EECS4415 Big Data Systems

Fall 2018

Assignment 2 (10%): Distributed Text Analytics using Python

Due Date: 11:59 pm on Friday, Nov 9, 2018

Objective

In this assignment, you will be designing and implementing MapReduce algorithms 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 set of the MapReduce algorithms compute *n*-grams of the lyrics; the second set computes *k*-skip-*n*-grams of the lyrics; the third set computes an inverted index of the lyrics. These are important statistics and tools commonly used in computational linguistic and information retrieval tasks.

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 and outputs in a directory (e.g., assignment2), and use the following command to electronically submit your files:

```
% submit 4415 a2 umapper.py ureducer.py unigrams.txt bmapper.py breducer.py bigrams.txt
tmapper.py treducer.py trigrams.txt frequency-computation.txt skipgrammapper.py
skipgramreducer.py skipgrams.txt iimaper.py iireducer.py inverted-index.txt 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. Distributed Computation of *n*-grams (35%, 5% each)

In the fields of computational linguistics, an ***n*-gram** is a contiguous sequence of *n* items from a given sample of text. For this part of the assignment you can assume that items are **words** collected from song lyrics. An *n*-gram of size 1 is referred to as a "unigram"; of size 2 is a "bigram"; of size 3 is a "trigram". For example, given the text input "`I love ice cream`" the following unigrams, bigrams and trigrams are computed:
unigrams → `("I", "love", "ice", "cream")`
bigrams → `("I love", "love ice", "ice cream")`
trigrams → `("I love ice", "love ice cream")`

Your task is to design and implement MapReduce algorithms that given a collection of English songs:

- compute the number of occurrences of each **unigram** in the song collection (`umapper.py`, `ureducer.py`) and output the results in a file called `unigrams.txt`
- compute the number of occurrences of each **bigram** in the song collection (`bmapper.py`, `breducer.py`) and output the results in a file called `bigrams.txt`
- compute the number of occurrences of each **trigram** in the song collection (`tmapper.py`, `treducer.py`) and output the results in a file called `trigrams.txt`
- how would you modify these scripts in order to compute the **frequency** of each of the quantities (instead of the number of occurrences)? Provide a short answer in plain text (up to half a page) with the name `frequency-computation.txt`

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:

The following webpage provides useful information on how to test your scripts first locally and then in the Hadoop environment:
https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

B. Distributed Computation of *k*-skip-*n*-gram (25%)

In the field of computational linguistics, in particular language modeling, skip-grams are a generalization of *n*-grams in which the words need not be consecutive in the text under consideration, but may leave gaps that are skipped over. Formally, while an *n*-gram is a contiguous sequence of *n* items from a given sample of text, a ***k*-skip-*n*-gram** is a length-*n* sequence where the components occur at distance at most *k* from each other. For example, given the text input:

`"I love ice cream a lot"`

the set of *1*-skip-*2*-grams (*k*=1, *n*=2) includes all the bigrams, and in addition the subsequences:

`"I ice", "love cream", "ice a", "cream lot"`

Your task is to design and implement MapReduce algorithms that given a collection of English songs:

- compute the number of occurrences of each **1-skip-2-gram** in the song collection (`skipgrammapper.py, skipgramreducer.py`) and output the results in a file called `skipgrams.txt`

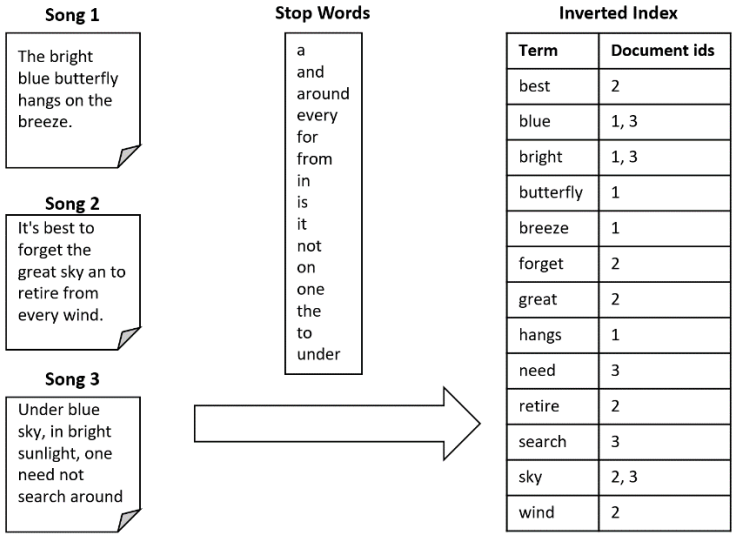
The collection of songs is provided in a file `songdata.csv` that follows the same format as the original data set provided by Kaggle. The contents of the file might vary when testing your code.

Running the script:

Similar to instructions provided in Part A.

C. Distributed Construction of an Inverted Index (40%)

In information retrieval, an inverted index is an index data structure that stores a mapping from words to the collection of documents that they appear in. Your task is to build an index that maps words that appear in lyrics to their songs. In other words, given a collection of songs, an inverted index is a dictionary where each word is associated with a list of the song ids in which that word appears in. See the example below:



Term	Document ids
best	2
blue	1, 3
bright	1, 3
butterfly	1
breeze	1
forget	2
great	2
hangs	1
need	3
retire	2
search	3
sky	2, 3
wind	2

Your task is to design and implement MapReduce algorithms that given a collection of English songs:

- compute the inverted index of the song collection (`iimaper.py`, `iireducer.py`) and output the results in a file called `inverted-index.txt`

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:

Similar to instructions provided in Part A.