EECS4415 Big Data Systems

Fall 2018

Assignment 1 (10%): Text Analytics using Python

Due Date: 11:59 pm on Friday, Oct 19, 2018

Objective

In this assignment, you will be writing python programs/scripts for performing basic analytics on textual data. The dataset is coming from a collection of 55000+ English song lyrics from LyricsFreak[1] and can be accessed here (registration to Kaggle is required to download the raw dataset):

Dataset (`songdata.csv`): https://www.kaggle.com/mousehead/songlyrics

The first program (`dstats.py`) performs descriptive analytics of the dataset. The second (`songprofiling`) and third (`artistprofiling.py`) programs perform statistical analysis to determine the importance of words in the lyrics of a specific song or a specific artist/band. The fourth (`songsim.py`) and fifth (`artistsim.py`) programs are tools for comparing two songs or two artists.

Important Notes:

- You must use the *submit* command to electronically submit your solution by the due date.
- All programs are to be written using Python 3.
- Your programs should be tested on the *docker image that we provided* before being submitted.
- To get full marks, your code must be well-documented.

What to Submit

When you have completed the assignment, move or copy your python scripts in a directory (e.g., assignment1), and use the following command to electronically submit your files within that directory:

```
% submit 4415 a1 dstats.py songprofiling.py artistprofiling.py songsim.py artistsim.py team.txt
```

The `team.txt` file includes information about the team members (*first name, last name, student ID, login, yorku email*). You can also submit the files individually after you complete each part of the assignment– simply execute the *submit* command and give the filename that you wish to submit. Make sure you name your files **exactly** as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command:

```
% submit -l 4415 a1
```

[1] https://www.lyricsfreak.com/

A. Descriptive Statistics (24%, 4% each)

Write a python program (`dstats.py`) that given a collection of English song lyrics **computes** and **prints out (in the STDOUT)** the following statistics:

- number of artists/bands in the collection (`numOfArtists`)
- number of songs in the collection (`numOfSongs`)
- average number of songs per artist/band (`avgNumOfSongs`)
- average number of unique words per song in the collection (`avgNumOfWords`)
- average number of unique words per song of an artist/band, sorted by artist/band name in an alphanumerically ascending order, i.e., a->z (`pairsOfArtistAvgNumOfWords`)
- plot a bar chart that shows the *top-10* pairs found in the previous bullet, where the *x-axis* represents the artists/band and the *y-axis* represents the average number of words. Note that the top-1 artist/band is the one with the largest average number of unique words per song.

The collection of songs is provided in a file `songdata.csv` that follows the same format as the original data set provided by Kaggle. The contents of the file might vary when testing your code.

Running the script:

Your script should be run as follows:

```
% python3 dstats.py < songdata.csv
```

Hints:

In order to perform a more meaningful text analysis, you need to use a *word tokenizer* to tokenize the song lyrics into individual words. You also need to eliminate *stopwords* — the very common words in English — and words just one character long, as they are not interesting for the analysis. The Natural Language Toolkit[2] library (`nltk`) provides this functionality off the shelf.

Use the `matplotlib.pyplot` module to create the plot. You can follow the example at pythonspot[3] about using `matplotlib` to create a bar chart: https://pythonspot.com/en/matplotlib-bar-chart/

[2] https://www.nltk.org/
[3] https://pythonspot.com

B. Creating Profiles of Songs and Artists (50%, 25% each)

Write two python programs with the following descriptions:

- `songprofiling.py`: Given a collection of English songs, it computes and prints out (in the STDOUT) the profile of each song in the collection; the profile of a song consists of the ***top-50*** more important words of its lyrics, based on the *tf-idf* score. Before printing out, you need to sort the pairs of (*word*, *score*) in descending order of *score*.
- `artistprofiling.py`: Given a collection of English songs, it computes and prints out (in the STDOUT) the profile of each artist/band in the collection; the profile of an artist/band consists of the ***top-100*** more important words of their song lyrics, based on the *tf-idf* score. Before printing out, you need to sort the pairs of (*word*, *score*) in descending order of *score*.

The collection of songs is provided in a file `songdata.csv` that follows the same format as the original data set provided by Kaggle. The contents of the file might vary when testing your code.

Running the script:

Your scripts should be run as follows:

```
% python3 songprofiling.py < songdata.csv

% python3 artistprofiling.py < songdata.csv
```

Hints:

The `scikit-learn` has a built-in tf-idf implementation that you can use off the shelf. You are also welcome to build your own tf-idf implementation, based on the tutorial material presented in class (let us know if you have followed this path!).

C. Comparing Songs and Artists (26%, 13% each)

Write two python programs with the following descriptions:

- `songsim.py`: Given a collection of English songs, it computes and prints out (in the STDOUT) the similarity between two songs in the collection. It determines the similarity between two songs by computing the Jaccard index of their profiles (re-use the song profile of Question B).
- `artistsim.py`: Given a collection of English songs, it computes and prints out (in the STDOU) the similarity between two artists in the collection. It determines the similarity between two artists by computing the Jaccard index of their profiles (re-use the artist/band profile of Question B).

The collection of songs is provided in a file `songdata.csv` that follows the same format as the original data set provided by Kaggle. The contents of the file might vary when testing your code.

Running the script:

Your script should be run as follows:

```
% python3 songsim.py songdata.csv <song1_id> <song2_id>

% python3 artistsim.py songdata.csv <artist1_id> <artist2_id>
```

Assume that the first song in the collection (as reading from top to bottom) has `song_id=1`, the second has `song_id=2`, etc. Similarly, assume that the first artist has `artist_id=1`, the second has `artist_id=2`, etc. The largest song id should be equal to the number of songs in the collection. The largest artist id should be equal to the number of unique artists/bands.

Hints:

For the Jaccard index, it suffice to consider its simplest form, defined as the size of the intersection divided by the size of the union of two sample sets A and B. By now, you won't be surprised to find out that the Jaccard index can be computed using the `sklearn.metrics` module. Shouldn't be hard to implement it yourself.

Observe that in this example we are not using the STDIN redirection symbol (<) to read from the file. Instead we provide a list of arguments that are passed to the programs as command line arguments. The `argparse` module makes it easy to write user-friendly command-line interfaces.