UNIT 3

Mining Data Streams: Introduction to streams concepts, stream data model and architecture, stream computing, sampling data in a stream, filtering streams, counting distinct elements in a stream, estimating moments, counting oneness in a window, decaying window, Real-time Analytics Platform (RTAP) applications, Case studies – real time sentiment analysis, stock market predictions.

Mining Data Streams

Mining data streams is a subfield of data mining and machine learning that deals with the analysis of continuously arriving data, where the data is often too large to be stored and processed in traditional databases. The main objective of data stream mining is to extract valuable and meaningful information from high-velocity, high-volume, and highly dynamic data sources in real-time.

Data stream mining can be applied in various domains such as finance, healthcare, transportation, telecommunications, environmental monitoring, and social media analysis. Some common applications of data stream mining include fraud detection, network intrusion detection, web analytics, and patient monitoring.

The process of data stream mining involves several key steps, including:

1. **Data collection:** Collecting and aggregating data from various sources in real-time.
2. **Preprocessing:** Cleaning and transforming the data to prepare it for analysis.
3. **Feature extraction:** Selecting relevant features and reducing the dimensionality of the data.
4. **Model building:** Building models to classify, cluster, or summarize the data.
5. **Model evaluation:** Evaluating the performance of the models on the data stream.

Data stream mining algorithms are designed to handle the unique challenges of continuously arriving data, including high velocity, high volume, and concept drift, where the underlying patterns in the data change over time. Some common techniques used in data stream mining include:

- **Sampling:** Selecting a subset of the data to reduce the computational complexity.
- **Window-based processing:** Keeping track of the most recent data and discarding the older data.
- **Evolutionary algorithms:** Adaptive algorithms that change their behavior over time as they process new data.
- **Frequent pattern mining:** Identifying patterns that occur frequently in the data stream.
- **Anomaly detection:** Identifying unusual or unexpected data points in the stream.

There are several challenges in data stream mining, including the ability to handle large amounts of data, the need for real-time processing, and the ability to handle concept drift. To address these challenges, data stream mining algorithms must be scalable, efficient, and able to process data in real-time.

Data stream mining is a challenging but important area of data analysis that is relevant to many industries. It involves the analysis of high-velocity, high-volume, and highly dynamic data sources to extract valuable and meaningful information in real-time. The use of specialized algorithms and tools is necessary to handle the unique challenges of data stream mining.

Introduction to streams concepts

1. **Stream:** A stream is a continuous flow of data that arrives in real-time and is processed as it arrives. Unlike traditional data, which is stored and processed in batches, data streams are never-ending and require real-time processing. The data in streams can come from various sources such as financial transactions, network traffic, website visits, and sensor readings.
2. **Data Stream Mining:** Data stream mining is a subfield of data mining and machine learning that deals with the analysis of continuously arriving data streams. The objective of data stream mining is to extract valuable and meaningful information from high-velocity, high-volume, and highly dynamic data sources in real-time. This information can be used for various purposes such as fraud detection, network intrusion detection, web analytics, and patient monitoring.
3. **Data Stream Characteristics:** Data streams are often characterized by their high velocity, high volume, and high variety. High velocity refers to the speed at which data is generated and arrives, while high volume refers to the amount of data generated. High variety refers to the diverse types of data that can be generated, such as numerical, categorical, and textual data. Data streams are also highly dynamic, meaning that the underlying patterns in the data can change over time, which is known as concept drift.
4. **Data Stream Sources:** Data streams can be generated from various sources, including financial transactions, network traffic, website visits, and sensor readings. Financial

transactions can provide information about the purchase behavior of customers, while network traffic can provide information about network usage and security. Website visits can provide information about user behavior on websites, while sensor readings can provide information about the physical environment.

5. **Real-time Processing:** Data stream mining requires real-time processing, where the data is analyzed as it arrives and the results are updated in real-time. This is necessary because the data streams are never-ending and require immediate processing to extract valuable information.
6. **Concept Drift:** Concept drift refers to the change in the underlying patterns in the data over time. This can be due to various factors such as changes in customer behavior, network usage patterns, or environmental conditions. Concept drift can have a significant impact on the accuracy of data stream mining algorithms, as the algorithms must continuously adapt to changes in the data.
7. **Sampling:** Sampling is a technique for reducing the computational complexity of data stream mining by selecting a subset of the data for processing. Sampling can help to reduce the amount of data that needs to be processed and make data stream mining more efficient.
8. **Window-based Processing**: Window-based processing is a technique for handling high-velocity data streams by keeping track of only the most recent data and discarding older data. This can help to reduce the amount of memory required to store the data and make data stream mining more efficient.
9. **Evolutionary Algorithms:** Evolutionary algorithms are algorithms that adapt over time as they process new data. This can help to address concept drift in data streams and improve the accuracy of the algorithms. Evolutionary algorithms can be applied to various tasks in data stream mining, including classification, clustering, and anomaly detection.
10. **Frequent Pattern Mining:** Frequent pattern mining is the process of identifying patterns that occur frequently in the data stream. This can be useful for understanding the behavior of customers, network usage patterns, or environmental conditions. Frequent pattern mining can be used to extract information such as the items that are commonly purchased together or the websites that are frequently visited by users.
11. **Anomaly Detection:** Anomaly detection is the process of identifying unusual or unexpected data points in the data stream. This can be useful for detecting fraud, network intrusions

Stream Data Model and Architecture

A stream data model is a representation of the data that is generated in a data stream. It is designed to handle the high-velocity, high-volume, and highly dynamic nature of data streams. The stream data model includes the following aspects:

1. **Structure of data:** The stream data model defines the structure of the data, including the number and types of attributes in the data, and their relationships with each other.
2. **Data values:** The stream data model represents the data values generated by the data stream. This can include numerical, categorical, and textual data.
3. **Data patterns:** The stream data model also includes information about the patterns in the data, such as trends and dependencies between attributes.
4. **Dynamic nature:** The stream data model must be able to handle the highly dynamic nature of data streams, with data values changing rapidly over time.
5. **Scalability:** The stream data model must be scalable to handle increasing amounts of data and provide fast and accurate results.
6. **Fault tolerance:** The stream data model must be fault-tolerant and able to handle failures without losing data or interrupting processing.
7. **Performance:** The stream data model must be designed for high performance, allowing for real-time processing and fast and accurate results.

Overall, the stream data model is a critical component of a data stream architecture and must be carefully designed to meet the demands of processing high-velocity, high-volume, and highly dynamic data streams.

A data stream architecture is a set of components and technologies used to process and analyze data streams. The architecture is designed to handle the high-velocity, high-volume, and highly dynamic nature of data streams. A data stream architecture typically includes the following components:

1. **Data Ingestion:** The data ingestion component is responsible for collecting and integrating data from various sources. It must be able to handle diverse types of data, including numerical, categorical, and textual data, and support integration with different types of data sources, such as databases, APIs, and sensors.

2. **Data Processing:** The data processing component is responsible for transforming and processing the raw data into a format suitable for analysis. This can include tasks such as data cleaning, data normalization, and feature extraction.
3. **Data Storage:** The data storage component is responsible for storing the processed data in a manner that is optimized for real-time access and analysis. This can include databases, data warehouses, or distributed storage systems, such as Apache Kafka or Apache Cassandra.
4. **Data Analysis:** The data analysis component is responsible for performing the actual analysis of the data, including tasks such as statistical analysis, machine learning, and data visualization.
5. **Stream Management:** The stream management component is responsible for managing the flow of data through the architecture, including controlling the rate of data ingestion, monitoring the processing of data, and ensuring that data is stored and analyzed in a timely manner.
6. **User Interfaces**: The user interfaces component provides a user-friendly interface for accessing and analyzing the data, including dashboards, visualizations, and reporting tools.

Overall, the data stream architecture must be designed to support the high-velocity, high-volume, and highly dynamic nature of data streams and provide fast and accurate results. The architecture must also be scalable, fault-tolerant, and optimized for performance to meet the demands of processing large amounts of data in real-time.

Stream Computing

Stream computing is:

1. **Real-time processing:** Stream computing processes data as it is generated, producing results in real-time without waiting for the entire data set to be collected and processed.
2. **Continuous data analysis:** Stream computing analyzes data continuously as it is generated, producing updated results in real-time.
3. **High-velocity data handling:** Stream computing is designed to handle large amounts of continuous and high-velocity data.
4. **Data from various sources:** Stream computing integrates data from various sources, including databases, APIs, and sensors, in real-time.

5. **Data processing algorithms:** Stream computing uses algorithms and models designed for stream computing, such as complex event processing and machine learning algorithms.
6. **Data storage optimization:** Stream computing stores processed data in a manner optimized for real-time access and analysis.
7. **Real-time data analysis:** Stream computing analyzes processed data in real-time, using techniques such as statistical analysis, machine learning, and data visualization.
8. **Stream management:** Stream computing manages the flow of data through the architecture, including controlling the rate of data ingestion and monitoring the processing of data.
9. **Fast, informed decision-making:** Stream computing enables organizations to make faster, more informed decisions based on real-time data.

Overall, stream computing provides a real-time, highly scalable, and flexible approach to processing and analyzing data streams.

Sampling Data in a Stream

Sampling data in a stream is a process of selecting a subset of data from a continuous and high-volume data stream for processing and analysis. The goal of sampling is to reduce the amount of data to be processed while preserving the overall characteristics of the data stream. This is important in stream computing because the volume of data being generated and processed is often too large to be handled efficiently in its entirety.

There are two main types of sampling techniques used in stream data:

1. **Random Sampling:** In random sampling, data points are selected randomly from the data stream. This method is used to obtain a representative sample of the data stream and is appropriate when the data distribution is unknown. The sample size is determined by the desired level of accuracy and the data distribution. Random sampling can be performed in a number of ways, including simple random sampling, stratified random sampling, and cluster sampling.
2. **Reservoir Sampling:** In reservoir sampling, a fixed-size sample of the data stream is maintained and updated at each step. This method is appropriate when the data

distribution is known or when the number of data points in the stream is unknown. In this method, an initial sample of data points is selected randomly and the sample is updated at each step by randomly replacing a data point in the sample with a new data point from the stream. The sample size is determined by the desired level of accuracy and the computational complexity of the sampling process.

The choice of sampling technique depends on the specific requirements of the application, such as the desired accuracy, computational complexity, and the data distribution. Sampling is often used in stream computing to reduce the processing overhead, minimize storage requirements, and reduce the latency of real-time data analysis.

Overall, sampling data in a stream is an important technique for managing the volume and complexity of data in stream computing. By reducing the amount of data to be processed, sampling enables organizations to make more efficient use of their computational resources and to process data in real-time with minimal delay.

Reservoir Sampling Algorithm:

Suppose we have a stream of data with an unknown number of elements, and we want to randomly select k elements from the stream, where k is a fixed number.

1. Initialize an empty reservoir of size k.
2. For each incoming element in the stream:

    - If the reservoir is not yet full, add the incoming element to the reservoir.
    - If the reservoir is full, randomly select an index i between 0 and the current index n (where n is the total number of elements seen so far) inclusive.
        - If i < k, replace the element at index i with the incoming element.

At the end of the stream, the k elements in the reservoir will be a random sample of the data stream.

Example:

Suppose we have a stream of integers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and we want to randomly select 3 elements from the stream.

1. Initialize an empty reservoir of size 3.
2. The first 3 elements are added to the reservoir since it is not yet full. The reservoir now contains: [1, 2, 3].
3. For the fourth element (4), a random index between 0 and 3 (inclusive) is selected. Suppose index 1 is selected. The element at index 1 in the reservoir (which is currently 2) is replaced with the incoming element 4. The reservoir now contains: [1, 4, 3].
4. For the fifth element (5), a random index between 0 and 4 (inclusive) is selected. Suppose index 0 is selected. The element at index 0 in the reservoir (which is currently 1) is replaced with the incoming element 5. The reservoir now contains: [5, 4, 3].
5. For the sixth element (6), a random index between 0 and 5 (inclusive) is selected. Suppose index 4 is selected. Since index 4 is greater than the size of the reservoir (which is 3), we discard the incoming element 6.
6. Steps 3-5 are repeated for the remaining elements in the stream. At the end of the stream, the reservoir contains a random sample of 3 elements from the stream.

Random Sampling Algorithm:

Suppose we have a stream of data with an unknown number of elements, and we want to randomly select k elements from the stream, where k is a fixed number.

1. Initialize a counter n to 0.
2. For each incoming element in the stream:

    - Increment n.
    - With probability k/n, add the incoming element to a sample reservoir of size k.
        - If the reservoir is already full, randomly select an index i between 0 and k-1 inclusive.
            - Replace the element at index i with the incoming element.

At the end of the stream, the k elements in the sample reservoir will be a random sample of the data stream, with each element having an equal probability of being selected.

Example:

Suppose we have a stream of integers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and we want to randomly select 3 elements from the stream.

1. Initialize a counter n to 0.
2. For the first element (1), increment n to 1. Since k/n is 3/1, we add the incoming element to the sample reservoir. The reservoir now contains: [1].
3. For the second element (2), increment n to 2. Since k/n is 3/2, we add the incoming element to the sample reservoir with probability 3/2. Suppose the element is not added to the reservoir, so the reservoir still contains: [1].
4. For the third element (3), increment n to 3. Since k/n is 3/3, we add the incoming element to the sample reservoir. The reservoir now contains: [1, 3].
5. Steps 3-4 are repeated for the remaining elements in the stream. At each step, we update the sample reservoir based on the probability of adding the incoming element. At the end of the stream, the reservoir contains a random sample of 3 elements from the stream.

Filtering Streams

Filtering streams is the process of selecting relevant data from a continuous and high-volume data stream, and removing irrelevant or unwanted data. This process is an essential step in stream data processing, as it helps to reduce the volume of data that needs to be analyzed and processed, and enables organizations to make more efficient use of their computational resources.

There are two main types of filtering techniques used in stream computing:

1. **Pre-filtering:** In pre-filtering, data is filtered at the source of the data stream, before it is processed or stored. This type of filtering is used to eliminate data that is not needed for a specific application. For example, an application that is only interested in monitoring temperature data from a set of sensors might use pre-filtering to eliminate all other data, such as humidity or pressure data, before it is processed.
2. **Post-filtering:** In post-filtering, data is filtered after it has been processed or stored. This type of filtering is used to refine the results of a data analysis. For example, an

application that has processed a large volume of data to detect outliers might use post-filtering to eliminate outliers that are not relevant to the specific application.

Filtering can be performed based on a number of criteria, such as data values, data quality, data distribution, and data relevance. For example, data can be filtered based on its value, by eliminating data points that fall outside of a specified range. Data can also be filtered based on its quality, by eliminating data points that are missing or contain errors.

The choice of filtering technique and criteria depends on the specific requirements of the application, such as the desired accuracy, computational complexity, and the data distribution. It is also important to consider the characteristics of the data stream, such as its volume, velocity, and variability, when selecting a filtering technique.

Overall, filtering streams is an important step in stream computing, as it enables organizations to manage the volume and complexity of data, and to make more efficient use of their computational resources. By reducing the amount of data to be processed, filtering enables organizations to process data in real-time with minimal delay.

Bloom Filter

A Bloom filter is a probabilistic data structure used to test whether an element is a member of a set. The steps for implementing a Bloom filter are as follows:

1. **Initialize the Bloom filter:** Choose a size for the Bloom filter and initialize all bits to 0. For example, let's say we choose a size of 20 bits and initialize the bits as follows: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].
2. **Hash the element:** Hash the element to be added to the set into multiple hash values. For example, let's say we want to add the element "apple" to the set. We hash the element "apple" into two hash values, h1 = 7 and h2 = 11.
3. **Set bits:** Set the bits corresponding to the hash values to 1. Our Bloom filter now looks like this: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0].
4. **Query the Bloom filter:** To check if an element is in the set, hash the element and check if all the bits corresponding to the hash values are set to 1. For example, let's say we want to check if the element "banana" is in the set. We hash the element "banana"

into two hash values, h1 = 9 and h2 = 14. Since both bits corresponding to these hash values are 0, we can conclude that "banana" is not in the set.

5. **False positives:** Since the Bloom filter is a probabilistic data structure, there is a possibility of false positives. That is, the Bloom filter may indicate that an element is in the set when it is not. For example, if we query for the element "cherry" with hash values h1 = 7 and h2 = 11, the Bloom filter indicates that "cherry" is in the set, even though it was not added to the set previously.
6. **Tune the parameters:** The parameters of the Bloom filter, such as the size and number of hash functions, can be tuned to reduce the false positive rate. For example, increasing the size of the Bloom filter or using more hash functions reduces the false positive rate.
7. **False negatives:** False negatives are not possible with Bloom filters. That is, if the Bloom filter indicates that an element is not in the set, it is guaranteed to be correct.

Here's an example of how to compute hash functions for a Bloom filter:

Suppose we have a bit array of size 20 and k = 3 hash functions. Let's define three hash functions as follows:

- Hash function 1: Multiply the element by 3 and take the modulus of 20.
- Hash function 2: Multiply the element by 7 and take the modulus of 20.
- Hash function 3: Multiply the element by 13 and take the modulus of 20.

Now let's insert the element "apple" into the Bloom filter:

1. Apply each hash function to the element "apple" to obtain k hash values:
    - Hash function 1: ("apple" * 3) % 20 = 2
    - Hash function 2: ("apple" * 7) % 20 = 3
    - Hash function 3: ("apple" * 13) % 20 = 1
2. Set the bits at the indices obtained in step 1 to 1 in the bit array: bitArray[2] = 1, bitArray[3] = 1, bitArray[1] = 1

Counting distinct elements in a stream

Counting distinct elements in a stream refers to the process of determining the number of unique elements in a data stream, in real-time. This is an important operation in stream computing, as it enables organizations to gain insight into the diversity of the data, and to track the frequency of occurrence of elements over time.

There are several algorithms and data structures that can be used to count distinct elements in a stream, including:

1. **Hash-based algorithms:** These algorithms use hash tables to store the elements and their counts. A hash function is used to map each element to a unique index in the hash table. The time complexity of these algorithms is typically O(1) for both insertion and retrieval of elements, making them well-suited for large and high-volume data streams.
2. **Reservoir sampling:** This algorithm randomly selects a fixed number of elements from the data stream, and uses them to estimate the number of distinct elements in the entire stream. This algorithm is simple and efficient, but its accuracy depends on the size of the reservoir.
3. **Bloom filters:** This is a probabilistic data structure that is used to determine whether an element is in the stream or not. It is based on a bit array and a set of hash functions. The Bloom filter is used to eliminate duplicates before inserting elements into a hash table, which is used to count the distinct elements.
4. **Count-Min Sketch:** This is a probabilistic data structure that is used to estimate the frequency of occurrence of elements in a data stream. It is based on a matrix of counters and a set of hash functions. The Count-Min Sketch is used to estimate the count of each distinct element in the stream.

The choice of algorithm and data structure for counting distinct elements in a stream depends on the specific requirements of the application, such as the desired accuracy, computational complexity, and the volume and velocity of the data stream.

In conclusion, counting distinct elements in a stream is an important operation in stream computing, as it enables organizations to gain insight into the diversity of the data and to track the frequency of occurrence of elements over time. By using the appropriate algorithm and data

structure, organizations can efficiently count distinct elements in large and high-volume data streams in real-time.

Flajolet–Martin algorithm

The Flajolet-Martin algorithm is a probabilistic algorithm used for counting the number of distinct elements in a stream of data. The algorithm makes use of hash functions to map the data items to a bit array and then analyzes the length of the longest trailing zero sequence in the binary representation of the hash values to estimate the number of distinct elements.

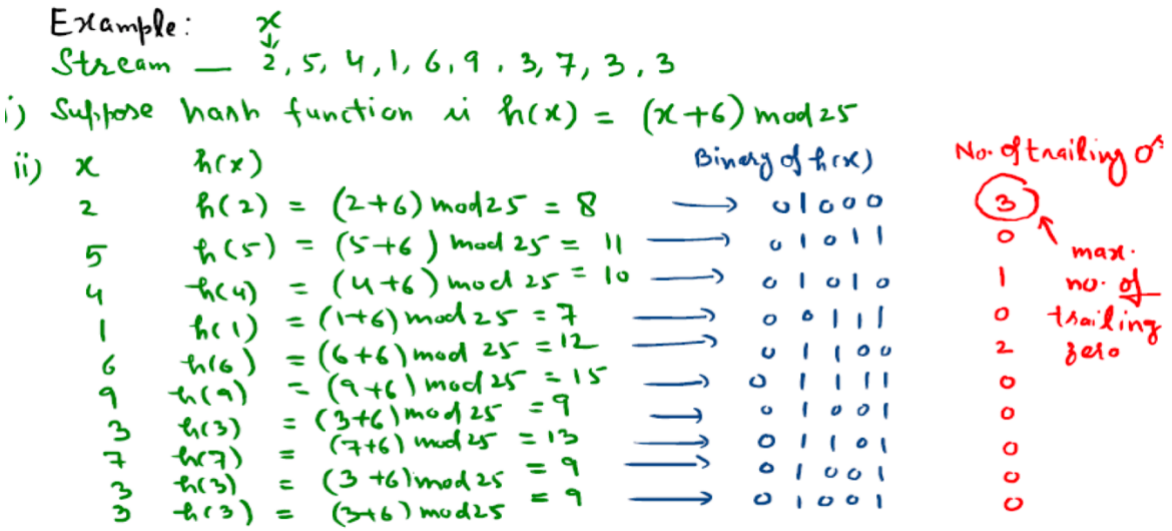
Here is a step-by-step example of the Flajolet-Martin algorithm:

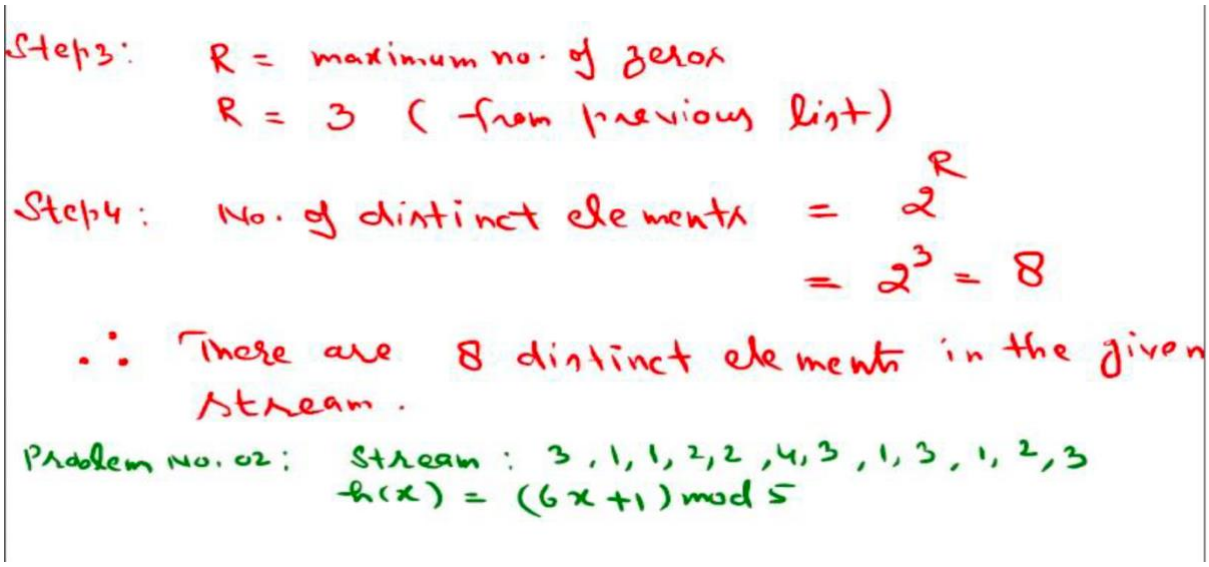
1. Define a hash function: The first step is to define a hash function that maps the data elements to a fixed-size bit array. The hash function should have the property that different elements map to different hash values with high probability.
2. Initialize the counters: For each bit position in the hash values, initialize a counter that will keep track of the number of hash values that have a zero at that position.
3. Process the data stream: For each data item in the stream, compute its hash value and increment the counter for each bit position where the corresponding bit in the hash value is zero.
4. Estimate the number of distinct elements: The number of distinct elements can be estimated by taking the median of the counter values across all bit positions.

Pseudo Code-Stepwise Solution:

- Selecting a hash function h so each element in the set is mapped to a string to at least log2n bits.
- For each element x, r(x)= length of trailing zeroes in h(x)
- R= max(r(x))
  => Distinct elements= 2^R

Example





Estimating Moments

Estimating moments in mining data streams refers to the process of computing summary statistics of a data stream, such as the mean, variance, and skewness, in real-time. Moments are important features of a data stream, as they provide information about the central tendency, dispersion, and shape of the data.

Aim of Estimating Moments:
To compute distribution of frequencies of different elements in a stream.

1st Moment = Sum of all mi (length of the stream)

2nd Moment = Sum of all mi^2(Surprise Number)
Where mi is frequency of value i.

Measuring Data Uniformity: Estimating Moments
Moments for Stream: 1, 3, 2, 1, 2, 3, 4, 3, 1, 2, 3, 1
• Occurrences: 1(x4), 2(x3), 3(x4), 4(x1)
• 0th Moment = $4^0 + 3^0 + 4^0 + 1^0 = 4$ (Distinct items)
• 1st Moment = $4^1 + 3^1 + 4^1 + 1^1 = 12$ (Stream length)
• 2nd Moment = $4^2 + 3^2 + 4^2 + 1^2 = 42$ (Surprise number)

There are several algorithms for estimating moments in a data stream, including:

1. **AMS (Algorithm for the estimation of Moments in streams):** This is a well-known algorithm for estimating the mean, variance, and skewness of a data stream. It uses a combination of sliding windows and random sampling to reduce the variance of the estimates, and to handle large and high-volume data streams.
2. **t-Digest:** This is a recent algorithm for estimating quantiles and moments of a data stream. It is based on a compact and scalable data structure, the t-Digest, which is used to store the values of the data stream. The t-Digest is used to estimate the moments and quantiles of the data stream with high accuracy and low variance.
3. **Count-Sketch:** This is a probabilistic data structure that is used to estimate the frequency of occurrence of elements in a data stream. It is based on a matrix of counters and a set of hash functions. The Count-Sketch is used to estimate the moments of the data stream, such as the mean, variance, and skewness, by aggregating the counts of the elements.

The choice of algorithm for estimating moments in a data stream depends on the specific requirements of the application, such as the desired accuracy, computational complexity, and the volume and velocity of the data stream.

Alon-Matias-Szegedy (AMS) algorithm

The Alon-Matias-Szegedy (AMS) algorithm is a probabilistic algorithm used for estimating the number of distinct elements in a stream of data. The algorithm works by randomly sampling a subset of the data stream and using the number of unique elements in the sample to estimate the number of unique elements in the entire stream.

Here is a step-by-step example of the AMS algorithm:

1. **Choose a sample size:** The first step is to choose the sample size, which determines the number of elements that will be randomly selected from the data stream. A larger sample size will result in a more accurate estimate, but also requires more processing.
2. **Initialize the sample:** Create an empty sample.
3. **Process the data stream:** For each data item in the stream, randomly decide whether to include it in the sample or not. The probability of including an element in the sample should be proportional to the sample size.
4. **Count the number of unique elements in the sample:** Use a hash table or other data structure to count the number of unique elements in the sample.
5. **Estimate the number of distinct elements:** The number of distinct elements in the entire stream can be estimated by multiplying the number of unique elements in the sample by the ratio of the size of the stream to the sample size.

EXAMPLE

For example, having this toy problem\data stream:

a, b, c, b, d, a, c, d, a, b, d, c, a, a, b

We calculate the second moment like this:

5^2 + 4^2 + 3^2 + 3^2 = 59

(because 'a' occurs 5 times in the data stream, 'b' 4 times, and so on)

Because we cannot store all the data stream in memory, we can use an algorithm for the estimate of the second moment:

The **Alon-Matias-Szegedy Algorithm (AMS algorithm)**, that estimate the second moment using this formula:

E(n *(2 * X.value − 1))

In which X is an univocal element of the stream, randomically selected, and X.value is a counter, that, as we read the stream, add to 1 each time we encounter another occurrence of the x element from the time we selected it.

n represents the length of the data stream, and "E" is the mean notation.

Doing an example with the previous data stream, let's assume we selected "a" at the 13th position of the data stream, "d" at the 8th and "c" at the 3th. We haven't selected "b".

```
a, b, c, b, d, a, c, d, a, b, d, c, a, a, b
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      x              x              x
```

Selected like this, we have:

X.element = "a" X.value = 2

X.element = "c" X.value = 3

X.element = "d" X.value = 2

The estimate by the AMS algorithm is:

(15*(2 * 2 - 1) + 15*(2 * 3 - 1) + 15*(2 * 2 - 1))/3 = 55

Which is pretty near the true value of the second moment calculated before (59).

Counting oneness in a window

Counting the number of distinct elements in a window of a data stream, also known as the "counting oneness in a window", is a common task in data stream mining. The goal is to keep track of the number of unique elements in a sliding window of a data stream, as new data is continuously generated and added to the stream.

There are several algorithms for counting the number of distinct elements in a window of a data stream, including:

1. **Bloom Filter:** This is a probabilistic data structure that is used to determine if an element is in a set or not. It is based on a bit array and a set of hash functions. The Bloom Filter is used to count the number of distinct elements in a window of a data stream by counting the number of set bits in the bit array.
2. **Count-Min Sketch:** This is a probabilistic data structure that is used to estimate the frequency of occurrence of elements in a data stream. It is based on a matrix of counters and a set of hash functions. The Count-Min Sketch is used to count the number of

distinct elements in a window of a data stream by aggregating the counts of the elements in the matrix of counters.

3. **Space-Saving Algorithm:** This is a deterministic algorithm that is used to maintain a summary of the most frequent elements in a data stream. It is based on a priority queue and a set of counters. The Space-Saving Algorithm is used to count the number of distinct elements in a window of a data stream by maintaining a summary of the most frequent elements in the window and discarding the least frequent elements.

The choice of algorithm for counting the number of distinct elements in a window of a data stream depends on the specific requirements of the application, such as the desired accuracy, computational complexity, and memory requirements.

In conclusion, counting the number of distinct elements in a window of a data stream is a common task in data stream mining. By using the appropriate algorithm, organizations can efficiently keep track of the number of unique elements in a sliding window of a data stream, even for large and high-volume data streams.

Datar-Gionis-Indyk-Motwani (DGIM) algorithm

The Datar-Gionis-Indyk-Motwani (DGIM) algorithm is a probabilistic algorithm used for estimating the number of ones in a sliding window of binary data. The algorithm works by keeping track of "buckets" that represent ranges of ones in the window, and updating the buckets as the window slides.

Example: from Slides

Decaying Window

A decaying window is a sliding window technique used in data analytics that assigns higher weights to recent data points and lower weights to older data points. This approach helps to give more significance to recent information in the analysis and reduces the impact of older, potentially less relevant data. The exact decay function used can vary, such as exponential decay or linear decay, and the decaying window is commonly used in time series analysis, trend analysis, and anomaly detection.

Real-time Analytics Platform (RTAP) applications

Real-Time Analytics Platform (RTAP) applications are software systems designed to process and analyze large amounts of data in real-time, providing quick insights and enabling real-time decision making. Here are some of the common applications of RTAPs in detail:

1. **Financial trading:** In high-frequency trading, RTAPs are used to analyze real-time market data and execute trades within milliseconds. These platforms use algorithms to make informed decisions based on market trends and other market data.
2. **Log analysis:** RTAPs can be used to process and analyze log data in real-time, helping organizations identify and respond to any security threats or system issues.
3. **Fraud detection:** RTAPs can help detect fraudulent activities by analyzing transactional data in real-time. This enables organizations to quickly identify and respond to any potential fraud.
4. **IoT:** RTAPs can be used to process and analyze data from IoT devices in real-time, providing real-time insights into the performance and usage of these devices.
5. **Customer Experience:** RTAPs can be used to analyze customer data in real-time, providing real-time personalization to enhance the customer experience.
6. **Supply chain management:** RTAPs can be used to monitor the real-time performance of the supply chain, allowing organizations to optimize operations and make informed decisions.
7. **Marketing:** RTAPs can be used to analyze marketing data in real-time, enabling organizations to optimize campaigns based on real-time results and adjust strategies as needed.

In conclusion, RTAPs are powerful tools that provide real-time insights into a variety of data types, enabling organizations to make informed decisions and respond to changing conditions quickly.

Real time sentiment analysis

1. **Definition:** Real-time sentiment analysis is the process of identifying and extracting opinions and emotions from a given piece of text in real-time.

2. **Purpose:** The goal of real-time sentiment analysis is to provide immediate insights into the tone and emotion behind a piece of text, be it a social media post, review, or comment.
3. **Input:** Sentiment analysis algorithms take in natural language text as input, which can be in the form of a tweet, Facebook post, review, etc.
4. **Output:** The output of sentiment analysis algorithms is typically a sentiment score, which can range from negative to positive, and indicates the overall polarity of the text.
5. **Techniques:** Sentiment analysis algorithms make use of various techniques such as machine learning, natural language processing, and lexicon-based analysis.
6. **Applications:** Real-time sentiment analysis is used in a variety of applications such as marketing and advertising, customer service, politics, and social media monitoring.
7. **Example:** Consider a tweet that says "I love this new phone! It's the best one I've ever had." The sentiment analysis algorithm would likely assign a positive sentiment score to this text, indicating that the writer is expressing a positive opinion.

Stock Market Analysis

Bitcoin prediction using Time series analysis