Shakespeare_WordTrimming

September 21, 2020

0.1 Trimming Player Line

The goal of this notebook is to trim the PlayerLine catagory of data to use to develop a model. Some of the operations, such as removing stop words, are expensive. To avoid running the operations each time, I downloaded the results of the word trimming to a new file, `PlayerLine_trimmed.csv`.

The steps in this file are taken from : https://towardsdatascience.com/nlp-for-beginners-cleaning-preprocessing-text-data-ae8e306bef0f

[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
```

[2]:
```
import nltk
# nltk.download('stopwords')
# nltk.download('wordnet')
from bs4 import BeautifulSoup
import string
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
```

[3]:
```
data = pd.read_csv('~/Documents/EECS/EECS_731/HW/EECS731_2/data/
↪1028_2124_bundle_archive/Shakespeare_data.csv')
```

[4]:
```
data.columns
```

[4]:
```
Index(['Dataline', 'Play', 'PlayerLinenumber', 'ActSceneLine', 'Player',
       'PlayerLine'],
      dtype='object')
```

[5]:
```
data.dtypes
```

[5]:
```
Dataline              int64
Play                 object
```

```
PlayerLinenumber    float64
ActSceneLine         object
Player               object
PlayerLine           object
dtype: object
```

What if we made the player a number? So that we could talk about what players are saying what? What if we only looked at the major players… could count how many times they speak and only use the ones that pop up the most.

Need to delete the columns where NaN is present for player.

[6]:
```
data = data.dropna()
```

[7]:
```
data = data.drop(columns="Dataline")
```

[8]:
```
data
```

[8]:
```
                  Play  PlayerLinenumber ActSceneLine         Player  \
3              Henry IV               1.0        1.1.1  KING HENRY IV
4              Henry IV               1.0        1.1.2  KING HENRY IV
5              Henry IV               1.0        1.1.3  KING HENRY IV
6              Henry IV               1.0        1.1.4  KING HENRY IV
7              Henry IV               1.0        1.1.5  KING HENRY IV
…                   …                 …            …              …
111390  A Winters Tale              38.0      5.3.179        LEONTES
111391  A Winters Tale              38.0      5.3.180        LEONTES
111392  A Winters Tale              38.0      5.3.181        LEONTES
111393  A Winters Tale              38.0      5.3.182        LEONTES
111394  A Winters Tale              38.0      5.3.183        LEONTES

                                            PlayerLine
3                So shaken as we are, so wan with care,
4            Find we a time for frighted peace to pant,
5         And breathe short-winded accents of new broils
6                To be commenced in strands afar remote.
7             No more the thirsty entrance of this soil
…                                                   …
111390  Is troth-plight to your daughter. Good Paulina,
111391       Lead us from hence, where we may leisurely
111392            Each one demand an answer to his part
111393   Perform'd in this wide gap of time since first
111394           We were dissever'd: hastily lead away.

[105152 rows x 5 columns]
```

Naming Issue In "A Comedy of Errors", there is ANTIPHOLUS OF SYRACUSE and ANTIPHOLUS OF EPHESUS and I think that that the ANTIPHOLUS part of the name got sepa-

rated so we can delete all of thos cells with the entry for player as ANTIPHOLUS and rename the player columns that say OF EPHESUS and OF SYRACUSE to ANTIPHOLUS OF SYRACUSE and ANTIPHOLUS OF EPHESUS

In "Love's Labour's Lost" the character Don Armado is named DON in the `PlayerLine` column and ADRIANO DE ARMADO in the `Player` column. So, we can delete the cells for DON in the `PlayerLine` and rename in to Don Armando in the `Player` column.

[9]:
```
data.drop(data.loc[data['PlayerLine']=="ANTIPHOLUS"].index, inplace=True)
data['Player'].replace({'OF SYRACUSE': 'ANTIPHOLUS OF SYRACUSE', 'OF EPHESUS':␣
↪'ANTIPHOLUS OF EPHESUS'})
```

[9]:
```
3         KING HENRY IV
4         KING HENRY IV
5         KING HENRY IV
6         KING HENRY IV
7         KING HENRY IV
              …
111390          LEONTES
111391          LEONTES
111392          LEONTES
111393          LEONTES
111394          LEONTES
Name: Player, Length: 104977, dtype: object
```

[10]:
```
data.drop(data.loc[data['PlayerLine']=="DON"].index, inplace=True)
data['Player'].replace({'ADRIANO DE ARMADO': 'DON ARMADO',})
```

[10]:
```
3         KING HENRY IV
4         KING HENRY IV
5         KING HENRY IV
6         KING HENRY IV
7         KING HENRY IV
              …
111390          LEONTES
111391          LEONTES
111392          LEONTES
111393          LEONTES
111394          LEONTES
Name: Player, Length: 104876, dtype: object
```

0.2 Preprocessing text data

0.2.1 Goal

My goal was to preprocess the text data to use it for analysis in the `ShakespeareAnalysis` notebook. I wanted to reason if a line was positive or negative and see if that could help increase the accuracy when identifying the player.

First, I removed punctuation from each of the lines. We really only care about important words, not the punction around those words.

[11]:
```
def remove_punctuation(text):
        no_punct = "".join([c for c in text if c not in string.punctuation])
        return no_punct
```

[12]:
```
data['PlayerLine'] = data['PlayerLine'].apply(lambda x: remove_punctuation(x))
data.head()
```

[12]:
```
       Play  PlayerLinenumber ActSceneLine         Player  \
3  Henry IV               1.0        1.1.1  KING HENRY IV
4  Henry IV               1.0        1.1.2  KING HENRY IV
5  Henry IV               1.0        1.1.3  KING HENRY IV
6  Henry IV               1.0        1.1.4  KING HENRY IV
7  Henry IV               1.0        1.1.5  KING HENRY IV

                                    PlayerLine
3           So shaken as we are so wan with care
4      Find we a time for frighted peace to pant
5  And breathe shortwinded accents of new broils
6         To be commenced in strands afar remote
7      No more the thirsty entrance of this soil
```

Here, I broke the strings into a list of words based on spaces.

[13]:
```
tokenizer = RegexpTokenizer(r'\w+')
```

[14]:
```
data['PlayerLine'] = data['PlayerLine'].apply(lambda x: tokenizer.tokenize(x.
↪lower()))
data['PlayerLine'].head(10)
```

[14]:
```
3         [so, shaken, as, we, are, so, wan, with, care]
4      [find, we, a, time, for, frighted, peace, to, …
5      [and, breathe, shortwinded, accents, of, new, …
6         [to, be, commenced, in, strands, afar, remote]
7      [no, more, the, thirsty, entrance, of, this, s…
8      [shall, daub, her, lips, with, her, own, child…
9      [nor, more, shall, trenching, war, channel, he…
10     [nor, bruise, her, flowerets, with, the, armed…
11           [of, hostile, paces, those, opposed, eyes]
12     [which, like, the, meteors, of, a, troubled, h…
Name: PlayerLine, dtype: object
```

Next, we must remove stop words. Stop words are words like “the” that get in the way of understanding the meaning of a line.

[15]:
```
def remove_stopwords(text):
    words = [w for w in text if w not in stopwords.words('english')]
    return words
```

[16]:
```
data['PlayerLine'] = data['PlayerLine'].apply(lambda x: remove_stopwords(x))
data['PlayerLine'].head(5)
```

[16]:
```
3                              [shaken, wan, care]
4             [find, time, frighted, peace, pant]
5    [breathe, shortwinded, accents, new, broils]
6              [commenced, strands, afar, remote]
7                       [thirsty, entrance, soil]
Name: PlayerLine, dtype: object
```

Now, we cut off prefixes and suffixes. For example, a conjugated word like "slept" would just become sleep. This helps us reason about the words without caring about every tense the word could be in.

[17]:
```
lemmatizer = WordNetLemmatizer()

def word_lemmatizer(text):
    lem_text = [lemmatizer.lemmatize(i) for i in text]
    return lem_text
```

[18]:
```
data['PlayerLine'].apply(lambda x : word_lemmatizer(x))
data['PlayerLine'].head(5)
```

[18]:
```
3                              [shaken, wan, care]
4             [find, time, frighted, peace, pant]
5    [breathe, shortwinded, accents, new, broils]
6              [commenced, strands, afar, remote]
7                       [thirsty, entrance, soil]
Name: PlayerLine, dtype: object
```

Finally, you can add the data back together using the join function.

[19]:
```
stemmer = PorterStemmer()

def word_stemmer(text):
        stem_text = " ".join([stemmer.stem(i) for i in text])
        return stem_text
```

[20]:
```
data['PlayerLine'].apply(lambda x : word_stemmer(x))
data['PlayerLine'].head(5)
```

[20]:
```
3                              [shaken, wan, care]
4             [find, time, frighted, peace, pant]
5    [breathe, shortwinded, accents, new, broils]
```

```
6              [commenced, strands, afar, remote]
7                       [thirsty, entrance, soil]
Name: PlayerLine, dtype: object
```

0.3 Exporting data

Here, after we finished trimming, I exported the data to a csv. I used the first csv when performing the analysis.

[21]:
```
data.to_csv('PlayerLine_trimmed_2.csv')
```

There are 104876 rows of data. For Model training, we only want to use 80% of the data for training. I will create a subset of the data for training and a subset of the data for testing. Can use `data_train = data.sample(frac=0.8)` to get 80% of the data but then you can't get the other 20% for testing.

[22]:
```
data_shuf = shuffle(data)
data_train = data_shuf[:84000]
data_test = data_shuf[-21052:]
```

[23]:
```
data_train
```

[23]:

	Play	PlayerLinenumber	ActSceneLine	Player	PlayerLine
16895	As you like it	36.0	2.7.183	AMIENS	[thou, art, seen]
110371	A Winters Tale	145.0	4.4.577	FLORIZEL	[good, camillo]
63004	Merchant of Venice	37.0	3.2.259	BASSANIO	[ever, blotted, paper, gentle, lady]
68185	A Midsummer nights dream	31.0	3.2.146	DEMETRIUS	[thou, holdst, thy, hand, let, kiss]
77634	Pericles	13.0	4.3.56	DIONYZA	[like, one, superstitiously]
…	…	…	…	…	…
61721	Merchant of Venice	15.0	1.3.30	SHYLOCK	[prophet, nazarite, conjured, devil]
67686	A Midsummer nights dream	3.0	2.2.33	OBERON	[thou, wakest, thy, dear]
1139	Henry IV	52.0	2.4.138	FALSTAFF	[call, thee, coward, ill, see, thee, damned, e…
50668	King Lear	70.0	2.4.245	KING LEAR	[plaguesore, embossed, carbuncle]
77175	Pericles	36.0	3.2.125	CERIMON	[lend, hands, next, chamber, bear]

```
[84000 rows x 5 columns]
```

0.3.1 Mindless Musings

The following are just mindless things I did when brainstorming for this project. I kept them in case I wanted to reference them in future projects.

Lets just look at the play and the player to find the most popular player's for each play. You can then say, if I am in play A, this player will speak the most.

[24]:
```
play_player = data.drop(columns="PlayerLinenumber")
play_player = play_player.drop(columns="ActSceneLine")
play_player = play_player.drop(columns="PlayerLine")
play_player = play_player[play_player["Play"] == "Henry IV"]
```

[25]:
```
play_player
```

[25]:
```
          Play         Player
3     Henry IV  KING HENRY IV
4     Henry IV  KING HENRY IV
5     Henry IV  KING HENRY IV
6     Henry IV  KING HENRY IV
7     Henry IV  KING HENRY IV
…        …            …
3199  Henry IV  KING HENRY IV
3200  Henry IV  KING HENRY IV
3201  Henry IV  KING HENRY IV
3202  Henry IV  KING HENRY IV
3203  Henry IV  KING HENRY IV

[3044 rows x 2 columns]
```

`count_player` is a list of the most common Players in the Shakespear Plays. Could say `value_counts(normalize=True)` to normalize all the values.

[26]:
```
count_player = data_train['Player'].value_counts()
count_player
```

[26]:
```
GLOUCESTER        1433
HAMLET            1197
IAGO               888
FALSTAFF           869
KING HENRY V       818
                  …
HORTENSIA            1
Outlaws              1
HERNIA               1
```

```
Third Musician       1
All The Lords        1
Name: Player, Length: 917, dtype: int64
```

We can also put the counts into bins to better see the distrubtion of how many times a player speaks.

[27]:
```
count_player.value_counts(bins=20)
```

[27]:
```
(-0.433, 72.6]      614
(72.6, 144.2]       110
(144.2, 215.8]       70
(215.8, 287.4]       58
(287.4, 359.0]       16
(502.2, 573.8]       11
(430.6, 502.2]       11
(359.0, 430.6]       10
(645.4, 717.0]        6
(573.8, 645.4]        5
(860.2, 931.8]        2
(717.0, 788.6]        1
(788.6, 860.2]        1
(1146.6, 1218.2]      1
(1361.4, 1433.0]      1
(1289.8, 1361.4]      0
(931.8, 1003.4]       0
(1003.4, 1075.0]      0
(1075.0, 1146.6]      0
(1218.2, 1289.8]      0
Name: Player, dtype: int64
```

We see that most people are mentioned between 0 and 91.5 times. I guess we could make a model that says, in any shakespeare play, there will be x players who speak more than once, and y players who only speak once.