UNIT 10 MINING DATA STREAMS

10.0 INTRODUCTION

In the previous Unit of this Block, you have gone through the concepts relating to mining of Big data, where difference distance measures and techniques were discussed to uncover hidden pattern in Big data. However, there are certain applications, like satellite communication data, sensor data etc. which produce a continuous stream of data. This data stream can be regular or irregular, homogenous or heterogeneous, partly stored or completely stored etc. Such data streams are processed using various techniques.

This unit explains the characteristics and models of data stream processing. It also identifies the challenges or stream processing and introduces you to various techniques for processing of data streams. You may refer to further readings of this Unit for more details on data streams and data stream processing.

10.1 OBJECTIVES

After going through this Unit, you would be able to:

- Define the characteristics of data streams
- Discuss models of data stream processing
- Explain the uses of data streams
- Illustrate the example of data stream queries
- Explain the role of Bloom filter in data stream
- Explain an algorithm related to data stream processing.

10.2 DATA STREAMS

Usually, the data resides in a database or a distributed file system from where users can access the same data repeatedly, as it is available to the users whenever they need it. But there are some applications where the data does not reside in a database, or if it does the database is so large that the users cannot query it fast enough to answer questions about it. One such example is, data being received from a weather satellites.

Answering queries about this sort of data requires clever approximation techniques and methods for compressing data in a way that allows us to answer the queries, we need to answer.

Thus, mining data stream is a process to extract knowledge in real time from a large amount of volatile data, which comes in an infinite stream. The data is volatile because it is continuously changing and evolving over time. The system does not store the data in the database due to limited amount of resources. How would you anlayse this data stream? The following section presents the models for data stream processing.

10.2.1 Model For Data Stream Processing

As stated in the previous section, an infinite amount of data arrives continuously in a data stream. Assume D is the data stream which is the sequence of transactions and can be defined as:

$$D = (T_1, T_2, \ldots, T_i, T_{i+1}, \ldots, T_j)$$

Where: T_1: is the 1st transaction, T_2: is the 2nd transaction, T_i: is the ith transaction and T_j: jth transaction.

There are three different models for data stream processing, namely, Landmark, Sliding Windows and Damped, as shown in Figure 1 and discussed as follows:

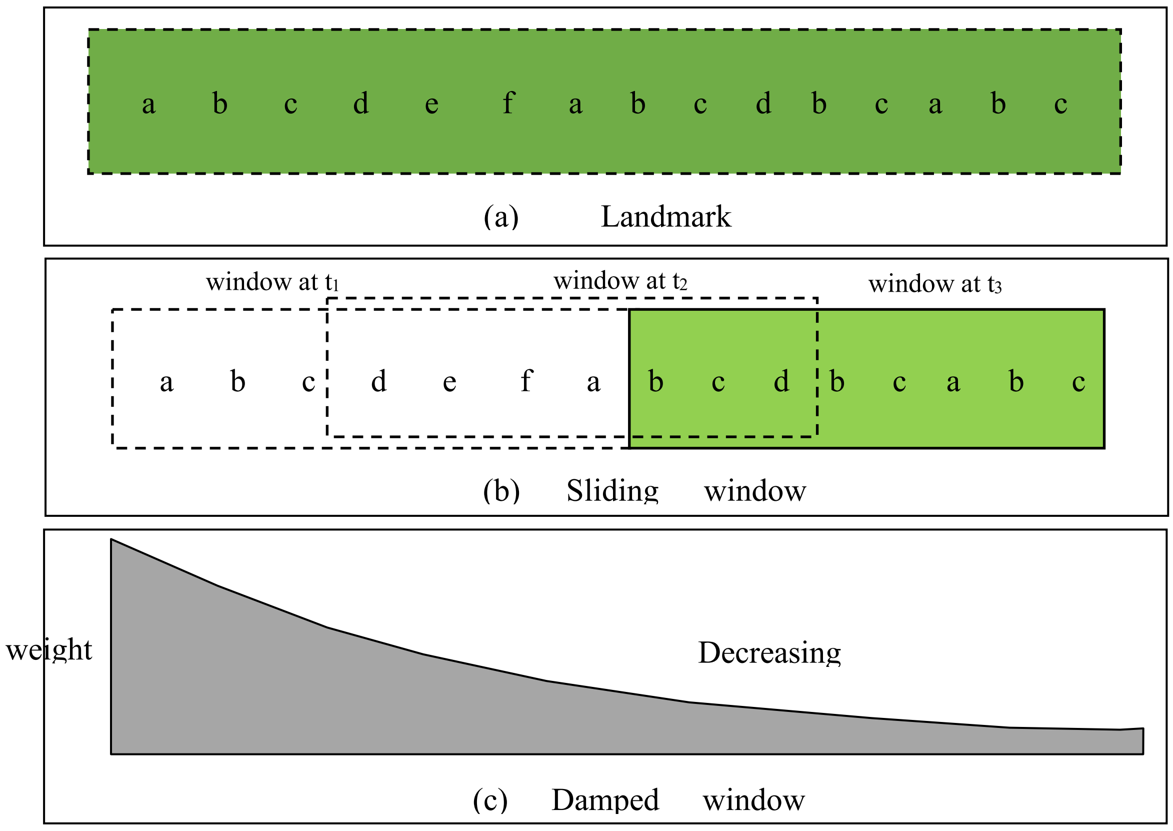

Figure1: Model for data stream processing

(a) Landmark model:

This model finds the frequently used items in entire data stream from a specific time (known as landmark) till present. In other words, the model finds out frequent items starting from T_i to current time T_t from the window $W[i,t]$, where i represents the landmark time. However, if i=1, then the model finds out the frequent items over entire data stream. In this type of model, all time-points are treated equally after the starting time. However, this model is not suitable to find items in the most recent data streams. The examples of landmark model include stock monitor system, which observes and reports on global stock market.

(b) Sliding Windows model:

This model stores recent data in sliding window from a certain range and discard old data items. The size of the sliding window may vary according to the type of application used. Suppose the size of the sliding window is *w* and current time is *t*, the model finds the data on the sliding window- *W[t-w+1, t]*. The window will update its size according to the current time. The model does not store the data that arrive before the time *t-w+1*. In other words, the part of the data stream that is in the range of the sliding window are retrieved at a particular time point.
However, the data will not be processed if the arrival rate of data is higher than the processing rate. In this case, most of the data points will be dropped.

(c) Damped model:

This model is also called as Time-Fading model as it assigns more weight to the recent transactions in data stream and this weight keeps on decreasing with age. In other words, the older transactions have less weight as compared to the newer transactions in the data stream. This model is mostly used in those applications where new data has more impact on mining results in comparison to the old data and the impact of old data decreases with time.

You may use any model for data stream processioning, but how are data streams managed? Next section discusses the data stream management system.

10.3 DATA STREAM MANAGEMENT

The fundamental difference between Data Base Management System (DBMS) and Data Stream Management System (DSMS) is who controls how data enters into the system.

(a) DBMS:
In a DBMS, the staff associated with the management of the database usually insert data into the system using a bulk loader or even explicit SQL INSERT commands. The staff can decide how much data to be loaded into the system. The staff can also decide when and how fast to load the data into the system. *For example,* the records of students stored in schools or colleges or universities.

(b) DSMS:
In a DSMS, the management cannot control the rate of input. *For example,* the search queries that arrive at Google search engine are generated by random people around the globe, who search for information at their respective pace. Google staff literally has no control over the rate of arrival of queries. They have to design and architect their system in such a way that it can easily deal with the varying data rate.

Data Stream Management System (DSMS) extracts knowledge from multiple data streams by eliminating undesirable elements, as shown in Figure 2. DSMS is important where the input data rate is controlled externally. For example, Google queries.

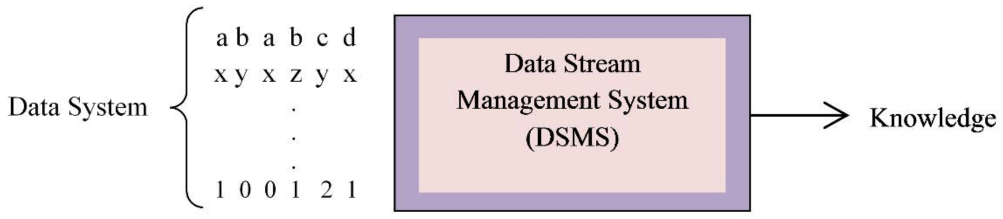

Figure 2: A simple outline of Data Stream Management System

The sources of data streams include Internet traffic, online transactions, satellite data, sensors data, live event data, real-time surveillance systems, etc.

Figure 3 shows the detailed view of a data stream management system. The components of this system are described as follows:

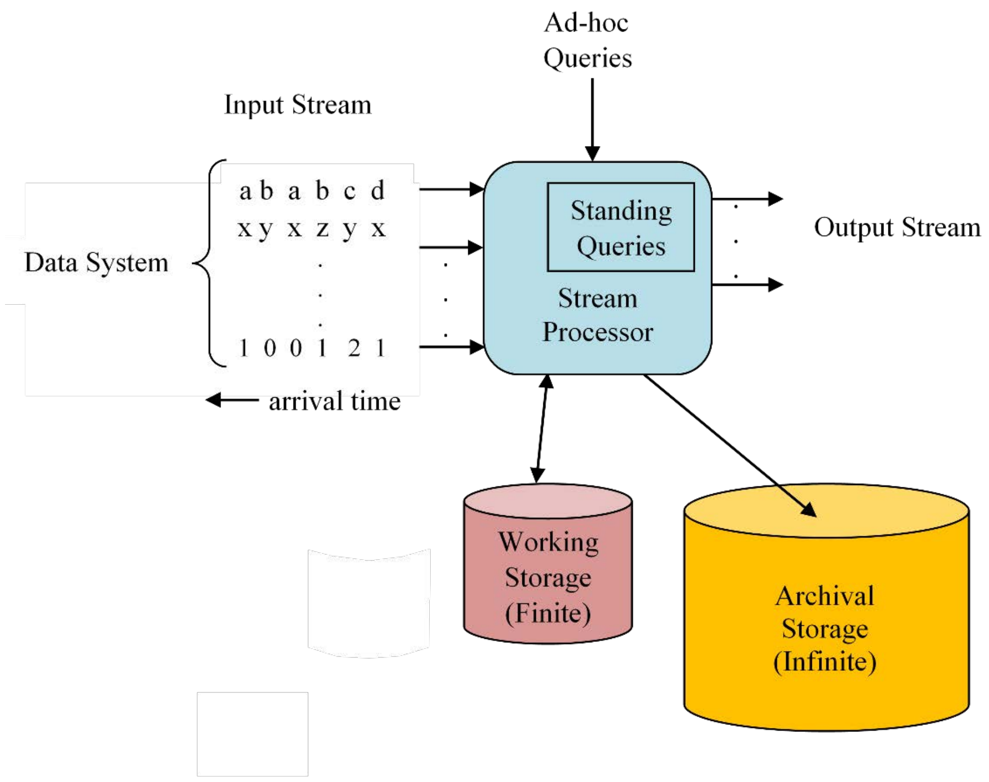

Figure 3: A detailed view of Data Stream Management System

- **Processor:** The processor is a software that executes the queries on the data stream. There can be multiple processors working together. The processor may store some standing queries and also allows ad-hoc queries (refer to section 10.3.1) to be issued by the users.

- **Streams Entering:** There can be several streams entering in the system. Conventionally, we will assume that the elements at the right end of the stream have arrived more recently. The time goes backward to the left i.e. the further left the earlier the element entered the system.

- **Output:** The system makes output in response to the standing queries and the ad-hoc queries (refer to section 10.3.1).

- **Archival Storage:** There is a massive archival storage and we cannot assume the archival storage is architected like a database system. Further, we can use appropriate indices or other tools to efficiently answer the queries from that data. We only know that if we had to reconstruct the history of the streams it could take a long time.

- **Limited Working Storage:** It might be a main memory, or a flash storage, or even magnetic disk. But we assume that it holds important parts of the input streams in a way that supports fast execution of query.

10.3.1 Queries Of Data Stream

Streams can be carried in two modes:

(a) Ad-hoc queries:

This is similar to the way we query a database system in which we make a query once and expect an answer to the query based on the current state of the system. *For example,* what is the maximum value seen so far in data stream, *D*, from its beginning to the exact time the query is asked?
This question can be answered by keeping a single value- the maximum- and updating it (if necessary), every time a new stream element arrives.

(b) Standing queries:

In this type of query, the users write the query once. However as compared to ad-hoc queries, the difference is that here, the users expect the system to report the answer available at all times perhaps outputting a new value each time the answer changes.
For example, report each new maximum value ever seen in data stream, *D*.
This question can be answered by keeping one value- the maximum (MAX)- and each new element is compared with the MAX. If it is larger, then we output the value and update the MAX to be that value.

10.3.2 Examples Of Data Stream

The following are the examples of data stream sources:

(a) Mining query streams:

The first example is the query stream and a search engine like Google. For instance, Google Trends wants to find out which search queries are much more frequent today than yesterday. These queries represent issues of rising public interest. Answering such a standing query requires looking back for at most two days in the query stream that is quite a lot, perhaps billions of queries. But it is little compared with the stream of all Google queries ever issued.

(b) Mining click streams:

Click streams are another source of a very rapid input. A site like Yahoo has many millions of users each day and the average user probably clicks a dozen times or more. A question worth answering is which URLs are getting clicked-on, a lot more, in the past one hour than normal. Interestingly, while some of these events reflect breaking news stories, many also represent a broken link. For instance, when people cannot get the page they want, they often click on it several times before giving up. So, sites mine their click streams to detect broken links.

(c) IP packets can be monitored at a switch:

The Internet Protocol (IP) packets can be monitored at a switch. The elements of the stream are IP packets typically and the switch can store a lot of information about the packets including the response speed of different network links and the points of origin and destination of the packets. This information could be used to advise the switch and the best routing for a packet to detect a Denial of Service (DoS) attack.

10.3.3 Issues And Challenges Of Data Stream

The issues and challenges of data stream are discussed as follows:

- **Input tuples:** The input elements are the tuples of a very simple kind such as bits or integers. There are one or more input ports at which data arrives. The arrival rates of input tuples are very high on these input ports.

- **Arrival rate:** The arrival rate of data is fast enough that it is not feasible for the system to store all the arriving data and at the same time make it instantaneously available for any query that we might want to perform on the data.

- **Critical calculations:** The algorithms for data stream are general methods that use a limited amount of storage (perhaps only main memory) and still enables to answer important queries about the content of the stream. However, it becomes difficult to perform critical calculations about the data stream with such a limited amount of memory.

Check Your Progress 1:

1. Define the data stream processing. Which model of data stream processing is useful in finding stock market trends? Justify.

2. Differentiate between DBMS and DSMS. Why all the data of data streams is not stored?

3. How are Standing queries different to ad-hoc queries?

4. What is use of mining click streams?

10.4 DATA SAMPLING IN DATA STREAMS

If the data on the stream is arriving too rapidly, we may not want to or need to look at every stream element. Perhaps we can work with just a small sample of the values in the stream.

While taking samples, you need to consider the following two issues:

- First, the sample should be unbiased
- Second, the sampling process must preserve the answer to the query or queries that you may like to ask about the data

Thus, you require a method that preserves the answer to the queries. The following example explains the concepts stated above:

Google has a stream of search queries that are generated at all the times. You might want to know what fraction of search queries received over a period (say last month) were unique? This query can also be written as: What fraction of search queries have only one occurrence in the entire month?

One of the sampling techniques that can be used for data streams to answer the queries may be to randomly select 1/10th of the stream. For example, if you want to know what fraction of the search queries are a single word queries; you can compute that fraction using the 1/10th sample of the data stream. The computed fraction would be very close to the actual fraction of single word queries over the entire data stream. Thus, over the month, you would be testing 1/10th of the data stream as sample. If those queries are selected at random, the deviation from the true answer will be extremely small.

However, the query – to find the fraction of unique queries, cannot be answered correctly from a random sample of the stream.

10.4.1 The Representative Sample

In this section, we discuss about the need of a representative sample with the help of an example. The query that needs to be answered is – "to find the fraction of unique search queries":

- We know that the length of a sample is 10% of the length of the whole stream (as mentioned above in section 10.4). The problem is that the probability of a given query appearing to be unique in the sample gets distorted because of the sampling.

- Suppose a query is unique in the data stream. It has only a 1/10th of chance of being selected for the sample. It implies that fraction of truly unique queries that may get selected into the sample is the same as for the entire stream. If we could only count the truly unique queries in the sample, we would get the right answer, as we are trying to find proportions.

- However, suppose a search query appears exactly twice in the whole stream. The chance that the first occurrence will be selected for the sample is 10% and the chance that the second occurrence will not be selected is 90%. Multiply those and we have a 9% chance of this query occurrence being unique in the sample.

- Moreover, the first occurrence could not be selected but the second is selected, this also has 9% chance. Thus, a query that really occurs twice but may be selected as unique in the sample with the probability of a total of 18%.

- Similarly, the queries that appears on the stream three times has a 24.3% chance of being selected as unique queries in the sample.

- In fact, any query no matter how many times it appears in the original stream, has at least a small chance of being selected as unique query in the sample.

So, when you count the number of unique queries in the sample, it will be an overestimate of the true fraction of unique queries.

In the example given above, the problem was that we performed sampling on the basis of the position in the stream, rather than the value of the stream element. In other words, we assumed that we flipped a 10-sided coin every time a new element arrived in the stream. The consequences are that when a query occurs at several positions in the stream, we decide independently whether to add or not to add the query into the sample. However, this is not the sampling which we are interested in, for answering the query about finding the unique queries.

We want to pick 1/10thof the search queries, not 1/10th of the instances of search queries in the stream. We can make a random decision when we see a search query for the first time.

If we kept a table/list of our decision for each search query we have ever seen, then each time a query appears, we can look it up in the table. If the query is found in the table/list, then we can either add it to the sample or not to add it to the sample. But if you did not find the query in the table, then you can flip the ten sided coin to decide what to do with it and we crawled the query and the outcome from the table.

However, it will be hard to manage and lookup such a table with each stream element. Fortunately, there is a much simpler way to get the same effect without storing anything in the list by using a Hash function.

- Select a hash function, which maps the search queries into 10 buckets i.e. 0 to 9.
- Apply the hash function when a search query arrives, if it is mapped to bucket 0, then add it to the sample, otherwise if it maps to any of the other 9 buckets, then do not add it to the sample.

The advantage of this approach is that all occurrences of the same query would be mapped to the same bucket, as the same hash function is applied. As a result, you do not need to know whether the search query that just arrived has been seen before or not.

Therefore, the fraction of unique queries in the sample is the same as for the stream as a whole. The result of sampling this way is that 1/10th of the queries are selected for the sample.

If selected, then the query appears in the sample exactly as many times as it does in the entire data stream. Thus, the fraction of unique queries in the sample should be exactly as it is in the data stream.

What if the total sample size is limited?

Suppose you want your sample not to be a fixed fraction of the total stream, but a fixed number of samples from the stream.

> In that case, the solution would be to perform hashing to a large number of buckets, such that the resulting sample just stays within the size limit.

As more stream elements are added, your sample gets too large. In that case, you can pick one of the buckets that you have included in the sample and delete all the stream elements from the sample that hash to that bucket. Organizing the sample by bucket can make this decision process efficient.

With this different way of sampling, the stream unique query problem will be addressed as:

- ➢ You still want a 10% sample for the search queries, however, eventually even the 10% sample will become too large. Hence, you want the ability to throw out some fraction of sampled elements. You may have to do it repeatedly. If any-one occurrence of the query is thrown out, then all other occurrences of the same query are thrown out.

- ➢ Perform hashing to 100 buckets for our example, but for a real data stream you may require a million buckets or even more, as long as you want 10% sample.

- You could choose any 10 buckets out of 100 buckets for the sample. For the present example, let us choose bucket 0 to bucket 9.

- If the sample size gets too big, then you pick one of the buckets to remove the samples, say bucket 9. It means that you delete those elements from the sample that hash to bucket 9 while retaining those that hash to bucket 0 to bucket 8. You just returned bucket 9 to the available space. Now, your sample is 9% of the stream.

- In future, you add only those new stream elements to sample that hash to bucket 0 through bucket 8.

- Sooner or later, even the 9% sample will exceed our space bound. So, you remove those elements from the sample that hash to bucket 8, then to bucket 7, and so on.

Sampling Key-Value Pairs

The idea, which has been explained with the help of an example given above, is really an instance of a general idea. A data stream can be any form of key-value pairs. You can choose sample by picking a random key set of a desired size and take all key value pairs whose key falls into the accepted set regardless of the associated value.

In our example, the search query itself was the key with no associated value. In general, we select our sample by hashing keys only, the associated value is not part of the argument of the hash function.

You can select an appropriate number of buckets for acceptance and add to sample each key-value pair whose key hashes to one of the accepting buckets.

Example: Salary Ranges

- Assume that a data stream elements are tuples with three components - ID for some employee, department that employee works for and the salary of that employee.
  StreamData=tuples(EmpID, Department, Salary)

- For each department, there is a salary range, which is the difference between the maximum and minimum salaries and is computed from the salaries of all the employees of that department.
  Query: What is the average salary range within a given department?

Assuming that you want to use a 10% sample of those stream tuples to estimate the average salary range. Picking 10% of the tuples at random would not work for a given department, as you are likely to be missing one or both of the employees with the minimum or maximum salary in that department. This will result in computation of a lower difference between the MAX and MIN salaries in the sample for a department.

Key= Department

Value= (EmpID, Salary)

The right way to sample is to treat only the department component of tuples as the key and the other two components: employee ID and salary as part of the value. In general, both the key and value parts can consist of many components.

If you sample this way, you would be sampling a subset of the departments. But for each department in the sample, you get all its employee salary data, and you can compute the true salary range for that department.

When you compute the average of the ranges, you might be off a little because you are sampling the ranges for some departments rather than averaging the ranges for all the departments. But that error is just random noise introduced by the sampling process and not a bias in one direction or another.

Check Your Progress 2:

1. What are the different ways of sampling data stream?

2. Explain any one example of sampling data.

3. What is the purpose of sampling using <Key, Value> pair? Why did you choose department as the key in the example?

10.5 FILTERING OF DATA STREAMS

In the previous section, you were answering queries using the recent window items. What would you do if you want to extract information from the entire data stream? You may have to use the filtering of data stream for this. In this section, we discuss about one data steam filter called Bloom filter.

10.5.1 Bloom filter

Bloom filters enable us to select only those items in the stream that are on some list or a set of items. In general, Bloom filter is used for cases where the number of items on the list or set is so large, that you cannot do a comparison of each stream element with element of the list or check the set membership.

Need of Bloom filters:

Let us explain the need of Bloom filter with the help of an example application of a search engine.

- Web crawler performs many crawling tasks and uses different processors to crawl pages.

- The crawler maintains a list of all URLs in the database that it has already found. Its goal is to explore the web pages in each of these URLs to find the additional URLs that are linked to these web pages.

- It assigns these URLs to any of a number of parallel tasks, these tasks stream back the URLs they find in the links they discover on a page.

- However, it is not expected to have the same URL to get into the list twice. Because in that case you will be wasting your time in crawling the page twice. So, each time a URL comes back to the central controller, it needs to determine whether it has seen that URL before and discard the second report so that you could create an index, say a hash table, to make it efficient to look up the URL and see whether it is already among those web pages that has been in the index.

- But the number of URLs are extremely large and such an index will not fit in the main memory. It may be required to be stored in the secondary memory (hard disk), requiring disk access every time a URL is to be checked. This would be very time consuming process.

Therefore, you need a Bloom filter.

- When a URL arrives in a stream, pass it through a Bloom filter. This filter will determine if the URL has already been visited or not.

- If the filter says it has not been visited earlier, the URL will be added to the list of URLs that needs to be crawled. And eventually it will be assigned to some crawling task.

- But the Bloom filter can have false positives, which would result in assigning some of the URLs as already visited, while in fact that it was not.

- The good news- if a Bloom filter says that the URL has never been seen. Then that is true i.e., there are no false negatives.

Working of Bloom filter:

- Bloom filter itself is a large array of bits, perhaps several times as many bits as there are possible elements in the stream.

- The array is manipulated through a collection of hash functions. The number of hash functions can be one, although several hash functions are better. In some situations, even a few dozen hash functions may be a good choice.

- Each hash function maps a stream element to one of the positions in the array.

- During initialization all the bits of the array are initialized to 0.

- Now, when a stream element, say x, arrives, we compute the value of $h_i(x)$ for each hash function h_i that are to be used for Bloom filtering.

- A hash function maps a stream element to an index value on Bloom filter array. In case, this index value is 0, then it is changed to 1.

The following example explains the working of Bloom filter in details.

Example of Bloom filters:

For example, the size of an array that is going to be used for Bloom filter is of 11 bits, i.e., N=11. Also, assume that the stream that is to be filtered consists of only unsigned integers of 12 bits, i.e., Stream elements=unsigned integers. Further, for the purpose of this example, let us use only two hash functions h_1 and h_2, as given below:

- The first hash function h_1 maps an integer x to a hash value $h_1(x)$ as follows:
    - Write the binary value of integer x. Select the bits at the odd bit positions starting from the left most (least significant) bit.
    - Extract these odd bits of x to another binary, say x_{odd}
    - Take modulo as: (x_{odd} modulo 11) to map x into hash value $h_1(x)$.
- $h_2(x)$ is computed in exactly the same manner except that it collects the even bit positions of the binary representation to create x_{even}. The modulo is also computed using x_{even} modulo 11.

Next, you will initialize all the array values of the Bloom filter to zero. Assuming that the set of valid stream elements is {25, 159, 585}, you train the Bloom filter as:

Initial Bloom filter contents=00000000000, which is representation as:

Bloom Filter Array Index	0	1	2	3	4	5	6	7	8	9	10
Bloom Filter	0	0	0	0	0	0	0	0	0	0	0

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Binary Equ	2048	1024	512	256	128	64	32	16	8	4	2	1
$x = 25$	0	0	0	0	0	0	0	1	1	0	0	1
x_{odd}		0		0		0		1		0		1
$h_1(x)$	000101 = 5 in decimal; $h_1(x) = 5 \bmod 11 = 5$											
x	0	0	0	0	0	0	0	1	1	0	0	1
x_{even}	0		0		0		0		1		0	
$h_2(x)$	000010 = 2 in decimal; $h_2(x) = 2 \bmod 11 = 2$											

Bloom filter contents after inserting hash values of 25

Bloom Filter Array Index	0	1	2	3	4	5	6	7	8	9	10
Bloom Filter	0	0	1	0	0	1	0	0	0	0	0

Stream element= 159 and Bloom filter contents=00100100000

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Binary Equ	2048	1024	512	256	128	64	32	16	8	4	2	1
$x = 159$	0	0	0	0	1	0	0	1	1	1	1	1
x_{odd}		0		0		0		1		1		1
$h_1(x)$	000111 = 7 in decimal; $h_1(x) = 7 \bmod 11 = 7$											
x	0	0	0	0	1	0	0	1	1	1	1	1
x_{even}	0		0		1		0		1		1	
$h_2(x)$	001011 = 11 in decimal; $h_2(x) = 11 \bmod 11 = 0$											

Bloom filter contents after inserting hash values of 159

Bloom Filter Array Index	0	1	2	3	4	5	6	7	8	9	10
Bloom Filter	1	0	1	0	0	1	0	1	0	0	0

Stream element= 585 and Bloom filter contents=10100101000

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Binary Equ	2048	1024	512	256	128	64	32	16	8	4	2	1
$x = 585$	0	0	1	0	0	1	0	0	1	0	0	1
x_{odd}		0		0		1		0		0		1
$h_1(x)$	001001 = 9 in decimal; $h_1(x) = 9 \bmod 11 = 9$											
x	0	0	1	0	0	1	0	0	1	0	0	1
x_{even}	0		1		0		0		1		0	
$h_2(x)$	010010 = 18 in decimal; $h_2(x) = 18 \bmod 11 = 7$											

Bloom filter contents after inserting hash values of 585

Bloom Filter Array Index	0	1	2	3	4	5	6	7	8	9	10
Bloom Filter	1	0	1	0	0	1	0	1	0	1	0

<u>How to test membership to the set using Bloom filters:</u>

Assuming that you are using Bloom filter to test the membership of a 12-bit unsigned integer to the set {25, 159, 585}. The bloom filter for this set is 10100101010 (as shown above). Find if the stream element is member of the set or not.

Lookup element *y* = 118

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Binary Equ	2048	1024	512	256	128	64	32	16	8	4	2	1
y = 118	0	0	0	0	0	1	1	1	0	1	1	0
y_{odd}		0		0		1		1		1		0
$h_1(y)$	001110 = 14 in decimal; $h_1(y)$ = 14 mod 11 = 3											
y	0	0	0	0	0	1	1	1	0	1	1	0
y_{even}	0		0		0		1		0		1	
$h_2(y)$	000101 = 5 in decimal; $h_2(y)$ = 5 mod 11 = 5											

Bloom Filter Array Index	0	1	2	3	4	5	6	7	8	9	10
Bloom Filter	1	0	1	0	0	1	0	1	0	1	0
Checking for 118				1		1					
				M		0					

Since, there is a mismatch represented by M, therefore, *y* is not a member of the set and it can be filtered out.

However, there can be false positives, when you use Bloom filter. For example:

Lookup element *y* = 115

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Binary Equ	2048	1024	512	256	128	64	32	16	8	4	2	1
y= 115	0	0	0	0	0	1	1	1	0	0	1	1
y_{odd}		0		0		1		1		0		1
$h_1(y)$	001101 = 13 in decimal; $h_1(y)$ = 13 mod 11 = 2											
y	0	0	0	0	0	1	1	1	0	0	1	1
y_{even}	0		0		0		1		0		1	
$h_2(y)$	000101 = 5 in decimal; $h_2(y)$ = 5 mod 11 = 5											

Bloom Filter Array Index	0	1	2	3	4	5	6	7	8	9	10
Bloom Filter	1	0	1	0	0	1	0	1	0	1	0
Checking for 115			1			1					
			0			0					

Since, there is No mismatch, therefore, y is a member of the set, and it cannot be filtered out. However, you may notice this is a false positive.

10.6 ALGORITHM TO COUNT DIFFERENT ELEMENTS IN STREAM

In stream processing sometimes instead of exact solution, you can accept approximate solutions. One such algorithm, which is used to count different elements in a stream in a single pass was given by Flajolet-Martin. This algorithm is discussed below:

Steps of the algorithm:

1. Pick a hash function h that maps each of the *n* elements of the data stream to at least $\log_2(n)$ bits.
2. For each stream element *a*, let r(*a*) be the number of trailing 0's in h(*a*).

3. Record R = the maximum r(*a*) seen.
4. Estimated number of different elements= 2^R.

Example:

Given a good uniform distribution of numbers as shown in Table 1. It has eight different elements in the stream.
Probability that the right-most set bit is at position 0 = ½
At position 1 = 1/2 * 1/2 = 1/4
At position 2 =1/2 * 1/2 *1/2 = 1/8
…
At position n = $1/2^n$

Table 1: An Example of Flajolet-Martin Algorithm

	Number	Binary Representation	Position of the rightmost set bit
	0	000	-
	1	001	0
Uniform Distribution	2	010	1
	3	011	0
	4	100	2
	5	101	0
	6	110	1
	7	111	0

(Assuming that the index value of least significant bit is 0)

It implies that the probability of the right-most set bit drops by a factor of 1/2 with every position from the LSB to the MSB as shown in Figure 4.

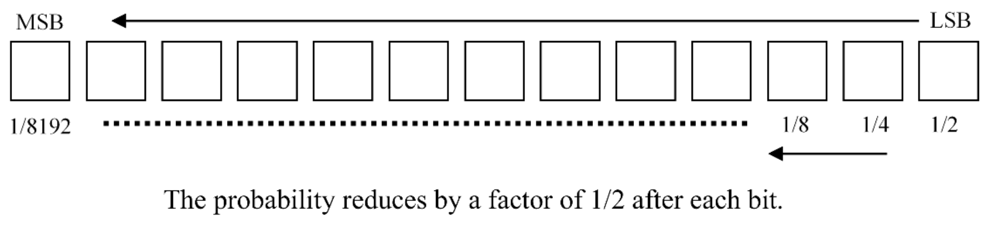

Figure 4: Probability in Flajolet-Martin Algorithm [4]

By keeping the record of these positions of the right-most set bit, say ρ, for each element in the stream. We will expect position of rightmost set bit = 0 to be 0.5, ρ = 1 to be 0.25, etc. Also consider that *m* is the number of distinct elements in the stream.

- This probability will come to 0 when bit position *b* is greater than log *m*
- This probability will be non-zero, when *b* <= log *m*

Therefore, if we find the right-most unset bit position *b* such that the probability = 0, we can say that the number of unique elements will approximately be 2^b. This forms the core intuition behind the Flajolet Martin algorithm.

A detailed discussion on this algorithm can be referred from the further reading.

Check Your Progress 3:

1. Explain how Bloom filter can be used to identify emails that are not from a selected group of addresses.

2. Explain the Flajolet-Martin algorithm.

10.7 SUMMARY

This unit introduces the concept of mining data streams. Data streams can be processed using different models. Three such models (landmark, sliding windows, and damped model) for data stream processing are introduced in this unit. Further, data stream management system (DSMS) is explained. In addition, different types of queries of data stream namely, ad-hoc and standing queries are discussed followed by examples of these queries. The issues and challenges of Data streams and data sampling in data streams with examples of representation sample and sampling Key-Value pairs are discussed in this unit. This unit also explains bloom filter with its need, working, and related examples. In the end, this unit shows the algorithm to approximately count different elements in stream with example.

10.8 ANSWERS

Check Your Progress 1:

1. The data stream is a process to extract knowledge in real time from a large amount of volatile data, which comes in an infinite stream of data. The data is volatile because it is continuously changing and evolving over time. The system does not store the data in the database due to limited amount of resources. The landmark model of data stream processing is useful in finding stock market trends because it finds the frequently used items in entire data stream from a specific time till present and all time-points are treated equally after the starting time.

2. In a DBMS, the staff associated with the management of the database usually insert data into the system. In a DSMS, the management cannot control the rate of input. All data of data streams is not stored due to limited amount of resources.

3. In standing queries, the users expect the system to report the answer available at all time perhaps outputting a new value each time the answer changes. In ad-hoc queries, the users make a query once and expect an answer to the query based on the current state of the system, which is available only at current time.

4. The use of mining click streams is to know which URLs are getting clicked-on, a lot more, in the past one hour than normal.

Check Your Progress 2:

1. The representative samples and sampling key-value pairs are different ways of sampling data stream.

2. Suppose that you want a 10% sample for the search queries. You could choose any 10 buckets out of 100 buckets for the sample. For the present example, let us choose bucket 0 to bucket 9. If the sample size gets too big, then you pick one of the buckets to remove the samples, say bucket 9. It means that you delete those elements from the sample that hash to bucket 9 while retaining those that hash to bucket 0 to bucket 8. You just returned bucket 9 to the available space. Now, our sample is 9% of the stream. In future, you add only those new stream elements to sample that hash to bucket 0 through bucket 8. Sooner or later, even the 9% sample will exceed our space bound. So, you remove those elements from the sample that hash to bucket 8, then to bucket 7, and so on.

3. The purpose of sampling using <Key, Value> pair is that you can choose a sample by picking a random key set of a desired size and take all key value pairs whose key falls into the accepted set regardless of the associated value. The department is chosen as a key in the example because department is a constant used to define the data set.

Check Your Progress 3:

1. When an email arrives in the stream, it will pass through a Bloom filter. Bloom filter is a large array of bits which is equal to the possible elements in the stream. The array is manipulated through a collection of hash functions. Each hash function maps a stream element to one of the positions in the array. During initialization all the bits of the array are initialized to 0.

   First, you will initialize all the arrays to zero. You will then select each email address from the allowable/selected group of address and apply every hash function on these email addresses to compute hash values. A hash function maps a stream element to an index value on Bloom filter array. In case this index value is 0, then it is changed to 1. Now, your filter is set.

   Now, when a email from a mail id say *rakesh@gmail.com*, arrives, you compute the value of $h_i(x)$ for each hash function h_i that are to be used for Bloom filtering. If all these hash values are 1 in the filter, it means this email id is one of the selected/allowed email id. Please note there can be false positives through.

2. Explain the Flajolet-Martin algorithm with the help of an example.

   The Flajolet-Martin algorithm is used to count different elements in a stream in a single pass. The steps of the algorithm are:

   A. Pick a hash function h that maps each of the n elements of the data stream to at least $\log_2(n)$ bits.
   B. For each stream element a, let $r(a)$ be the number of trailing 0's in $h(a)$.
   C. Record R = the maximum $r(a)$ seen.
   D. Estimate = 2^R.

10.9 REFERENCES/FURTHER READINGS

References

[1] Mansalis, Stratos, et al. "An evaluation of data stream clustering algorithms." Statistical Analysis and Data Mining: The ASA Data Science Journal 11.4 (2018): 167-187

[2] Albert C. “Introduction to Stream Mining.” https://towardsdatascience.com/introduction-to-stream-mining-8b79dd64e460

[3] “Mining Data Streams.” http://infolab.stanford.edu/~ullman/mmds/ch4.pdf

[4] Bhyani, A., “Approximate Count-Distinct using Flajolet Martin Algorithm.” https://arpitbhayani.me/blogs/flajolet-martin