ALGORITHMS FOR MASSIVE DATA PROBLEMS

Supervised By:
Dr. Benny George Kenkireth

Presented By:
Devaishi Tiwari(170101021)

OBJECTIVE

The objective of this thesis project is as follows,

- Introducing the field of massive data analysis and communicate its usefulness in different sectors.
- Discussing and implementing various existing algorithms related to massive data and evaluate their performance and correctness theoretically.

INTRODUCTION

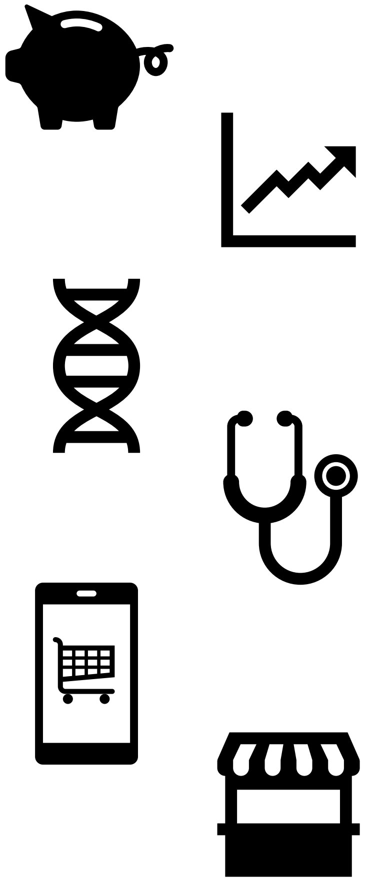
- At this day and age, data is growing faster than ever. According to IDC's study – "The Digital Universe in 2020", we would have approximately 40 trillion gigabytes data by the end of the year 2020.

- This rapid growth in the amount of data has led to existence of a new field called **Big / Massive Data Analysis**.

- According to Gartner, "Big Data are **high volume, high velocity,** or **high-variety** information assets that require new forms of processing to enable enhanced decision making, insight discovery, and process optimization."

APPLICATIONS

- Banking and Security:
  - Predicting stocks' and shares' prices.
  - Monitoring the banks and trading markets for fraudulent practices.
- Healthcare:
  - Finding out early-stage symptoms of various diseases.
  - Helping in efficient diagnosis of the patients.
- Customer Behaviour Analysis:
  - Analyzing the customer’s choices and behaviour for growth.
  - Enhancing the customer experience.



METHODS OF COMPUTATION

- In **Data Streaming** method, we assume that the data items arrive one at a time, instead of being stored inside the memory as a block.

- In **Data Sampling** method, we store only a small section of data into the memory. This small section is used, instead of the original data, to perform computations in less space.

- In **Data Sketching** method, we go through the data stream and quickly process it to generate an in-memory summary, called **sketch**, for the whole data.

STREAMING

COUNT OF DISTINCT ELEMENTS

- Let us assume that we have a data stream of length n, $\{a_1, a_2, a_3, \ldots a_n\}$ where a_i can take up any value from 1 to m.

- Some of the most intuitive approach will be as follows,
  - Create a bit-vector to record which of the m symbols have occurred, in the stream, so far - $O(m)$ **space**
  - Maintain all the distinct elements in a single set – $O(n.\log_2(m))$ **space**

- Note that we need **at least $O(m)$ space** for any deterministic algorithm to find out the count of distinct numbers

BETTER ALGORITHM

- We can approximate the count using the minimum element in the stream. Let D be the set of distinct elements present in the data stream. Let the minimum element present in D be **Min**. Now the expected value for Min will be

$$
\begin{aligned}
E(Min) &\approx \frac{m}{2} \; if \; |D| = 1 \\
&\approx \frac{m}{3} \; if \; |D| = 2 \; and \; so \; on... \\
&\approx \frac{m}{|D|+1}
\end{aligned}
$$

- Thus, the approx value of the count of distinct elements will be $|D| \approx \frac{m}{Min} - 1$

MAJORITY ELEMENT

- A **majority element** is an element that occurs **more than n/2** times in the stream.

- Let us consider the example of an election. Suppose there are **m** candidates and **n** number of votes, coming in as a stream of integers $\{a_1, a_2, a_3, \ldots a_n\}$ where a_i can take up any value from 1 to m.

- Some of the most intuitive approach will be as follows,
  - Store the number of occurrences of each symbol - $O(m.\log_2(n))$ space
  - Store all the distinct elements in a set, along with their frequency – $O(n.\log_2(n))$ space

BETTER ALGORITHM

Algorithm 1 Moore: Majority Element Algorithm

```
element ← 0
count ← 0
for each i do
    if i = element then
        count ← count + 1
    else
        count ← count − 1
    if count = 0 then
        element ← i
        count ← 1
```

- We store a single element and a counter.
- For each incoming element ai, we check if the element is equal to the stored element.
  - If both the elements are equal, we **increase the counter by 1**,
  - else we **decrease the counter by 1**.
- If the counter reaches 0, we **set the stored element to ai and the counter to 1**.
- The saved value, at the end of the stream, is the majority element.

MOST FREQUENT ELEMENTS

Algorithm 2 Misra-Gries: Frequent Elements Algorithm

$setElements \leftarrow \phi$
$count \leftarrow 0$
for *each* i **do**
    if $a_i \in setElements$ **then**
        $count_i \leftarrow count_i + 1$
    else if $|setElements| < k + 1$ **then**
        $setElements \leftarrow setElements \cup \{a_i\}$
        $count_i \leftarrow 1$
    else
        for *all* $j \in setElements$ **do**
            $count_j \leftarrow count_j - 1$
            if $count_j = 0$ **then**
                $setElements \leftarrow setElements - \{j\}$

O(k.logn + k.logm) Space

SAMPLING

SAMPLING IN MATRIX ALGORITHMS

- Standard matrix algorithms like matrix multiplication, single value decomposition, linear regression etc. require around $O(n^3)$ time for $n \times n$ matrices.
- As the size of the matrix increases, these algorithms become slow. Hence, we need a faster method to estimate the results.
- In which case, we sample the rows and columns of the matrix to create a random sub-matrix to replace the original large matrix.
  - How to choose these rows and columns?

LENGTH SQUARED SAMPLING

- An intuitive approach could be to consider **all the columns equally likely**, i.e.select the columns uniformly at random.
- The **Squared Length** of a column c in a n × m matrix A is defined as,

$$S.L.(c) = \sum_{i=1}^{n} a_{ic}^2$$

- In length squared sampling, we assign probabilities that are **proportional to the squared lengths of the column**. This way, significant columns will have more probability as compare to insignificant columns

MATRIX MULTIPLICATION

- Select **s** number of columns, one at a time, in s independent trials. Let p_k be the probability of choosing the k^{th} column, based on length square sampling.

$$P = \left[\frac{A(:,k_1)}{\sqrt{sp_{k_1}}}, \frac{A(:,k_2)}{\sqrt{sp_{k_2}}}, \ldots \frac{A(:,k_s)}{\sqrt{sp_{k_s}}}\right] \qquad Q = \left[\frac{B(k_1,:)}{\sqrt{sp_{k_1}}}, \frac{B(k_2,:)}{\sqrt{sp_{k_2}}}, \ldots \frac{B(k_s,:)}{\sqrt{sp_{k_s}}}\right]^T$$

- On substituting matrices A and B by matrices P and Q respectively, our matrix multiplication results in PQ such that,

$$E(\|AB - PQ\|^2) \leq \frac{\|A\|^2 \|B\|^2}{s}$$

SKETCHING

SIMILARILITY IN WEB PAGES

- Instead of caching millions of web pages, we can use sketching to create a sketch of the web pages in the memory. This sketch can be used to estimate the similarity between two or more web pages.

- Web pages can be considered as a **string of words**. We simply take the sequence of words in the page as a string, viewing each word as a character.

- This string can then be easily represented as a **set of substrings of some length k**. Thus, if we find out a way to find resemblance in sets, our work is done!

RESEMBLANCE IN SETS

$$resemblance(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Since size of A and B will be large, calculating the resemblance will become a tough task. One way to approximate is to select a subset of these elements as follows,

- **Option-I**: Randomly choose k elements from both A and B.
- **Option-II**: Rename all elements using a random permutation and choose k smallest elements from both A and B.
- **Option-III**: Select all the elements that are divisible by some randomly chosen integer m from both A and B.

FUTURE WORK

The future work will include the following tasks,

- Replicating a large data stream to implement the streaming algorithms listed above.
- Implementing approximate matrix multiplication using sampling method.
- Implementing a sketching method to estimate the resemblance between two or more documents.
- Analysing the performance of these algorithms and comparing it with the theoretical results.

THANK YOU