation of text classification methods for literary study. Literary and Linguistic Computing 23(3): 327-343. Retrieved from: doi:10.1093/IIC/fqn015