ShakespeareAnalysis

September 21, 2020

0.1 Goal

The goal of this notebook is to develop a model that predicts the player based on features given.

0.2 Results

I used word trimming functions in the `Shakespeare_WordTrimming` notebook to get the important words from each PlayerLine. I then used the `textblob` function to assign a sentiment rating for the line (if the line was positive or negative). I then used the Play, Act, Scene, Line and polarity to determine who the player was. The models I used were Decision Trees and Random Forest. They both gave around a 70% accuracy for determing the Player.

[1]:
```
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn import tree
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from nltk.probability import FreqDist
import nltk
from sklearn.preprocessing import LabelEncoder
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from textblob import TextBlob
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

[2]:
```
data = pd.read_csv('~/Documents/EECS/EECS_731/HW/EECS731_2/data/
↪PlayerLine_trimmed.csv')
```

[3]:
```
data
```

[3]:

	Unnamed: 0	Play	PlayerLinenumber	ActSceneLine	\
0	3	Henry IV	1.0	1.1.1	
1	4	Henry IV	1.0	1.1.2	
2	5	Henry IV	1.0	1.1.3	
3	6	Henry IV	1.0	1.1.4	

```
4                7        Henry IV                1.0        1.1.5
…              …              …                  …          …
104871      111390  A Winters Tale               38.0      5.3.179
104872      111391  A Winters Tale               38.0      5.3.180
104873      111392  A Winters Tale               38.0      5.3.181
104874      111393  A Winters Tale               38.0      5.3.182
104875      111394  A Winters Tale               38.0      5.3.183

                Player                                          PlayerLine
0       KING HENRY IV                          ['shaken', 'wan', 'care']
1       KING HENRY IV      ['find', 'time', 'frighted', 'peace', 'pant']
2       KING HENRY IV  ['breathe', 'shortwinded', 'accents', 'new', '…
3       KING HENRY IV         ['commenced', 'strands', 'afar', 'remote']
4       KING HENRY IV                    ['thirsty', 'entrance', 'soil']
…                 …                                                 …
104871        LEONTES     ['trothplight', 'daughter', 'good', 'paulina']
104872        LEONTES        ['lead', 'us', 'hence', 'may', 'leisurely']
104873        LEONTES                ['one', 'demand', 'answer', 'part']
104874        LEONTES  ['performd', 'wide', 'gap', 'time', 'since', '…
104875        LEONTES           ['disseverd', 'hastily', 'lead', 'away']

[104876 rows x 6 columns]
```

Somewhere, the unnamed 0 column was generated. We don't need this.

[4]:
```
del data['Unnamed: 0']
```

Encoding Play and Player We need to encode the play and player as numbers if we are going to reason about them. I used the LabelEncoder() function to do this.

[5]:
```
le = LabelEncoder()
le.fit(data['Player'])
data['Player'] = le.transform(data['Player'])
```

[6]:
```
le.fit(data['Play'])
data['Play'] = le.transform(data['Play'])
data
```

[6]:
```
        Play  PlayerLinenumber ActSceneLine  Player  \
0          9               1.0        1.1.1     457
1          9               1.0        1.1.2     457
2          9               1.0        1.1.3     457
3          9               1.0        1.1.4     457
4          9               1.0        1.1.5     457
…      …               …          …      …
104871     2              38.0      5.3.179     494
104872     2              38.0      5.3.180     494
```

```
104873     2              38.0      5.3.181     494
104874     2              38.0      5.3.182     494
104875     2              38.0      5.3.183     494

                                                PlayerLine
0                                ['shaken', 'wan', 'care']
1             ['find', 'time', 'frighted', 'peace', 'pant']
2       ['breathe', 'shortwinded', 'accents', 'new', '…
3              ['commenced', 'strands', 'afar', 'remote']
4                          ['thirsty', 'entrance', 'soil']
…                                                     …
104871     ['trothplight', 'daughter', 'good', 'paulina']
104872        ['lead', 'us', 'hence', 'may', 'leisurely']
104873                ['one', 'demand', 'answer', 'part']
104874  ['performd', 'wide', 'gap', 'time', 'since', '…
104875           ['disseverd', 'hastily', 'lead', 'away']

[104876 rows x 5 columns]
```

Seperating Features Here, I sepearate the ActSceneLine into separate columns for Act Scene Line. Then, in the model, I could reference the act, scene, and line individually.

[7]:
```
data['ActSceneLine'] = data['ActSceneLine'].astype(str)
actsceneline = data.ActSceneLine.str.split(pat='.', n=-1, expand=True)
data['Act'] = actsceneline[0]
data['Scene'] = actsceneline[1]
data['Line'] = actsceneline[2]

del data['ActSceneLine']

data
```

[7]:
```
         Play  PlayerLinenumber  Player  \
0           9               1.0     457
1           9               1.0     457
2           9               1.0     457
3           9               1.0     457
4           9               1.0     457
…       …                 …       …
104871      2              38.0     494
104872      2              38.0     494
104873      2              38.0     494
104874      2              38.0     494
104875      2              38.0     494

                                                PlayerLine Act Scene Line
0                                ['shaken', 'wan', 'care']   1     1    1
```

```
1           ['find', 'time', 'frighted', 'peace', 'pant']   1     1    2
2       ['breathe', 'shortwinded', 'accents', 'new', '…   1     1    3
3             ['commenced', 'strands', 'afar', 'remote']   1     1    4
4                         ['thirsty', 'entrance', 'soil']   1     1    5
…                                                     …  ..    …  …
104871     ['trothplight', 'daughter', 'good', 'paulina']   5     3  179
104872        ['lead', 'us', 'hence', 'may', 'leisurely']   5     3  180
104873               ['one', 'demand', 'answer', 'part']   5     3  181
104874  ['performd', 'wide', 'gap', 'time', 'since', '…   5     3  182
104875          ['disseverd', 'hastily', 'lead', 'away']   5     3  183

[104876 rows x 7 columns]
```

Turning Text into data I used the steps found at : https://sanjayasubedi.com.np/nlp/nlp-feature-extraction/ to binary encode the set of player lines. This assignes number to the words for feature engineering. It's output can be seen in the table below were it appears to use a one-hot encoding to represent which words are used in each line.

[8]:
```
vocab = sorted(set(word for sentence in data['PlayerLine'] for word in sentence.
↪split()))
# print(len(vocab), vocab)
```

[9]:
```
def binary_tranform(text):
    output = np.zeros(len(vocab))
    words = set(text.split())
    for i,v in enumerate(vocab):
        output[i] = v in words
    return output
```

[10]:
```
vec = CountVectorizer(binary=True)
vec.fit(data['PlayerLine'])
# print([w for w in sorted(vec.vocabulary_.keys())])
```

[10]:
```
CountVectorizer(binary=True)
```

[11]:
```
data_playerLine = pd.DataFrame(vec.transform(data['PlayerLine']).toarray(),␣
↪columns=sorted(vec.vocabulary_.keys()))
data_playerLine
```

[11]:
```
        10  2d  2s  4d  5s  6d  8d  aaron  aarons  abaissiez  …  zenelophon  \
0        0   0   0   0   0   0   0      0       0          0  …           0
1        0   0   0   0   0   0   0      0       0          0  …           0
2        0   0   0   0   0   0   0      0       0          0  …           0
3        0   0   0   0   0   0   0      0       0          0  …           0
4        0   0   0   0   0   0   0      0       0          0  …           0
…      ..  ..  ..  ..  ..  ..  ..    …       …          …  …         …
104871   0   0   0   0   0   0   0      0       0          0  …           0
```

```
104872   0   0   0   0   0   0   0      0       0          0  …            0
104873   0   0   0   0   0   0   0      0       0          0  …            0
104874   0   0   0   0   0   0   0      0       0          0  …            0
104875   0   0   0   0   0   0   0      0       0          0  …            0

        zenith  zephyrs  zir  zo  zodiac  zodiacs  zone  zounds  zwaggered
0            0        0    0   0       0        0     0       0          0
1            0        0    0   0       0        0     0       0          0
2            0        0    0   0       0        0     0       0          0
3            0        0    0   0       0        0     0       0          0
4            0        0    0   0       0        0     0       0          0
…          …      …  …  ..     …       …   …     …       …
104871       0        0    0   0       0        0     0       0          0
104872       0        0    0   0       0        0     0       0          0
104873       0        0    0   0       0        0     0       0          0
104874       0        0    0   0       0        0     0       0          0
104875       0        0    0   0       0        0     0       0          0

[104876 rows x 27071 columns]
```

[12]:
```
fdist = FreqDist(data['PlayerLine'])
fdist.most_common(9)
```

[12]:
```
[('[]', 699),
 ("['lord']", 155),
 ("['sir']", 82),
 ("['know']", 50),
 ("['ay']", 40),
 ("['well']", 39),
 ("['say']", 39),
 ("['madam']", 38),
 ("['whats', 'matter']", 31)]
```

Finding frequency distribution in words From the previous encoding, I could then look at the frequency distrubution in the words.

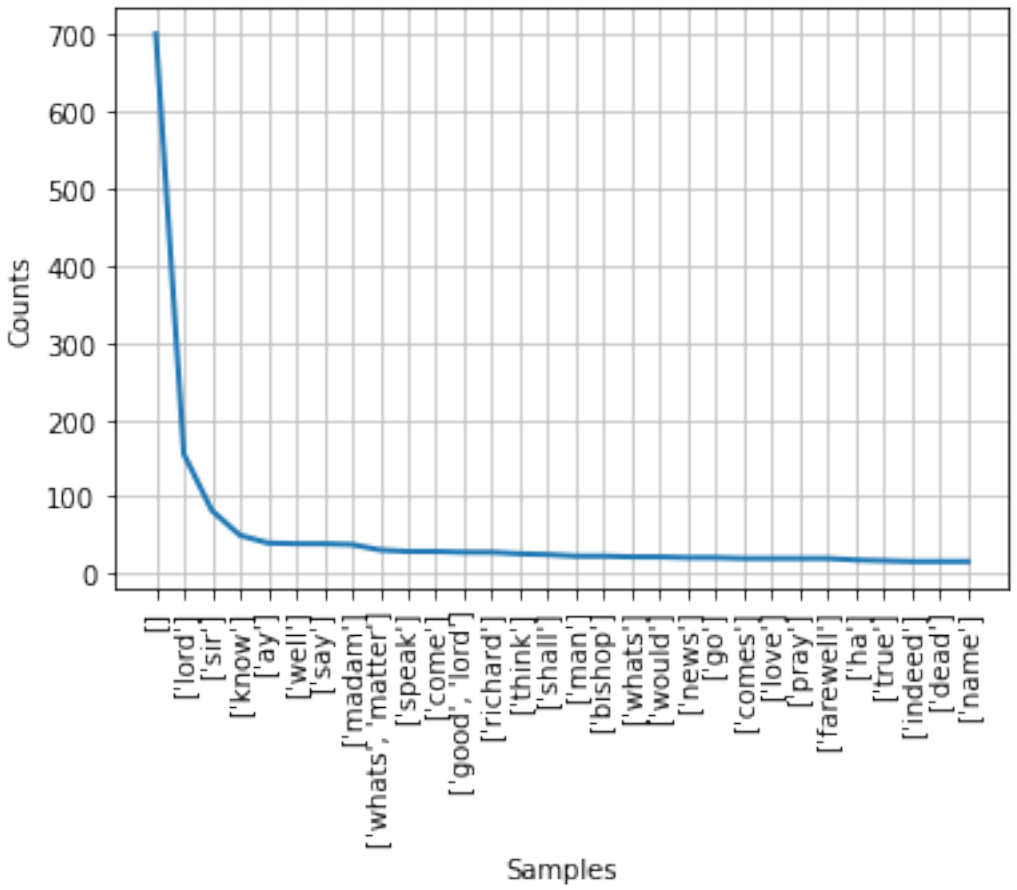
I took the steps to do this from https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk

[13]:
```
from nltk.probability import FreqDist
fdist = FreqDist(data['PlayerLine'])
print(fdist)
```

```
<FreqDist with 100157 samples and 104876 outcomes>
```

[14]:
```
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
```

```
plt.show()
```



0.2.1 Findings

From this plot, I can see that "lord" and "sir" are the most frequently used words. At one point, I thought there was something I could do with this information, but I couldn't find a way to use it. I tried to use this for feature engineering but it didn't work like I thought. I enjoyed learning about frequency distribution so I kept it in the report.

[15]:
```
from nltk import classify
from nltk import NaiveBayesClassifier
```

0.2.2 Labeling PlayerLine as positive or negative

Here, I used `textblob` to determine the polarity of a PlayerLine. The closer to -1 a line is, the more negitive it is. The closer to 1 the polarity is, the more positive the line is.

[16]:
```
data[['polarity', 'subjectivity']] = data['PlayerLine'].apply(lambda Text: pd.
↪Series(TextBlob(Text).sentiment))
```

[17]:
```
data
```

[17]:
```
         Play  PlayerLinenumber  Player  \
0           9               1.0     457
1           9               1.0     457
2           9               1.0     457
3           9               1.0     457
4           9               1.0     457
…       …                 …       …
104871      2              38.0     494
104872      2              38.0     494
104873      2              38.0     494
104874      2              38.0     494
104875      2              38.0     494

                                                PlayerLine Act Scene Line  \
0                                ['shaken', 'wan', 'care']   1     1    1
1             ['find', 'time', 'frighted', 'peace', 'pant']   1     1    2
2         ['breathe', 'shortwinded', 'accents', 'new', '…   1     1    3
3                ['commenced', 'strands', 'afar', 'remote']   1     1    4
4                           ['thirsty', 'entrance', 'soil']   1     1    5
…                                                       …  ..   …  …
104871      ['trothplight', 'daughter', 'good', 'paulina']   5     3  179
104872         ['lead', 'us', 'hence', 'may', 'leisurely']   5     3  180
104873                  ['one', 'demand', 'answer', 'part']   5     3  181
104874  ['performd', 'wide', 'gap', 'time', 'since', '…   5     3  182
104875            ['disseverd', 'hastily', 'lead', 'away']   5     3  183

         polarity  subjectivity
0       -0.200000      0.150000
1        0.000000      0.000000
2        0.136364      0.454545
3       -0.100000      0.200000
4        0.000000      0.000000
…           …            …
104871   0.700000      0.600000
104872   0.000000      0.000000
104873   0.000000      0.000000
104874   0.075000      0.366667
104875   0.000000      0.000000

[104876 rows x 9 columns]
```

1 Classification Problem

I decided I wanted to classify the Player based on the Play, Act, Scene, and the polarity of the line. My hope was that we could train a model to recognize that if a line is negative in a certian play,

scene, and act, it would likely be from a certain player.

1.0.1 Random Forest

I found support for creating a Random Forest here: https://www.datacamp.com/community/tutorials/random-forests-classifier-python

[18]:
```
labels = data['Player']
features = data[['Play', 'Act', 'Scene', 'Line','polarity']]
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
↪test_size=0.20)
```

[19]:
```
clf=RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
```

[20]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6851163234172387
```

1.0.2 Random Forest, again

I tried using the random forest again without including the Line in the features. This resulted in much lower accuracy.

[21]:
```
labels = data['Player']
features = data[['Play', 'Act', 'Scene','polarity']]
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
↪test_size=0.20)
```

[22]:
```
clf=RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.41151792524790237
```

1.0.3 Decision Trees

Finally, I used the decision tree classification algorithm to fit the data. This resulted in 70% accuracy.

I found support for creating Decision Tree's in python here: https://www.datacamp.com/community/tutorials/decision-tree-classification-python

[23]:
```
labels = data['Player']
features = data[['Play', 'Act', 'Scene', 'Line','polarity']]
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
↪test_size=0.20)
```

[24]:
```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7047578184591915
```

1.0.4 Final Remarks

I would love to know what could have made this model more accuarate. I attempted to use the PlayerLine to reason about the Player based on if the line was positive or negative but that didn't work as well as I had hoped. The accuracy of the decision tree and the random forest using polarity as a feature resulted in around 70% accuracy, which is lower than the ideal 90%+ accuracy.

[ ]: