UNIT IV

CHAPTER 4

Mining Big Data Streams

University Prescribed Syllabus

The Stream Data Model : A DataStream-Management System, Examples of Stream Sources, Stream Queries, Issues in Stream Processing. Sampling Data in a Stream : Sampling Techniques, Filtering Streams: The Bloom Filter Counting Distinct Elements in a Stream : The Count-Distinct Problem, The Flajolet-Martin Algorithm, Combining Estimates, Space Requirements. Counting Ones in a Window : The Cost of Exact Counts, The Datar-Gionis-Indyk, Motwani Algorithm, Query Answering in the DGIM Algorithm.

Self-learning Topics : Streaming services like Apache Kafka/Amazon Kinesis/Google Cloud DataFlow. Standard spark streaming library. Integration with IOT devices to capture real time stream data.

4.1 THE STREAM MODEL

- As data arrives in various streams, it's important to stream the data as the data which arrives cannot be stored immediately and the probability of losing the data is more.
- To mine the data, we need to create the sample of stream data and to filter the stream to eliminate most of the "undesirable" data elements.

- To summarize the large stream of data, consider the fixed length window consisting of last n elements.
- The data stream model is considered for the management of data, along with the stream queries and issues in stream processing
- In recent years, advances in hardware technology have facilitated the ability to collect data continuously. Simple transactions of everyday life such as using a credit card, a phone or browsing the web lead to automated data storage. Similarly, advances in information technology have lead to large flows of data across IP networks.
- In many cases, these large volumes of data can be mined for interesting and relevant information in a wide variety of applications. When the volume of the underlying data is very large, it leads to a number of computational and mining challenges :
- With increasing volume of the data, it is no longer possible to process the data efficiently by using multiple passes. Rather, one can process a data item at most once.
- This leads to constraints on the implementation of the underlying algorithms. Therefore, stream mining algorithms typically need to be designed so that the algorithms work with one pass of the data.
- In most cases, there is an inherent temporal component to the stream mining process. This is because the data may evolve over time.
- This behavior of data streams is referred to as temporal locality. Therefore, a straightforward adaptation of one-pass mining algorithms may not be an effective solution to the task. Stream mining algorithms need to be carefully designed with a clear focus on the evolution of the underlying data.

- Another important characteristic of data streams is that they are often mined in a distributed fashion. Furthermore, the individual processors may have limited processing and memory.
- Common examples of streaming data sources include :
  - IoT sensors
  - Real-time advertising platforms
  - Server and security logs
  - Click-stream data from apps and websites

4.1.1 Data Stream Mining Characteristics

- **Continuous Stream of Data :** High amount of data in an infinite stream. we do not know the entire dataset
- **Concept Drifting :** The data change or evolves over time
- **Volatility of data :** The system does not store the data received (Limited resources). When data is analysed it's discarded or summarised.

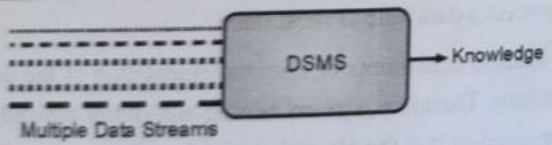


Fig. 4.1.1 : Data stream management system

Next, is discussed how to analyse Data Streams. Data-Based Techniques rely on analysing a representative subset of data. This technique also is used as pre-processing for Data Stream algorithms. On the Other Hand, Mining Techniques are enhanced versions of traditional Data Mining Algorithms.

4.2 DATA BASED TECHNIQUES

Sampling is based on selecting subset of data uniformly distributed.

- **Reservoir Sampling** (Fixed Size Sample) This algorithm sample a subset m of the data from the stream. After the i^{th} item is chosen with probability 1 / i and if the i element is already sampled its randomly replace a sampled item.
- **Min-Wise Sampling** is based on assigning a random γ in range 0 to 1 to a subset of samples m. When the system retrieves m elements, we select the sample with the minimum γ.
- **Sketching** is based on reducing the dimensionality of the dataset. Sketch is based on performing a linear transformation of the data delivering a summarization of the Stream. See also Count-min Sketch.
- **Approximation Techniques** are based on maintaining only a subset of data and discarding previous data (sliding windows).
- **Sequence based :** The size of the window depends on the number of observations. The window store γ elements and when a new element arrives, the last element is removed if the window is full.
- **Timestamp based :** The size of the window depends on time. The window is bounded in instant T_n and T_{n+1} and holds the elements received in this period.
- At a high level, MapReduce breaks input data into fragments and distributes them across different machines. The input fragments consist of key-value pairs. Parallel map tasks process the chunked data on machines in a cluster. The mapping output then serves as input for the reduce stage. The reduce task combines the result into a particular key-value pair output and writes the data to HDFS.

- The Hadoop Distributed File System usually runs on the same set of machines as the MapReduce software. When the framework executes a job on the nodes that also store the data, the time to complete the tasks is reduced significantly.

4.2.1 Data Stream Management System (DSMS)

UQ. Explain with block diagram architecture of Data stream Management System. **(MU - Dec. 19, 10 Marks)**

UQ. Explain abstract architecture of Data Stream Management System (DSMS). **(MU - Dec. 16, 10 Marks)**

UQ. What is Data Stream Management System? Explain with block diagram. **(MU - May 17, 10 Marks)**

- We can view a stream processor as a kind of data-management system, the high-level organization which is shown in Fig. 4.2.1.
- Any number of streams can enter the system. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform.
- The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system. The latter system controls the rate at which data is read from the disk, and therefore never has to worry about data getting lost as it attempts to execute queries.
- Streams may be archived in a large archival store, but we assume it is not possible to answer queries from the archival store.
- It could be examined only under special circumstances using time-consuming retrieval processes.

- There is also a working store, into which summaries or parts of streams may be placed, and which can be used for answering queries.
- The working store might be disk, or it might be main memory, depending on how fast we need to process queries. But either way, it is of sufficiently limited capacity that it cannot store all the data from all the streams.

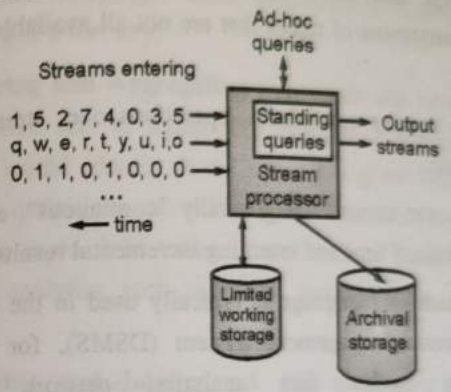


Fig. 4.2.1 : A data-stream-management system

- A Data Stream Management System (DSMS) is a computer software system to manage continuous **Data Streams**. It is similar to a **Database Management System** (DBMS), which is, however, designed for static data in conventional **Databases**.
- A DBMS also offers a flexible query processing so that the information needed can be expressed using queries. However, in contrast to a DBMS, a DSMS executes a continuous query that is not only performed once, but is permanently installed.
- Therefore, the query is continuously executed until it is explicitly uninstalled. Since most DSMS are data-driven, a continuous query produces new results as long as new data arrive at the system. This basic concept is similar to **Complex Event Processing** so that both technologies are partially coalescing.

4.3 STREAMSQL

- StreamSQL is a query language that extends SQL with the ability to process real-time data streams. SQL is primarily intended for manipulating relations (also known as tables), which are finite bags of tuples (rows).
- StreamSQL adds the ability to manipulate streams, which are infinite sequences of tuples that are not all available at the same time.
- Because streams are infinite, operations over streams must be monotonic.
- Queries over streams are generally "continuous", executing for long periods of time and returning incremental results.
- The StreamSQL language is typically used in the context of a Data Stream Management System (DSMS), for applications including market data analytics, network monitoring, surveillance, e-fraud detection and prevention, clickstream analytics and real-time compliance (anti-money laundering, RegNMS, MiFID).
- Other streaming and continuous variants of SQL include StreamSQL.io, Kafka KSQL, SQLStreamBuilder, WSO2 Stream Processor, SQLStreams, SamzaSQL, and Storm SQL.

4.3.1 StreamSQL Operations

- It extends the type system of SQL to support streams in addition to tables. Several new operations are introduced to manipulate streams.
- **Selecting from a stream :** A standard SELECT statement can be issued against a stream to calculate functions (using the target list) or filter out unwanted tuples (using a WHERE clause). The result will be a new stream.

- **Stream-Relation Join** : A stream can be joined with a relation to produce a new stream. Each tuple on the stream is joined with the current value of the relation based on a predicate to produce 0 or more tuples.
- **Union and Merge** : Two or more streams can be combined by unioning or merging them. Unioning combines tuples in strict FIFO order. Merging is more deterministic, combining streams according to a sort key.
- **Windowing and Aggregation** : A stream can be windowed to create finite sets of tuples. For example, a window of size 5 minutes would contain all the tuples in a given 5-minute period. Window definitions can allow complex selections of messages, based on tuple field values. Once a finite batch of tuples is created, analytics such as count, average, max, etc., can be applied.
- **Windowing and Joining** : A pair of streams can also be windowed and then joined together. Tuples within the join windows will combine to create resulting tuples if they fulfill the predicate.

4.3.2 Examples of Stream Sources

GQ. Explain different types of Stream sources.

Sensor Data

- Imagine a temperature sensor bobbing about in the ocean, sending back to a base station a reading of the surface temperature each hour. The data produced by this sensor is a stream of real numbers. It is not a very interesting stream, since the data rate is so low. It would not stress modern technology, and the entire stream could be kept in main memory, essentially forever.

- Now, give the sensor a GPS unit, and let it report surface height instead of temperature. The surface height varies quite rapidly compared with temperature, so we might have the sensor send back a reading every tenth of a second. If it sends a 4-byte real number each time, then it produces 3.5 megabytes per day. It will still take some time to fill up main memory, let alone a single disk.
- But one sensor might not be that interesting. To learn something about ocean behavior, we might want to deploy a million sensors, each sending back a stream, at the rate of ten per second. A million sensors isn't very many; there would be one for every 150 square miles of ocean. Now we have 3.5 terabytes arriving every day, and we definitely need to think about what can be kept in working storage and what can only be archived.

☞ **Image Data**

- Satellites often send down to earth streams consisting of many terabytes of images per day. Surveillance cameras produce images with lower resolution than satellites, but there can be many of them, each producing a stream of images at intervals like one second.
- London is said to have six million such cameras, each producing a stream.

☞ **Internet and Web Traffic**

- A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs. Normally, the job of the switch is to transmit data and not to retain it or query it. But there is a tendency to put more capability into the switch, e.g., the ability to detect denial-of-service attacks or the ability to reroute packets based on information about congestion in the network.

- Web sites receive streams of various types. For example, Google receives several hundred million search queries per day. Yahoo! accepts billions of "clicks" per day on its various sites. Many interesting things can be learned from these streams. For example, an increase in queries like "sore throat" enables us to track the spread of viruses. A sudden increase in the click rate for a link could indicate some news connected to that page, or it could mean that the link is broken and needs to be repaired.

4.3.3 Stream Queries

UQ. With respect to data stream querying, give example of Ad-hoc queries and standing queries.

(MU - May 17, 5 Marks)

- There are two ways that queries get asked about streams. We show in Fig. 4.2.1 a place within the processor where standing queries are stored. These queries are, in a sense, permanently executing, and produce outputs at appropriate times.

Example 4.3.1

The stream produced by the ocean-surface-temperature sensor. It might have a standing query to output an alert whenever the temperature exceeds 25 degrees centigrade. This query is easily answered, since it depends only on the most recent stream element.

- Alternatively, we might have a standing query that, each time a new reading arrives, produces the average of the 24 most recent readings. That query also can be answered easily, if we store the 24 most recent stream elements. When a new stream element arrives, we can drop from the working store the 25th most recent element, since it will never again be needed.

- Another query we might ask is the maximum temperature ever recorded by that sensor. We can answer this query by retaining a simple summary: the maximum of all stream elements ever seen. It is not necessary to record the entire stream. When a new stream element arrives, we compare it with the stored maximum, and set the maximum to whichever is larger.
- We can then answer the query by producing the current value of the maximum. Similarly, if we want the average temperature over all time, we have only to record two values: the number of readings ever sent in the stream and the sum of those readings. We can adjust these values easily each time a new reading arrives, and we can produce their quotient as the answer to the query.
- The other form of query is ad-hoc, a question asked once about the current state of a stream or streams. If we do not store all streams in their entirety, as normally we cannot, then we cannot expect to answer arbitrary queries about streams. If we have some idea what kind of queries will be asked through the ad-hoc query interface, then we can prepare for them by storing appropriate parts or summaries of streams as in Example 4.3.1.
- If we want the facility to ask a wide variety of ad-hoc queries, a common approach is to store a sliding window of each stream in the working store. A sliding window can be the most recent n elements of a stream, for some n, or it can be all the elements that arrived within the last t time units, e.g., one day. If we regard each stream element as a tuple, we can treat the window as a relation and query it with any SQL query. Of course the stream-management system must keep the window fresh, deleting the oldest elements as new ones come in.

Example 4.3.2

Web sites often like to report the number of unique users over the past month. If we think of each login as a stream element, we can maintain a window that is all logins in the most recent month. We must associate the arrival time with each login, so we know when it no longer belongs to the window. If we think of the window as a relation Logins(name, time), then it is simple to get the number of unique users over the past month. The SQL query is: SELECT COUNT(DISTINCT(name)) FROM Logins WHERE time >= t; Here, t is a constant that represents the time one month before the current time. Note that we must be able to maintain the entire stream of logins for the past month in working storage. However, for even the largest sites, that data is not more than a few terabytes, and so surely can be stored on disk.

4.4 KEY ISSUES IN BIG DATA STREAM ANALYSIS

UQ. What are the challenges of querying on large data stream? (MU - May 18, 5 Marks)

Scalability

- One of the main challenges in big data streaming analysis is the issue of scalability. The big data stream Is experiencing exponential growth in a way much faster than computer resources.
- The processors follow Moore's law, but the size of data is exploding. Therefore, research efforts should be geared towards developing scalable frameworks and algorithms that will accommodate data stream computing mode, effective resource allocation strategy and parallelization issues to cope with the ever-growing size and complexity of data.

Integration

- Building a distributed system where each node has a view of the data flow, that is, every node performing analysis with a small number of sources, then aggregating these views to build a global view is non-trivial.

- An integration technique should be designed to enable efficient operations across different datasets.

☞ Fault-tolerance

- High fault-tolerance is required in life-critical systems.
- As data is real-time and infinite in big data stream computing environments, a good scalable high fault-tolerance strategy is required that allows an application to continue working despite component failure without interruption.

☞ Timeliness

- Time is of the essence for time-sensitive processes such as mitigating security threats, thwarting fraud, or responding to a natural disaster.
- There is a need for scalable architectures or platforms that will enable continuous processing of data streams which can be used to maximize the timeliness of data.
- The main challenge is implementing a distributed architecture that will aggregate local views of data into global view with minimal latency between communicating nodes.

☞ Consistency

- Achieving high consistency (i.e. stability) in big data stream computing environments is non-trivial as it is difficult to determine which data are needed and which nodes should be consistent.
- Hence a good system structure is required.

☞ Heterogeneity and incompleteness

- Big data streams are heterogeneous in structure, organisations, semantics, accessibility and granularity. The challenge here is how to handle an always ever-increasing data, extract meaningful content out of it, aggregate and correlate streaming data from multiple sources in real-time.

- A competent data presentation should be designed to reflect the structure, diversity and hierarchy of the streaming data.

☞ Load balancing

- A big data stream computing system is expected to be self-adaptive to data streams changes and avoid load shedding.
- This is challenging as dedicating resources to cover peak loads 24/7 is impossible and load shedding is not feasible when the variance between the average load and the peak load is high.
- As a result, a distributing environment that automatically streams partial data streams to a global centre when local resources become insufficient is required.

☞ High throughput

- Decision with respect to identifying the sub-graph that needs replication, how many replicas are needed and the portion of the data stream to assign to each replica is an issue in big data stream computing environment.
- There is a need for good multiple instances replication if high throughput is to be achieved.

☞ Privacy

- Big data stream analytics created opportunities for analyzing a huge amount of data in real-time but also created a big threat to individual privacy.
- According to the International Data Cooperation (IDC), not more than half of the entire information that needs protection is effectively protected.
- The main challenge is proposing techniques for protecting a big data stream dataset before its analysis.

Accuracy

- One of the main objectives of big data stream analysis is to develop effective techniques that can accurately predict future observations.
- However, as a result of inherent characteristics of big data such as volume, velocity, variety, variability, veracity, volatility, and value, big data analysis strongly constrain processing algorithms spatio-temporally and hence stream-specific requirements must be taken into consideration to ensure high accuracy.

4.5 SAMPLING TECHNIQUES FOR EFFICIENT STREAM PROCESSING

UQ. Describe any two sampling techniques for big data with the help of examples. **(MU - May 16, 10 Marks)**

4.5.1 Sliding Window

- This is the simplest and most straightforward method. A first-in, first-out (FIFO) queue with size n and a skip / sub-sampling factor $k \geq 1$ is maintained.

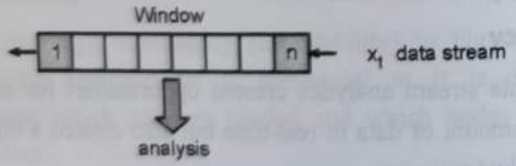


Fig. 4.5.1 : Sliding Window

- In addition to that, a stride factor $s \geq 1$ describes by how many time-steps the window is shifted before analyzing it.

Advantage

(1) Simple to implement.

(2) Deterministic reservoir can be filled very fast from the beginning.

Drawbacks

The time history represented by the reservoir R is short; long-term concept drifts cannot be detected easily outliers can create noisy analyses.

4.5.2 Unbiased Reservoir Sampling

- A reservoir R is maintained such that at time $t > n$ the probability of accepting point x(t) in the reservoir is equal to n/t.
- The algorithm [1] is as follows :
  - Fill the reservoir R with the first n points of the stream.
  - At time $t > n$ replace a randomly chosen (equal probability) entry in the reservoir R with acceptance probability n/t.
- This leads to a reservoir R(t) such that each point x(1)...x(t) is contained in R(t) with equal property n/t.

Advantages

(1) The reservoir contains data points from all history of the stream with equal probability.

(2) Very simple implementation; adding a point requires only O(1)

Drawbacks

A concept drift cannot be compensated; the oldest data point x(1) is equal important in this sampling technique as the latest data point x(t).

4.5.3 Biased Reservoir Sampling

The bias function associated with the r-th data point at the time of arrival of the t-th point ($r \leq t$) is given by f(r, t) and is related to the probability p(r, t) of the r-th point belonging to the reservoir at the time of arrival of the t-th point. Specifically, p(r, t) is proportional to f(r, t). The function f(r, t) is monotonically decreasing with t

(for fixed r) and monotonically increasing with r (for fixed t). Therefore, the use of a bias function ensures that recent points have higher probability of being represented in the sample reservoir. Next, we define the concept of a bias sensitive sample S(t), which is defined by the bias function f(r, t).

Definition : Let f(r, t) be the bias function for the r-th point at the arrival of the t-th point. A biased sample S(t) at the time of arrival of the t-th point in the stream is defined as a sample such that the relative probability p(r, t) of the r-th point belonging to the sample S(t) (of size n) is proportional to f(r, t).

Algorithm 4.5.1

We start off with an empty reservoir with capacity n = pin/λ, and use the following replacement policy to gradually fill up the reservoir. Let us assume that at the time of (just before) the arrival of the t-th point, the fraction of the reservoir filled is F(t) ∈ [0, 1]. When the (t + 1)-th point arrives. We add it to the reservoir with insertion probability pin. However, we do not necessarily delete one of the old points in the reservoir. We flip a coin with success probability F(t). In the event of a success, we randomly pick one of the points in the reservoir, and replace its position in the sample array by the incoming (t + 1)-th point. In the event of a failure, we do not delete any of old points and simply add the (t + 1)-th point to the reservoir. In the latter case, the number of points in the reservoir (current sample size) increases by 1.

The algorithm has a lower insertion probability, and a corresponding reduced reservoir requirement. In this case, the value of λ and n are decided by application specific constraints, and the value

of pin is set to n · λ. We show that the sample from the modified algorithm shows the same bias distribution, but with a reduced reservoir size.

- In biased reservoir sampling Alg. 4.5.1, [2] the probability of a data point x(t) being in the reservoir is a decreasing function of its lingering time within R.
- So the probability of finding points of the sooner history in R is high. Very old data points will be in R with very low probability.

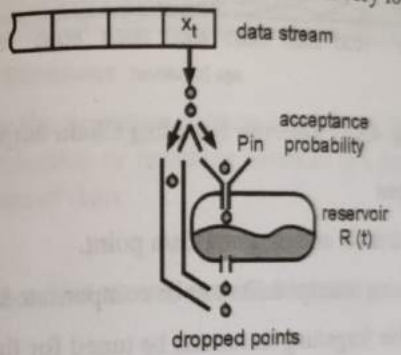


Fig. 4.5.2

☞ **Illustration of the Biased Reservoir Sampling**

The probability that point x(r) is contained in R(t) equals to

$$P(r, t) = e^{-\lambda (t-r)}$$

- So this is an exponential forgetting. For details of the algorithm, see [2] and the goreservoir package.
- The example from the github.com\andremueller\goservoir package shows the lingering time of a stack of unbiased reservoir samplers.

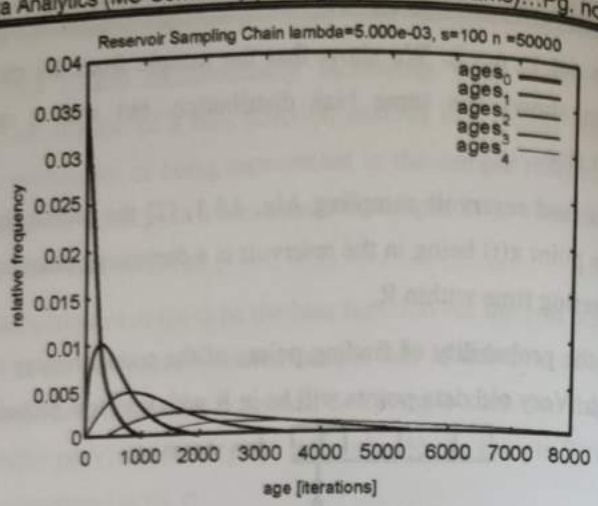


Fig. 4.5.3 : Reservoir Sampling Chain output

☞ Advantages

(1) O(1) algorithm for adding a new data point.

(2) Slowly moving concept drifts can be compensated.

(3) An adjustable forgetting factor can be tuned for the application of interest.

☞ Drawbacks

It is a randomized technique. So the algorithm is non-deterministic. However, the variance might be estimated by running an ensemble of independent reservoirs.

$$R_1 \ldots R_B$$

4.5.4 Histograms

A histogram is maintained while observing the data stream. Hereto, data points are sorted into intervals/buckets

$$[l_i, u_i]$$

- If the useful range of the observed values is known in advance, a simple vector with counts and breakpoints could do the job.

- V-optimal histograms tries to minimize the variance within each histogram bucket.
- Proposes an algorithm for efficiently maintaining an approximate V-optimum histogram from a data stream. This is of relevance for interval data, such as a time-series of temperature values; i.e., absolute value and distance between values have a meaning.

4.6 FILTERING STREAMS : BLOOM FILTER WITH ANALYSIS

GQ. Explain filtering streams.

☞ Filtering Streams

It identifies the sequence patterns in a stream. Stream filtering is the process of selection or matching instances of a desired pattern in a continuous stream of data.

Example

Assume that a data stream consists of tuples

Filtering steps :

(i) Accept the tuples that meet a criterion in the stream,

(ii) Pass the accepted tuples to another process as a stream and

(iii) Discard remaining tuples.

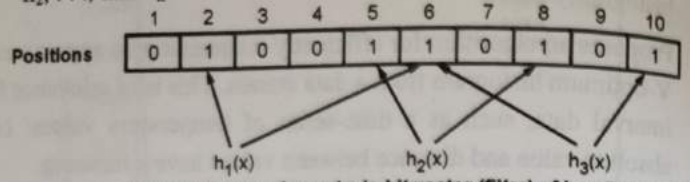
UQ. How bloom filter is useful for big data analytics? Explain with one example. **(MU - Dec. 18, 10 Marks)**

UQ. Explain the concept of Blooms Filter using an example. **(MU - Dec. 17, 10 Marks)**

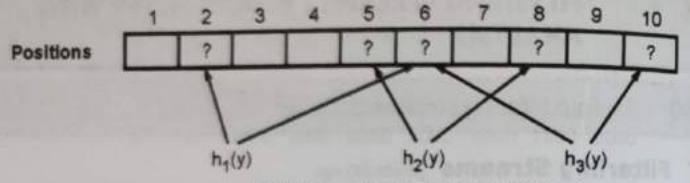
☞ Bloom Filter with Analysis

- A simple space-efficient data structure introduced by Burton Howard Bloom in 1970. The filter matches the membership of an element in a dataset.
- The filter is basically a bit vector of length m that represent a set $S = \{x_1, x_2, \ldots, x_m\}$ of m elements,

- Initially all bits 0. Then, define k independent hash functions, h_1, h_2, . . ., and h_k.



(a) Inserting an element x in bit vector (filter) of length m = 10



(b) Finding element y is the Filter

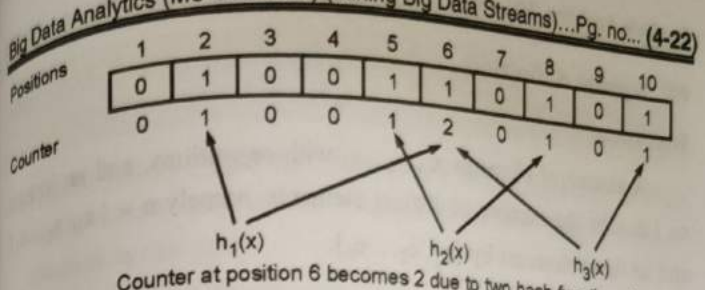
(a) Inserting an element x in bit vector (filter) of length m = 10, (b) finding an element y in an example of Bloom filter

Fig. 4.6.1

- Each of which maps (hashes) some element x in set S to one of the m array positions with a uniform random distribution.
- Number k is constant, and much smaller than m. That is, for each element $x \in S$, the bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$. [$\in$ is symbol in set theory for 'contained in'.]

☞ Counting Bloom Filter: A Variant of Bloom Filter

- It mmaintains a counter for each bit in the Bloom filter. The counters corresponding to the k hash values increment or decrement, whenever an element in the filter is added or deleted, respectively.
- As soon as a counter changes from 0 to 1, the corresponding bit in the bit vector is set to 1. When a counter changes from 1 to 0, the corresponding bit in the bit vector is set to 0. The counter basically maintains the number of elements that hashed.



Counter at position 6 becomes 2 due to two hash functions h_1 and h_3, when an element x is inserted in bit vector (filter) of lenght m = 10

Fig. 4.6.2 : Example of counting bloom filter

4.7 COUNTING DISTINCT ELEMENTS IN A STREAM

It relates to finding the number of dissimilar elements in a data stream. The stream of data contains repeated elements. This is a well-known problem in networking and databases.

4.7.1 The Count-Distinct Problem

UQ. What do you mean by Counting Distinct Element in a stream? Illustrate with an example working of an Flajolet-martin algorithm used to count number of distinct elements.

(MU - Dec. 19, 10 Marks)

- The count-distinct problem (also known in applied mathematics as the cardinality estimation problem) is the problem of finding the number of distinct elements in a data stream with repeated elements.
- This is a well-known problem with numerous applications.
- The elements might represent IP addresses of packets passing through a router, unique visitors to a web site, elements in a large database, motifs in a DNA sequence, or elements of RFID/sensor networks.

☞ Formal definition

Instance

A stream of elements $x_1, x_2, ...x_3$ with repetitions, and an integer m. Let n be the number of distinct elements, namely $n = \{x_1, x_2, ...x_3\}$ and let these elements by $\{e_1, e_2, ...e_n\}$.

☞ Objective

Find an estimate n of n using only m storage units, where $m << n$

An example of an instance for the cardinality estimation problem is the steam a,b,a,c,d,b,d. For this instance.

$$n = |\{a, b, c, d\}| = 4$$

☞ Naive solution [edit]

The naive solution to the problem is as follows :

- Inititalize a counter, c, to zero, $c \leftarrow 0$,
- Initialize an efficient dictionary data structure, D, such as hash table or search tree in which insertion and membership can be performet quickly.

For each element x_i, a membership query is issued.

If x_i is not a member of D ($x_i \notin D$)

Add x_i to D

Increase c by one, $c \leftarrow c + 1$

Otherwise ($x_i \in D$) do nothing.

Output n = c.

4.7.2 Flajolet Martin Algorithm

UQ. What do you mean by Counting Distinct Elements in a stream? Illustrate with an example working of a Flajolet – Martin Algorithm used to count number of distinct elements. **(MU - Dec. 19, 10 Marks)**

UQ. Give problem in Flajolet-Martin (FM) algorithm to count distinct element in a stream. **(MU - Dec. 16, 5 Marks)**

- Flajolet Martin Algorithm, also known as FM algorithm, is used to approximate the number of unique elements in a data stream or database in one pass.
- The highlight of this algorithm is that it uses less memory space while executing.
- Pseudo Code-Stepwise Solution :

1. Selecting a hash function h so each element in the set is mapped to a string to at least $\log_2 n$ bits.
2. For each element x, r(x) = length of trailing zeroes in h(x)
3. R = max(r(x)) = > Distinct elements = 2_R

☞ Reasons for using Flajolet Martin algorithm

- Let us compare this algorithm with our conventional algorithm using python code. Assume we have an array (stream in code) of data of length 20 with 8 unique elements.
- Using the brute force approach to find the number of unique elements in the array, each element is taken into consideration. Another array (st_unique in code) is formed for unique elements.
- Initially, the new array is empty (st_unique length equals zero), so naturally, the first element is not present in it.
- The first element is considered to be unique as it does not exist in the new array and thus a copy of the first element is inserted into the new array (1 is appended to st_unique)
- Similarly, all the elements are checked, if they are already present in the new array, they are not considered to be unique, else a copy of the element is inserted into the new array.

- Running the brute force algorithm for our array, we will get 8 elements in the new array.
- If each element takes 20 bytes of data, the new array will take 8*20 = 160 bytes memory to run the algorithm.

```
stream=[1,2,3,4,5,6,4,2,5,9,1,6,3,7,1,2,2,4,2,1]
print('Using conventional Algorithm:')
start_time = time.time()
st_unique=[]
for i in stream:
 if i in st_unique:
 continue
 else:
 st_unique.append(i)
 print('distinct elements',len(st_unique))
print("--- %s seconds ---" % (time.time() - start_time))
```

- For the same array, if use the FM algorithm for the same array, we define a variable (maxnum in code) that stores the maximum number of zeroes at the end.
- For each value in the array(stream in code), we run a loop to covert its hash function in the form ax + b mod c,(a = 1, b = 6 and c = 32 in this case) into binary (we place [2:] at the end because in python when converted to binary, the number starts with '0b').
- We run another loop to find if the number of zeroes at the end exceeds the maximum number of zeroes.
- In this case, if each variable occupies 2 bytes of data, the whole program takes 4*20= 80 bytes of data, i.e. half of the memory used in the above case.
- Here, we have considered the variables maxnum, sum, val, and j. We have not considered i, time, and stream as they are common in both of the codes.


```
stream=[1,2,3,4,5,6,4,2,5,9,1,6,3,7,1,2,2,4,2,1]
print('Using Flajolet Martin Algorithm:')
import time
start_time = time.time()
maxnum=0
for i in range(0,len(stream)):
val= bin((1*stream[i] + 6) % 32)[2:]


sum=0
for j in range(len(val)-1,0,-1):

if val[j]=='0':
sum+=1
else:
break
if sum>maxnum:
maxnum=sum
print('distict elements', 2**maxnum)
print("--- %s seconds ---" % (time.time() - start_time))
```

- For small values of m(where m is the number of unique elements), the brute force approach can work, but for large data sets or data streams, where m is very large, a lot of space is required. The compiler may not let us run the algorithm in some cases.
- This is where the Flajolet Martin Algorithm can be used.
- Not only does it occupy less memory, but it also shows better results in terms of time in seconds when the python code is run which can be shown in our output as we calculated seconds taken by both algorithms by using time.time() in python.

- As shown in the output, it can clearly be said that the FM algorithm takes very little time as compared to the conventional algorithm.

```
Using  Flajolet Martin Algorithm :
distict elements 8
- - - 0.00011801719665527344 seconds - -
Using conventional Algorithm
distinct elements 8
---
7.295608520507812e-05 seconds - - -
```

4.8 COMBINING ESTIMATES

GQ. Explain the combining estimates method.

- The strategy for combining the estimates of m, the number of distinct elements, that we obtain by using many different hash functions.
- Our first assumption would be that if we take the average of the values 2^R that we get from each hash function, we shall get a value that approaches the true m, the more hash functions we use. Consider a value of r such that 2^r is much larger than m. There is some probability p that we shall discover r to be the largest number of 0's at the end of the hash value for any of the m stream elements. Then the probability of finding r + 1 to be the largest number of 0's instead is at least p/2. However, if we do increase by 1 the number of 0's at the end of a hash value, the value of 2^R doubles.
- Consequently, the contribution from each possible large R to the expected value of 2^R grows as R grows, and the expected value of 2^R is actually infinite.

- Another way to combine estimates is to take the median of all estimates. The median is not affected by the occasional outsized value of 2^R, so the worry described above for the average should not carry over to the median. Unfortunately, the median suffers from another defect: it is always a power of 2. Thus, no matter how many hash functions we use, should the correct value of m be between two powers of 2, say 400, then it will be impossible to obtain a close estimate.
- There is a solution to the problem, however. We can combine the two methods. First, group the hash functions into small groups, and take their average. Then, take the median of the averages.
- It is true that an occasional outsized 2^R will bias some of the groups and make them too large. However, taking the median of group averages will reduce the influence of this effect almost to nothing.
- Moreover, if the groups themselves are large enough, then the averages can be essentially any number, which enables us to approach the true value m as long as we use enough hash functions. In order to guarantee that any possible average can be obtained, groups should be of size at least a small multiple of $\log_2 m$.

4.8.1 Space Requirements

- As we read the stream it is not necessary to store the elements seen. The only thing we need to keep in main memory is one integer per hash function; this integer records the largest tail length seen so far for that hash function and any stream element.
- If we are processing only one stream, we could use millions of hash functions, which is far more than we need to get a close estimate. Only if we are trying to process many streams at the

same time would main memory constrain the number of hash functions we could associate with any one stream.

- In practice, the time it takes to compute hash values for each stream element would be the more significant limitation on the number of hash functions we use.

4.9 COUNTING ONES IN A WINDOW

UQ. Give two applications for counting the number of 1's in a long stream of binary values. Using a stream of binary digits, illustrate how the DGIM algorithm will find the number of 1's ? **(MU - May 18, 5 Marks)**

- The Cost of Exact Counts Sliding window model - data elements arrive at every instant; each data element expires after exactly N time steps; and the portion of data that is relevant to gathering statistics or answering queries is the set of the last N elements to arrive.
- Able to count exactly the number of 1s in the last k bits for any k<=N, it is necessary to store all N bits of the window. Problem :
- Most counting applications N is still so large that it cannot be stored on disk or there are so many streams that windows for all cannot be stored.
- Another method: Random Sampling – this will fail when the 1s may not arrive in a uniform manner.

4.9.1 Datar-Gionis-Indyk-Motwani

UQ. Explain DGIM algorithm for counting ones in stream with example. **(MU - May 19, Dec. 18, 10 Marks)**

UQ. Using an example bit stream explain the working of the DGIM algorithm to count number of 1's in a data stream. **(MU - May 16, 10 Marks)**

The simplest case of an algorithm called DGIM. This version of the algorithm uses $O(\log^2 N)$ bits to represent a window of N bits and enables the estimation of the number of 1s in the window with an error of no more than 50%.

Divide the window into buckets, consisting of – The timestamp of its right 1 – The number of 1s in the bucket. This number must be power of 2, and we refer to the number of 1s as the size of the bucket.

The right end of a bucket always starts with a position with a 1.

- Number of 1s must be power of 2.
- Either one or two buckets with the same power of 2 number of 1s exists.
- Buckets do not overlap in timestamps
- Buckets are stored by size.
- Buckets disappear when their end-time is N time units in the past.

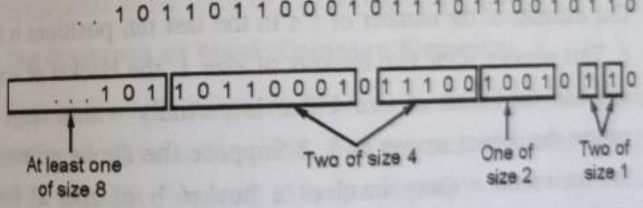


Fig. 4.9.1 : A bit-stream divided into buckets following the DGIM rules

4.9.2 Query Answering in the DGIM Algorithm

- Suppose we are asked how many 1's there are in the last k bits of the window, for some $1 \leq k \leq N$. Find the bucket b with the earliest timestamp that includes at least some of the k most recent bits. Estimate the number of 1's to be the sum of the sizes of all the buckets to the right (more recent) than bucket b, plus half the size of b itself

- **Example :** Suppose the stream is that of Fig. 4.9.1, and k = 10. Then the query asks for the number of 1's in the ten rightmost bits, which happen to be 0110010110. Let the current timestamp (time of the rightmost bit) be t. Then the two buckets with one 1, having timestamps t – 1 and t – 2 are completely included in the answer.
- The bucket of size 2, with timestamp t – 4, is also completely included. However, the rightmost bucket of size 4, with timestamp t–8 is only partly included. We know it is the last bucket to contribute to the answer, because the next bucket to its left has timestamp less than t – 9 and thus is completely out of the window.
- On the other hand, we know the buckets to its right are completely inside the range of the query because of the existence of a bucket to their left with timestamp t – 9 or greater.
- Our estimate of the number of 1's in the last ten positions is thus 6. This number is the two buckets of size 1, the bucket of size 2, and half the bucket of size 4 that is partially within range. Of course the correct answer is 5. 2 Suppose the above estimate of the answer to a query involves a bucket b of size 2j that is partially within the range of the query.
- Let us consider how far from the correct answer c our estimate could be. There are two cases: the estimate could be larger or smaller than c.
- **Case 1 :** The estimate is less than c. In the worst case, all the 1's of b are actually within the range of the query, so the estimate misses half bucket b, or $2j-1$ 1's. But in this case, c is at least 2j; in fact it is at least $2j+1 - 1$, since there is at least one bucket of each of the sizes $2j - 1, 2j - 2, \ldots, 1$. We conclude that our estimate is at least 50% of c.

- **Case 2 :** The estimate is greater than c. In the worst case, only the rightmost bit of bucket b is within range, and there is only one bucket of each of the sizes smaller than b. Then $c = 1 + 2j - 1 + 2j - 2 + \cdots + 1 = 2j$ and the estimate we give is $2j-1 + 2j-1 + 2j-2 + \cdots + 1 = 2j + 2j - 1 - 1$. We see that the estimate is no more than 50% greater than c.

4.9.3 Decaying Windows

GQ. Explain Decaying windows method.

It useful in applications which need identification of most common elements. Decaying window concept assigns more weight to recent elements. The technique computes a smooth aggregation of all the 1's ever seen in the stream, with decaying weights. When element further appears in the stream, less weight is given. The effect of exponentially decaying weights is to spread out the weights of the stream elements as far back in time as the stream flows.

The Problem of Most-Common Elements

- Suppose we have a stream whose elements are the movie tickets purchased all over the world, with the name of the movie as part of the element. We want to keep a summary of the stream that is the most popular movies "currently." While the notion of "currently" is imprecise, intuitively, we want to discount the popularity of a movie like Star Wars–Episode 4, which sold many tickets, but most of these were sold decades ago.
- On the other hand, a movie that sold n tickets in each of the last 10 weeks is probably more popular than a movie that sold 2n tickets last week but nothing in previous weeks. One solution would be to imagine a bit stream for each movie.
- The i^{th} bit has value 1 if the i^{th} ticket is for that movie, and 0 otherwise. Pick a window size N, which is the number of most

recent tickets that would be considered in evaluating popularity. Then, use the method of Section 4.6 to estimate the number of tickets for each movie, and rank movies by their estimated counts.

- This technique might work for movies, because there are only thousands of movies, but it would fail if we were instead recording the popularity of items sold at Amazon, or the rate at which different Twitter-users tweet, because there are too many Amazon products and too many tweeters. Further, it only offers approximate answers

☞ Definition of the Decaying Window

- An alternative approach is to redefine the question so that we are not asking for a count of 1's in a window. Rather, let us compute a smooth aggregation of all the 1's ever seen in the stream, with decaying weights, so the further back in the stream, the less weight is given.
- Formally, let a stream currently consist of the elements a1, a2, . . ., at, where a1 is the first element to arrive and at is the current element. Let c be a small constant, such as 10–6 or 10–9. Define the exponentially decaying window for this stream to be the sum

$$\sum_{i=0}^{t=1} a_{t-i}(1-c)^i$$

- The effect of this definition is to spread out the weights of the stream elements as far back in time as the stream goes.
- In contrast, a fixed window with the same sum of the weights, 1/c, would put equal weight 1 on each of the most recent 1/c elements to arrive and weight 0 on all previous elements. The distinction is suggested by Fig. 4.9.2.

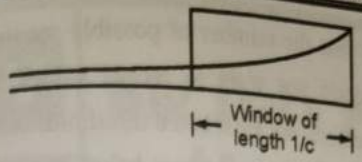


Fig. 4.9.2 : A decaying window and a fixed-length window of equal weight

It is much easier to adjust the sum in an exponentially decaying window than in a sliding window of fixed length. In the sliding window, we have to worry about the element that falls out of the window each time a new element arrives. That forces us to keep the exact elements along with the sum, or to use an approximation scheme such as DGIM. However, when a new element at + 1 arrives at the stream input, all we need to do is :

1. Multiply the current sum by 1 – c. 2. Add a_{t+1}.

The reason this method works is that each of the previous elements has now moved one position further from the current element, so its weight is multiplied by 1 – c. Further, the weight on the current element is $(1 - c)^0 = 1$, so adding a_{t+1} is the correct way to include the new element's contribution.

☞ Finding the Most Popular Elements

- The problem of finding the most popular movies in a stream of ticket sales. We shall use an exponentially decaying window with a constant c, which you might think of as 10–9 . That is, we approximate a sliding window holding the last one billion ticket sales.
- For each movie, we imagine a separate stream with a 1 each time a ticket for that movie appears in the stream, and a 0 each time a ticket for some other movie arrives. The decaying sum of the 1's measures the current popularity of the movie.

- We imagine that the number of possible movies in the stream is huge, so we do not want to record values for the unpopular movies. Therefore, we establish a threshold, say 1/2, so that if the popularity score for a movie goes below this number, its score is dropped from the counting. For reasons that will become obvious, the threshold must be less than 1, although it can be any number less than 1.
- When a new ticket arrives on the stream, do the following :
  1. For each movie whose score we are currently maintaining, multiply its score by (1 – c).
  2. Suppose the new ticket is for movie M. If there is currently a score for M, add 1 to that score. If there is no score for M, create one and initialize it to 1.
  3. If any score is below the threshold 1/2, drop that score.

Chapter Ends...

❑❑❑

UNIT V

CHAPTER 5

Big Data Mining Algorithms

University Prescribed Syllabus

Frequent Pattern Mining : Handling Larger Datasets in Main Memory Basic Algorithm of Park, Chen, and Yu. The SON Algorithm and MapReduce. Clustering Algorithms : CURE Algorithm. Canopy Clustering, Clustering with MapReduce Classification Algorithms: Overview SVM classifiers, Parallel SVM, KNearest Neighbor classifications for Big Data, One Nearest Neighbour.

Self-learning Topics : Standard libraries included with spark like graphX, MLlib

5.1 FREQUENT PATTERN MINING

5.1.1 Handling Larger Datasets in Main Memory Basic Algorithm of Park, Chen, and Yu

PCY (Park-Chen-Yu) Algorithm

In pass 1 of A-Priori, most memory is idle. We store only individual item counts. We use the idle memory to reduce memory required in pass 2.

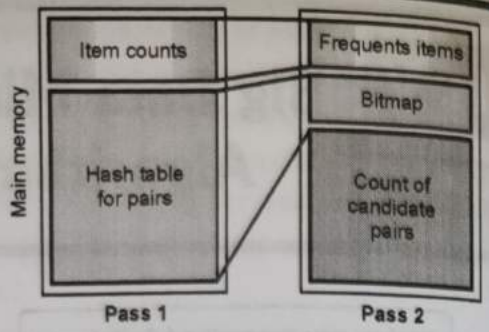


Fig. 5.1.1 : Organization of main memory for the first two passes of the PCY Algorithm

☞ Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory. Keep a count for each bucket into which pairs of items are hashed v For each bucket just keep the count, not the actual pairs that hash to the bucket.

FOR (each basket)

FOR (each item in the basket)

add 1 to item's count;

FOR (each pair of items)

hash the pair to a bucket;

add 1 to the count for that bucket;

- Pairs of items need to be generated from the input file; they are not present in the file.
- We are not just interested in the presence of a pair, but we need to see whether it is present at least s (support) times.
- If a bucket contains a frequent pair, then the bucket is surely frequent v However, even without any frequent pair, a bucket can still be frequent. So, we cannot use the hash to eliminate any member (pair) of a "frequent" bucket.

- But, for a bucket with total count less than s, none of its pairs can be frequent. Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of 2 frequent items).
- E.g., even though {A}, {B} are frequent, count of the bucket containing {A,B} might be < s

☞ Pass 2 : Only count pairs that hash to frequent buckets

- Replace the buckets by a bit-vector: 1 means the bucket count exceeded the support s (call it a frequent bucket); 0 means it did not.
- 4-byte integer counts are replaced by bits, so the bit-vector requires 1/32 of memory. Also, decide which items are frequent and list them for the second pass.
- Count all pairs {i, j} that meet the conditions for being a candidate pair :
  1. Both i and j are frequent items
  2. The pair {i, j} hashes to a bucket whose bit in the bit vector is 1 (i.e., a frequent bucket).
- Both conditions are necessary for the pair to have a chance of being frequent.

5.1.2 The SON Algorithm and MapReduce

The SON algorithm lends itself well to a parallel-computing environment. Each of the chunks can be processed in parallel, and the frequent itemsets from each chunk combined to form the candidates. We can distribute the candidates to many processors, have each processor count the support for each candidate in a subset of the baskets, and finally sum those supports to get the support for each candidate itemset in the whole dataset. This process does not have to

be implemented in MapReduce, but there is a natural way of expressing each of the two passes as a MapReduce operation.

- **First Map Function :** Take the assigned subset of the baskets and find the itemsets frequent in the subset using the algorithm. As described there, lower the support threshold from s to ps if each Map task gets fraction p of the total input file. The output is a set of key-value pairs (F, 1), where F is a frequent itemset from the sample. The value is always 1 and is irrelevant.
- **First Reduce Function :** Each Reduce task is assigned a set of keys, which are itemsets. The value is ignored, and the Reduce task simply produces those keys (itemsets) that appear one or more times. Thus, the output of the first Reduce function is the candidate itemsets.
- **Second Map Function :** The Map tasks for the second Map function take all the output from the first Reduce Function (the candidate itemsets) and a portion of the input data file. Each Map task counts the number of occurrences of each of the candidate itemsets among the baskets in the portion of the dataset that it was assigned. The output is a set of key-value pairs (C, v), where C is one of the candidate sets and v is the support for that itemset among the baskets that were input to this Map task.
- **Second Reduce Function :** The Reduce tasks take the itemsets they are given as keys and sum the associated values. The result is the total support for each of the itemsets that the Reduce task was assigned to handle. Those itemsets whose sum of values is at least s are frequent in the whole dataset, so the Reduce task outputs these itemsets with their counts. Itemsets that do not have total support at least s are not transmitted to the output of the Reduce task.

5.1.3 Clustering Algorithms

5.1.3(A) CURE Algorithm

- CURE (Clustering Using Representatives), assumes a Euclidean space. However, it does not assume anything about the shape of clusters; they need not be normally distributed, and can even have strange bends, S-shapes, or even rings. Instead of representing clusters by their centroid, it uses a collection of representative points, as the name implies.

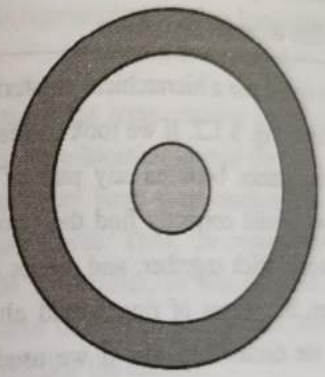


Fig. 5.1.2 : Two clusters, one surrounding the other

Example 5.1.1 : Fig. 5.1.2 is an illustration of two clusters. The inner cluster is an ordinary circle, while the second is a ring around the circle. This arrangement is not completely pathological. A creature from another galaxy might look at our solar system and observe that the objects cluster into an inner circle (the planets) and an outer ring (the Kuyper belt), with little in between.

5.1.3(B) Initialization in CURE

1. Take a small sample of the data and cluster it in main memory. In principle, any clustering method could be used, but as CURE is designed to handle oddly shaped clusters, it is often advisable to

use a hierarchical method in which clusters are merged when they have a close pair of points.

2. Select a small set of points from each cluster to be representative points. These points should be chosen to be as far from one another as possible, using the method.

3. Move each of the representative points a fixed fraction of the distance between its location and the centroid of its cluster. Perhaps 20% is a good fraction to choose. Note that this step requires a Euclidean space, since otherwise, there might not be any notion of a line between two points.

Example 5.1.2 : We could use a hierarchical clustering algorithm on a sample of the data from Fig. 5.1.2. If we took as the distance between clusters the shortest distance between any pair of points, one from each cluster, then we would correctly find the two clusters. That is, pieces of the ring would stick together, and pieces of the inner circle would stick together, but pieces of ring would always be far away from the pieces of the circle. Note that if we used the rule that the distance between clusters was the distance between their centroids, then we might not get the intuitively correct result. The reason is that the centroids of both clusters are in the center of the diagram.

For the second step, we pick the representative points. If the sample from which the clusters are constructed is large enough, we can count on a cluster's sample points at greatest distance from one another lying on the boundary of the cluster. Fig. 5.1.3 suggests what our initial selection of sample points might look like.

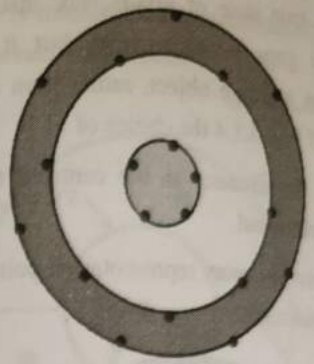


Fig. 5.1.3 : Select representative points from each cluster, as far from one another as possible

Finally, we move the representative points a fixed fraction of the distance from their true location toward the centroid of the cluster. In Fig. 5.1.3 both clusters have their centroid in the same place: the center of the inner circle. Thus, the representative points from the circle move inside the cluster, as was intended. Points on the outer edge of the ring also move into their cluster, but points on the ring's inner edge move outside the cluster.

5.1.3(C) Completion of the CURE Algorithm

The next phase of CURE is to merge two clusters if they have a pair of representative points, one from each cluster, that are sufficiently close. The user may pick the distance that defines "close." This merging step can repeat, until there are no more sufficiently close clusters.

Example 5.1.3 : The situation of Fig. 5.1.4 serves as a useful illustration. There is some argument that the ring and circle should really be merged, because their centroids are the same. For instance, if the gap between the ring and circle were much smaller, it might well be argued that combining the points of the ring and circle into a single

---

cluster reflected the true state of affairs. For instance, the rings of Saturn have narrow gaps between them, but it is reasonable to visualize the rings as a single object, rather than several concentric objects. In the case of Fig. 5.1.4 the choice of :

1. The fraction of the distance to the centroid that we move the representative points and
2. The choice of how far apart representative points of two clusters need to be to avoid merger.

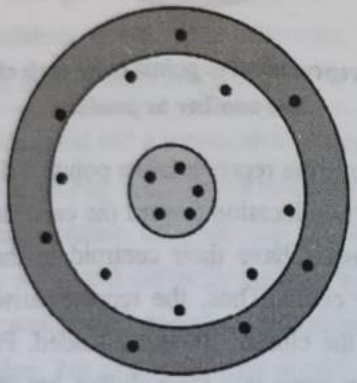


Fig. 5.1.4 : Moving the representative points 20% of the distance to the cluster's centroid

The last step of CURE is point assignment. Each point p is brought from secondary storage and compared with the representative points. We assign p to the cluster of the representative point that is closest to p.

5.1.4 Canopy Clustering

Canopy clustering is a fast and approximate clustering technique. It divides the input data points into overlapping clusters called canopies. Two different distance thresholds are used for the estimation of the cluster centroids. Canopy clustering can provide a quick approximation of the number of clusters and initial cluster centroids of

a given dataset. It is mainly used to understand the data and provide input to algorithms such as k-means.

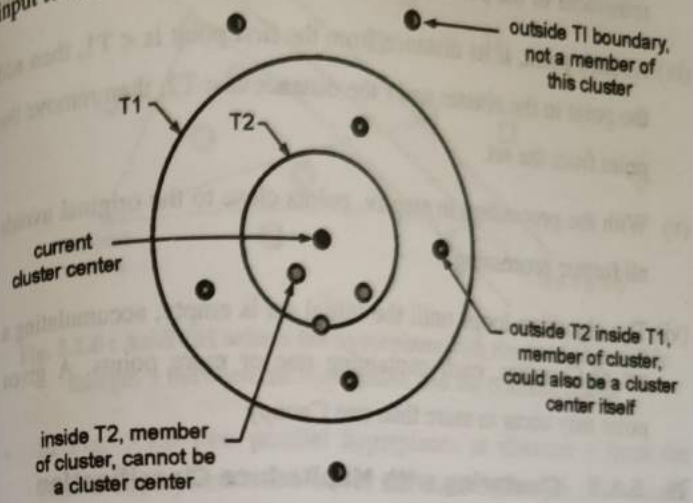


Fig. 5.1.5 : Canopy Clustering

- It is an unsupervised pre-clustering algorithm, often used as pre processing step for the K-means algorithm or the Hierarchical clustering algorithm. It speeds up the clustering operations on large data sets, whereby the other algorithms may be impractical due to the size of the data set. Briefly the algorithm may be stated as :
  - (i) Cheaply partitioning the data into overlapping subsets (called "canopies")
  - (ii) Perform more expensive clustering, but only within these canopies

Algorithm

(i) Two distance thresholds T1 and T2 are decided such that T1 > T2

(ii) A set of points are considered and remove one at random.

(iii) Create a Canopy containing this point and iterate through the remainder of the point set.

(iv) At each point, if its distance from the first point is < T1, then add the point to the cluster and f the distance is < T2, then remove the point from the set.

(v) With the processing in step iv, points close to the original avoids all further processing.

(vi) The algorithm loops until the initial set is empty, accumulating a set of Canopies, each containing one or more points. A given point may occur in more than one Canopy

5.1.5 Clustering with MapReduce Classification Algorithms

5.1.5(A) Overview SVM Classifiers

An SVM selects one particular hyperplane that not only separates the points in the two classes, but does so in a way that maximizes the margin – the distance between the hyperplane and the closest points of the training set.

The goal of an SVM is to select a hyperplane $w.x + b = 0$ that maximizes the distance γ between the hyperplane and any point of the training set. The idea is suggested by Fig. 5.1.6. There, we see the points of two classes and a hyperplane dividing them.

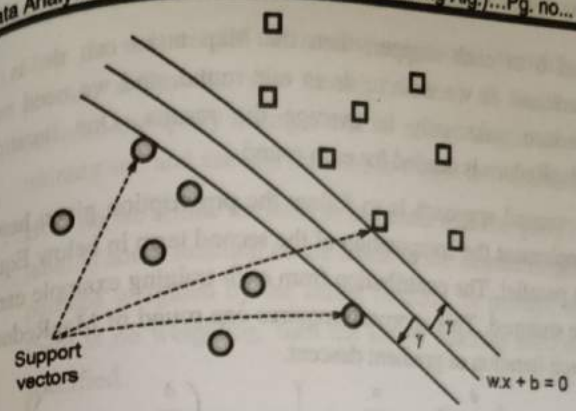


Fig. 5.1.6 : An SVM selects the hyperplane with the greatest possible margin γ between the hyperplane and the training points

- In Fig. 5.1.6 two parallel hyperplanes at distance γ from the central hyperplane $w.x + b = 0$, and these each touch one or more of the support vectors. The latter are the points that actually constrain the dividing hyperplane, in the sense that they are all at distance γ from the hyperplane.
- The SVM improves upon perceptrons by finding a separating hyperplane that not only separates the positive and negative points, but does so in a way that maximizes the margin – the distance perpendicular to the hyperplane to the nearest points. The points that lie exactly at this minimum distance are the support vectors. Alternatively, the SVM can be designed to allow points that are too close to the hyperplane, or even on the wrong side of the hyperplane, but minimize the error due to such misplaced points.

5.1.5(B) Parallel SVM

- Start with the current w and b, and in parallel do several iterations based on each training example. Then average the changes for each of the examples to create a new w and b. If we distribute w

and b to each mapper, then the Map tasks can do as many iterations as we wish to do in one round, and we need use the Reduce tasks only to average the results. One iteration of MapReduce is needed for each round.

- A second approach is to follow the prescription given here, but implement the computation of the second term in below Equation in parallel. The contribution from each training example can then be summed. This approach requires one round of MapReduce for each iteration of gradient descent.

$$f(w, b) = \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C \sum_{i=1}^{n} \max\left\{0, 1 - y_i\left(\sum_{j=1}^{d} w_j x_{ij} + b\right)\right\}$$

The first term encourages small w, while the second term, involving the constant C that must be chosen properly, represents the penalty for bad points.

5.1.6 KNearest Neighbor Classifications for Big Data

- In this approach to machine learning, the entire training set is used as the model. For each ("query") point to be classified, we search for its k nearest neighbors in the training set. The classification of the query point is some function of the labels of these k neighbors.
- The training set is first preprocessed and stored. The decisions take place when a new example, called the query example arrives and must be classified.
- There are several decisions we must make in order to design a nearest neighbor-based algorithm that will classify query examples. Like :

(1) What distance measure do we use?

(2) How many of the nearest neighbors do we look at?

(3) How do we weight the nearest neighbors? Normally, we provide a function (the kernel function) of the distance between the query example and its nearest neighbors in the training set, and use this function to weight the neighbors.

(4) How do we define the label to associate with the query? This label is some function of the labels of the nearest neighbors, perhaps weighted by the kernel function, or perhaps not. If there is no weighting, then the kernel function need not be specified.

5.1.6(A) One Nearest Neighbour

The simplest cases of nearest-neighbor learning are when we choose only the one neighbor that is nearest the query example. In that case, there is no use for weighting the neighbors, so the kernel function is omitted. There is also typically only one possible choice for the labeling function: take the label of the query to be the same as the label of the nearest neighbor.

Example : Fig. 5.1.7 shows some of the examples of dogs that last appeared in Fig. 5.1.7. We have dropped most of the examples for simplicity, leaving only three Chihuahuas, two Dachshunds, and two Beagles. Since the height-weight vectors describing the dogs are two-dimensional, there is a simple and efficient way to construct a Voronoi diagram for the points, in which the perpendicular bisectors of the lines between each pair of points is constructed. Each point gets a region around it, containing all the points to which it is the nearest. These regions are always convex, although they may be open to infinity in one direction. 2 It is also a surprising fact that, even though there are O(n2) perpendicular bisectors for n points, the Voronoi diagram can be found in O(n log n) time.

In Fig. 5.1.7 we see the Voronoi diagram for the seven points. The boundaries that separate dogs of different breeds are shown solid, while the boundaries

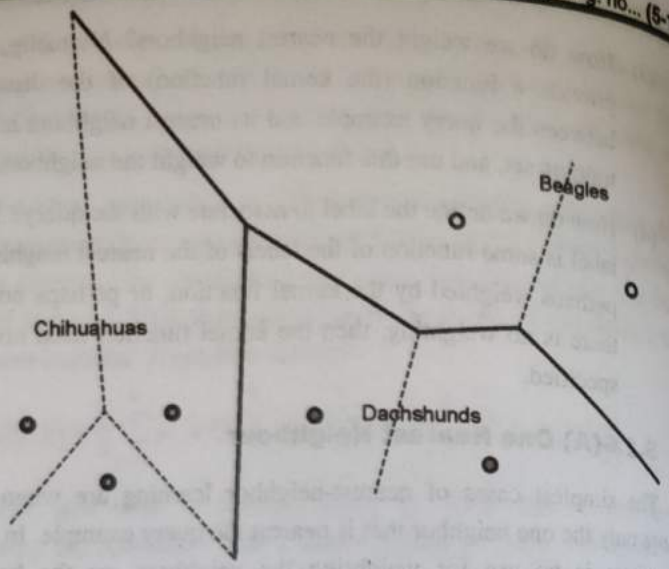


Fig. 5.1.7 : Voronoi diagram for the three breeds of dogs

- Between dogs of the same breed are shown dashed. Suppose a query example q is provided. Note that q is a point in the space of Fig. 5.1.7. We find the region into which q falls, and give q the label of the training example to which that region belongs. It is not too hard to find the region of q. We have to determine to which side of certain lines q falls.
- To compare a vector x with a hyperplane perpendicular to a vector w. In fact, if the lines that actually form parts of the Voronoi diagram are pre-processed properly, we can make the determination in O(log n) comparisons; it is not necessary to compare q with all of the O(n log n) lines that form part of the diagram.

Chapter Ends..

□□□

UNIT VI
CHAPTER 6
Big Data Analytics Applications

University Prescribed Syllabus

Link Analysis : PageRank Definition, Structure of the web, dead ends, Using Page rank in a search engine, Efficient computation of Page Rank : PageRank Iteration Using MapReduce, Topic sensitive Page Rank, link Spam, Hubs and Authorities, HITS Algorithm. **Mining Social-Network Graphs :** Social Networks as Graphs, Types, Clustering of Social Network Graphs, Direct Discovery of Communities, Counting triangles using Map-Reduce. **Recommendation Engines :** A Model for Recommendation Systems, Content-Based Recommendations, Collaborative Filtering

Self-learning Topics : Sample applications like social media feeds, multiplayer game interactions, retail industry, financial data analysis. Use case like location data, real-time stock trades, log monitoring etc

6.1 LINK ANALYSIS

6.1.1 PageRank Definition

- PageRank is a function that assigns a real number to each page in the Web. The intent is that the higher the PageRank of a page, the more "important" it is. There is not one fixed algorithm for assignment of PageRank, and in fact variations on the basic idea can alter the relative PageRank of any two pages.

- Think of the Web as a directed graph, where pages are the nodes, and there is an arc from page p_1 to page p_2 if there are one or more links from p_1 to p_2. Fig. 6.1.1 is an example of a tiny version of the Web, where there are only four pages. Page A has links to each of the other three pages; page B has links to A and D only; page C has a link only to A, and page D has links to B and C only.

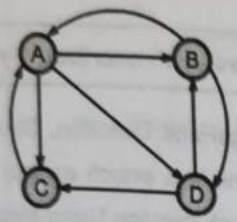


Fig. 6.1.1 : A hypothetical example of the Web

- Suppose a random surfer starts at page A in Fig. 6.1.1. There are links to B, C, and D, so this surfer will next be at each of those pages with probability 1/3, and has zero probability of being at A. A random surfer at B has, at the next step, probability 1/2 of being at A, 1/2 of being at D, and 0 of being at B or C. In general, we can define the transition matrix of the Web to describe what happens to random surfers after one step. This matrix M has n rows and columns, if there are n pages. The element m_{ij} in row i and column j has value 1/k if page j has k arcs out, and one of them is to page i. Otherwise, $m_{ij} = 0$.

Example 6.1.1 : The transition matrix for the Web of Fig. 6.1.1 is,

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- In this matrix, the order of the pages is the natural one, A, B, C, and D. Thus, the first column expresses the fact, already

discussed, that a surfer at A has a 1/3 probability of next being at each of the other pages. The second column expresses the fact that a surfer at B has a 1/2 probability of being next at A and the same of being at D. The third column says a surfer at C is certain to be at A next. The last column says a surfer at D has a 1/2 probability of being next at B and the same at C.

- The probability distribution for the location of a random surfer can be described by a column vector whose j^{th} component is the probability that the surfer is at page j. This probability is the (idealized) PageRank function.

- Suppose we start a random surfer at any of the n pages of the Web with equal probability. Then the initial vector v_0 will have 1/n for each component. If M is the transition matrix of the Web, then after one step, the distribution of the surfer will be Mv_0, after two steps it will be $M(M\ v_0) = M^2\ v_0$, and so on. In general, multiplying the initial vector v_0 by M a total of i times will give us the distribution of the surfer after i steps.

- To see why multiplying a distribution vector v by M gives the distribution $x = Mv$ at the next step, we reason as follows. The probability x_i that a random surfer will be at node i at the next step, is $\sum_j m_{ij}v_j$. Here, m_{ij} is the probability that a surfer at node j will move to node i at the next step (often 0 because there is no link from j to i), and v_j is the probability that the surfer was at node j at the previous step. This sort of behavior is an example of the ancient theory of Markov processes. It is known that the distribution of the surfer approaches a limiting distribution v that satisfies $v = Mv$, provided two conditions are met :

1. The graph is strongly connected; that is, it is possible to get from any node to any other node.
2. There are no dead ends: nodes that have no arcs out.

- Fig. 6.1.1 satisfies both these conditions. The limit is reached when multiplying the distribution by M another time does not change the distribution. In other terms, the limiting v is an eigenvector of M (an eigenvector of a matrix M is a vector v that satisfies $v = \lambda M v$ for some constant eigenvalue λ). In fact, because M is stochastic, meaning that its columns each add up to 1, v is the principal eigenvector (its associated eigenvalue is the largest of all eigenvalues).
- M is stochastic, the eigenvalue associated with the principal eigenvector is 1. The principal eigenvector of M tells us where the surfer is most likely to be after a long time. Recall that the intuition behind PageRank is that the more likely a surfer is to be at a page, the more important the page is. We can compute the principal eigenvector of M by starting with the initial vector v0 and multiplying by M some number of times, until the vector we get shows little change at each round. In practice, for the Web itself, 50–75 iterations are sufficient to converge to within the error limits of double-precision arithmetic.

6.1.2 Structure of the Web

- Fig. 6.1.2. There was a large strongly connected component (SCC), but there were several other portions that were almost as large.

1. The in-component, consisting of pages that could reach the SCC by following links, but were not reachable from the SCC.
2. The out-component, consisting of pages reachable from the SCC but unable to reach the SCC.
3. Tendrils, which are of two types. Some tendrils consist of pages reachable from the in-component but not able to reach the in-component. The other tendrils can reach the out-component.

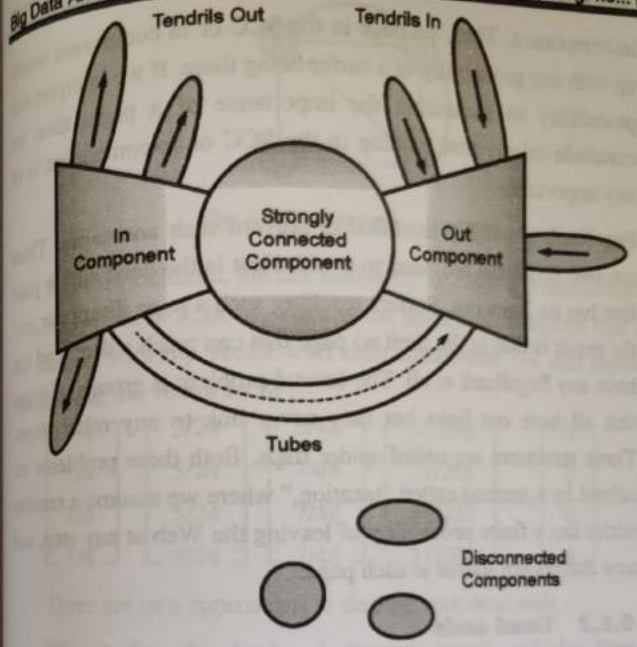


Fig. 6.1.2 : The "bowtie" picture of the Web

There were small numbers of pages found either in :

(a) Tubes, which are pages reachable from the in-component and able to reach the out-component, but unable to reach the SCC or be reached from the SCC.

(b) Isolated components that are unreachable from the large components (the SCC, in- and out-components) and unable to reach those components.

- Several of these structures violate the assumptions needed for the Marko process iteration to converge to a limit. For example, when a random surfer enters the out-component, they can never leave.
- As a result, surfers starting in either the SCC or in-component are going to wind up in either the out component or a tendril off the

in-component. Thus, no page in the SCC or in component winds up with any probability of a surfer being there. If we interpret this probability as measuring the importance of a page, then we conclude falsely that nothing in the SCC or in-component is of any importance.

- PageRank is usually modified to prevent such anomalies. There are two problems are need to avoid. First is the dead end, a page that has no links out. Surfers reaching such a page disappear, and the result is that in the limit no page that can reach a dead end can have any PageRank at all. The second problem is groups of pages that all have out links but they never link to any other pages. These structures are called spider traps. Both these problems are solved by a method called "taxation," where we assume a random surfer has a finite probability of leaving the Web at any step, and new surfers are started at each page.

6.1.3 Dead ends

- A page with no link out is called a dead end. If we allow dead ends, the transition matrix of the Web is no longer stochastic, since some of the columns will sum to 0 rather than 1.
- A matrix whose column sums are at most 1 is called substochastic. If we compute $M^i v$ for increasing powers of a substochastic matrix M, then some or all of the components of the vector go to 0. That is, importance "drains out" of the Web, and we get no information about the relative importance of pages.

Example 6.1.2 : The matrix M that describes :

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

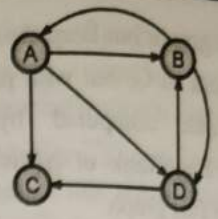


Fig. 6.1.3 : C is now a dead end

It is substochastic, but not stochastic, because the sum of the third column, for C, is 0, not 1. Here is the sequence of vectors that result by starting with the vector with each component 1/4, and repeatedly multiplying the vector by M :

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

There are two approaches to dealing with dead ends :

1. We can drop the dead ends from the graph, and also drop their incoming arcs. Doing so may create more dead ends, which also have to be dropped, recursively. However, eventually we wind up with a strongly-connected component, none of whose nodes are dead ends. Recursive deletion of dead ends will remove parts of the out-component, tendrils, and tubes, but leave the SCC and the in-component, as well as parts of any small isolated components.
2. We can modify the process by which random surfers are assumed to move about the Web. This method, which we refer to as "taxation".

- If we use the first approach, recursive deletion of dead ends, then we solve the remaining graph G by whatever means are appropriate, including the taxation method if there might be spider traps in G.

- Then, we restore the graph, but keep the PageRank values for the nodes of G. Nodes not in G, but with predecessors all in G can have their PageRank computed by summing, over all predecessors p, the PageRank of p divided by the number of successors of p in the full graph.
- Now there may be other nodes, not in G, that have the PageRank of all their predecessors computed. These may have their own PageRank computed by the same process. Eventually, all nodes outside G will have their PageRank computed; they can surely be computed in the order opposite to that in which they were deleted.

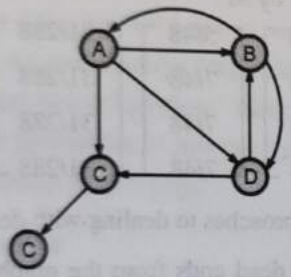


Fig. 6.1.4 : A graph with two levels of dead ends

6.1.4 Using Page Rank in a Search Engine

- Having seen how to calculate the PageRank vector for the portion of the Web that a search engine has crawled, we should examine how this information is used.
- Each search engine has a secret formula that decides the order in which to show pages to the user in response to a search query consisting of one or more search terms (words).
- Google is said to use over 250 different properties of pages, from which a linear order of pages is decided.
- First, in order to be considered for the ranking at all, a page has to have at least one of the search terms in the query.

- Normally, the weighting of properties is such that unless all the search terms are present, a page has very little chance of being in the top ten that are normally shown first to the user.
- Among the qualified pages, a score is computed for each, and an important component of this score is the PageRank of the page. Other components include the presence or absence of search terms in prominent places, such as headers or the links to the page itself.

6.1.5 Efficient Computation of Page Rank

- To compute the PageRank for a large graph representing the Web, we have to perform a matrix–vector multiplication on the order of 50 times, until the vector is close to unchanged at one iteration. To a first approximation, the MapReduce method is suitable. However, we must deal with two issues :

1. The transition matrix of the Web M is very sparse. Thus, representing it by all its elements is highly inefficient. Rather, we want to represent the matrix by its nonzero elements.
2. We may not be using MapReduce, or for efficiency reasons we may wish to use a combine with the Map tasks to reduce the amount of data that must be passed from Map tasks to Reduce tasks.

6.1.5(A) PageRank Iteration Using MapReduce

- One iteration of the PageRank algorithm involves taking an estimated PageRank vector v and computing the next estimate v ′ by

$$v' = \beta M v + (1 - \beta)^{e/n}$$

- β is a constant slightly less than 1, e is a vector of all 1's, and n is the number of nodes in the graph that transition matrix M represents.

- If n is small enough that each Map task can store the full vector v in main memory and also have room in main memory for the result vector v', then there is little more here than a matrix – vector multiplication. The additional steps are to multiply each component of Mv by constant β and to add $(1 - \beta)/n$ to each component.

6.1.6 Topic Sensitive Page Rank

There are several improvements we can make to PageRank. One, to be studied in this section, is that we can weight certain pages more heavily because of their topic. The mechanism for enforcing this weighting is to alter the way random surfers behave, having them prefer to land on a page that is known to cover the chosen topic. In the next section, we shall see how the topic-sensitive idea can also be applied to negate the effects of a new kind of spam, called "'link spam," that has developed to try to fool the PageRank algorithm.

6.1.7 Link Spam

- When it became apparent that PageRank and other techniques used by Google made term spam ineffective, spammers turned to methods designed to fool the PageRank algorithm into overvaluing certain pages.
- The techniques for artificially increasing the PageRank of a page are collectively called link spam. In this section we shall first examine how spammers create link spam, and then see several methods for decreasing the effectiveness of these spamming techniques, including TrustRank and measurement of spam mass.

6.1.8 Hubs and Authorities

- An idea called 'hubs and authorities' was proposed shortly after PageRank was first implemented. The algorithm for computing hubs and authorities bears some resemblance to the computation

of PageRank, since it also deals with the iterative computation of a fixed point involving repeated matrix-vector multiplication. However, there are also significant differences between the two ideas, and neither can substitute for the other.

- This hubs-and-authorities algorithm, sometimes called HITS (hyperlinkinduced topic search), was originally intended not as a preprocessing step before handling search queries, as PageRank is, but as a step to be done along with the processing of a search query, to rank only the responses to that query.
- We shall, however, describe it as a technique for analyzing the entire Web, or the portion crawled by a search engine. There is reason to believe that something like this approach is, in fact, used by the Ask search engine.

6.1.9 HITS Algorithm

Hyperlink Induced Topic Search Algorithm (HITS) identifies good authorities and hubs for a query topic by assigning two scores to a page :

(i) Authority Score

A page is a good authoritative page with respect to a given query if it is referenced by many pages related to that query.

(ii) Hub Score

A page is a good hub page with respect to a given query if it points to many good authoritative pages with respect to that query.

The steps are as follows :

(1) Submit query q to a search engine. Let S be the set of top 'n' pages returned by the search engine.

(2) Expand s into large sit : T (base set) Add pages that are pointed to by any page in S. Add pages that point to any page in S.

(3) Find the sub graph SG of the web graph that is induced by T.

(4) Compute authority score and hub score of each web page in T using sub graph SG (V, E).

Given a page P, Let :

a(p) - Authority score

b(p) - Hub score

(p, q) - directed edge from p to q

(5) Apply following operations

(i) **Operation I :** Update each a(P) as the sum of all hub scores of web pages point to P.

(ii) **Operation O :** Update each h(p) as the sum of all authority scores of web pages pointed to by P.

(6) To apply the operations to all pages of the web graph at once, we can use matrix representation of operations I and O

- Let A be the adjacency matrix of SG : A (p, q) is 1 if P has a link to q, else the entry is O.
- Let hi be vector of hub scores after i iterations.
- Let ai be vector of authority score after i operations.
- Operation I : ai = ATbi-1
- Operation O : hi = Aai

(7) As the values may increase beyond bounds, normalization is done after step 6 such that maximum value is 1.

(8) Repeat until scores coverage.

(9) Sort pages in descending order of authority scores.

(10) Display the top authority pages.

6.2 SOCIAL NETWORKS AS GRAPHS, CLUSTERING OF SOCIAL-NETWORK GRAPHS, DIRECT DISCOVERY OF COMMUNITIES IN A SOCIAL GRAPH

6.2.1 Social Networks as Graphs

- There is collection of entities that participate in network. Typically, people, but could be something else too. There is at least one relationship between entities of the network. For example : friends
- **Sometimes Boolean :** two people are either friends or they are not. They may have a Discrete degree. E.g. friends, family, acquaintances, or none. It could be real number : the fraction of the average day that two people. An example spends talking to each other.
- There is an assumption of non randomness or locality. This condition is the hardest to formalize, but the intuition is that relationships tend to cluster. That is, if entity A is related to both B and C, then there is a higher probability than average that B and C are related.
- Social networks are naturally modeled as graphs, which we sometimes refer to as a social graph. The entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network.
- If there is a degree associated with the relationship, this degree is represented by labeling the edges. Often, social graphs are undirected, as for the Face book friend's graph. But they can be directed graphs, as for example the graphs of followers on Twitter or Google+.

Ex. 6.2.1 : Fig. Ex. 6.2.1 is an example of a tiny social network. The entities are the nodes A through G. The relationship, which we might think of as "friends," is represented by the edges. For instance, B is friends with A, C, and D.Is this graph really typical of a social network, in the sense that it exhibits locality of relationships?

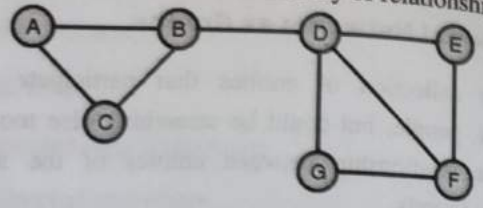


Fig. Ex. 6.2.1 : Example of a small social network

☑ Soln. :

- Check for the non-randomness criterion. In a random graph (V, E) of 7 nodes and 9 edges, if XY is an edge, YZ is an edge, what is the probability that XZ is an edge?
- For a large random graph, it would be close to

$$\frac{|E|}{|V|C2} = \frac{9}{21} \sim 0.43$$

- **Small graph :** XY and YZ are already edges, so compute within the rest. So the probability is $\frac{|E|-2}{(|V|C2-2)} = \frac{7}{19} = 0.37$.
- Now let's compute what is the probability for this graph in particular. For each X, check possible YZ and check if YZ is an edge or not Example : if X = A, YZ = {BC}, it is an edge.

X =	YZ =	Yes / Total
A	dBC	$\frac{1}{1}$
B	AC, AD, CD	$\frac{1}{3}$

X =	YZ =	Yes/ Total
E	DF	$\frac{1}{1}$
F	DE, DG, EG	$\frac{2}{3}$

X =	YZ =	Yes / Total
C	AB	$\frac{1}{1}$
D	BE, BG, BF, EF, EG, FG	$\frac{2}{6}$

X =	YZ =	Yes/ Total
G	DF	$\frac{1}{1}$
Total		$\frac{9}{16} \sim 0.56$

☞ Varieties of Social Networks

Telephone networks

- Nodes are phone numbers.
- AB is an edge if A and B talked over phone within the last one week, or month, or ever.
- Edges could be weighted by the number of times phone calls were made, or total time of conversation.

☞ Email networks : nodes are email addresses

- AB is an edge if A and B sent mails to each other within the last one week, or month, or ever.
- One directional edges would allow spammers to have edges.
- Edges could be weighted.
- Other networks: collaboration network – authors of papers, jointly written papers or not.
- Also networks exhibiting locality property

☞ Collaboration networks

- Nodes represent individuals who have published research papers. There is an edge between two individuals who published one or more papers jointly.
- Option-ally, we can label edges by the number of joint publications. The communities in this network are authors working on a particular topic.

- An alternative view of the same data is as a graph in which the nodes are papers. Two papers are connected by an edge if they have at least one author in common. Now, we form communities that are collections of papers on the same topic.

☞ **Other Examples of Social Graphs**

- Many other phenomena give rise to graphs that look something like social graphs, especially exhibiting locality.
- Examples include: information networks(documents, web graphs, patents), infrastructure networks (roads, planes, water pipes, power grids), biological networks (genes, proteins, food-webs of animal seating each other), as well as other types, like product co-purchasing networks(e.g., Group on).

6.2.2 Clustering of Social-Network Graphs

GQ. Explain clustering of Social-Network Graphs with example. **(10 Marks)**

An important aspect of social networks is that they contain communities of entities that are connected by many edges. These typically correspond to groups of friends at school or groups of researchers interested in the same topic, for example.

☞ **Distance Measures for Social-Network Graphs**

- When the edges of the graph have labels, these labels might be usable as a distance measure, depending on what they represented. But when the edges are unlabeled, as in a "friends" graph, there is not much we can do to define a suitable distance.
- Our first instinct is to assume that nodes are close if they have an edge between them and distant if not. Thus, we could say that the distance d (x, y) is 0 if there is an edge (x, y) and 1 if there is no such edge. We could use any other two values, such as 1 and ∞, as long as the distance is closer when there is an edge.

- Neither of these two-valued "distance measures" – 0 and 1 or 1 and ∞ – is a true distance measure.

☞ **Applying Standard Clustering Methods**

- In particular, suppose we use as the inter cluster distance the minimum distance between nodes of the two clusters. Hierarchical clustering of a social-network graph starts by combining some two nodes that are connected by an edge.
- Successively, edges that are not between two nodes of the same cluster would be chosen randomly to combine the clusters to which their two nodes belong. The choices would be random, because all distances represented by an edge are the same.

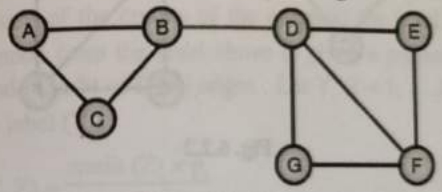


Fig. 6.2.1

☞ **Point-assignment approach**

- in k-means randomly picked nodes might be in the same cluster
- randomly chosen and another as far away as possible doesn't do much better (.i.e. E and G)
- even choosing B and F – problem with assignment of D

☞ **Betweenness**

This method based on finding the edges that are **least likely to be inside a community.** Betweenness of and edge (a, b) is the number of pairs of nodes x and y such that the edge (a, b) lies on the shortest path between x and y. High value suggests that (a, b) runs between two different communities – a and b do not belong to the same community

The Girvan-Newman Algorithm

To exploit betweenness, we need to calculate the number of shortest paths going through each edge

Girvan-Newman Algorithm visits each node X once and computes the number of shortest paths from X to other nodes through each of the edges.

- **Step I :**

BFS : Start at a node X, perform a BFS with X as root

Observe : level of node Y = length of shortest path from X to Y

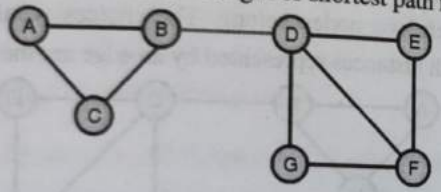


Fig. 6.2.2

Edges between level are called "DAG" edges

Each DAG edge is part of at least one shortest path from X

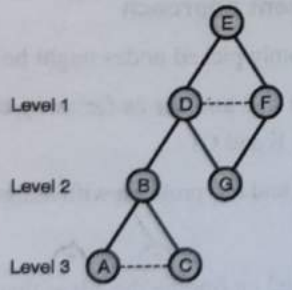


Fig. 6.2.3 : Newman Algorithm

- **Step II :**

Label each node by the number of shortest paths that reach it from the root the sum of the labels of its parents

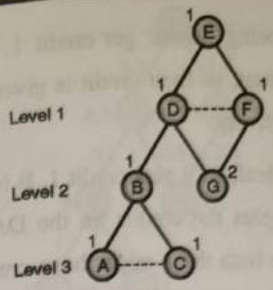


Fig. 6.2.4 : Newman Algorithm

- **Step III :**

- do the calculation starting from the bottom according to rules:
- leaf gets a credit of 1 node that is not a leaf gets a credit equal to 1 plus the sum of the credits of the edges to the level below edge entering node from the level above is given a proportional share of that node Credit of DAG edges : Let Y_i (I = 1, ...k) be parents of Z, p_i = label (Y_i)

$$\text{credit}(Y, Z) = \frac{\text{credit}(Z) \times p_i}{(p_i + \ldots p_i)}$$

- **Intuition :** a DAG edge Y, Z gets the share of credit of Z proportional to the ≠ of shortest paths from X to Z going through Y,Z
- **Finally :** Repeat Steps 1, 2 and 2 with each node as root. For each edge, betweenness = sum credits obtained in all iterations / 2

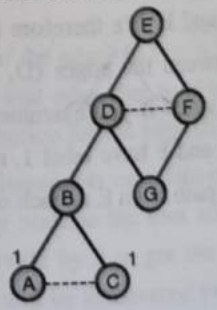


Fig. 6.2.5 : Newman Algorithm

- First, A and C, being leaves, get credit 1. Each of these nodes have only one parent, so their credit is given to the edges (B, A) and (B, C), respectively.
- At level 2, G is a leaf, so it gets credit 1. B is not a leaf, so it gets credit equal to 1 plus the credits on the DAG edges entering it from below. Since both these edges have credit 1, the credit of B is 3. Intuitively 3 represents the fact that all shortest paths from E to A, B, and C go through B. Fig. 6.2.6 shows the credits assigned so far.

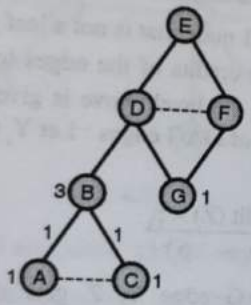


Fig. 6.2.6 : Newman Algorithm – levels 3 and 2

- Now, let us proceed to level 1. B has only one parent, D, so the edge (D, B) gets the entire credit of B, which is 3. However, G has two parents, D and F. We therefore need to divide the credit of 1 that G has between the edges (D, G) and (F, G). In what proportion do we divide? If you examine the labels of Fig. 6.2.4. you see that both D and F have label 1, representing the fact that there is one shortest path from E to each of these nodes.

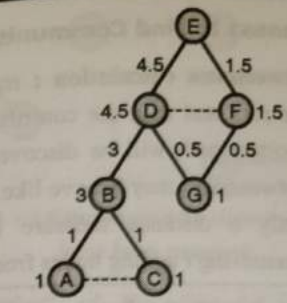


Fig. 6.2.7 : Newman Algorithm

- Thus, we give half the credit of G to each of these edges; i.e., their credit is each $1 / (1 + 1) = 0.5$.
- Had the labels of D and F in Fig. 6.2.7 been 5 and 3, meaning there were five shortest paths to D and only three to F, then the credit of edge (D, G) would have been 5/8 and the credit of edge (F, G) would have been 3/8.
- Now, we can assign credits to the nodes at level 1. D gets 1 plus the credits of the edges entering it from below, which are 3 and 0.5. That is, the credit of D is 4.5. The credit of F is 1 plus the credit of the edge (F, G), or 1.5. Finally, the edges (E, D) and (E, F) receive the credit of D and F, respectively, since each of these nodes has only one parent. These credits are all shown in Fig. 6.2.7.
- The credit on each of the edges in Fig. 6.2.7 is the contribution to the betweenness of that edge due to shortest paths from E. For example, this contribution for the edge (E, D) is 4.5.
- To complete the betweenness calculation, we have to repeat this calculation for every node as the root and sum the contributions. Finally, we must divide by 2 to get the true betweenness, since every shortest path will be discovered twice, once for each of its endpoints.

Using Betweenness to Find Communities

To complete betweenness calculation : repeat these steps for every node as the root and sum the contributions divide by 2, because every shortest path will be discovered twice, once for each endpoints betweenness may behave like a distance measure, but is not exactly a distance measure ordering edges by betweenness and removing / adding nodes from graph

Ex. 6.2.1 : We see it with the betweenness for each edge in Fig. Ex.6.2.1. The calculation of the betweenness will be left to the reader. The only tricky part of the count is to observe that between E and G there are two shortest paths, one going through D and the other through F. Thus, each of the edges (D, E), (E, F), (D, G), and (G, F) are credited with half a shortest path.

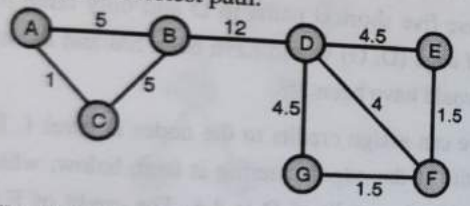


Fig. Ex. 6.2.1 : Betweenness scores for the graph

☑ **Soln. :** Clearly, edge (B,D) has the highest betweenness, so it is removed first. That leaves us with exactly the communities we observed make the most sense, namely: {A, B, C} and {D, E, F, G}. However, we can continue to remove edges.

Next to leave are (A, B) and (B, C) with a score of 5, followed by (D, E) and (D, G) with a score of 4.5. Then, (D, F), whose score is 4, would leave the graph. We see in Fig. Ex. 6.2.2 the graph that remains.

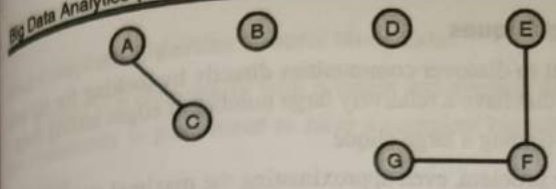


Fig. Ex. 6.2.2 : All the edges with betweenness 4 or more have been removed

The "communities" of Fig. Ex. 6.2.2 look strange. One implication is that A and C are more closely knit to each other than to B. That is, in some sense B is a "traitor" to the community {A, B, C} because he has a friend D outside that community. Likewise, D can be seen as a "traitor" to the group {D, E, F, G}, which is why in Fig. Ex. 6.2.2, only E, F, and G remain connected.

6.2.3 Direct Discovery of Communities in a Social Graph

UQ. Explain the Clique Percolation Method (CPM) used in direct discovery of communities in a social graph with example.
(MU - Dec. 18, 10 Marks)

UQ. What is a "Community" in a social network Graph ?
(MU - May 18, 5 Marks)

UQ. What is a "Community" in a social Network Graph? Explain any one algorithm for finding communities in asocial graph.
(MU - Dec. 17, 10 Marks)

Discovering communities directly by looking for subsets of the nodes that have a relatively large number of edges among them. Interestingly, the technique for doing this search on a large graph involves finding large frequent item sets

☞ Finding Cliques

We want to discover communities directly by looking for subsets of the nodes that have a relatively large number of edges among them

first thought-finding a large clique

NP-complete problem, even approximating the maximal clique is hard

It is possible to have a set of nodes with almost all edges between them, yet with relatively small cliques

☞ Complete Bipartite Graphs

- Graph that consists of 's' nodes on one side and 't' nodes on the other side with all 'st' possible edges between them
- It is possible to guarantee that a bipartite graph with many edges has a large complete bipartite sub graph (unlike cliques)
- It might be regarded as the nucleus of community

☞ Finding Complete Bipartite Sub graphs

- We are given a large bipartite graph G , and we want to find instances of Ks,t within it. It is similar to finding frequent item sets. Items" on the left side (Ks,t – t nodes there) assumption that $t \leq s$ The baskets" on the right side. member of the basket are the nodes from left side connected to that node. support threshold s frequent item set of size t and s of the baskets, in which all those items appear, form an instance of Ks,t

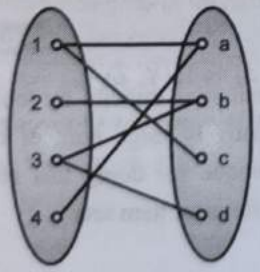


Fig. 6.2.8 : The bipartite graph

Why Complete Bipartite Graphs Must Exist

If there is a community with n nodes and average degree d, then this community is guaranteed to have a complete bipartite sub graph $K_{s,t}$ when :

$$n\frac{\binom{d}{t}}{\binom{n}{t}} \geq s$$

which approximately is :

$$n\left(\frac{d}{n}\right) \geq s$$

6.2.4 Counting Triangles using Map-Reduce

Assume that the nodes of a graph are numbered 1, 2, . . . , n. We use a relation E to represent edges. To avoid representing each edge twice, we assume that if E(A, B) is a tuple of this relation, then not only is there an edge between nodes A and B, but also, as integers, we have A < B.

This requirement also eliminates loops (edges from a node to itself), which we generally assume do not exist in social-network graphs anyway, but which could lead to "triangles" that really do not involve three different nodes. Using this relation, we can express the set of triangles of the graph whose edges are E by the natural join.

$$E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z)$$

To understand this join, we have to recognize that the attributes of the relation E are given different names in each of the three uses of E. That is, we imagine there are three copies of E, each with the same tuples, but with a different schema. In SQL, this join would be written using a single relation E(A, B) as follows :

```
SELECT e1.A, e1.B, e2.B
```

FROM E e1, E e2, E e3

WHERE e1.A = e2.A AND e1.B = e3.A AND e2.B = e3.B

In this query, the equated attributes e1.A and e2.A are represented in our join by the attribute X. Also, e1.B and e3.A are each represented by Y; e2.B and e3.B are represented by Z. Each triangle appears once in this join. The triangle consisting of nodes v1, v2, and v3 is generated when X, Y , and Z are these three nodes in numerical order, i.e., X < Y < Z. For instance, if the numerical order of the nodes is $v_1 < v_2 < v_3$, then X can only be v_1, Y is v_2, and Z is v_3.

6.3 A MODEL FOR RECOMMENDATION SYSTEMS, CONTENT-BASED RECOMMENDATIONS, COLLABORATIVE FILTERING

UQ. How recommendation is done based on properties of product? Explain with suitable example. **(MU - May 19, 10 Marks)**

UQ. What are Recommendation Systems ? **(MU - Dec. 17, 5 Marks)**

6.3.1 A Model for Recommendation Systems

A recommendation system is also called as Recommender system. A recommender system is really an automated system to filter some entities can be any products, ads, people, movies or songs. And we see this from all over on a daily basis from Amzon, Netfix, Youtube, FilpCart etc.

A recommender system captures the pattern of people's behaviour and use it to predict what else they might want or like.

For example, we watch a movie and then later on we get a recommendation for a different movie based on the power of previous viewing history. It could also be a product that we bought and then we get a recommendation for another product based on the previous

product viewing or purchase history. And recommender doesn't work only in what products we are being shown, but also in what order the products are being ranked.

For example, if you've recently purchased a book on Machine Learning in Python and you've enjoyed reading it, it's very likely that you'll also enjoy reading a book on Data Visualization. People also tend to have similar tastes to those of the people they're close to in their lives. Recommender systems try to capture these patterns and similar behaviours, to help predict what else you might like.

Applications

- What to buy?

  E-commerce, books, movies, shoes, etc.
- Where to eat?
- Which job to apply to?
- Who you should be friends with?

  LinkedIn, Facebook
- Personalize your experience on the web
- News platforms, news personalization
- Recommender systems function with two kinds of information:
- **Characteristic information**. This is information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
- **User-item interactions.** This is information such as ratings, number of purchases, likes, etc.
- Why the Recommendation system?
- Benefits users in finding items of their interest.
- Help item providers in delivering their items to the right user.
- Identity products that are most relevant to users.

- Personalized content.
- Help websites to improve user engagement.

Advantages of Recommender system

1. Broder exposure
2. Possibility of continual usage or purchase of products
3. Provides better experience

6.3.2 Applications for Recommendation System

UQ. Clearly explain two applications for Recommendation system. **(MU - Dec. 17, 5 Marks)**

(1) **Entertainment** : recommendations for movies, music, and IPTV.

(2) **Content** : personalized newspapers, recommendation for documents, recommendations of Web pages, e-learning applications, and e-mail filters.

(3) **E-commerce** : recommendations for consumers of products to buy such as books, cameras, PCs etc.

(4) **Services** : recommendations of travel services, recommendation of experts for consultation, recommendation of houses to rent, or matchmaking services.

6.3.3 Types of Recommender Systems

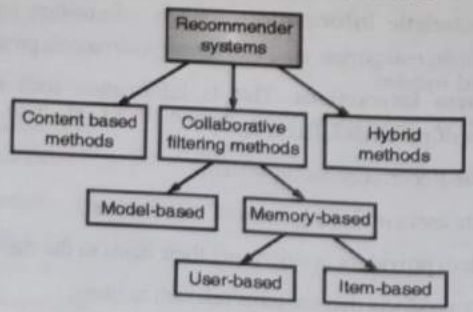


Fig. 6.3.1

Collaborative filtering

- Collaborative filtering focuses on collecting and analyzing data on user behavior, activities, and preferences, to predict what a person will like, based on their similarity to other users.
- To plot and calculate these similarities, collaborative filtering uses a matrix style formula. An advantage of collaborative filtering is that it doesn't need to analyze or understand the content (products, films, books). It simply picks items to recommend based on what they know about the user.

Content-based filtering

- Content-based filtering works on the principle that if you like a particular item, you will also like this other item.
- To make recommendations, algorithms use a profile of the customer's preferences and a description of an item (genre, product type, color, word length) to work out the similarity of items using cosine and Euclidean distances.
- The downside of content-based filtering is that the system is limited to recommending products or content similar to what the person is already buying or using. It can't go beyond this to recommend other types of products or content. For example, it couldn't recommend products beyond homeware if the customer had only brought homeware.

Hybrid model

- A hybrid recommendation engine looks at both the meta (collaborative) data and the transactional (content-based) data. Because of this, it outperforms both.
- In a hybrid recommendation engine, natural language processing tags can be generated for each product or item (movie, song), and vector equations used to calculate the similarity of products.

- A collaborative filtering matrix can then be used to recommend items to users depending on their behaviors, activities, and preferences. Netflix is the perfect example of a hybrid recommendation engine. It takes into account both the interests of the user (collaborative) and the descriptions or features of the movie or show (content-based).

6.3.4 Content based Recommendations

- The gist of this approach is that we match users to the content or items they have liked or bought. Here the attributes of the users and the products are important.
- For example, for movie recommendations, we use features such as director, actors, movie length, genre, etc. to find similarity between movies. Furthermore, we can extract features like sentiment score and tf-idf scores from movie descriptions and reviews. (The tf-idf score of a word reflects how important a word is to a document in a collection of documents). **The aim of content-based recommendation is to create a 'profile' for each user and each item.**
- Consider an example of recommending news articles to users. Let's say we have 100 articles and a vocabulary of size N. We first compute the tf-idf score for each of the words for every article. Then we construct 2 vectors:

1. **Item vector :** This is a vector of length N. It contains 1 for words that have a high tf-idf score in that article, otherwise 0.
2. **User vector :** Again a 1xN vector. For every word, we store the probability of the word occurring (i.e. having a high tf-idf score) in articles that the user has consumed. Note here, that the user vector is based on the attributes of the item (tf-idf score of words in this case).

- Once we have these profiles, we compute similarities between the users and the items. The items that are recommended are the ones that 1) the user has the highest similarity with or 2) has the highest similarity with the other items the user has read. There are multiple ways of doing this. Let's look at 2 common methods:

1. **Cosine similarity**
   - To compute similarity between the user and item, we simply take the cosine similarity between the user vector and the item vector. This gives us user-item similarity.
   - To recommend items that are most similar to the items the user has bought, we compute cosine similarity between the articles the user has read and other articles. The ones that are most similar are recommended. Thus this is item-item similarity.

$$\text{cosine}(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}$$

   - Cosine similarity is best suited when you have high dimensional features, especially in information retrieval and text mining.

2. **Jaccard similarity**
   - Also known as intersection over union, the formula is as follows :

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- This is used for item-item similarity. We compare item vectors with each other and return the items that are most similar.
- Jaccard similarity is useful only when the vectors contain binary values. If they have rankings or ratings that can take on multiple values, Jaccard similarity is not applicable.
- This method is useful when we have a whole lot of 'external' features, like weather conditions, market factors, etc. which are not a property of the user or the product and can be highly variable. For example, the previous day's opening and closing price play an important role in determining the profitability of investing in a particular stock.
- This comes under the class of supervised problems where the label is whether the user liked/clicked on a product or not (0/1) or the rating the user gave that product or the number of units the user bought.
- For example, if a user likes movies such as 'Mission Impossible' then we can recommend him the movies of 'Tom Cruise' or movies with the genre 'Action'.
- In this filtering, two types of data are used. First, the likes of the user, the user's interest, user's personal information such as age or, sometimes the user's history too. This data is represented by the user vector. Second, information related to the product's known as an item vector. The item vector contains the features of all items based on which similarity between them can be calculated.

Fig. 6.3.2

- The recommendations are calculated using cosine similarity. If 'A' is the user vector and 'B' is an item vector then cosine similarity Values calculated in the cosine similarity matrix are sorted in descending order and the items at the top for that user are recommended.

6.3.5 Content-Based Recommendations Advantages and Disadvantages

Sr. No.	Advantages	Disadvantages
1.	Does not depend on data of other users.	When we have a new user, without much information about his transactions, we cannot make accurate recommendations
2.	There is no cold start problem for new items. This is because, using the item features we can easily find items it is similar to.	Clear-cut groups of similar products may result in not recommending different products. E may end up recommending a small subset over and over again

Sr. No.	Advantages	Disadvantages
3.	Recommendation results are interpretable.	If there is limited information about the content, it is difficult to clearly discriminate between items and group recommendations

6.3.6 Collaborative Filtering

- The recommendations are done based on the user's behavior. History of the user plays an important role. For example, if the user 'A' likes 'Coldplay', 'The Linkin Park' and 'Britney Spears' while the user 'B' likes 'Coldplay',
- 'The Linkin Park' and 'Taylor Swift' then they have similar interests. So, there is a huge probability that the user 'A' would like 'Taylor Swift' and the user 'B' would like 'Britney Spears'. This is the way collaborative filtering is done.

Fig. 6.3.3

- The underlying assumption of the collaborative filtering approach is that if A and B buy similar products, A is more likely to buy a product that B has bought than a product which a random person has bought.

- Unlike content based, there are no features corresponding to users or items here. All we have is the Utility Matrix. This is what it looks like :

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- A, B, C, D are the users, and the columns represent movies. The values represent ratings (1 – 5) a user has given a movie. In other cases, these values could be 0/1 depending on whether the user watched the movie or not. There are a 2 broad categories that collaborative filtering can be split into :

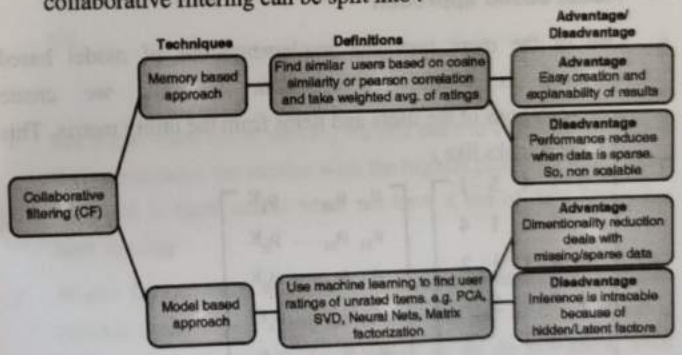


Fig. 6.3.4

Memory based approach

- For the memory based approach, the utility matrix is memorized and recommendations are made by querying the given user with the rest of the utility matrix. Let's consider an example of the same: If we have m movies and u users, we want to find out how much user i likes movie k.

$$\bar{y}_i = \frac{1}{|I_i|} \sum_{j \in I_i} y_{ij}$$

This is the mean rating that user *i* has given all the movies she/he has rated. Using this, we estimate his rating of movie k as follows :

Similarity between users a and i

$$\hat{y}_{ik} = \bar{y}_i + \frac{1}{\sum_{a \in U_k} |w_{ia}|} \sum_{a \in U_k} w_{ia}(y_{ak} - \bar{y}_a)$$

a's rating of k – a's average ratings

All users that have rated k

Similarity between users *a* and *i* can be computed using any methods like cosine similarity / Jaccard similarity / Pearson's correlation coefficient, etc. These results are very easy to create and interpret, but once the data becomes too sparse, performance becomes poor.

☞ Model based approach

- One of the more prevalent implementations of model based approach is Matrix Factorization. In this, we create representations of the users and items from the utility matrix. This is what it looks like :

$$\begin{bmatrix} 5 & 1 & 4 & 5 & 1 \\ & 5 & 2 & 1 & 4 \\ 1 & 4 & 1 & 1 & 2 \\ 4 & 1 & 5 & 5 & 4 \\ 5 & 3 & 3 & & 4 \\ 1 & 5 & 1 & 1 & 1 \\ 5 & 1 & 5 & 5 & 4 \end{bmatrix} \approx \begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_1 K \\ \mu_{21} & \mu_{22} & \cdots & \mu_2 K \\ \mu_{31} & \mu_{32} & \cdots & \mu_3 K \\ \mu_{41} & \mu_{42} & \cdots & \mu_4 K \\ \mu_{51} & \mu_{52} & \cdots & \mu_5 K \\ \mu_{61} & \mu_{62} & \cdots & \mu_6 K \\ \mu_{71} & \mu_{72} & \cdots & \mu_7 K \end{bmatrix} \times \begin{bmatrix} \upsilon_{11} & \upsilon_{21} & \upsilon_{31} & \upsilon_{41} & \upsilon_{61} \\ \upsilon_{12} & \upsilon_{22} & \upsilon_{32} & \upsilon_{42} & \upsilon_{62} \\ \vdots & & & & \\ \upsilon_1 k & \upsilon_2 k & \upsilon_3 k & \upsilon_4 k & \upsilon_6 k \end{bmatrix}$$

$$\approx \begin{bmatrix} 0.2 & 3.4 \\ 3.6 & 1.0 \\ 2.6 & 0.6 \\ 0.9 & 3.7 \\ 2.0 & 3.4 \\ 2.9 & 0.5 \\ 0.8 & 3.9 \end{bmatrix} \times \begin{bmatrix} 0.0 & 1.5 & 0.1 & 0.0 & 0.7 \\ 1.3 & 0.0 & 1.2 & 1.4 & 0.7 \end{bmatrix}$$

- Thus, our utility matrix decomposes into U and V where U represents the users and V represents the movies in a low dimensional space.
- This can be achieved by using matrix decomposition techniques like SVD or PCA or by learning the 2 embedding matrices using neural networks with the help of some optimizer like Adam, SGD etc.

$$\hat{y}_{ij} = u_i \cdot \upsilon_j$$

- For a user *i* and every movie j we just need to compute rating *y* to and recommend the movies with the highest predicted rating. This approach is most useful when we have a ton of data and it has high sparsity.
- Matrix factorization helps by reducing the dimensionality, hence making computation faster. One disadvantage of this method is that we tend to lose interpretability as we do not know what exactly elements of the user/item vectors mean.

Chapter Ends...

❑❑❑