COMP3210 Assignment 1 Documentation

NOTE: If you want further details on how I pre-processed my data go to the Task1_and_preprocessing Jupiter notebook which provides comments and explanations of my techniques used to pre-process the data for my MapReduce functions.

Table of Contents

Task 1

Module 1: Read the Tweet Dataset from MongoDB and Curate data:

- This is done through importing the Tweets file into Studio3T

Module 2: Keyword extraction from the text of the Tweet

```
client = MongoClient = MongoClient(port = 27017)
db = client["COMP3210-ASSIGN1"]
tweets = db["10000 tweets"]
```

✓ 0.2s

```
def remove_stopwords(words):
    result =[]
    for w in words:
        if w not in stopwords.words("english"):
            result.append(w)
    return result

def stem_words(words):
    ps = PorterStemmer()
    result =[ps.stem(w) for w in words]
    return result

def find_keywords_in_text(words):
    text = re.sub(r"[^\w]", " ", words)
    words = word_tokenize(text)
    words=remove_stopwords(words)
    words=stem_words(words)
    keywords=nltk.pos_tag(words)
    return keywords

#converting keywords into a common speator to text
def keywords_to_csv(keywords):
    csv:str= ""
    for k in keywords:
        column: str = k[0]+","+k[1]
        csv = csv + column + "\n"
    return csv
```

✓ 0.2s

```
#Loops through each tweet, and extracts keywords
# updates the database with the keywords in CSV format
# 1 for using and 0 for not using
for t in tweets.find({},{"actor":1, "body":1, "_id":0}):
    keywords = find_keywords_in_text(t["body"])
    csv=keywords_to_csv(keywords)
    tweets.update_many({"actor.id":t["actor"]["id"]}, {"$set":{"keyword":csv}})
```

[10]

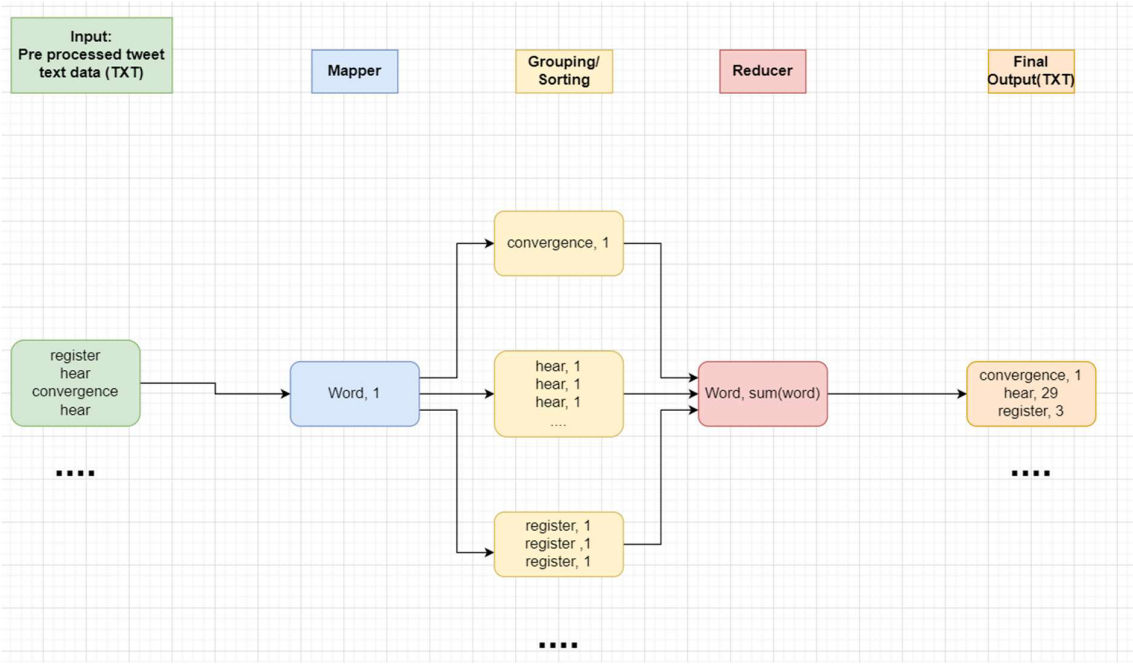
```
Output exceeds the size limit. Open the full output data in a text editor
('regist', 'NN')
('convergence2016', 'NN')
('hear', 'VBP')
('chellemelbourn', 'NN')
('talk', 'NN')
('manag', 'NN')
('chang', 'NN')
('digit', 'JJ')
```

Module 4: update the original Tweet in MongoDB

Works correctly as keyword is found within a key at the end of each document:

Task 2

Flowchart



Pseudocode

```
Task 2 - Number of occurances of words within tweets

BEGIN

compile preprocessing words <- compiling words on each line of the preprocessed text file
CLASS number of occurances()
    METHOD mapper(words):
       FOR LOOP through all compile processing words
          RETURN each word with the number 1

    METHOD reducer(words)
        RETURN each word and its total sum of occurances

RUN number of occurances class

END
```

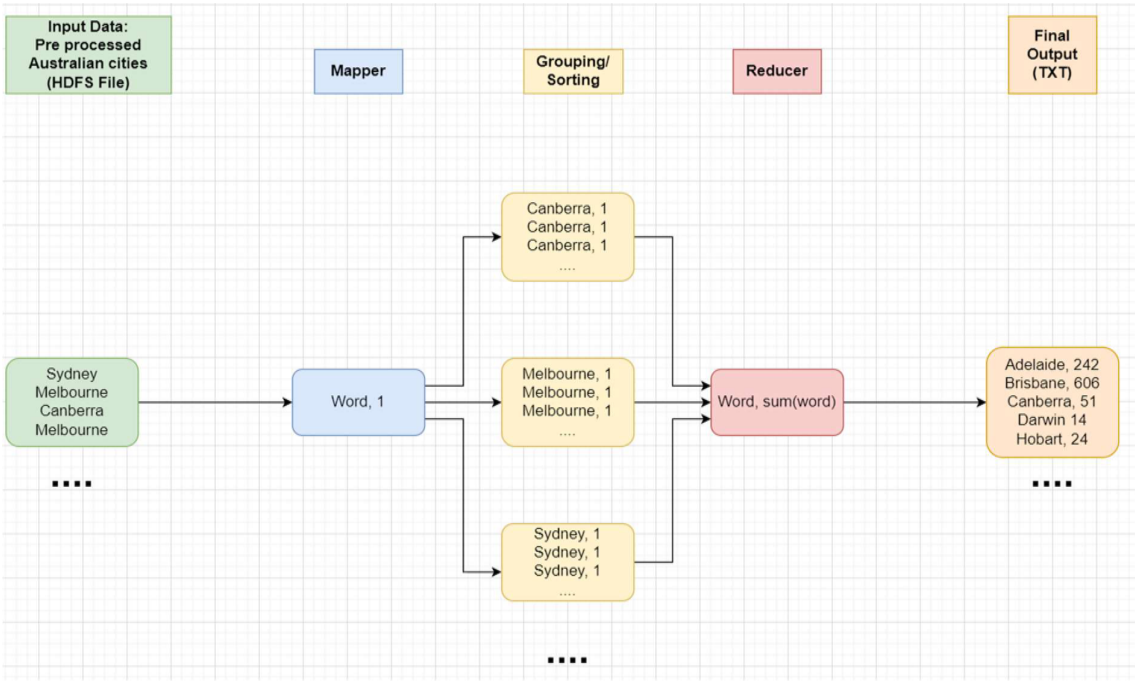
Discussion

For this section I had to clean the data this consisted of removing URLs, users, hashtags, character normalization, removing punctation, special characters and numbers, making text lower and removing stop words. To apply these functions, I had to add all post body text into a string then apply the functions through one big function called ‘clean_data’. Then finally I word tokenized to get a list of each word, remove stop words and wrote each word one by one into a text file.

For this task I used 1 mapper and reducer. Mapper iterated through the preprocessed text file to assign the words as keys and assign a value of 1 to each. Reducer was used to display and combined the total sum of each word within all post text data.

Task 3

Flowchart



Pseudocode

```
Task 3 - Number occurances of Australian cities in post locations

BEGIN

compile preprocessing words <- compiling words on each line of the preprocessed text file
CLASS number of Australin city occurances():
    METHOD mapper(city):
        FOR LOOP through all compile processing city words
            RETURN each city with the number 1

    METHOD reducer(city)
        RETURN each city and its total sum of occurances

RUN number of Australin city occurances class

END
```

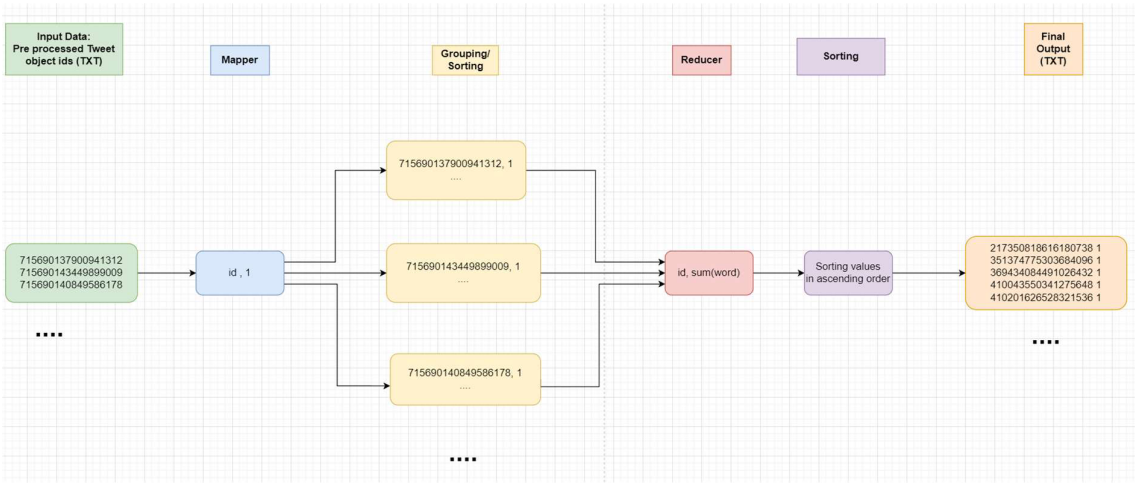
Discussion

To preprocess this data, I create a list called aus_cities which included all Australian cities within the timezone key. So, each time iterated through the data and found one of those cities it would write that city within my text file.

I used 1 mapper and reducer following a similar if not same process as task 2's mapper and reducer but instead of words within posts it was Australian cities.

Task 4 – Merge sort

Flowchart



Pseudocode

```
Task 4 p1- Merger sorting ids

BEGIN

compile objectids <- compiling objectids on each line of the preprocessed text file
CLASS merge sorting ids():
    METHOD mapper(ids):
        FOR LOOP through all ids
            RETURN each id as integer and with number 1

    METHOD reducer(ids):
        RETURN ids and total sum of occurances

ENABLE SORTING as TRUE (ascending)

RUN merge sorting ids

END
```

Discussion

We only need one mapper and reduce. Similarly, to Task 2 and Task 3 we only use the:

- Mapper for collecting the objectids within our pre-processed text file and assign a 1 to each one
- Reducer for combining duplicate objects ids and adding together.

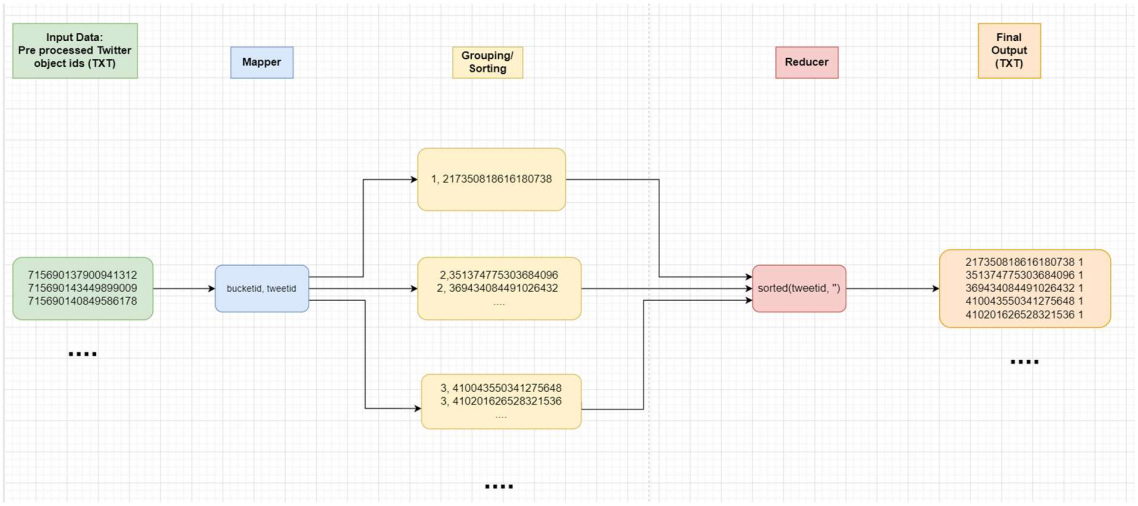
Once these methods are complete, I use the MrJob sorting function outside the class to sort the results into ascending order:

-
```
#merge sort
MRJob.SORT_VALUES = True
```

This is valid as according to sources such as Data-flair training included “Merge sort is the default feature of MapReduce. One cannot change the MapReduce sorting method, the reason is that data comes from the different nodes to a single point, so the best algorithm that can be used here is the merge sort” [1].

Task 4 – Bucket sort

Flowchart



Pseudocode

```
Task 4 p2 - Bucket sorting ids

BEGIN

CLASS bucket sorting ids():
    METHOD mapper(twitterid):
        CREATE largest bucket of size 7*10^17
        CREATE size of each bucket as 1*10^17 (7 buckets of 1*10^17)
        STORE twitter ids and convert to integer
        ALLOCATE each twitter id to one of the seven buckets
        RETURN twitter ids assigned to each bucket id they belong into
    METHOD reducer(bucketid, twitterid
        FOR LOOP through all twitterids within buckets, combine together and sort within ascending order
            RETURN twitterid, '' symbol to symbolise blank

RUN bucket sorting ids

END
```

Discussion

I analyse the data through firstly determine the size of each object ids, it turned out that all ids were the same size of 18 digits long. To determine how I wanted to develop my buckets I also decided to check what starting numbers are contained within the ids as shown below:

```
leng = []
for t in tweets.find():
    leng.append(t["object"]["id"].split(':')[2][0])

leng = set(leng)
print(leng)
```

[4] ✓ 2.1s

```
{'2', '7', '3', '4', '5', '6'}
```

It included digits starting with 2, 3,4, 5, 6, 7. After that I thought I would do an extra step of determining how many included within each group.

```
two = 0
three = 0
four = 0
five = 0
six = 0
seven = 0
eight = 0
total = 0
for t in tweets.find():
    if t["object"]["id"].split(':')[2][0] == '2':
        two+=1
    if t["object"]["id"].split(':')[2][0] == '3':
        three+=1
    if t["object"]["id"].split(':')[2][0] == '4':
        four+=1
    if t["object"]["id"].split(':')[2][0] == '5':
        five+=1
    if t["object"]["id"].split(':')[2][0] == '6':
        six+=1
    if t["object"]["id"].split(':')[2][0] == '7':
        seven+=1
    total+=1
print({"Two" :two, "Three" :three, "Four" :four, "Five" :five, "Six" :six, "Seven": seven, "Eight": eight})
```

[67]

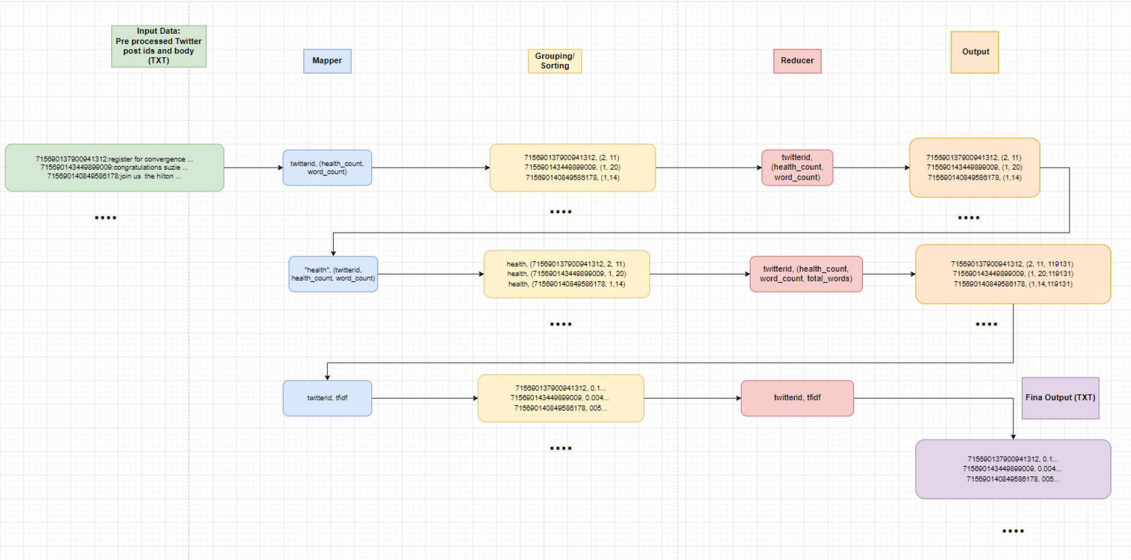
```
{'Two': 1, 'Three': 2, 'Four': 6, 'Five': 11, 'Six': 65, 'Seven': 9915, 'Eight': 0}
```

Based on this I decided to break the data into 7 buckets. As 6 would mean I would have to split by a complex number.

In this task I included 1 Mapper and reducer where the mapper developed the buckets for the sort and assigned each id into a bucket. The reducer then combines and iterates through all the ids while sorting them into ascending order.

Task 5 - TFIDF of Health in each tweet text

Flowchart



Pseudocode

```
Task 5 - TFIDF for health

BEGIN


CLASS health tfidf():
    METHOD STEPS
        CREATE links/relationships with each mapper and reducer together into steps (done through MRJOB.STEP)

    METHOD mapper frequency of each word within a document (data):
        SPLIT data, both twitter id and post text
        STORE Twitter id from split element 0
        STORE Post text from split element 1
        STORE length of words in each document
        FOR LOOP through words:
            IF words contains 'health':
                THEN RETURN Twitter id and Document word count

    METHOD reducer sum of frequency of words(keys, values):
        STORE values into a variable and converts to a list
        STORE values element of 0 into word count
        STORE 1 into health count
        RETURN twitter id, (health count and word count within a list)

    METHOD mapper move data to values(twitter id and values):
        BREAK values into variables health count and word count
        RETURN "heath" and Twitter id, health count and word count into the values side

    METHOD reducer sum of the total health count within all documents(keys, values):
        STORE values into a variable and converts to a list
        STORE the len of values into a new variable called total health count
        FOR LOOP through values
            RETURN twitter id, (health count, word count, total health count into a list)

    METHOD mapper calculating the tfidf for health in each document(twitter id, values)
        BREAK values into all value variables from previous reducer
        STORE 10000 to the total documents variables
        CALCULATE the TF by dividing health count by word count
        CALCULATE the IDF by getting log10 of total documents divided by the total health count
        CALCULATE the TFIDF through multiply the TF  by the IDF
        RETURN Twitter id and the TFIDF of the word health

    METHOD reducer displaying the TFIDF (key,value):
        RETURN key (Twitter id), value TFIDF

RUN Health tfidf

END
```

Discussion:

Firstly, the tfidf is the result of multiplying the Term Frequency by the Inverse Document Frequency. Where:

- Term frequency = frequency of word (health)/ the total number of words in the document
- Inverse Document Frequency = log(number of documents/number of documents containing word (health)

I learnt this concept in the unit COMP3220.

I Chose three mappers and reducers because:

- Mapper 1 calculates the occurrence of health in each document and word count of the document for only documents that contain health.
- Reducer 1 passes the data from the mapper, then makes sure both health count and word count are combined into a list under values while twitter id is the key.
- Mapper 2 based on our results from previous step combines the key and value into a value for the reducer.
- Reducer 2 then based on the values iterates through the posts and calculates the total number of words within all documents.

- Mapper 3 we now have all the values we need to calculate the tfidf (we already know that total number of tweets in the document is 10000. Once we got the tfidf we yield the twitter id and the tfidf value for each document based on the word health.
- Reducer 3, we iterate through list and yield it to our text file with the help of next ().

Overall, I was on the right track however I was unable to grab the total number of times health appears in each document so I manually assigned the health value to 1 for the tf score, this is the closest results I can get if we assume health appears in each document once (more founds within the Task 5 section of the Jupiter notebook).

References

- [1] DataFlair Team, 2018. Available at: https://data-flair.training/forums/topic/which-sorting-algorithm-is-used-in-mapreduce/#:~:text=Reducer%2D%20Merge%20sort%20is%20used,here%20is%20the%20merge%20sort.