 November 20, 2015



**DINH Quang-Huy**

DATA MINING

TWITTER PROJECT

# Introduction

In this project, we were given a couple folders of data from Twitter. The data are the tweet that were collected based on an event or a location.

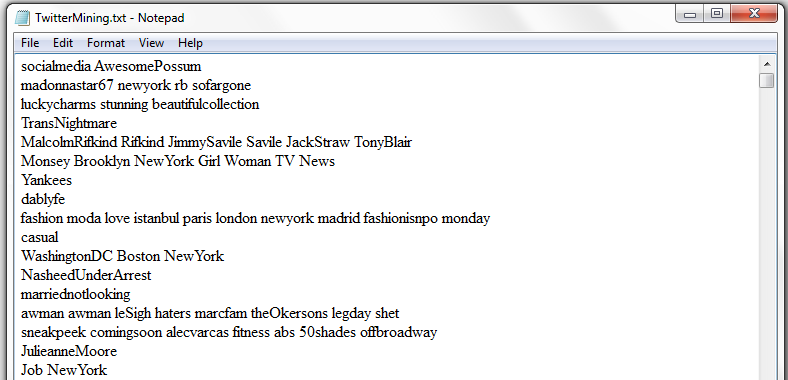
Our mission is to process those data, refine them, analysis and find out the trend that is hidden inside of those data.

The main tasks would follow the following steps:

1. Read data from file, refine and filter data.
2. Build an undirected graph from that refined data.
3. Find the dense subgraph(s)
4. Analysis the results.

# Filtering

This filtering step is the very first task that we have to do in the order to filter out the spamming tweets. The idea behind is that the spamming post (from spammers or robots) will try to replicate the post from other people to get the attention. Hence, they are likely to create the tweet with the hashtags are identical to others. Following the instruction of TA, we will filter out all the tweets which have the identical (in number and in word) hashtags. After that, we store the processed data to an output file that look like the below screen-captured photo:

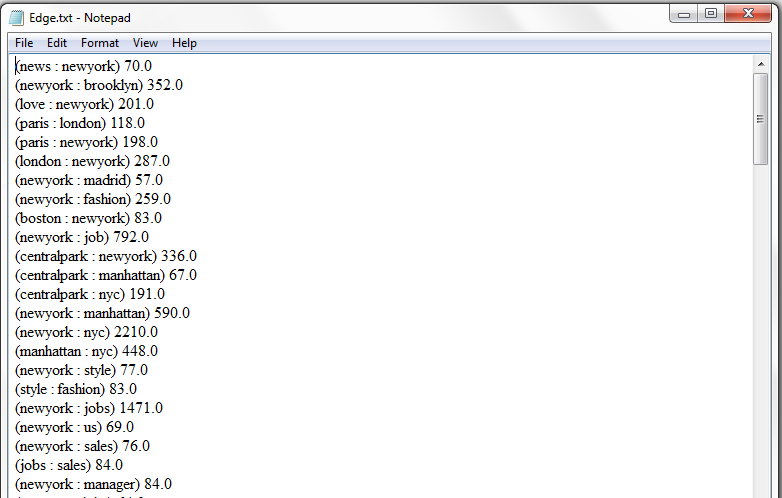


Moreover, we also store the Users’ id into another file. This user txt file will be used after when we need to count the distinct number of users that contribute into the dense subgraph.

# Build the graph

Using the JGraphT library, we were able to build the whole graph G for the data we had. This is a weighted graph that has weights on each of its edges. Hence, we have to calculate the weight for each edge from the data. For each pair of hashtag in the same tweet, we will count 1 for the weight of the edge created by those 2 hashtags.

Finally we will get a data structure in which we have the edge and the its weight.



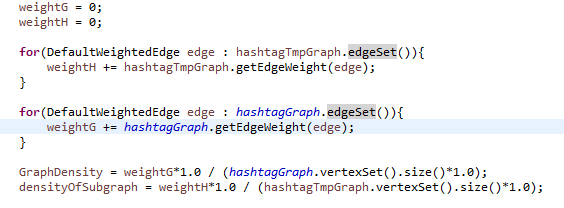
In order for an easier calculation, we calculate the weight of the node (vertex) in graph by the sum of the weight of all edge from/to it (this is an undirected graph, so we do not care “from” or “to”). And store the result into a hashmap, where the data is stored in the form of a pair key-value <K, V>.

# Find the dense subgraph(s)

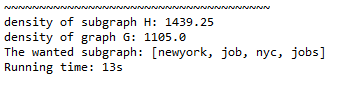
We decided to implement the finding function in 2 ways.

* One is to follow the greedy methods which is to delete from the smallest weight node to find the last nodes (number of nodes smaller than 11) that create the dense subgraph. For this method, we will apply the idea of parallel algorithm as in the slide of the Professor. But this method can only find a densest subgraph.
* The second one is a modification of Charikar’s algorithm for weighted graphs. It can obtain many dense subgraphs. The algorithm is as follows [1]:
  + At each step remove a random *edge* of lowest weight.
  + Then find all connected components
  + Recurse down on each component, and return the maximal density subgraph.
  + By repeated executions of the algorithm the hope is that different dense components will be revealed, that can overlap.

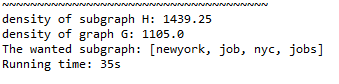
For whatever method we go with, we have to calculate the density of the big graph and the subgraph. We calculate by sum up the weight of all edges and divide by the number of nodes (this is a variant version of the density that we learned in class Number of edge/Number of vertex)



For the first method (parallel greedy), we have the running is quite fast (~14 seconds for the 1st folder (NewYorkOneWeek)). And we got the result as below:



The running time you see above is 13 seconds (exclude the time for reading and filtering data from file). If counted in, it should be 35 seconds:



The second method, however, has a big complexity and takes much time to finish. We can only run that algorithm with around 1000 tweets.

However, we think that if we apply the second method until we get a certain amount of nodes (e.g. 1000 nodes) then we turn into the first algorithm with greedy strategy, we will be able to speed up the algorithm.

GibHub link for our code: <https://github.com/qhdinh/TwitterMining>

[1] Paul Horn, “Finding dense components in weighted graphs” slide, 12-2-02