EECS4415 Big Data Systems

Fall 2019

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

Due Date: 11:59 pm on Friday, Nov 8, 2019

Objective

In this assignment, you will be designing and implementing MapReduce algorithms for performing distributed analytics on textual data. The dataset is a subset of Yelp[1]'s was originally put together for the Yelp Dataset Challenge and contains seven CSV files including information about businesses across 11 metropolitan areas in four countries and can be accessed here (registration to Kaggle is required):

Yelp dataset: https://www.kaggle.com/yelp-dataset/yelp-dataset/version/6

The first set of the MapReduce algorithms compute *n*-grams of business tips; the second set computes popularity of a business; the third set computes an inverted index of the business categories. 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 checkinsmapper.py
checkinsreducer.py checkinsbyday.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. You may check the status of your submission as following:

```
% submit -l 4415 a2
```

[1] Yelp. http://www.yelp.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 yelp business tips. 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 business tips:

- compute the number of occurrences of each **unigram** in the tips 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 tips 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 tips 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 business tips is provided in a file `yelp_tip.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 business popularity by day of the week (25%)

Design and implement MapReduce algorithms that given a collection of yelp checkins, compute the aggregated number of checkins per day of the week for each business. Your mapper and reducer should be named `checkinsmapper.py` and `checkinsreducer.py`, respectively they should output the results in a file called `checkinsbyday.txt`

The output file `checkinsbyday.txt` looks like:

```
Yelp_business_1, Mon, #checkins
Yelp_business_1, Tue, #checkins
                …
Yelp_business_1, Sun, #checkins
Yelp_business_2, Mon, #checkins
Yelp_business_2, Tue, #checkins
                …
Yelp_business_2, Sun, #checkins
                …
```

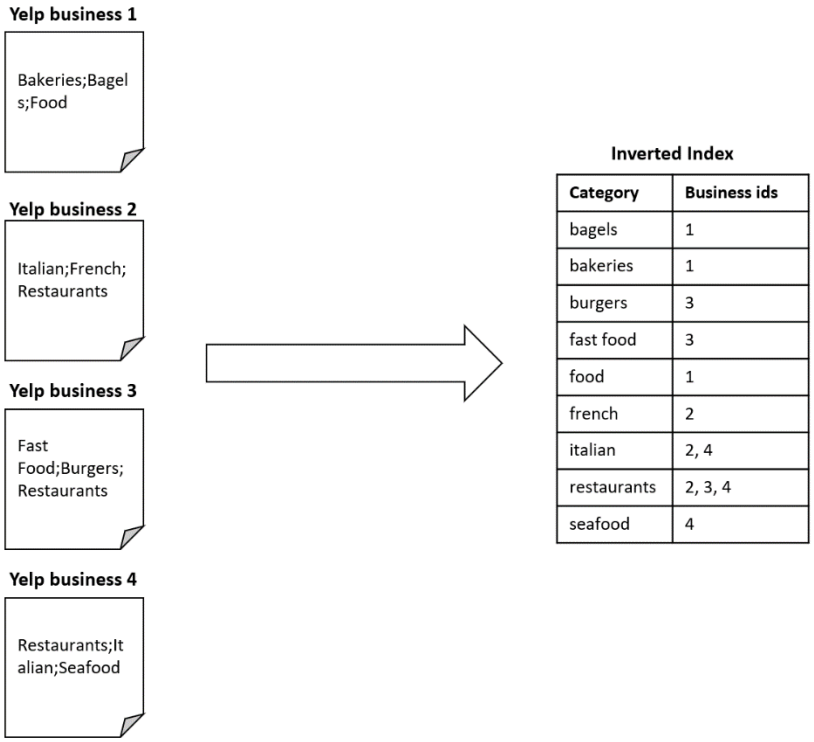
The checkin information is provided in a file `yelp_checkin.csv` that follows the same format as the original data set provided by Kaggle. The contents of the file might vary when testing your code.

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 *a collection of documents* that they appear in. Your task is to build an inverted index that maps categories (of businesses) to businesses. In other words, given a collection of businesses (as provided by Yelp), an inverted index is a dictionary where each category is associated with a list of the business ids that belong to that category. See the example below:



Inverted Index

Category	Business ids
bagels	1
bakeries	1
burgers	3
fast food	3
food	1
french	2
italian	2, 4
restaurants	2, 3, 4
seafood	4

Your task is to design and implement MapReduce algorithms that given a collection of Yelp businesses:

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

The collection of businesses is provided in a file `yelp_business.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.