## I. Histogram:

**1. What was your approach for building this histogram?**

The basic technique is to use fixed size min-heap, insert entries till we reach the max size of the heap. Now, for the subsequent entries compare the minimum value (stays at the root, since it is a min-heap) with the current entry, keep the one with higher value. Repeat the process for all entries in the file.

My plan was to show the top 10 likes in a histogram. The initial design has a fixed size min-heap (size = 10, node as a tuple (like\_text, like\_count) ) taking in the values as they come in. And In case we encounter the same tuple, I tried increasing the value by 1 using heap\_increase\_key(heap, old, new). If it is a new tuple, I was just pushing it on to the heap. But as I progress, I realized that this approach has a flaw. For instance, say heap\_size =5, and assume that the first five entries we encountered so far are distinct. Now, for the sixth entry we need to make a choice if they have the same value (like\_count) and at this point, we have no way to know how many likes would be encountered for each node further.

So I changed the approach to read the entire file and obtain the total like counts for each like. Now I iterate over this map data structure and insert each entry into my fixed size min-heap. In this approach, we can make a choice for sixth entry since we know the total like count. Please refer to histogram for top five likes in the given data <https://github.com/naveenkothamasu/DataEngineering/blob/master/bargraph_final.png>

**2. How does your approach scale to 1 terabyte of like and interest data?**

My plan is to break the file into multiple files. I wrote a script breakInputFile.py which breaks the given file into files with each consisting of 1000 lines. Since the maximum like count is an aggregate function, we can use MapR on Hadoop File System to run my program (histogram.py) on each data node and save the response in a resultFile. Now, we can run histogram.py again on this resultFile to obtain the final result.

## II. High-frequency Pairs

1. What was your approach for building this histogram?

The basic approach is the same as Section I (Maintain a fixed size min-heap and keep replacing the root if the value is lower compared to current value).

Here we do not need to generate all possible pairs of comments across users. The trick is to generate possible pairs of the likes within each line and keep them in a dictionary. Now, the approach is similar to Section I.

2. How does your approach scale as the number of likes increases linearly?

My plan is to break the file into multiple files. I wrote a script breakInputFile.py which breaks the given file into files with each consisting of 1000 lines. Since the maximum like count is an aggregate function, we can use MapR on Hadoop File System to run my program (histogram.py) on each data node and save the response in a resultFile. Now, we can run histogram.py again on this resultFile to obtain the final result.

## Source Code:

<https://github.com/naveenkothamasu/DataEngineering>

## Challenges:

|  |  |  |
| --- | --- | --- |
| Challenge | Solution | Constraints/ Assumptions |
| Identification of the language from its script | Language Detection Ex:Google Translate | Learning models for script recognition are not perfect. |
| Text is in foreign languages | Language Translation APIs | Language translation APIs are not perfect. Say s1 (base language) and s2 (foreign language) refer to same sentence in different languages. With the existing NLP techniques such as text classification, we can get better results as explained below (in my opinion ). The question is how likely is s2 referring to s1, call this “sentence-equivalent metric”, try to compute this metric for each sentence (since the likes here is relatively low) and pick the sentence with the highest sentence-equivalent metric. |
| Rephrased sentence (in the same language) | I did a project in my undergrad to map the key words in a sentence to SQL keywords at the backend, but as we know, this approach has a very limited scope. | In my opinion, this has been in the spotlight of NLP research for decades. Ex: Ontology, Semantic web |
| Conversion of symbols (smileys, sms lingo etc) into words | A look up map can be maintained to replace these symbols with the actual text |  |