Assignment 1
COMP3210 Big Data, S1, 2023
SID: 46558608, Name: Yue Long Zheng
Video Link : https://www.youtube.com/watch?v=Ky5Lv5oN-HE

1 Task 1

Before any work is done create a database and import the json file with all the tweets into the database.

1.1 Keywords

Use the pymongo library iterate through all tweets in the database. For each tweet, find the body of the tweet, which is the text of the tweet. Then use the KeyBERT [1] library to extract keywords out of the text of the tweet. Then filter those keywords for English words and then format the keywords into comma-separated value(CSV) format. Finally, add the keywords the database.

Source file for the code can be found in file `keywords.py`

1.2 Named Entities

Again iterate through all the tweets in the database. For each tweet remove hashtags, links and mentions from the text of the tweet. Then use the spaCy [2] library to extract named entities. Then turn the named entities to a CSV and then add the CSV to the tweet in the database.

Source file for the code can be found in file `named_entities.py`

1.3 Topic

From each tweet clean the text of the tweet. Now use the gensim [3] library to build a topic model from the cleaned text. Then take a topic from the topic model and format it into a CSV. Finally, add the CSV to the tweet in the database.

Source file for the code can be found in file `topic.py`

1.4 Sentiment

For each tweet clean the text of the tweet. Then use the vaderSentiment [4] library to extract the sentiment of the text. The sentiment includes a negative percent, positive percent, neutral percent and overall rating. Format the sentiment into a CSV format and then update the database with the sentiment analysis.

Source file for the code can be found in file `sentiment.py`

Figure 1: Sample of updated database

2 Task 2

Use the mrjob package to write MapReduce jobs. First create a program called `generate_text.py`, which goes through the database and write the text of all tweets into a file called `tweet_text.txt`. The create a program called `Task2.py` to implement the MapReduce program to count the frequency of all the words in the text file. Source file for task 2 can be found in `Task2.py`. Output for task 2 can be found in `Output2.py`.

Algorithm 1: Pseudocode for Task 2

Input: tweet_text.txt

mapper:
go through each word in a line
return word, 1

combiner:
just return word, sum of the counts from the mapper

reducer:
just return word, sum of all the sums from the combiner

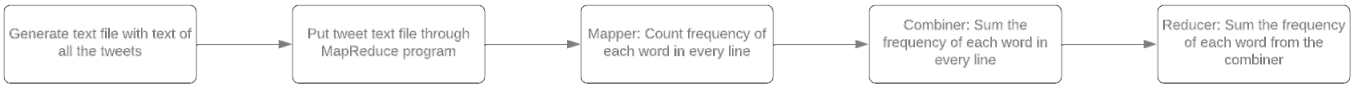

Figure 2: Flowchart for task 2

3 Task 3

Add to the `generate_text.py` program to go through the database of all the tweets and write the location data to a text file, called `tweet_location.txt`. Then create a file called `Task3.py` that uses MapReduce to count the number of Australian cities in the text file.

The file `Task3.py` first reads a file called `au.csv`, which contains a list of Australian cities. The program creates then creates a list of Austrlian cities. Then the read each line of the text file `tweet_location.txt` to see if the location is an Australian city. If it is an Australian city then it counts it.

The source file for task 3 can be found in `Task3.py`. Output for task 3 can be found in the text file called `Output3.txt`.

Algorithm 2: Pseudocode for Task 3

Input: tweet_location.txt

Create a list of Australian cities.

mapper:
Read line from text file
if *If location is in list of Australian cities* **then**
  return location, 1

combiner:
just return location, sum of counts from mapper

reducer:
just return location, sum of counts from combiner

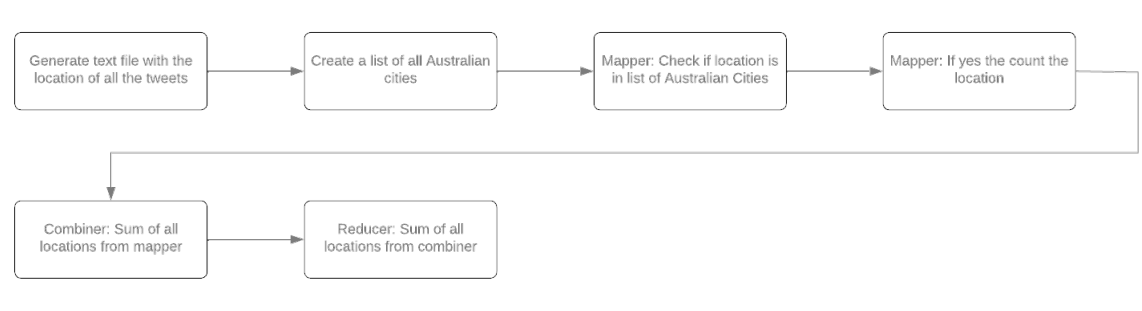

Figure 3: Flowchart for task 3

4 Task 4

4.1 MapReduce

Again add to the `generate_text.py` program to go through the database of all the tweets and write the id data to a text file, called `tweet_id.txt`. Then create a file called `Task4.py` which will count the frequency of tags and sum of the frequency of tags. Afterwards sort the tags.

Algorithm 3: Pseudocode for Task 4

Input: tweet_id.txt

mapper:
just return the line, 1

combiner:
just return the tag and the sum of the counts from mapper

reducer_count_tags:
just return the tag and the sum of the sum from the combiner

reducer_sort_tags:
sort the tags
then return the sorted tags

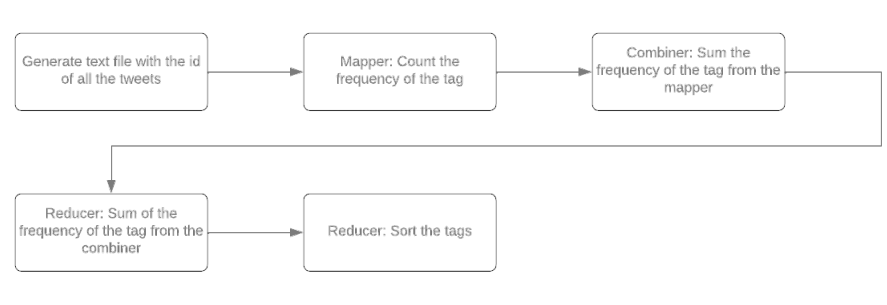

Figure 4: Flowchart for task 4

4.2 Merge Sort and Compare

Create a file called `Task4a.py` that implements a merge sort algorithm. The program `Task4a.py` will read the text file with all the tweet ids, `tweet_id.txt`. The program then adds the ids to a list and then merge sort the list. For both the merge sort and MapReduce, take starting time and the ending time of the program.

Source file for MapReduce can be found in `Task4.py` and the source file for merge sort can be found in `Task4a.py`.

The MapReduce program takes 0.58 seconds and the merge sort algorithm takes 0.04 seconds. The time taken for the MapReduce program can be found at the bottom of the text file named `Output4.txt`. Similarly, the time for merge sort can be found in the text file `Output4a.txt`

5 Task 5

Go through the database, using the `pymongo` package, and add all the keywords created in Task 1 and add them to a list. Then go through the text file, `tweet_text.txt`, and add all the words found in the text file into a list.
Now use `dask.bag` package to count the frequency of each of the words in the list. This is used to calculate the term-frequency of each word and the total number of words across all the tweets. Also use `dask.bag` to calculate the number of times a word appears across all the tweets. When calculating frequencies, the result can be transformed into a dictionary. This dictionary is then filtered, using a dictionary comprehension, to return the frequencies of all the keywords.
Finally, collect all this information into a `pandas` dataframe and calculate the TF-IDF values for each word. Then sort the dataframe by TF-IDF values. At the top of the `Task5.py` file there is a variable named `N`. This variable can be changed to show keywords with the top `N` TF-IDF values.
Term-Frequency of a word is calculated by

$$\text{Term-Frequency of word} = \frac{\text{Frequency of word}}{\text{Total word count of all documents}}$$

Inverse Document Frequency of a word is calculated by the following

$$\text{Inverse Document Frequency of word} = ln\Big(\frac{\text{Number of Documents}}{\text{Number of documents a word appears in}}\Big)$$

Then TF-IDF values of a word is calculated by multiplying the two above values.

Pandas dataframe was used in calculating TF-IDF values, when `DASK` dataframes could be used instead. On the iLearn forums, it was recommended to combine the text from all the tweets so that a word will only have one TF-IDF score. However, this makes calculating inverse document frequency tricky as the number of documents could be either 1 or 10,000 (The number of tweets in the database). If 1 was chosen then the inverse document frequency of all words would be 0 and the TF-IDF values for all words would be trivial. So 10,000 was chosen, though this approach in calculating TF-IDF is not like the approach taken in packages like `sk-learn`

References

[1] M. Grootendorst, "Keybert: Minimal keyword extraction with bert.," 2020.

[2] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020.

[3] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks, (Valetta, MT), pp. 45–50, University of Malta, 5 2010.

[4] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text.," in *ICWSM* (E. Adar, P. Resnick, M. D. Choudhury, B. Hogan, and A. H. Oh, eds.), The AAAI Press, 2014.