University of Southampton, School of Electronics and Computer Science
Foundation of Data Science - December 11, 2015
Analysis of unstructured microblogging data.

Nikolaos Perrakis
np4g15@soton.ac.uk
https://github.com/Iolaum/MongoDB_CW

Setting up our system.

We start by setting up our system. My laptop works in windows 7 64-bit. I download and install MongoDB version 3.0.7. We have python 2.7.10 installed already and it will be the programming language we will be using to complete the assignment. We will be accessing the database through two methods. Through the command prompt and through pymongo python package.

We download the relevant file and store them in a directory created for this course work. We also create a new project on GitHub and initialize it on the same folder. We configure gitignore so it does not include the database folders and any other irrelevant files that don't need to be submitted.

Data Preprocessing.

We have our data in a CSV format and we now need to clean them and make sure they are properly formatted before they are sent to the database. We split the data and create a smaller CSV file to get a preview of our data to familiarise ourselves with our database.

As a first step we run the script PreProcess1.py to make sure that everything in our csv file is properly readable by Python. We verify that every entry is an ordinary string according to python. Then we proceed to do some more in depth cleaning. From our inspection we saw that we had some lines in csv file that did not have the proper structure of 6 entries. that usually happened because the text entry of the tweets had the newline character. We also saw that some users had incorrect member id's.

To address those issues we created and run the PreProcess2.py script. For the first issue we handle each line differently if it has 6 entries or not. If it has 6 entries the entry is stored in the file output1.csv. If it doesn't we concatenate the entry with the next one until we get line with 6 entries (that we don't concatenate but handle properly). Improper entries are stored on file output2.csv for later processing[1]. For the second issue assigned a dummy id (which is numerical less than 1 but higher than 0) to differentiate them but to not have values that are the same with existing users.

Querying our Database.

Entering the main part of our coursework we describe the queries we made.

The first question was straightforward to implement with script Question1.py . All we had to do is implement mongodb's distinct method. Our results are 119218 distinct users.

The second question was a bit tricky until I found out how to implement mongodb's aggregate method. The implementation is on script Question2.py and the total tweets by the 10 most active users are 32344 which consist of 2.216% of the total tweets.

The third question we answered in script Quetion3.py . We sort according to the timestamp label and get the following results: Earliest Date: 2014-06-22 23:00:00 Latest Date: 2014-06-30 21:59:59

[1]The improper entries (427) are very few compared to the proper ones (1459434) consisting only 0.03% of the tweets. Because the questions we are asked to answer are about general trends their impact is negligible and we can ignore them.

For the fourth question there is a shortcut. We don't need to query the data because the time difference between all the tweets adds up to the difference between the first and last tweet. Therefore all we need to do in script Quetion4.py is to convert the dates to seconds and divide by the number of tweets minus 1. Our result is 0.4711 seconds averagely per tweet.

The fifth and seventh questions we answered together in script Question5+7.py . For both of them we have to process the length of the message. Therefore we decided to query the whole database once[2]. We introduce to variables which count the total length of the messages we have processed and the total number of hashtags we encountered. A minor problem we encountered is that some entries weren't usable by the length method immediately and had to be converted. They were only 45 and we count their length after converting them and counting them on a separate variable. None of them included hashtags. Our results are: Averaget Length: 71.38 Average Hashtags : 0.316

The sixth quesion we answered by using map reduce. We use separate queries for unigrams and bigrams. For unigrams we write a javascript script in the mapper to separate each word, clean it of special character's and emit it with value 1. For bigrams we create a list of the words instead of emmiting and on a subsequent for loop we select bigrams from the list and we emit them. Our results are:
the 368135 – in the 35934
to 324153 – of the 23761
a 299124 – for the 22853
I 283814 – on the 22825
and 184945 – to be 22347
you 179331 – 00 mm 16593
in 179008 – to the 15348
of 162046 – at the 14814
for 160072 – for a 14736
is 159087 – I have 14667

The eight question we answered with map reduce as well. It is my favorite implementation mapping 100 segments on numbers 0 to 99 of the area in question. Our results are:

Segment 27 with 323894 tweets and coordinates: MinLat 51.4 - MaxLat 52.4 MinLng -0.9 - Max Lng 0.1

[2]Map Reduce method may be more efficient but I hadn't understood it at the time and the query gets finished a lot faster compared to the (harder) queries where I use Mar Reduce.