CSE 587 Data-intensive Computing Spring 2014
Aditya Sawe (adityamu) - 50097199
Rahul Singh (rsingh33) - 50097213

BIG- DATA CONTENT RETRIEVAL, STORAGE AND ANALYSIS FOUNDATIONS OF DATA-INTENSIVE COMPUTING.

Link to Twitter Dataset: https://www.dropbox.com/sh/ac92gzd0siffd5x/NmKezFkz0i

Part 0. Tweet Collector

This part is more of a preprocessing step wherein we run the tweet collector program to get the tweets on the files. We have collected the tweets for several hash tags like #Obama, #president, #NBA , #IPL, #MH370 etc. We have collected these tweets in different files for easier management and handling. Our tweet collection size is more than 2 gigs.

Part 1. Top Trending Hash Tags

This part is about finding the top trending Hash Tags which occur more frequently than others. This part is basically done in four phases as follows:

1. Cleaning the Data
   Which includes eliminating all the symbols and stop words that occur more frequently in documents like the articles a, an, the and other words like and, go, to etc. and the symbols like “+”, ”-”, ”/”, ”*”, ”?”, ”=”, ”~” etc.

2. Running the Word Count
   Which includes running the word count program on the cleaned data it will give you a document with all the words and their respective counts.

3. Parsing the Document
   Which includes parsing the word count document in four different documents UserNames which contains the user names of the users, WordCount which contains the words and their counts, HashTags which contains all the HashTags and their counts, Trends which contains top trending words including both the hash tags and words.

4. Sorting the Trends Document
   Now it’s time to get the most trending words and hash tags so for that we pick top 5 entries for the top 5 trending words, this is done by sorting all the documents.

Implementation:

This part has a straightforward implementation we have followed the four steps given above. First part i.e cleaning the data is very basic we have picked all the stock words and symbols we possibly could and replaced them with white spaces, after that we ran the word count program which just counts the words in the mapper and emits them to reducer where in individual counts are added and emitted. In the third step we just segregate the words, hash tags (#) and users (@) and the trends file contains both the hash tags and words and in the last step we sort the words to get the trends.

Output

On running the code on the dataset of 500 MB the top 5 HashTags obtained are
#MUFC
#NEWS
#CLIPPERS
#DONALDSTERLING
#PLAYOFFS

On running the code on the dataset of 500 MB the top 5 most occurring words obtained are
CLIPPERS
NEWS
US
OWNER
NBA

On running the code on the dataset of 500 MB the top 5 Trends obtained are
CLIPPERS
NEWS
#MUFC
OWNER
#NEWS

On running the code on the dataset of 500 MB the top 5 users whose name occurred the most are
@MagicJohnson
@Read_Liam_Payne
@cjbycookie
@ManUtd
@LilTunechi

Part 2. Pairs and Stripes

Pairs Approach:

Documents containing tweets and the corresponding contents make up the input
key-value pairs. The mapper processes each input document, one line at a time, and emits intermediate key-value pairs with each co-occurring hash tags as the key and the integer one (i.e the count) as the value. This is straightforwardly accomplished by two nested loops: the outer loop iterates over all words (the left element in the pair), and the inner loop iterates over all neighbors of the first word (the right element in the pair). The neighbors of a Hash Tags can either be defined in terms of a sliding window or some other contextual unit such as a sentence.

Implementation:

Basic implementation follows the standard algorithm given in Lin and Dyer's book with a slight modifications and optimizations here and there.

In Mapper:

The program starts with a mapper reading from the tweets document line by line, splitting the line and storing it in an array, after that each line is sorted on the basis of the left word, this is an additional step we do to make sure there is no redundancy that means we don't end up considering the same word in different orientation more than once.

For example
a b = b a we should consider it only once.

Till now all we have done is arranging the data in particular order. Main functionality starts with two for loops one reading each hash tags and the second reading all its neighbors and emitting them to the reducer.
In order to support this implementation and reduce clutter in mapper class we have created an extra class called Wordpair.
Wordpair class does nothing but creates the pairs of hash tags i.e gives certain functionalities like giving the count of hash tags, gives the pairs of co-occurring hash tags , and method to compare to key value pairs.

In mapper we also form a special type of key value pair where key is appended with the * to make it special and different from the normal key value. It is done for the calculation of relative frequencies which is done in the reducer.

In Partitioner:

So this is emitted to the partitioner from mapper which basically does a simple functionality of mapping all the same keys to same reducer by calculating their hashes and comparing them and it computes hashes in order to distribute the data equally to reducers where reducer begins its functionality.

In Reducer:

The drawback of absolute counts is that it doesn't take into account the fact that some words appear more frequently than others. Some hash tags may co-occur frequently with some other hash tag simply because one of the words is very common. A simple remedy is to convert absolute counts into relative frequencies. In order to calculate the relative frequency we have modified the pairs approach a little bit.
In order to achieve this we created a data structure to solve the problem of storing the counts of one particular key i.e a hash tag. We are doing this because it was really hard to get the relative counts of hash tags which we will use as the denominator of the relative frequency formula, where the numerator will be the occurrence of a pair. So here in the mapper what we do is we mark every key with an * (asterisk) so that we know that this is a special case for relative frequency and value here is the count of that particular key with all the neighbors. So what happens is first the reducer checks what type of key value pair we have received if it is a key with * it will store in into the data structure and keep on doing that till it receives all the keys with * for that particular key and ultimately it will sum all the values and emit the sum.

If the key received by the reducer is not special that is it does not contain a * then reducer will normally sum all the co-occurrences and emit them.

Sample Output

```
word=#B674 neighbor=#Mikimoto    Count= 1  Relative Frequency= 1.0
word=#BAE neighbor=#FBI    Count= 10  Relative Frequency= 0.3333333333333333
word=#BAE neighbor=#Farnborough    Count= 10  Relative Frequency= 0.3333333333333333
word=#BAE neighbor=#NCA    Count= 10  Relative Frequency= 0.3333333333333333
word=#BAH neighbor=#BBC    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#BBCNEWS    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#BYU    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#Bath    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Belfast    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Bradford    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Breaking    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#BreakingNews    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#Bristol    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Burlington    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#CNN    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#California    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Carlisle    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Cleveland    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Columbia    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#Cougars    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Derby    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Dundee    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#FOX    Count= 4  Relative Frequency= 0.03571428571428571
word=#BAH neighbor=#Friday    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Greensboro    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Harrisburg    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Illinois    Count= 1  Relative Frequency= 0.008928571428571428
```

```
word=#BAH neighbor=#Indianapolis    Count= 4  Relative Frequency= 0.03571428571428571
word=#BAH neighbor=#Iran    Count= 6  Relative Frequency= 0.05357142857142857
word=#BAH neighbor=#Kingston    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#LONDON    Count= 3  Relative Frequency= 0.026785714285714284
word=#BAH neighbor=#Lansing    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Lebanon    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Louisville    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Maryland    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#MinnesotaTimberwolves    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Mumbai    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Nebraska    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#NewOrleans    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#NewYork    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#Newark    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Northwest    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#OklahomaCity    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Oxford    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Qatar    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Qom    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Raleigh    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#SanAntonio    Count= 4  Relative Frequency= 0.03571428571428571
word=#BAH neighbor=#SaudiArabia    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#SecKerry    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Sheffield    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Southampton    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Spain    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Syria    Count= 3  Relative Frequency= 0.026785714285714284
word=#BAH neighbor=#Wales    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Wichita    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Wisconsin    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#Woman    Count= 3  Relative Frequency= 0.026785714285714284
word=#BAH neighbor=#Yahoo    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#York    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#android    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#egypt    Count= 3  Relative Frequency= 0.026785714285714284
word=#BAH neighbor=#google    Count= 3  Relative Frequency= 0.026785714285714284
word=#BAH neighbor=#googlenews    Count= 5  Relative Frequency= 0.044642857142857144
word=#BAH neighbor=#health    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#ipad    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#iphone    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#media    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAH neighbor=#newsfeed    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#palestine    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#politics    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#roid    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#sms    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#socialtimes    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#uae    Count= 1  Relative Frequency= 0.008928571428571428
word=#BAH neighbor=#wonder    Count= 3  Relative Frequency= 0.026785714285714284
word=#BAH neighbor=#world    Count= 2  Relative Frequency= 0.017857142857142856
word=#BAMTO neighbor=#leapmotion    Count= 1  Relative Frequency= 1.0
word=#BATB neighbor=#BATBENDS2NEEDSS3    Count= 1  Relative Frequency= 4.084967320261438E-4
word=#BATB neighbor=#BATBEndS2NeedS3    Count= 25  Relative Frequency= 0.010212418300653595
word=#BATB neighbor=#BATBEndS2NeedsS3    Count= 2  Relative Frequency= 8.169934640522876E-4
word=#BATB neighbor=#BatBEndS2NeedS3    Count= 1652  Relative Frequency= 0.6748366013071896
word=#BATB neighbor=#BatBS2EndNeedS3    Count= 1  Relative Frequency= 4.084967320261438E-4
word=#BATB neighbor=#BatBWriters    Count= 1  Relative Frequency= 4.084967320261438E-4
word=#BATB neighbor=#BatBnevergiveup    Count= 1  Relative Frequency= 4.084967320261438E-4
word=#BATB neighbor=#BatbEndS2NeedS3    Count= 746  Relative Frequency= 0.3047385620915033
```

```
word=#BATB neighbor=#BatbEndS2NeedsS3    Count= 2  Relative Frequency= 8.169934640522876E-4
word=#BATB neighbor=#Beasties    Count= 1  Relative Frequency= 4.084967320261438E-4
```

Stripes Approach:

Like the pairs approach, co-occurring word pairs are generated by two nested
loops. However, the major difference is that instead of emitting intermediate key-value pairs for each co-occurring word pair, co-occurrence information is first stored in an associative array, denoted H. The mapper emits key-value pairs with words as keys and corresponding associative arrays as values, where each associative array encodes the co-occurrence counts of the neighbors of a particular word (i.e., its context). The MapReduce execution framework guarantees that all associative arrays with the same key will be brought together in the reduce phase of processing.

Implementation

We have implemented it the same way as the pair approach it's just that before we emit the key value pairs we aggregate them into a MapWritable this approach is better because here the number of key value pairs emitted is way less than in pairs approach so it makes it more efficient than pairs.

In Mapper

The program starts with a mapper reading from the tweets document line by line, splitting the line and storing it in an array, after that each line is sorted on the basis of the left word, this is an additional step we do to make sure there is no redundancy that means we don't end up considering the same word in different orientation more than once.

The mapper here scans through the array and counts the particular key (Hash Tag) and stores it into the MapWritable (which is a data structure explained below) without emitting it directly. When the line is complete, then the MapWritable which contains the associated neighbors and their counts is send to the reducer as value along with the considered Hash Tag as the key.

Major advantage of this approach is that you don't have to distinguish the keys.
You don't have to make special keys with * in order to identify them as Relative frequency calculators instead the MapWritable itself serves the purpose.
This idea is made clearer in the given example.

One instance of MapWritable is {<B> (10),
<C> (12),
<Z> (100)}
Where <B, C, Z>are Co-occurring Hash Tags and the numbers are their respective counts. Remember this is just the value of one emit of mapper this is in regard with some key say <A>.

This examples makes it clearer that you are already sending the local counts for relative frequency and you just have to add them in Reducer to get the denominator which is explained below in Reducer. Now you are transmitting everything to the reducer all the information it needs to calculate the relative frequency.

In Reducer

Here in reducer we create a new MapWritable in order to store the keys and their corresponding aggregating counts i.e when we receive the MapWritable from the mapper we extract the key say <A> from the above example and we keep on taking the counts from co-occurring neighbors of <A> in this case <B,C,Z>'s counts which is 10, 12, 100 and add them together which gives the total count of A used to calculate its frequency and we also get the local count of co-occurrence by counting the total number of MapWritable's received from Mapper to Reducer in respect of each key which we do with help of a counter variable. For the example above it is <A, B> is 10, <A, C> is 12 so on and so forth. The above results are then emitted by the Reducer.

Sample Output

```
#AP	#NewOrleans :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Burlington :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Qom :  Count= 8.0 : Relative Frequency= 0.04938271604938271
#AP	#uk :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#ArabSpring :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Tampa :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#Zanjan :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#TMZ :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Cairo :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#EPA :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Canterbury :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#Yahoo :  Count= 4.0 : Relative Frequency= 0.024691358024691357
#AP	#Kansas :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#CNN :  Count= 3.0 : Relative Frequency= 0.018518518518518517
#AP	#BYU :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#googlenews :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#Bradford :  Count= 8.0 : Relative Frequency= 0.04938271604938271
#AP	#Wakefield :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#google :  Count= 10.0 : Relative Frequency= 0.06172839506172839
#AP	#Wichita :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#sms :  Count= 3.0 : Relative Frequency= 0.018518518518518517
#AP	#SantaFe :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Preston :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#android :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#CampAshraf :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Ardebil :  Count= 3.0 : Relative Frequency= 0.018518518518518517
#AP	#chicago :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Nevada :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#Benghazi :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#IRS :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#iphone :  Count= 3.0 : Relative Frequency= 0.018518518518518517
#AP	#SecKerry :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#reuters :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#Syria :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Vermont :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#Extortion17 :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#Obama :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP	#BBCNEWS :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP	#politics :  Count= 7.0 : Relative Frequency= 0.043209876543209874
```

```
#AP     #MinnesotaTimberwolves :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Bahrain :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Mississippi :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Midwest :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP     #Nottingham :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #DonaldSterling :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #York :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #palestine :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #StAlbans :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #News :  Count= 13.0 : Relative Frequency= 0.08024691358024691
#AP     #Iraq :  Count= 3.0 : Relative Frequency= 0.018518518518518517
#AP     #Iran :  Count= 8.0 : Relative Frequency= 0.04938271604938271
#AP     #dandolaliga :  Count= 12.0 : Relative Frequency= 0.07407407407407407
#AP     #Woman :  Count= 3.0 : Relative Frequency= 0.018518518518518517
#AP     #Florida :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Missouri :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Salisbury :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #SouthDakota :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #AlJazeera :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #LosAngeles :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #NewHampshire :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP     #Georgia :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Harrisburg :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Olympia :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Fargo :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #LeMonde :  Count= 2.0 : Relative Frequency= 0.012345679012345678
#AP     #Winchester :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Greensboro :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #oman :  Count= 1.0 : Relative Frequency= 0.006172839506172839
#AP     #Cork :  Count= 6.0 : Relative Frequency= 0.037037037037037035


#APA14     #planning :  Count= 3.0 : Relative Frequency= 0.3333333333333333
#APA14     #CycleAtlanta :  Count= 1.0 : Relative Frequency= 0.1111111111111111
#APA14     #aging :  Count= 3.0 : Relative Frequency= 0.3333333333333333
#APA14     #CPlan :  Count= 1.0 : Relative Frequency= 0.1111111111111111
#APA14     #goblue :  Count= 1.0 : Relative Frequency= 0.1111111111111111

#APA2014     #bantamtowns :  Count= 3.0 : Relative Frequency= 1.0

#APBootCamp     #Bestteachever :  Count= 1.0 : Relative Frequency= 0.5
#APBootCamp     #RhinoZoo :  Count= 1.0 : Relative Frequency= 0.5
```

Part 3.   K-Means

K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done.

Implementation

In our implementation we start with very basic values of k i.e 1, 2, 3 as we don't know what the centroids are going to be , so we keep on running the iterations until the 3 values of k converges to the optimum values and do not change further. We want to find 3 centroids so we take 3 values of k.

In Mapper

Mapper doesn't do significant amount of work here. Basic job of mapper is to read from 2 different files as follows
First file contains number of followers of each user under consideration. This file is formatted in such a way that each row contains only one count of a particular user i.e for one user and the count for the next user is in the next line.
Second file contains the different values of k which are read by mapper and being updated after each iteration of reducer with the optimal values of k.

So exact functioning of mapper is it reads the values of three k's from centroids file and calculates the distance of each k with the count it reads from the count file which is fed into it and emits the count as the value and the value of k for which the distance is least as the key to the reducer.

In Reducer

Reducer in K-Means is even more simple all we do is the we extract the count received for each of the k from the mapper we sum them and take an average which is done by a counter variable and we emit the average as key and null as value which in updates the value k's in centroids file this is repeated unless the convergence is reached.

Use of Counters

The counters are used till the time you want to get the desired output counters will make your program run.

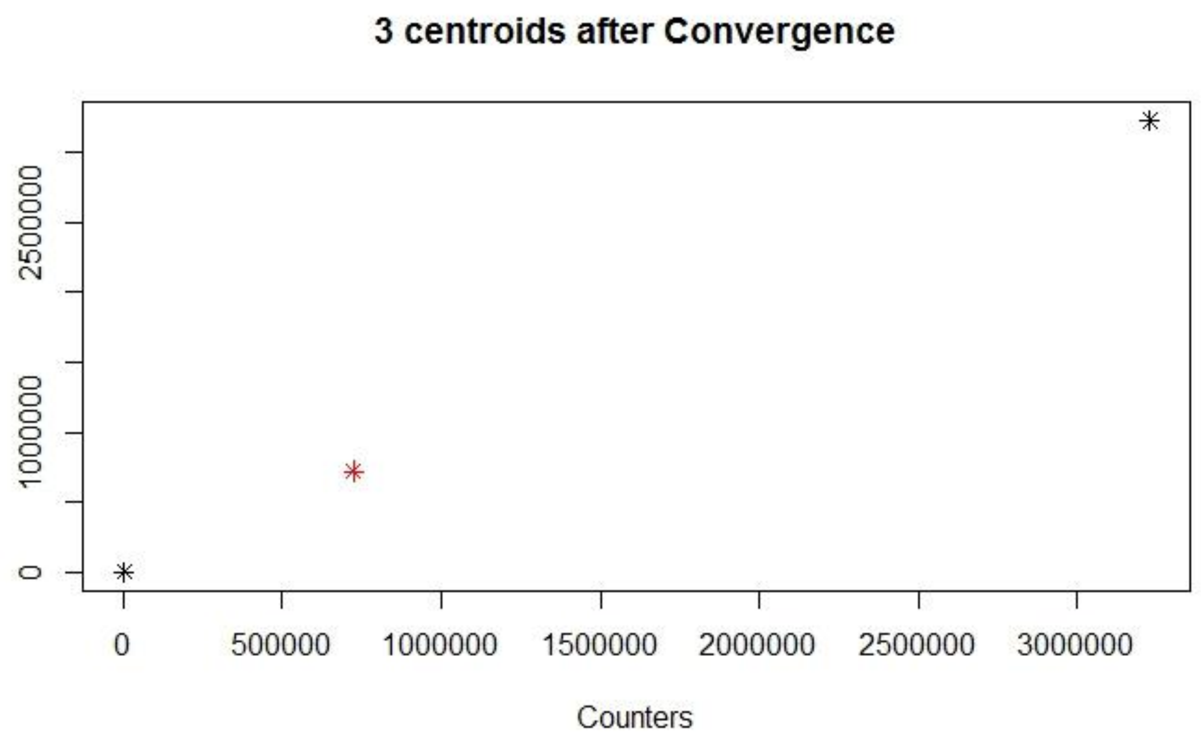
We have used the counters in this and the next part.
public enum MyCounters {
Counter;
}
To get the counters in driver class we have used

Job.getCounter

In reducer key we have previous and we calculate new centroids based on all the values if the value of the previous centroid is not equal to current centroid the counters will be incremented

The job will keep executing as long as the counter value stays above 0.
If key is not equal to average MyCounters is incremented.

Output
On running the K-Means algorithm on a fairly small dataset consisting of 2000 users the centroids obtained are
K = 0.456
K = 19.2348
K = 982.826



The above figure is generated using R for complete dataset and is multiplied by 1000 for nice visualizations.

Part 4: Shortest path

One of the most common and well-studied problems in graph theory is the single-source shortest path problem, where the task is to find shortest paths from a source node to all other nodes in the graph (or alternatively, edges can be associated with costs or weights, in which case the task is to compute lowest-cost or lowest-weight paths)

In the fourth and final part of the project we have to find the shortest path using parallel breadth first search on the three input graphs with different infinity values provided.

In Mapper

Mapper does not do many fancy things it just takes in node_id which is the id of node and a data structure as adjacency list which contains all the connected nodes to the given node.
After that it stores the minimum distance from that node to other nodes in a variable for further computation and ultimately it emits two things first its node_id, its adjacency list and its distance and it also emits the its minimum distance +1 to all the nodes in its adjacency list.

In Reducer

In reducer on every iteration a new node is created and for each distances minimum distance is found. Reducer just selects the minimum of these distances received from the mappers for each of those destination nodes and transmits the counters and keeps on iterating until the convergence.

Shortest path if calculated for the three graphs provided:
Large Graph
Small Graph
Graph with Distance

Use of Counters

In this case the counters will be incremented till the minimum distance for all the nodes become non infinite. When a particular is considered in the reducer if the min distance bet it and initial node is infinite the counter will be incremented.

Output

For shortest path with graph containing distance the output after the first step was as follows

```
1	0 2:1::3:1
2	1 3:1::4:1::
3	1 2:1::4:1::5:1::
4	125 5:1::
5	125 1:1::4:1::
```

The above output contains infinite distance i.e 125, the counter makes the job run again and so at second iteration we get the final output

```
1	0 2:1::3:1
2	1 3:1::4:1::
3	1 2:1::4:1::5:1::
4	2 5:1::
5	2 1:1::4:1::
```

The output for the large graph is obtained after 18 map reduce iterations

The output for the large graph is as follows

```
1    0 2:
10    12 9:20:
100    14 99:107:
101    13 82:105:
102    14 70:103:
103    15 102:104:105:
104    16 75:103:106:
105    14 101:103:107:
106    17 104:108:
107    15 100:105:
108    18 106:109:110:
109    19 108:
11    5 12:
110    19 108:
12    4 11:13:22:
13    3 3:12:14:
14    4 4:5:13:15:
15    5 14:16:
16    6 15:27:
17    8 7:27:
18    13 19:35:
19    12 18:20:21:
2    1 3:
20    11 8:10:19:
21    13 19:
22    5 12:24:
23    8 32:
24    6 22:32:
25    8 26:32:
26    8 25:27:
27    7 16:17:26:28:33:
28    8 27:44:
29    17 37:38:
3    2 2:13:
30    16 45:
31    8 32:
32    7 23:24:25:31:40:41:46:
33    8 27:42:43:
34    17 49:
35    14 18:50:
36    16 50:
37    18 29:
38    16 29:45:53:
39    16 45:
4    5 14:
40    8 32:55:
41    8 32:47:
42    9 33:
43    9 33:44:48:
44    9 28:43:52:
45    15 30:38:39:61:
46    8 32:56:60:
47    9 41:
48    10 43:
49    16 34:50:71:
5    5 14:
50    15 35:36:49:58:
```

```
51    17 58:
52    10 44:
53    17 38:62:
54    11 66:
55    9 40:
56    9 46:68:
57    13 43:69:
58    16 50:51:72:
59    13 69:
6    10 7:
60    9 46:66:67:
61    14 45:65:
62    18 53:63:64:
63    19 62:
64    19 62:
65    13 61:80:
66    10 54:60:79:
67    10 60:
68    10 56:77:78:
69    12 57:59:77:
7    9 6:8:17:
70    13 82:102:
71    17 49:72:
72    17 58:71:73:74:75:
73    18 72:
74    18 72:
75    17 72:76:104:
76    18 75:
77    11 68:69:82:98:
78    11 68:81:
79    11 66:80:81:
8    10 7:20:
80    12 65:79:89:
81    12 78:79:95:
82    12 70:77:101:
83    18 85:
84    18 85:
85    17 83:84:86:
86    16 85:87:
87    15 86:88:
88    14 87:89:91:
89    13 80:88:90:
9    13 10:
90    14 89:93:
91    15 88:92:
92    16 91:93:
93    15 90:92:94:
94    14 93:95:96:
95    13 81:94:
96    15 94:
97    14 99:
98    12 77:99:
99    13 97:98:100:
```

Similarly the output for the small graph without distance is

```
1    0 2:3:
2    1 3:4:
3    1 2:4:5:
4    2 5:
5    2 1:4:
```

References

1. Lin and Dyer Textbook
2. http://codingjunkie.net/cooccurrence/
3. https://mahout.apache.org/
4. http://www.stackoverflow.com