Project 3 - Mining Data Streams
TDT4305 - Big Data Architecture (Spring 2025)

DUE: **April 18, 2025 2:00 PM (kl. 14)**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk with other students on homework assignments.
  - You can share ideas but not code; you must submit your code individually or as a group of two.
  - All submitted codes will be compared against all codes submitted this semester and in previous semesters.
- **Programming:**
  - You are required to use Python 3.8 or newer versions for this assignment.
  - You can use external libraries for this project.
  - Ensure the dataset is in your project's "/data" directory.
  - Implement the missing code under ## *To-do!* in the Notebook provided.
  - Ensure the Notebook runs without errors, then save the Notebook for submission.
  - Use the markdown function to write in the Notebook.
- **Submissions:**
  - Note that if you submit your work multiple times, the most recent submission will be graded.
  - You can do this project individually or in a team of two people.
  - Only include the Notebook file in the .zip; do not include anything else, such as dataset or a PDF report.
  - When you are done, save the Notebook file in a single .zip file with the 'GroupNumber_DGIM.zip' and upload the .zip file on BlackBoard via the submission link for the assignment. Also, include the names of team members in the Notebook (or only your name if you are working on the assignment individually).

  Important: *Post all questions on Piazza and only send me an email in case of (rare) anomalies!*

Introduction

This project represents the conclusion of the 3-part series for the TDT4305 - Big Data Architecture course (Spring 2025). Like all culminations, this project aims to build upon concepts covered in previous projects and introduce relevant algorithms discussed in subsequent lectures. The project is divided into five distinct tasks. The first three tasks focus on **Mining Data Streams**, while the last two tasks center around **AdWords** and **Recommendation Systems**, respectively. Each task has a corresponding *Case*, making a total of 5 engaging *Cases*. Provide a concise response to each *Case* in the Notebook file.

---

1 DGIM (*25pts*)

In mining data streams, we are faced with two fundamental limitations. First, we do not know all the data in advance, and the system cannot store the entire stream accessibly. Despite not knowing all the data in advance, we aim to answer useful queries over a sliding window N. For example, counting the number of 1s in the last k elements of a stream consisting of 0s and 1s without storing the entire window N. An obvious solution will be to maintain a counter of all 1s and 0s seen so far and use this to answer query k. However, this solution has an implicit assumption that the data stream has a uniform distribution. In reality, this is rarely the case as data streams are non-stationary (distribution changes over time). DGIM provides an ingenious way to do this without the uniformity assumption while storing $\mathbf{O}(\log^2 \mathbf{N})$ bits rather than the entire window **N**.

1.1 DGIM Algorithm

Implement a DGIM algorithm that takes a stream path (> 1 million bits of 0s and 1s) and window size N as parameters and returns a tuple containing a list of buckets with timestamps and the end timestamp (see Notebook for output format). The position of each bucket in the list corresponds to the size of the bucket (Number of 1s). Specifically, the first bucket is size 1, the second is size 2, the third is size 4, and so on. For example, a list of buckets of this form **[[3, 7], [16, 23], [41], [22, 1]]** can be interpreted as having two buckets of size 1 with end-time of 3 and 7, two buckets of size 2 with end-time of 16 and 23, one bucket of size 4 with end-time of 41, and two buckets of size 8 with end-time of 22 and 1. The end-timestamp is the end-time of the last entry of size 1 in our bucket; for this example, the end-timestamp is 7.

1.2 Query the Bucket

In the previous step, we created a list of buckets with timestamp information about the count of 1s in a sliding window. If our aim is to count the entire 1s in a sliding window, then we have enough information from our list of buckets to answer that query. However, we want to take a step further to implement a function that, given an arbitrary query k, signifying the last k bits, can provide the total count of 1s within k bits provided **k < N**. Write a function that takes the output of the DGIM algorithm, the window size **N**, and a query **k** as input and returns the total count of **1s** in the last **k** bits.

Case 1: DGIM stores $\mathbf{O}(\log^2 \mathbf{N})$ bits rather than the entire window **N**. Knowing that the number of buckets used in the DGIM algorithm is $\log \mathbf{N}$, why then is the space complexity $\mathbf{O}(\log^2 \mathbf{N})$ rather than $\mathbf{O}(\log \mathbf{N})$?

---

2 Bloom Filter (*20pts*)

In data stream processing, while DGIM is useful in answering queries over a sliding window, Bloom filters, on the other hand, is a technique for rapidly (with a false positive trade-off) selecting elements with a specific property **x** from a stream. Imagine we own a startup website - TDT4305 - with a continuous stream of usernames being created. We don't have enough memory to store all usernames in memory or hash table. In this task, we want to use the Bloom filter to solve a crucial website problem, which is assigning usernames without having to store the entire stream of usernames seen so far in memory.

2.1 Create Bloom Filter

To create a bloom filter, we have provided about 500k usernames taken so far by users in our hypothetical website in the data_file.csv. Using this stream of usernames, implement a bloom filter function that takes as parameters an empty bloom filter array, a list of hash functions, and a data path and outputs an updated bloom filter with 0s and 1s. Use the equation below to create **h** number of random hash functions

$$H_h(s) = \left(\sum_{i=1}^{len(s)} s[i] \cdot p^i \right) \mod N \tag{1}$$

In this equation:

$H_h(s)$ represents the hash function.

$s[i]$ represents the individual elements of the username sequence s.

p^i represents a random prime number. Make sure that each hash functions have a different p.

N represents the size of the Bloom filter.

2.2 Verify Usernames

Implement the function "single verify usernames," which takes the updated Bloom filter (**B**), a list of hash functions, and a new username, then returns a code of 0 or 1 (1 - username taken and 0 - username available). Similarly, implement a function "group verify username" that accepts the updated **B**, list of hash functions, and file path containing streams of new usernames and returns the percentage of usernames seen so far. We have provided two test files for this verification. Test file 1 consists of approximately 100K usernames created from a subset of the data stream used in updating the Bloom filter. In contrast, test file 2 contains about 100k usernames never seen before by the Bloom filter (In

other words, available usernames - this is helpful for checking the false positive rate).

Case 2: Your favorite social media platform uses Bloom filters to assign new usernames. When you try to obtain a new username, 'Kazeem,' it says, "username is taken." Is there a possibility that the username is available? If yes, how do you present your case to the site admin?

After a bit of ingenuity on your part, you finally came up with 'KazeemTDT4305", and fortunately, it says this is available. However, a friend sitting next to you prompted that they know a friend of a friend that has this exact username on the same platform. Is this possible? Knowing that this friend has never taken a CS course on Big Data and has probably always doubted your brilliance. Clearly, this is the moment you have been waiting for to brandish your TDT4305 Big Data prowess. How can you effectively convince your friend that he is mistaken?

While you are at it, show a bit of altruism by solving this case for a Google user Google Support Center: https://support.google.com/mail/thread/14519892/creating-account-says-username-is-taken-but-it-is-available?hl=en

---

3 Flajolet-Martin (*10pts*)

The final query in the domain of data stream we are likely to ask is the number of unique users who visited our TDT4305 website this month. A basic solution is to maintain a hash table of all the distinct users seen so far. But as mentioned before, we are a startup, and consequently, we do not have the storage to maintain the set of elements we have seen so far. If we accept that the count may have a little error as a tradeoff for limited storage, then the Flajolet-Martin approach offers an elegant way to estimate the number of distinct users that visit our website. Implement a function to perform the Flajolet-Martin algorithm on an input stream and return the count of distinct elements in the input stream.

Case 3: How do we increase precision while using the Flajolet-Martin algorithm?

---

4 AdWords (*25pts*)

As a startup with talented minds working their hearts out, we have been able to leverage concepts covered in the Big Data course to manage resources efficiently to launch a sustainable website. Now it's time to make some money! Assume that our hypothetical TDT4305 website is beginning to gain some traction, and now different advertisers are vying for spots in our growing startup with a predetermined budget. Our aim is to make as much revenue as possible from our devoted users' queries, given a company's list of keywords and a budget.

4.1 Greedy Algorithm

We begin by implementing a greedy algorithm that takes a dictionary of a company as keys and a list of keywords and budget as values along with a sequence of corresponding user queries and returns the expected revenue generated from the advertising process (see the Notebook for the input dictionary format)

4.2 Balance Algorithm

No doubt, we have been able to generate some revenue using the greedy approach, but can we perform better? The Balance algorithm offers a unique way to improve the competitive ratio compared to the greedy approach.

Implement a Balance algorithm that takes the same input parameters as the greedy algorithm above and produces the expected revenue from the Ad campaign.

Case 4: What is the minimum and maximum possible revenue, and the competitive ratio of both the Greedy and Balance algorithm for the input data provided in the Notebook?

---

5 Recommendation System (*20pts*)

Along the way, you have effectively estimated the sales of products from our TDT4305 website, recovered your dream username, brandished your Big Data prowess to a friend, assisted a Google user, saved our TDT4305 startup tremendous storage and resources for both username processing and counting distinct users that visit our webpage, and that's not all, you helped our startup move from Greedy Ad campaign to a more effective Balance algorithm with a significantly higher competitive ratio. Not forgetting your feat of finding similar items with unprecedented speed in the previous project, there is no doubt that at this point, you might be tempted to call yourself a Big Data Scientist (a worthy appellation) and go for a long overdue vacation. But before you do, time for some entertainment *'problem.'*

Collaborative filtering (CF) exploits the quality judgments of other users to provide relevant recommendations. Remember your friend from Case 2. Clearly, you both have divergent interests in Big Data. Nonetheless, there's a possibility that your movie preferences coincide (You are friends for a reason, after all)

5.1 User-User Collaborative Filtering

Implement a user-user collaborative filtering function that takes in a rating matrix, a tuple of the movie-user index, the neighborhood|N|, and outputs the predicted rating of a movie by a user indicated in the movie-user index. Use the Cosine similarity measure in your implementation.

5.2 Item-Item Collaborative Filtering

The problem with user-user CF is that users' interests are unpredictable. Back to your friend, on second thought, if we can't trust your shared affinity for Computer science-related topics, why should we trust your movie preferences?

This leads us to item-item CF. The idea is that For item i, find other similar items and estimate the rating for item i based on ratings for similar items. While user-user CF identifies users similar to a particular user that rated the movie of interest, item-item CF identifies movies similar to the movie of interest rated by a particular user.

Similar to the user-user CF, implement a function that takes in the same parameters and outputs the predicted rating by a specific user for a particular movie.

Case 5: Observe the rating matrix, based on intuition, which gave a better prediction for movie 1 and user 5 rating between User-User CF and Item-Item CF.

Best of luck in your exams!