**IST-736 Text Mining**

**Homework-2&3**

Sharat Sripada (vssripad@syr.edu)

### Introduction

This week combines homework from week-2 and week-3 and involves the following topics:

* Word tokenization
* Stemming and Lemmatization
* Vectorization
* K-means clustering for text classification

The homework will extend the usage of the dataset used in week-1 from Twitter on topic AI/artificial intelligence for most part of the tasks and use an additional Kaggle dataset related to review of a Pink Floyd album for the clustering or classification exercise.

### Data source and collection

Data or tweets would be gathered from Twitter using the Python tweepy package/library. The code shows having to use consumer\_key, consumer\_secret, an access\_token and access\_token\_secret to setup an OAuthHandler and api handle (row 13).

We would continue to limit (via tweepy api knobs) to one hundred most recent tweets at the time/date of writing this document.

The auth parameters are setup as below:

Graphical user interface, text, application

Description automatically generated

Using filters like search string, date, and number of tweets to retrieve, a sample of tweets tagged with word *#ai* is stored in a python list *tweets = []*:

Graphical user interface, text

Description automatically generated

Further, for the clustering or classification exercise we will obtain the dataset or csv from the following URL:

<https://www.kaggle.com/michaelbryantds/reviews-of-pink-floyds-the-dark-side-of-the-moon?select=dsotm_reviews.csv>

A snapshot of the dataset in its raw form is shown here:

Graphical user interface

Description automatically generated with medium confidence

Fig: Music album review (csv -> pd dataframe)

#### Cleanup

As part of the data ingestion, we will clean up individual tweets for stop-words and punctuation marks:

Graphical user interface, text, application

Description automatically generated with medium confidence

Fig: A custom method to clean tweets

### Word Tokenization

In this exercise, we will use the nltk tokenize library and specifically consume the word\_tokenize function. The idea is, given text or a sentence we want to tokenize it into individual words. For this we first iterate through the tweets remove punctuation and other meta characters (like @, # etc.) apart from urls beginning with http/https in tweets. These are likely to have any impact for use-cases that may choose to tokenize text.

When removing the http/https url from tweet we will use a regex as show below – essentially parsing for words beginning with http and replacing it with null:

See output below showing a sample of the tokenized words:

Graphical user interface, text, application

Description automatically generated

Fig: Tokenized tweets or sentences

Stemming and Lemmatization

Stem is part of the word to which you can add inflectional affixes such as (-ed, -ize, -s, -de). Stemming is the process of removing the inflectional affixes. This is particularly useful in search engines and information retrieval where stemming is used to index words. And therefore, instead of storing all forms of words, the index can comprise merely stem words.

For this homework, we will use PorterStemmer.

To see the working of the stemmer, code was written to find words with affixes like ‘ed’ and ‘ing’ and what the stemmer would translate it into:

Text, letter

Description automatically generated

Fig: Illustration of working of PorterStemmer on dataset

Next, Lemmatization works like stemmer and has a similar use-case with search engines and information retrieval systems but particularly looks for meaning of a word. The output we will get after lemmatization is called ‘lemma’ which is a root word rather root stem, the output of stemming.

An example to bring out the difference between the two is as below:

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize('believes'))

print(word\_stemmer.stem('believes'))

**Output:**

Belief <- Related to lemmatization

Believ <- Related to stemmer

Finally, we use the nltk lemmatizer.lemmatize function to iterate through the tokenized words and translate them to lemma.

Text

Description automatically generated

Fig: Illustration of working of Lemmatizer on dataset

### Vectorization

Machine learning algorithms can operate only on numeric feature space, often expecting input as 2-dimensional array where rows are instances and columns are features. To perform machine learning on text, we need to transform our documents or text into vector representations such that we can apply numeric machine learning. This process is called vectorization and is the first step towards language-aware analysis.

Here we experiment with the following vectorization techniques:

* Count vectorization – Unigram and N-gram/Bi-gram term frequencies
* TF-IDF

For this we will import the *CountVectorizer* or *TfidfVectorizer* from the sklearn.feature\_extraction.text library.

#### Count vectorization

Using the unigram term frequency vectorization, we get *545* features and an overall dimensionality (100, 545).

Graphical user interface, text, application

Description automatically generated

Fig: CountVectorizer Unigram setup and results

Likewise with bi-gram or N-gram we get 708 features and overall dimensionality (100, 708).

Text, letter

Description automatically generated

Fig: CountVectorizer Ngram setup and results

Using the toarray() function, we the following numpy array (for uni-gram):

array([[0, 0, 0, ..., 0, 1, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 1, 0, ..., 0, 0, 0],

...,

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0]])

Although this may not be directly useful to interpret, this shows many cells comprise the value zero since words/features would be present in every tweet is therefore known as a Sparse Matrix.

#### TF-IDF

TF-IDF uses a technique related to the following formula:

W(x,y) = tf(x,y) \* log(N/dfx)

where:

W(x,y) = Word x within document y

tf(x,y) = Frequency of x in y

dfx = Number of documents containing x

N = Total number of documents

Using the sklearn library for TF-IDF, we obtain the following sparse matrix:

Table

Description automatically generated

Fig: TF-IDF sparse matrix output

To visualize this better, we will transpose the numpy array to a single dimensional dataframe and sort in descending order:

Graphical user interface, text, application, email

Description automatically generated

Fig: Results (sorted) TF-IDF results

Results show important word features are thankful, idea, because, created and reliable. While words like enzc, environmental, etc. withTF-IDF 0 are not important.

### Clustering using k-means

In this section we will explore the use of k-means clustering method to solve a text prediction problem. The sequence of steps is following:

1. Make a combined dataframe (randomized or shuffled) comprising data:
   1. corpus from twitter on topic artificial intelligence
   2. csv from Kaggle with reviews on an album on Pink Floyd
2. Apply the tfidf vectorizer
3. Run the kmeans clustering algorithm

Each of the steps is illustrated through code blocks:

Graphical user interface, text, application

Description automatically generated

Fig: Concat of data-frames (+ shuffling it) and applying vectorization

A picture containing text

Description automatically generated

Fig: Setting up params for k-means and creating the model

Once the model is built, we will visualize the centroids and the clusters where it belongs (sample **output**):

Cluster 0:

album

the

time

side

one

dark

it

music

great

moon

Cluster 1:

ai

machinelearning

100daysofcode

python

iot

datascience

artificial

5g

cybersecurity

ar

**NOTE**

Cluster-0 is album review and Cluster-1 is tweets related to artificial intelligence. The segregation based on the sample set of words here looks reasonable.

Finally, run the prediction on a review and see the classification result:

Text

Description automatically generated

### Conclusion

In this homework, we explored how text can be tokenized, stemmed, or lemmatized and eventually vectorized to be applied to machine-learning use-cases.

This in summary is everything that happens under the hood in applications like search engines and information retrieval systems.

Specifically, we demonstrate cleaning (removal of stop words, redundant text) in incoming text - concise storage and lookups (stemming and lemmatization) - vectorization to translate text to feature-sets and numeric data which can then be ingested by machine learning algorithms.

Finally, we demonstrate a use-case or application where, given a sentence or text we can correctly classify it to a cluster of documents. This can be thought of as a naïve information retrieval task.