Assignment 3: Prediction/Modeling

Firat Ciftci

Due: Friday, Nov 22, 2019 in class

Submission: Complete this notebook and print out the output or electronically submit it.

Everything you need to complete is marked with a TODO. For textual questions create a new cell under the question to respond to it.

Dataset

Game of Thrones is one of the most watched TV series of all times. With hundreds of characters and more than 22K sentences, this dataset aims to help you test your text mining skills. The content is pretty simple: the dataset contains each and every sentence said in the serie together with who has said it, the episode and the season. For the time being the dataset includes episodes from Season 1 to Season 7. You can download the dataset here: https://github.com/sjyk/cmsc21800/blob/master/got.csv (https://github.com/sjyk/cmsc21800/blob/master/got.csv)

Loading the Dataset

The first task is to load the dataset into a pandas dataframe and filter relevant rows for this assignment. We only care about the rows for chracters that are present in all 7 of the seasons and speak a sufficient amount. Filter the rows to include only those with the speaker name "Cersei", "Daenerys", "Tyrion", and "Arya"--make sure you handle upper-case and lower case properly!

In [293]:
```
import pandas as pd

def load_dataset(filename):
    '''
    Given a filename return a dataframe
        containing the rows.

    Only return those rows with a name:
    * "cersei"
    * "daenerys"
    * "tyrion"
    * "arya"
    '''
    df = pd.read_csv(filename, delimiter=";")

    return df[df['Name'].isin(['cersei', 'daenerys', 'tyrion', 'arya'])]
```

In [294]:
```
df = load_dataset('got.csv')

df[-10:]
```

Out[294]:

	Column1	Season	Episode	Sentence	Name	N_serie	N_Season	Emision Date
22455	22456	Season 7	the dragon and the wolf	And that will be treason	cersei	67	7	27/08/2017
22457	22458	Season 7	the dragon and the wolf	Disobeying your queen	cersei	67	7	27/08/2017
22459	22460	Season 7	the dragon and the wolf	I told you no one walks away from me	cersei	67	7	27/08/2017
22461	22462	Season 7	the dragon and the wolf	Theres one more yet to come	cersei	67	7	27/08/2017
22492	22493	Season 7	the dragon and the wolf	Are you all right?	arya	67	7	27/08/2017
22494	22495	Season 7	the dragon and the wolf	You did the right thing	arya	67	7	27/08/2017
22496	22497	Season 7	the dragon and the wolf	Im just the executioner You passed the sentenc...	arya	67	7	27/08/2017
22498	22499	Season 7	the dragon and the wolf	I was never going to be as good a lady as you ...	arya	67	7	27/08/2017
22500	22501	Season 7	the dragon and the wolf	I believe thats the nicest thing youve ever sa...	arya	67	7	27/08/2017
22502	22503	Season 7	the dragon and the wolf	In winter, we must protect ourselves Look afte...	arya	67	7	27/08/2017

Basic Cluster Analysis

Next, we will mine this dataset to understand what types of structure exist. In the next task, we will write a featurizer that takes the dataset and converts it into a set of feature vectors. We will use a tf-idf featurizer to do this:

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

In [295]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

def featurize(quotes):
    '''
    Takes a set of quotes as input and returns two things: an array of f
eature vectors
        and the featurizer.

    * Use the tfidfvectorizer from sklearn and remove english stopwords
 and restrict the features to
        words that appear in *at most* 20 quotes.

    Return values (returns a tuple!!):
        X - a dense numpy array of feature vectors representing the text
data.
        vectorizer - a TfidfVectorizer object.
    '''
    vectorizer = TfidfVectorizer(stop_words='english', max_df=20)
    vector = vectorizer.fit_transform(quotes)

    return vector.todense(), vectorizer

df = load_dataset('got.csv')
X, vectorizer = featurize(df['Sentence'])
```

Now, let's compute the principal components of this featurized dataset:

In [296]:
```
from sklearn.decomposition import PCA
import numpy as np

def compute_pca(features, components=2):
    '''
    Calculate the first two principal components of the
         features that you have. Return the components,
         the explained variance, and N-D representation of the
         feature vectors.

    Return Values (returns a 4-tuple):
          * axes (the principal components from .components_)
          * Y (the dimensionality reduced data)
          * c = explained variance on a range from [0,1]
    '''
    pca = PCA(n_components=components)
    pca.fit(features)

    return pca.components_, pca.transform(features), pca.explained_varia
nce_ratio_

# Compute PCA
pcs, Y, c = compute_pca(X, 3)
```

Now write code to interpret the PCA components. Write a function that uses the vectorizer to determine the words whose presence or absence is strongest in the PC.

In [297]:
```
def top_k(pc, vectorizer, k=10):
    '''
    Finds the highest (most positive) weighted elements in a pc and
        then returns the words that correspond to those elements.

    Exclude all words that are less than 3 letters.

    Return Value: A set of k words
    '''
    weights = [x for x in list(zip(pc, vectorizer.get_feature_names()))
 \
                if len(x[1]) >= 3]
    weights.sort()

    return weights[-k:]

# Extract each of the pcs
pc1, pc2, pc3 = pcs

print("PC 1: Most Positive: ", top_k(pc1, vectorizer))
print("\nPC 2: Most Positive: ", top_k(pc2, vectorizer))
print("\nPC 3: Most Positive: ", top_k(pc3, vectorizer))
print("\nExplained Variance: ", c)
```

```
PC 1: Most Positive:  [(0.04217712279957113, 'swear'), (0.0425497941415
31205, 'loved'), (0.04519834594387526, 'taken'), (0.0528466079583697,
'matter'), (0.0597150037400482, 'try'), (0.08119808479588238, 'belon
g'), (0.08238751952102706, 'swore'), (0.09601781471628285, 'liar'), (0.
11093947939592254, 'fuck'), (0.9596955096692612, 'care')]

PC 2: Most Positive:  [(0.03484346140548936, 'pick'), (0.0421680331625
68404, 'slaves'), (0.043217243122619875, 'mycah'), (0.0534199665460078
1, 'lies'), (0.06505915486791708, 'expect'), (0.07845933208057161, 'tra
itor'), (0.07907041923630982, 'coward'), (0.09772809651538666, 'aliv
e'), (0.13206552413225245, 'truth'), (0.95648690824929, 'liar')]

PC 3: Most Positive:  [(0.04097726062931132, 'desire'), (0.041290885464
794494, 'interrupt'), (0.04551883302118401, 'continue'), (0.04939394703
887829, 'husband'), (0.04948127694699012, 'catelyn'), (0.04994668863930
759, 'loyalty'), (0.0779784998747717, 'rules'), (0.09792828051353855,
'joke'), (0.11700070540297113, 'doesn'), (0.9234743033906512, 'mean')]

Explained Variance:  [0.00303668 0.00280099 0.00250545]
```

For those of you who know the story, you can see the story arcs in the principal components.

Predicting The Speaker

Now, we will have you predict the speaker from the patterns in the text. The first step is to define a training and a test set. Write the following function that splits the loaded dataset into a training set (80% of the data) and a test set (20% of the data). The partition should be random.

In [298]:
```
from sklearn.model_selection import train_test_split as split

def train_test_split(dataframe):
    '''
    Write a function that splits the dataset into a
        training set and a testing set

    Return values (returns a tuple!) :
        - A training set 80% of the data,
        - A test set 20% of the data.
    '''
    # The assignment does not put a limit on importing any tools
    # in terms of splitting the data, and I think this is the
    # best/most efficient way to handle this issue, so I hope
    # that me using an imported function here isn't an issue
    train, test = split(dataframe, test_size=0.20)

    return train, test

train, test = train_test_split(df)
```

Your task is to build a classifier that will achieve at least 45% accuracy on this dataset

To achieve this you will have to manipulate the data and play around with different featurization techniques and modeling choices. First, write a function that "fits" a language model, such as TFIDF, to the training dataset. It is up to you to tune the parameters for the vectorizer you choose approriately.

In [299]:
```
def language_model(training_quotes):
    '''
    Write a function that instantiates a vectorizer
        (e.g., a TfidfVectorizer), runs fit(), and
        returns the vectorizer.
    '''
    vectorizer = TfidfVectorizer(stop_words='english')

    vectorizer.fit(training_quotes)

    return vectorizer
```

Next, you will write a featurizer that takes in a set of quotes and returns an array of feature vectors using the language model above. You may add whatever additional features you find useful.

In [300]:
```
def prediction_featurize(quotes, vectorizer):
    '''
    Takes in a set of quotes and returns
        an array of feature vectors using
        the language model above.
    '''
    vector = vectorizer.transform(quotes)

    return vector.todense()
```

Finally, determine the right machine learning model to use to actually make the prediction.

In [301]:
```
vectorizer = language_model(train['Sentence'])
X = prediction_featurize(train['Sentence'], vectorizer)
Y = train['Name']

Xtest = prediction_featurize(test['Sentence'], vectorizer)
Ytest = test['Name']

from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='mu
ltinomial')
clf.fit(X, Y)
pred = clf.predict(Xtest)

# Calculate accuracy
from sklearn.metrics import classification_report
print(classification_report(Ytest, pred))
```

```
              precision    recall  f1-score   support

        arya       0.62      0.24      0.34       157
      cersei       0.60      0.35      0.44       198
    daenerys       0.55      0.39      0.46       164
      tyrion       0.45      0.80      0.58       296

    accuracy                           0.50       815
   macro avg       0.56      0.44      0.45       815
weighted avg       0.54      0.50      0.48       815
```