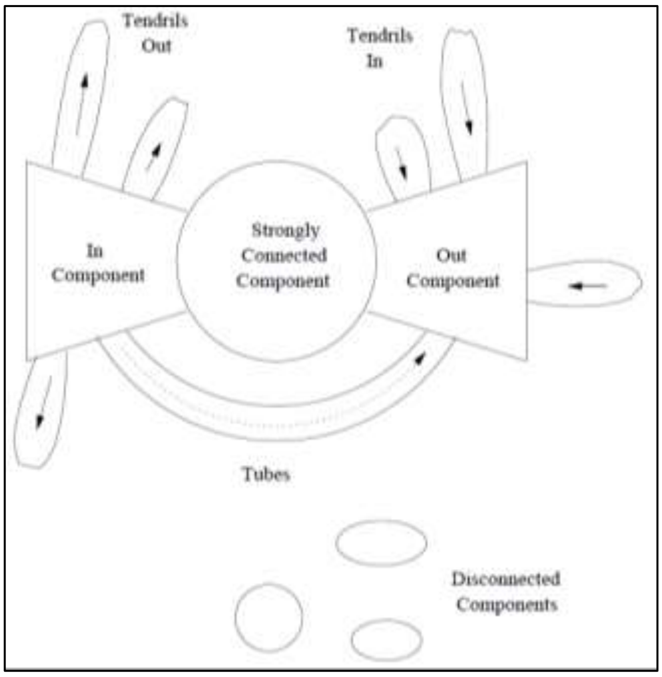
Big Data Analytics (BDA)

Big Data Analytics Applications

Link Analysis:

PageRank: PageRank is a function that assigns a real number to each page in the Web (or at least to that portion of the Web that has been crawled and its links discovered). 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.
Refer the notebook for PageRank problem calculation.

Structure of the Web:
It would be nice if the Web were strongly connected. However, it is not, in practice. An early study of the Web found it to have the structure shown in Fig. 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, but are not reachable from the out-component.



The "bowtie" picture of the Web

In addition, 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 Markov 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.
As a result, **PageRank** is usually modified to prevent such anomalies. There are really two problems we 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.

Dead Ends:

A dead end is a Web page with no links out. The presence of dead ends will cause the PageRank of some or all of the pages to go to 0 in the iterative computation, including pages that are not dead ends. We can eliminate all dead ends before undertaking a PageRank calculation by recursively dropping nodes with no arcs out. Note that dropping one node can cause another, which linked only to it, to become a dead end, so the process must be recursive.

Using PageRank in a Search Engine

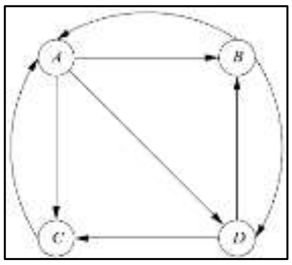
Each search engine has a secret formula that decides the order in which to show pages to the user in response to a search query consisting of one or more search terms (words). Google is said to use over 250 different properties of pages, from which a linear order of pages is decided. First, in order 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.

Efficient Computation of PageRank:

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 combiner with the Map tasks to reduce the amount of data that must be passed from Map tasks to Reduce tasks.

- **Representing Transition Matrices**

The transition matrix is very sparse, since the average Web page has about 10 out-links. If, say, we are analyzing a graph of ten billion pages, then only one in a billion entries is not 0. The proper way to represent any sparse matrix is to list the locations of the nonzero entries and their values. If we use 4-byte integers for coordinates of an element and an 8-byte double-precision number for the value, then we need 16 bytes per nonzero entry. That is, the space needed is linear in the number of nonzero entries, rather than quadratic in the side of the matrix.
However, for a transition matrix of the Web, there is one further compression that we can do. If we list the nonzero entries by column, then we know what each nonzero entry is, it is 1 divided by the out-degree of the page. We can thus represent a column by one integer for the out-degree, and one integer per nonzero entry in that column, giving the row number where that entry is located. Thus, we need slightly more than 4 bytes per nonzero entry to represent a transition matrix.
e.g. Consider the following graph whose transition matrix is



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

Following is the compact representation of this matrix.

Source	**Degree**	**Destinations**
A	3	B, C, D
B	2	A, D
C	1	A
D	2	B, C

For instance, the entry for A has degree 3 and a list of three successors. From that row of above table, we can deduce that the column for A in matrix M has 0 in the row for A (since it is not on the list of destinations) and 1/3 in the rows for B, C, and D. We know that the value is 1/3 because the degree column in the above table tells us there are three links out of 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 $v' = \beta Mv + (1 - \beta)e/n$

Where 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 − β)/n** to each component. However, it is likely, given the size of the Web today, that **v** is too much large to fit in main memory. As per the method of striping, where we break **M** into vertical stripes and break **v** into corresponding horizontal stripes, will allow us to execute the MapReduce process efficiently, with no more of **v** at any one Map task than can conveniently fit in main memory.

Topic Sensitive PageRank:

There are several improvements we can make to PageRank. One 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.

Motivation: Different people have different interests, and sometimes distinct interests are expressed using the same term in a query. The canonical example is the search query *jaguar*, which might refer to the animal, the automobile, a version of the MAC operating system, or even an ancient game console. If a search engine can deduce that the user is interested in automobiles, for example, then it can do a better job of returning relevant pages to the user. Ideally, each user would have a private PageRank vector that gives the importance of each page to that user. It is not feasible to store a vector of length many billions for each of a billion users, so we need to do something simpler.

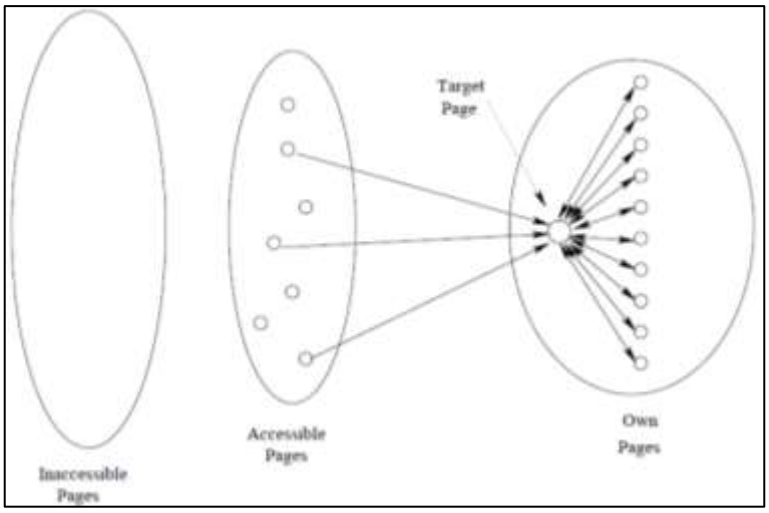
The topic-sensitive PageRank approach creates one vector for each of some small number of topics, biasing the PageRank to favor pages of that topic. We then endeavor to classify users according to the degree of their interest in each of the selected topics. While we surely lose some accuracy, the benefit is that we store only a short vector for each user, rather than an enormous vector for each user.

e.g. One useful topic set is the 16 top-level categories (sports, medicine, etc.) of the Open Directory (DMOZ). We could create 16 PageRank vectors, one for each topic. If we could determine that the user is interested in one of these topics, perhaps by the content of the pages they have recently viewed, then we could use the PageRank vector for that topic when deciding on the ranking of pages.

Using Topic-Sensitive PageRank-
In order to integrate topic-sensitive PageRank into a search engine, we must:
1. Decide on the topics for which we shall create specialized PageRank vectors.
2. Pick a teleport set for each of these topics, and use that set to compute the topic-sensitive PageRank vector for that topic.
3. Find a way of determining the topic or set of topics that are most relevant for a particular search query.
4. Use the PageRank vectors for that topic or topics in the ordering of the responses to the search query.
The third step is probably the trickiest, and several methods have been proposed. Some possibilities:
(a) Allow the user to select a topic from a menu.
(b) Infer the topic(s) by the words that appear in the Web pages recently searched by the user, or recent queries issued by the user.
(c) Infer the topic(s) by information about the user, e.g., their bookmarks or their stated interests on Facebook.

Refer notebook for topic-sensitive PageRank problem.

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**.



Architecture of Spam Farm-
A collection of pages whose purpose is to increase the PageRank of a certain page or pages is called a spam farm. Fig. shows the simplest form of spam farm. From the point of view of the spammer, the Web is divided into three parts:
1. Inaccessible pages: the pages that the spammer cannot affect. Most of the Web is in this part.
2. Accessible pages: those pages that, while they are not controlled by the spammer, can be affected by the spammer.
3. Own pages: the pages that the spammer owns and controls.

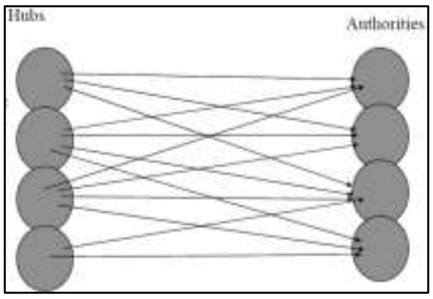
The spam farm consists of the spammer’s own pages, organized in a special way as seen on the right, and some links from the accessible pages to the spammer’s pages. Without some links from the outside, the spam farm would be useless, since it would not even be crawled by a typical search engine.
Concerning the accessible pages, it might seem surprising that one can affect a page without owning it. However, today there are many sites, such as blogs or newspapers that invite others to post their comments on the site. In order to get as much PageRank flowing to his own pages from outside, the spammer posts many comments such as “I agree. Please see my article at www.mySpamFarm.com.”
In the spam farm, there is one-page **t**, the target page, at which the spammer attempts to place as much PageRank as possible. There is a large number m of supporting pages, that accumulate the portion of the PageRank that is distributed equally to all pages (the fraction $1-\beta$ of the

PageRank that represents surfers going to a random page). The supporting pages also prevent the PageRank of **t** from being lost, to the extent possible, since some will be taxed away at each round. Notice that **t** has a link to every supporting page, and every supporting page links only to **t**.

Hubs and Authorities:
Authorities are pages that are recognized as providing significant, trustworthy, and useful information on a topic. In-degree (number of pointers to a page) is one simple measure of authority. However, in-degree treats all links as equal.
e.g. Page *i* is called an authority for the query "automobile makers" if it contains valuable information on the subject. Official web sites of car manufacturers, such as www.bmw.com, HyundaiUSA.com, www.mercedes-benz.com would be authorities for this search. Commercial web sites selling cars might be authorities on the subject as well. These are the ones truly relevant to the given query. These are the ones that the user expects back from the query engine.

Hubs are index pages that provide lots of useful links to relevant content pages (topic authorities). Their role is to advertise the authoritative pages. They contain useful links towards the authoritative pages. In other words, hubs point the search engine in the "right direction".
e.g. In real life, when you buy a car, you are more inclined to purchase it from a certain dealer that your friend recommends. Following the analogy, the authority in this case would be the car dealer, and the hub would be your friend. You trust your friend; therefore, you trust what your friend recommends.
In the world wide web, hubs for our query about automobiles might be pages that contain rankings of the cars, blogs where people discuss about the cars that they purchased, and so on.

HITS (Hyperlink-Induced Topic Search) Algorithm:
In the same time that PageRank was being developed, Jon Kleinberg a professor in the Department of Computer Science at Cornell came up with his own solution to the Web Search problem. He developed an algorithm that made use of the link structure of the web in order to discover and rank pages relevant for a particular topic. HITS is now part of the Ask search engine (www.Ask.com). It is based on mutually recursive facts-



- Hubs point to lots of authorities.
- Authorities are pointed to by lots of hubs.

Together they forma bipartite graph as shown in Fig.
Steps:

- Computes hubs and authorities for a particular topic specified by a normal query.
- First determines a set of relevant pages for the query called the base set *S*.
- Analyze the link structure of the web subgraph defined by *S* to find authority and hub pages in this set.

Refer Notebook for further steps and example.

Mining Social- Network Graphs:
Social Network as Graphs:
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 Facebook friends' graph. But they can be directed graphs, as for example the graphs of followers on Twitter or Google+.

Types of Social Network:

Telephone Networks: Here the nodes represent phone numbers, which are really individuals. There is an edge between two nodes if a call has been placed between those phones in some fixed period of time, such as last month, or "ever." The edges could be weighted by the number of calls made between these phones during the period. Communities in a telephone network will form from groups of people that communicate frequently: groups of friends, members of a club, or people working at the same company, for example.

Email Networks: The nodes represent email addresses, which are again individuals. An edge represents the fact that there was at least one email in at least one direction between the two addresses. Alternatively, we may only place an edge if there were emails in both directions. In that way, we avoid viewing spammers as "friends" with all their victims. Another approach is to label edges as weak or strong. Strong edges represent communication in both directions, while weak edges indicate that the communication was in one direction only. The communities seen in email networks come from the same sorts of groupings individuals working under same organization, etc. A similar sort of network involves people who text other people through their cell phones.

Collaboration Networks: Nodes represent individuals who have published research papers. There is an edge between two individuals who published one or more papers jointly. Optionally, 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. In fact, the data involved in Collaborative filtering, often can be viewed as forming a pair of networks, one for the customers and one for the products. Customers who buy the same sorts of products, e.g., science-fiction books, will form communities, and dually, products that are bought by the same customers will form communities, e.g., all science-fiction books.

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 animals eating each other), as well as other types, like product co-purchasing networks (e.g., Groupon).

Clustering of Social Network Graph:

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

If we were to apply standard clustering techniques to a social-network graph, our first step would be to define a distance measure. 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.

- **Applying Standard Clustering Methods**

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.

- **Betweenness**

Define the betweenness of an edge (a, b) to be the number of pairs of nodes x and y such that the edge (a, b) lies on the shortest path between x and y. To be more precise, since there can be several shortest paths between x and y, edge (a, b) is credited with the fraction of those shortest paths that include the edge (a, b). As in golf, a high score is bad. It suggests that the edge (a, b) runs between two different communities; that is, a and b do not belong to the same community.

- **The Girwan-Newman Algorithm**

The algorithm states rules as follows-
1. Each leaf in the DAG (a leaf is a node with no DAG edges to nodes at levels below) gets a credit of 1.
2. Each node that is not a leaf gets a credit equal to 1 plus the sum of the credits of the DAG edges from that node to the level below.
3. A DAG edge e entering node Z from the level above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e. Formally, let the parents of Z be Y1, Y2, . . . , Yk.

- **Using Betweenness to Find Communities**

The betweenness scores for the edges of a graph behave something like a distance measure on the nodes of the graph. It is not exactly a distance measure, because it is not defined for pairs of nodes that are unconnected by an edge, and might not satisfy the triangle inequality even when defined. However, we can cluster by taking the edges in order of increasing betweenness and add them to the graph one at a time. At each step, the connected components of the graph form some clusters. The higher the betweenness we allow, the more edges we get, and the larger the clusters become.

Direct Discovery of Communities

- **Finding Cliques**

Our first thought about how we could find sets of nodes with many edges between them is to start by finding a large clique (a set of nodes with edges between any two of them). However, that task is not easy. Not only is finding maximal cliques NP-complete, but it is among the hardest of the NP-complete problems in the sense that even approximating the maximal clique is hard. Further, it is possible to have a set of nodes with almost all edges between them, and yet have only relatively small cliques.

- **Complete Bipartite Graphs**

A complete bipartite graph consists of s nodes on one side and t nodes on the other side, with all st possible edges between the nodes of one side and the other present. We denote this graph by Ks,t. You should draw an analogy between complete bipartite graphs as subgraphs of general bipartite graphs and cliques as subgraphs of general graphs. In fact, a clique of s nodes is often referred to as a complete graph and denoted Ks, while a complete bipartite subgraph is sometimes called a bi-clique.

- **Finding Complete Bipartite Subgraphs**

Suppose we are given a large bipartite graph G , and we want to find instances of Ks,t within it. It is possible to view the problem of finding instances of Ks,t within G as one of finding frequent itemsets. For this purpose, let the “items” be the nodes on one side of G, which we shall call the left side. We assume that the instance of Ks,t we are looking for has t nodes on the left side, and we shall also assume for efficiency that $t \leq s$. The "baskets" correspond to the nodes on the other side of G (the right side). The members of the basket for node v are the nodes of the left side to which v is connected. Finally, let the support threshold be s, the number of nodes that the instance of Ks,t has on the right side.

Counting Triangles using MapReduce:

Suppose we have 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 $A < B$. Using this relation, we can express the set of triangles of the graph whose edges are E by the natural join

E(X, Y) ►◄ E(X,Z) ►◄ E(Y,Z)

The Map tasks divide the relation E into as many parts as there are Map tasks. Suppose one Map task is given the tuple E(u, v) to send to certain Reduce tasks. First, think of (u, v) as a tuple of the join term E(X, Y). We can hash u and v to get the bucket numbers for X and Y, but we don't know the bucket to which Z hashes. Thus, we must send E(u, v) to all Reducer tasks that correspond to a sequence of three bucket numbers (h(u), h(v), z) for any of the b possible buckets z.

But the same tuple E(u, v) must also be treated as a tuple for the term E(X,Z). We therefore also send the tuple E(u, v) to all Reduce tasks that correspond to a triple (h(u), y, h(v)) for any y. Finally, we treat E(u, v) as a tuple of the term E(Y,Z) and send that tuple to all Reduce tasks corresponding to a triple (x, h(u), h(v)) for any x. The total communication required is thus 3b key-value pairs for each of the m tuples of the edge relation E. That is, the minimum communication cost is O(mb) if we use b3 Reduce tasks.

Next, let us compute the total execution cost at all the Reduce tasks. Assume that the hash function distributes edges sufficiently randomly that the Reduce tasks each get approximately the same number of edges. Since the total number of edges distributed to the b3 Reduce tasks is O(mb), it follows that each task receives $O(m/b^2)$ edges. If we use the algorithm for finding triangle at each Reduce task, the total computation at a task is $O((m/b^2)^{3/2})$, or $O(m^{3/2}/b^3)$.
Since there are b^3 Reduce tasks, the total computation cost is O(m3/2).

Recommendation Systems:

There is an extensive class of Web applications that involve predicting user responses to options. Such a facility is called a recommendation system. two good examples of recommendation systems are:

1. Offering news articles to on-line newspaper readers, based on a prediction of reader interests.
2. Offering customers of an on-line retailer suggestions about what they might like to buy, based on their past history of purchases and/or product searches.

Recommendation systems use a number of different technologies. We can classify these systems into two broad groups.

• Content-based systems examine properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre.

• Collaborative filtering systems recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users. This sort of recommendation system can use the groundwork laid on similarity search and on clustering. However, these technologies by themselves are not sufficient, and there are some new algorithms that have proven effective for recommendation systems.

Applications of Recommendation Systems:

1. Product Recommendations: Perhaps the most important use of recommendation systems is at on-line retailers. We have noted how Amazon or similar on-line vendors strive to present each returning user with some suggestions of products that they might like to buy. These suggestions are not random, but are based on the purchasing decisions made by similar customers or on other techniques we shall discuss in this chapter.
2. Movie Recommendations: Netflix offers its customers recommendations of movies they might like. These recommendations are based on ratings provided by users, much like the ratings suggested in the example utility matrix of Fig. 9.1. The importance of predicting ratings accurately is so high, that Netflix offered a prize of one million dollars for the first algorithm that could beat its own recommendation system by 10%.1 The prize was finally won in 2009, by a team of researchers called "Bellkor's Pragmatic Chaos," after over three years of competition.
3. News Articles: News services have attempted to identify articles of interest to readers, based on the articles that they have read in the past. The similarity might be based on the similarity of important words in the documents, or on the articles that are read by people with similar reading tastes. The same principles apply to recommending blogs from among the millions of blogs available, videos on YouTube, or other sites where content is provided regularly.

Content-Based Recommendation:

1. **Item Profiles**

In a content-based system, we must construct for each item a profile, which is record or collection of records representing important characteristics of that item. In simple cases, the profile consists of some characteristics of the item that are easily discovered. For example, consider the features of a movie that might be relevant to a recommendation system.
1. The set of actors of the movie. Some viewers prefer movies with their favorite actors.
2. The director. Some viewers have a preference for the work of certain directors.
3. The year in which the movie was made. Some viewers prefer old movies; others watch only the latest releases.
4. The genre or general type of movie. Some viewers like only comedies, others dramas or romances.

2. **Discovering features of Documents**

There are many kinds of documents for which a recommendation system can be useful. For example, there are many news articles published each day, and we cannot read all of them. A recommendation system can suggest articles on topics a user is interested in, but how can we distinguish among topics? Web pages are also a collection of documents. Can we suggest pages a user might want to see? Likewise, blogs could be recommended to interested users, if we could classify blogs by topics.

3. **Obtaining Item Features from Tags**

Let us consider a database of images as an example of a way that features have been obtained for items. The problem with images is that their data, typically an array of pixels, does not tell us anything useful about their features. We can calculate simple properties of pixels, such as the average amount of red in the picture, but few users are looking for red pictures or especially like red pictures.