Chapter 18

Mining Web Data

"*Data is a precious thing, and will last longer than the systems themselves.*"—Tim Berners-Lee

18.1 Introduction

The Web is an unique phenomenon in many ways, in terms of its scale, the distributed and uncoordinated nature of its creation, the openness of the underlying platform, and the resulting diversity of applications it has enabled. Examples of such applications include e-commerce, user collaboration, and social network analysis. Because of the distributed and uncoordinated nature in which the Web is both created and used, it is a rich treasure trove of diverse types of data. This data can be either a source of knowledge about various subjects, or personal information about users.

Aside from the content available in the documents on the Web, the usage of the Web results in a significant amount of data in the form of user logs or Web transactions. There are two primary types of data available on the Web that are used by mining algorithms.

1. *Web content information:* This information corresponds to the Web documents and links created by users. The documents are linked to one another with hypertext links. Thus, the content information contains two components that can be mined either together, or in isolation.
   - *Document data:* The document data are extracted from the pages on the World Wide Web. Some of these extraction methods are discussed in Chap. 13.
   - *Linkage data:* The Web can be viewed as a massive graph, in which the pages correspond to nodes, and the linkages correspond to edges between nodes. This linkage information can be used in many ways, such as searching the Web or determining the similarity between nodes.
2. *Web usage data:* This data corresponds to the patterns of user activity that are enabled by Web applications. These patterns could be of various types.

- *Web transactions, ratings, and user feedback:* Web users frequently buy various types of items on the Web, or express their affinity for specific products in the form of ratings. In such cases, the buying behavior and/or ratings can be leveraged to make inferences about the preferences of different users. In some cases, the user feedback is provided in the form of textual user reviews that are referred to as *opinions.*
- *Web logs:* User browsing behavior is captured in the form of Web logs that are typically maintained at most Web sites. This browsing information can be leveraged to make inferences about user activity.

These diverse data types automatically define the types of applications that are common on the Web. In coordination with the different data types, the applications are also either content- or usage-centric.

1. *Content-centric applications:* The documents and links on the Web are used in various applications such as search, clustering, and classification. Some examples of such applications are as follows:
    - *Data mining applications:* Web documents are used in conjunction with different types of data mining applications such as clustering and categorization. Such applications are used frequently by Web portals for organizing pages.
    - *Web crawling and resource discovery:* The Web is a tremendous resource of knowledge about documents on various subjects. However, this resource is widely distributed on the Internet, and it needs to be discovered and stored at a single place to make inferences.
    - *Web search:* The goal in Web search is to discover high-quality, relevant documents in response to a user-specified set of keywords. As will be evident later, the notions of quality and relevance are defined both by the linkage and content structure of the documents.
    - *Web linkage mining:* In these applications, either actual or logical representations of linkage structure on the Web are mined for useful insights. Examples of logical representations of Web structure include social and information networks. Social networks are linked networks of users, whereas information networks are linked networks of users and objects.
2. *Usage-centric applications:* The user activity on the Web is mined to make inferences. The different ways in which user activity can be mined are as follows:
    - *Recommender systems:* In these cases, preference information in the form of either ratings for product items or product buying behavior is used to make recommendations to other like-minded users.
    - *Web log analysis:* Web logs are a useful resource for Web site owners to determine relevant patterns of user browsing. These patterns can be leveraged for making inferences such as finding anomalous patterns, user interests, and optimal Web site design.

Many of the aforementioned applications overlap with other chapters in the book. For example, content-centric data mining applications have already been covered in previous chapters of this book, especially in Chap. 13 on mining text data. Some of these methods do need to be modified to account for the additional linkage data. Many linkage mining applications are discussed in Chap. 19 on social network analysis. Therefore, this chapter will focus on the applications that are not primarily covered by other chapters. Among the content-centric applications, Web crawling, search, and ranking will be discussed. Among the usage-centric applications, recommender systems and Web log mining applications will be discussed.

This chapter is organized as follows. Sect. 18.2 discusses Web crawlers and resource discovery. Search engine indexing and query-processing methods are discussed in Sect. 18.3. Ranking algorithms are presented in Sect. 18.4. Recommender systems are discussed in Sect. 18.5. Methods for mining Web logs are discussed in Sect. 18.6. The summary is presented in Sect. 18.7.

18.2 Web Crawling and Resource Discovery

Web crawlers are also referred to as *spiders* or *robots*. The primary motivation for Web crawling is that the resources on the Web are dispensed widely across globally distributed sites. While the Web browser provides a graphical user interface to access these pages in an interactive way, the full power of the available resources cannot be leveraged with the use of only a browser. In many applications, such as search and knowledge discovery, it is necessary to download all the relevant pages *at a central location*, to allow machine learning algorithms to use these resources efficiently.

Web crawlers have numerous applications. The most important and well-known application is search, in which the downloaded Web pages are indexed, to provide responses to user keyword queries. All the well-known search engines, such as Google and Bing, employ crawlers to periodically refresh the downloaded Web resources at their servers. Such crawlers are also referred to as *universal crawlers* because they are intended to crawl all pages on the Web irrespective of their subject matter or location. Web crawlers are also used for business intelligence, in which the Web sites related to a particular subject are crawled or the sites of a competitor are monitored and incrementally crawled as they change. Such crawlers are also referred to as *preferential crawlers* because they discriminate between the relevance of different pages for the application at hand.

18.2.1 A Basic Crawler Algorithm

While the design of a crawler is quite complex, with a distributed architecture and many processes or threads, the following describes a simple sequential and universal crawler that captures the essence of how crawlers are constructed.

The basic crawler algorithm, described in a very general way, uses a seed set of Universal Resource Locators (URLs) S, and a selection algorithm $\mathcal{A}$ as the input. The algorithm $\mathcal{A}$ decides which document to crawl next from a current *frontier list* of URLs. The frontier list represents URLs extracted from the Web pages. These are the candidates for pages that can eventually be fetched by the crawler. The selection algorithm $\mathcal{A}$ is important because it regulates the basic strategy used by the crawler to discover the resources. For example, if new URLs are appended to the end of the frontier list, and the algorithm $\mathcal{A}$ selects documents from the beginning of the list, then this corresponds to a breadth-first algorithm.

The basic crawler algorithm proceeds as follows. First, the seed set of URLs is added to the frontier list. In each iteration, the selection algorithm $\mathcal{A}$ picks one of the URLs from the frontier list. This URL is deleted from the frontier list and then fetched using the

```
**Algorithm** *BasicCrawler*(Seed URLs: $S$, Selection Algorithm: $\mathcal{A}$)
begin
  $FrontierList = S$;
  repeat
    Use algorithm $\mathcal{A}$ to select URL $X \in FrontierSet$;
    $FrontierList = FrontierList - \{X\}$;
    Fetch URL $X$ and add to repository;
    Add all relevant URLs in fetched document $X$ to
       end of $FrontierList$;
  until termination criterion;
end
```

Figure 18.1: The basic crawler algorithm

HTTP protocol. This is the same mechanism used by browsers to fetch Web pages. The main difference is that the fetching is now done by an automated program using automated selection decisions, rather than by the manual specification of a link by a user with a Web browser. The fetched page is stored in a local repository, and the URLs inside it are extracted. These URLs are then added to the frontier list, provided that they have not already been visited. Therefore, a separate data structure, in the form of a hash table, needs to be maintained to store all visited URLs. In practical implementations of crawlers, not all unvisited URLs are added to the frontier list due to Web spam, spider traps, topical preference, or simply a practical limit on the size of the frontier list. These issues will be discussed later. After the relevant URLs have been added to the frontier list, the next iteration repeats the process with the next URL on the list. The process terminates when the frontier list is empty. If the frontier list is empty, it does not necessarily imply that the entire Web has been crawled. This is because the Web is not strongly connected, and many pages are unreachable from most randomly chosen seed sets. Because most practical crawlers such as search engines are *incremental* crawlers that refresh pages over previous crawls, it is usually easy to identify unvisited seeds from previous crawls and add them to the frontier list, if needed. With large seed sets, such as a previously crawled repository of the Web, it is possible to robustly crawl most pages. The basic crawler algorithm is described in Fig. 18.1.

Thus, the crawler is a graph search algorithm that discovers the outgoing links from nodes by parsing Web pages and extracting the URLs. The choice of the selection algorithm $\mathcal{A}$ will typically result in a bias in the crawling algorithm, especially in cases where it is impossible to crawl all the relevant pages due to resource limitations. For example, a breadth-first crawler is more likely to crawl a page with many links pointing to it. Interestingly, such biases are sometimes desirable in crawlers because it is impossible for any crawler to index the entire Web. Because the indegree of a Web page is often closely related to its *PageRank*, a measure of a Web page's quality, this bias is not necessarily undesirable. Crawlers use a variety of other selection strategies defined by the algorithm $\mathcal{A}$.

1. Because most universal crawlers are incremental crawlers that are intended to refresh previous crawls, it is desirable to crawl frequently changing pages. The change frequency can be estimated from repeated previous crawls of the same page. Some resources such as news portals are updated frequently. Therefore, frequently updated pages may be selected by the algorithm $\mathcal{A}$.

2. The selection algorithm $\mathcal{A}$ may specifically choose Web pages with high *PageRank* from frontier list. The computation of *PageRank* is discussed in Sect. 18.4.1.

A practice, a combination of factors are used by the commercial crawlers employed by search engines.

18.2.2 Preferential Crawlers

In the preferential crawler, only pages satisfying a user-defined criterion need to be crawled. This criterion may be specified in the form of keyword presence in the page, a topical criterion defined by a machine learning algorithm, a geographical criterion about page location, or a combination of the different criteria. In general, an arbitrary predicate may be specified by the user, which forms the basis of the crawling. In these cases, the major change is to the approach used for updating the frontier list during crawling.

1. The Web page needs to meet the user-specified criterion in order for its extracted URLs to be added to the frontier list.
2. In some cases, the anchor text may be examined to determine the relevance of the Web page to the user-specified query.
3. In context-focused crawlers, the crawler is trained to learn the likelihood that relevant pages are within a short distance of the page, even if the Web page is itself not directly relevant to the user-specified criterion. For example, a Web page on "*data mining*" is more likely to point to a Web page on "*information retrieval*," even though the data mining page may not be relevant to the query on "*information retrieval*." URLs from such pages may be added to the frontier list. Therefore, heuristics need to be designed to learn such context-specific relevance.

Changes may also be made to the algorithm $\mathcal{A}$. For example, URLs with more relevant anchor text, or with relevant tokens in the Web address, may be selected first by algorithm $\mathcal{A}$. A URL such as `http://www.golf.com`, with the word "*golf*" in the Web address may be more relevant to the topic of "*golf*," than a URL without the word in it. The bibliographic notes contain pointers to a number of heuristics that are commonly used for preferential resource discovery.

18.2.3 Multiple Threads

When a crawler issues a request for a URL and waits for it, the system is idle, with no work being done at the crawler end. This would seem to be a waste of resources. A natural way to speed up the crawling is by leveraging concurrency. The idea is to use multiple threads of the crawler that update a shared data structure for visited URLs and the page repository. In such cases, it is important to implement concurrency control mechanisms for locking or unlocking the relevant data structures during updates. The concurrent design can significantly speed up a crawler with more efficient use of resources. In practical implementations of large search engines, the crawler is distributed geographically with each "sub-crawler" collecting pages in its geographical proximity.

18.2.4 Combatting Spider Traps

The main reason that the crawling algorithm always visits distinct Web pages is that it maintains a list of previously visited URLs for comparison purposes. However, some shopping sites create dynamic URLs in which the last page visited is appended at the end of the user sequence to enable the server to log the user action sequences within the URL for future analysis. For example, when a user clicks on the link for *page2* from `http://www.examplesite.com/page1`, the new dynamically created URL will be `http://www.examplesite.com/page1/page2`. Pages that are visited further will continue to be appended to the end of the URL, even if these pages were visited before. A natural way to combat this is to limit the maximum size of the URL. Furthermore, a maximum limit may also be placed on the number of URLs crawled from a particular site.

18.2.5 Shingling for Near Duplicate Detection

One of the major problems with the Web pages collected by a crawler is that many duplicates of the same page may be crawled. This is because the same Web page may be mirrored at multiple sites. Therefore, it is crucial to have the ability to detect near duplicates. An approach known as *shingling* is commonly used for this purpose.

A k-shingle from a document is simply a string of k consecutively occurring words in the document. A shingle can also be viewed as a k-gram. For example, consider the document comprising the following sentence:

Mary had a little lamb, its fleece was white as snow.

The set of 2-shingles extracted from this sentence is "*Mary had*", "*had a*", "*a little*", "*little lamb*", "*lamb its*", "*its fleece*", "*fleece was*", "*was white*", "*white as*", and "*as snow*". Note that the number of k-shingles extracted from a document is no longer than the length of the document, and 1-shingles are simply the set of words in the document. Let S_1 and S_2 be the k-shingles extracted from two documents D_1 and D_2. Then, the shingle-based similarity between D_1 and D_2 is simply the Jaccard coefficient between S_1 and S_2

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}. \tag{18.1}$$

Typically, the value of k ranges between 5 and 10 depending on the corpus size and application domain. The advantage of using k-shingles instead of the individual words (1-shingles) for Jaccard coefficient computation is that shingles are less likely than words to repeat in different documents. There are r^k distinct shingles for a lexicon of size r. For $k \geq 5$, the chances of many shingles recurring in two documents becomes very small. Therefore, if two documents have many k-shingles in common, they are very likely to be near duplicates. To save space, the individual shingles are hashed into 4-byte (32-bit) numbers that are used for comparison purposes. Such a representation also enables better efficiency.

18.3 Search Engine Indexing and Query Processing

After the documents have been crawled, they are leveraged for query processing. There are two primary stages to the search index construction:

1. *Offline stage:* This is the stage in which the search engine preprocesses the crawled documents to extract the tokens and constructs an index to enable efficient search. A quality-based ranking score is also computed for each page at this stage.

2. *Online query processing:* This preprocessed collection is utilized for online query processing. The relevant documents are accessed and then ranked using both their relevance to the query and their quality.

The preprocessing steps for Web document processing are described in Chap. 13 on mining text data. The relevant tokens are extracted and stemmed. Stop words are removed. These documents are then transformed to the vector space representation for indexing.

After the documents have been transformed to the vector space representation, an inverted index is constructed on the document collection. The construction of inverted indices is described in Sect. 5.3.1.2 of Chap. 5. The inverted list maps each word identifier to a list of document identifiers containing it. The frequency of the word is also stored with the document identifier in the inverted list. In many implementations, the position information of the word in the document is stored as well.

Aside from the inverted index that maps words to documents, an index is needed for accessing the storage location of the inverted word lists relevant to the query terms. These locations are then used to access the inverted lists. Therefore, a *vocabulary index* is required as well. In practice, many indexing methods such as hashing and tries are commonly used. Typically, a hash function is applied to each word in the query term, to yield the logical address of the corresponding inverted list.

For a given set of words, all the relevant inverted lists are accessed, and the intersection of these inverted lists is determined. This intersection is used to determine the Web document identifiers that contain all, or most of, the search terms. In cases, where one is interested only in documents containing most of the search terms, the intersection of different subsets of inverted lists is performed to determine the best match. Typically, to speed up the process, two indexes are constructed. A smaller index is constructed on only the titles of the Web page, or anchor text of pages *pointing to the page*. If enough documents are found in the smaller index, then the larger index is not referenced. Otherwise, the larger index is accessed. The logic for using the smaller index is that the title of a Web page and the anchor text of Web pages pointing to it, are usually highly representative of the content in the page.

Typically, the number of pages returned for common queries may be of the order of millions or more. Obviously, such a large number of query results will not be easy for a human user to assimilate. A typical browser interface will present only the first few (say 10) results to the human user in a single view of the search results, with the option of browsing other less relevant results. Therefore, one of the most important problems in search engine query processing is that of *ranking*. The aforementioned processing of the inverted index does provide a content-based score. This score can be leveraged for ranking. While the exact scoring methodology used by commercial engines is proprietary, a number of factors are known to influence the content-based score:

1. A word is given different weights, depending upon whether it occurs in the title, body, URL token, or the anchor text of a pointing Web page. The occurrence of the term in the title or the anchor text of a Web page pointing to that page is generally given higher weight.

2. The number of occurrences of a keyword in a document will be used in the score. Larger numbers of occurrences are obviously more desirable.

3. The prominence of a term in font size and color may be leveraged for scoring. For example, larger font sizes will be given a larger score.

4. When multiple keywords are specified, their relative positions in the documents are used as well. For example, if two keywords occur close together in a Web page, then this increases the score.

The content-based score is not sufficient, however, because it does not account for the *reputation*, or the *quality*, of the page. It is important to use such mechanisms because of the uncoordinated and open nature of Web development. After all, the Web allows anyone to publish almost anything, and therefore there is little control on the quality of the results. A user may publish incorrect material either because of poor knowledge on the subject, economic incentives, or with a deliberately malicious intent of publishing misleading information.

Another problem arises from the impact of *Web spam*, in which Web site owners intentionally serve misleading content to rank their results higher. Commercial Web site owners have significant economic incentives to ensure that their sites are ranked higher. For example, an owner of a business on golf equipment, would want to ensure that a search on the word "*golf*" ranks his or her site as high as possible. There are several strategies used by Web site owners to rank their results higher.

1. *Content-spamming:* In this case, the Web host owner fills up repeated keywords in the hosted Web page, even though these keywords are not actually visible to the user. This is achieved by controlling the color of the text and the background of the page. Thus, the idea is to maximize the content relevance of the Web page to the search engine, without a corresponding increase in the *visible* level of relevance.

2. *Cloaking:* This is a more sophisticated approach, in which the Web site serves different content to crawlers than it does to users. Thus, the Web site first determines whether the incoming request is from a crawler or from a user. If the incoming request is from a user, then the actual content (e.g., advertising content) is served. If the request is from a crawler, then the content that is most relevant to specific keywords is served. As a result, the search engine will use different content to respond to user search requests from what a Web user will actually see.

It is obvious that such spamming will significantly reduce the quality of the search results. Search engines also have significant incentives to improve the quality of their results to support their paid advertising model, in which the *explicitly marked* sponsored links appearing on the side bar of the search results are truly paid advertisements. Search engines do not want advertisements (disguised by spamming) to be served as *bona fide* results to the query, especially when such results reduce the quality of the user experience. This has led to an adversarial relationship between search engines and spammers, in which the former use reputation-based algorithms to reduce the impact of spam. At the other end of Web site owners, a *search engine optimization (SEO)* industry attempts to optimize search results by using their knowledge of the algorithms used by search engines, either through the general principles used by engines or through reverse engineering of search results.

For a given search, it is almost always the case that a small subset of the results is more informative or provides more accurate information. How can such pages be determined? Fortunately, the Web provides several natural voting mechanisms to determine the reputation of pages.

1. *Page citation mechanisms:* This is the most common mechanism used to determine the quality of Web pages. When a page is of high quality, many other Web pages point to it. A citation can be logically viewed as a vote for the Web page. While the

number of in-linking pages can be used as a rough indicator of the quality, it does not provide a complete view because it does not account for the quality of the pages pointing to it. To provide a more holistic citation-based vote, an algorithm referred to as *PageRank* is used.

2. *User feedback or behavioral analysis mechanisms:* When a user chooses a Web page from among the responses to a search result, this is clear evidence of the relevance of that page to the user. Therefore, other similar pages, or pages accessed by other similar users can be returned. Such an approach is generally hard to implement in search because of limited user-identification mechanisms. Some search engines, such as Excite, have used various forms of relevance feedback. While these mechanisms are used less often by search engines, they are nevertheless quite important for *commercial recommender systems*. In commercial recommender systems, the recommendations are made by the Web site itself during user browsing, rather than by search engines. This is because commercial sites have stronger user-identification mechanisms (e.g., user registration) to enable more powerful algorithms for inferring user interests.

Typically, the reputation score is determined using *PageRank*-like algorithms. Therefore, if $IRScore$ and $RepScore$ are the content- and reputation-based scores of the Web page, respectively, then the final ranking score is computed as a function of these scores:

$$RankScore = f(IRScore, RepScore). \tag{18.2}$$

The exact function $f(\cdot,\cdot)$ used by commercial search engines is proprietary, but it is always monotonically related to both the $IRScore$ and $RepScore$. Various other factors, such as the geographic location of the browser, also seem to play a role in the ranking.

It should be pointed out, that citation-based reputation scores are not completely immune to other types of spamming that involve coordinated creation of a large number of links to a Web page. Furthermore, the use of anchor text of *pointing* Web pages in the content portion of the rank score can sometimes lead to amusingly irrelevant search results. For example, a few years back, a search on the keyword "*miserable failure*" in the Google search engine, returned as its top result, the official biography of a previous president of the United States of America. This is because many Web pages were constructed in a coordinated way to use the anchor text "*miserable failure*" to point to this biography. This practice of influencing search results by coordinated linkage construction to a particular site is referred to as *Googlewashing*. Such practices are less often economically motivated, but are more often used for comical or satirical purposes.

Therefore, the ranking algorithms used by search engines are not perfect but have, nevertheless, improved significantly over the years. The algorithms used to compute the reputation-based ranking score will be discussed in the next section.

18.4 Ranking Algorithms

The *PageRank* algorithm uses the linkage structure of the Web for reputation-based ranking. The *PageRank* method is independent of the user query, because it only precomputes the reputation portion of the score in Eq. 18.2. The *HITS* algorithm is query-specific. It uses a number of intuitions about how authoritative sources on various topics are linked to one another in a hyperlinked environment.

18.4.1 PageRank

The *PageRank* algorithm models the importance of Web pages with the use of the citation (or linkage) structure in the Web. The basic idea is that highly reputable documents are more likely to be cited (or in-linked) by other reputable Web pages.

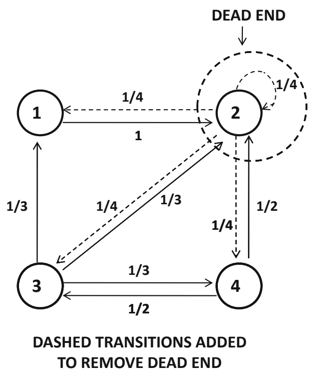
A random surfer model on the Web graph is used to achieve this goal. Consider a random surfer who visits random pages on the Web by selecting random links on a page. The long-term relative frequency of visits to any particular page is clearly influenced by the number of in-linking pages to it. Furthermore, the long-term frequency of visits to any page will be higher if it is linked to by other frequently visited (or *reputable*) pages. In other words, the *PageRank* algorithm models the reputation of a Web page in terms of its long-term frequency of visits by a random surfer. This long-term frequency is also referred to as the *steady-state probability.* This model is also referred to as the *random walk model.*

The basic random surfer model does not work well for all possible graph topologies. A critical issue is that some Web pages may have no outgoing links, which may result in the random surfer getting trapped at specific nodes. In fact, a probabilistic transition is not even meaningfully defined at such a node. Such nodes are referred to as *dead ends.* An example of a dead-end node is illustrated in Fig. 18.2a. Clearly, dead ends are undesirable because the transition process for *PageRank* computation cannot be defined at that node. To address this issue, two modifications are incorporated in the random surfer model. The first modification is to add links from the dead-end node (Web page) to all nodes (Web pages), including a self-loop to itself. Each such edge has a transition probability of $1/n$. This does not fully solve the problem, because the dead ends can also be defined on *groups of nodes.* In these cases, there are no outgoing links from *a group of nodes* to the remaining nodes in the graph. This is referred to as a *dead-end component*, or *absorbing component.* An example of a dead-end component is illustrated in Fig. 18.2b.

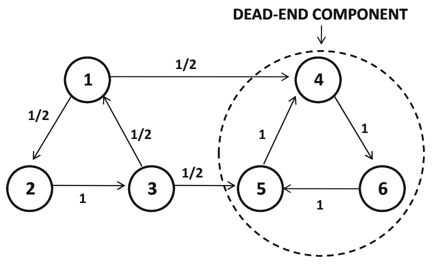
Dead-end components are common in the Web graph because the Web is not strongly connected. In such cases, the transitions at individual nodes can be meaningfully defined, but the steady-state transitions will stay trapped in these dead-end components. All the steady-state probabilities will be concentrated in dead-end components because there can be no transition out of a dead-end component after a transition occurs into it. Therefore, as long as even a minuscule probability of transition into a dead-end component[1] exists, *all* the steady-state probability becomes concentrated in such components. This situation is not desirable from the perspective of *PageRank* computation in a large Web graph, where dead-end components are not necessarily an indicator of popularity. Furthermore, in such cases, the final probability distribution of nodes in various dead-end components is not unique and it is dependent on the starting state. This is easy to verify by observing that random walks starting in different dead-end components will have their respective steady-state distributions concentrated within the corresponding components.

While the addition of edges solves the problem for dead-end nodes, an additional step is required to address the more complex issue of dead-end components. Therefore, aside from the addition of these edges, a *teleportation*, or *restart step* is used within the random surfer model. This step is defined as follows. At each transition, the random surfer may either jump to an arbitrary page with probability α, or it may follow one of the links on the page with probability $(1 - \alpha)$. A typical value of α used is 0.1. Because of the use of teleportation, the

[1] A formal mathematical treatment characterizes this in terms of the *ergodicity* of the underlying Markov chains. In ergodic Markov chains, a necessary requirement is that it is possible to reach any state from any other state using a sequence of one or more transitions. This condition is referred to as *strong connectivity.* An informal description is provided here to facilitate understanding.



(a) Dead-end node



(b) Dead-end component

Figure 18.2: Transition probabilities for *PageRank* computation with different types of dead ends

steady state probability becomes unique and independent of the starting state. The value of α may also be viewed as a *smoothing* or *damping probability*. Large values of α typically result in the steady-state probability of different pages to become more even. For example, if the value of α is chosen to be 1, then all pages will have the same steady-state probability of visits.

How are the steady-state probabilities determined? Let $G = (N, A)$ be the directed Web graph, in which nodes correspond to pages, and edges correspond to hyperlinks. The total number of nodes is denoted by n. It is assumed that A also includes the added edges from dead-end nodes to all other nodes. The set of nodes incident on i is denoted by $In(i)$, and the set of end points of the outgoing links of node i is denoted by $Out(i)$. The steady-state probability at a node i is denoted by $\pi(i)$. In general, the transitions of a Web surfer can be visualized as a *Markov chain*, in which an $n \times n$ transition matrix P is defined for a Web graph with n nodes. The *PageRank* of a node i is equal to the steady-state probability $\pi(i)$ for node i, in the Markov chain model. The probability[2] p_{ij} of transitioning from node i to node j, is defined as $1/|Out(i)|$. Examples of transition probabilities are illustrated in Fig. 18.2. These transition probabilities do not, however, account for teleportation which will be addressed[3] separately below.

Let us examine the transitions into a given node i. The steady-state probability $\pi(i)$ of node i is the sum of the probability of a teleportation into it and the probability that one of the in-linking nodes directly transitions into it. The probability of a teleportation into the node is exactly α/n because a teleportation occurs in a step with probability α, and all nodes are equally likely to be the beneficiary of the teleportation. The probability of a transition into node i is given by $(1-\alpha) \cdot \sum_{j \in In(i)} \pi(j) \cdot p_{ji}$, as the sum of the probabilities of transitions from different in-linking nodes. Therefore, at steady-state, the probability of

[2]In some applications such as bibliographic networks, the edge (i, j) may have a weight denoted by w_{ij}. The transition probability p_{ij} is defined in such cases by $\frac{w_{ij}}{\sum_{j \in Out(i)} w_{ij}}$.

[3]An alternative way to achieve this goal is to modify G by multiplying existing edge transition probabilities by the factor $(1-\alpha)$ and then adding α/n to the transition probability between each pair of nodes in G. As a result G will become a directed clique with bidirectional edges between each pair of nodes. Such strongly connected Markov chains have unique steady-state probabilities. The resulting graph can then be treated as a Markov chain without having to separately account for the teleportation component. This model is equivalent to that discussed in the chapter.

a transition into node i is defined by the sum of the probabilities of the teleportation and transition events are as follows:

$$\pi(i) = \alpha/n + (1-\alpha) \cdot \sum_{j \in In(i)} \pi(j) \cdot p_{ji}. \tag{18.3}$$

For example, the equation for node 2 in Fig. 18.2a can be written as follows:

$$\pi(2) = \alpha/4 + (1-\alpha) \cdot (\pi(1) + \pi(2)/4 + \pi(3)/3 + \pi(4)/2).$$

There will be one such equation for each node, and therefore it is convenient to write the entire system of equations in matrix form. Let $\overline{\pi} = (\pi(1) \ldots \pi(n))^T$ be the n-dimensional column vector representing the steady-state probabilities of all the nodes, and let $\overline{e}$ be an n-dimensional column vector of all 1 values. The system of equations can be rewritten in matrix form as follows:

$$\overline{\pi} = \alpha\overline{e}/n + (1-\alpha)P^T\overline{\pi}. \tag{18.4}$$

The first term on the right-hand side corresponds to a teleportation, and the second term corresponds to a direct transition from an incoming node. In addition, because the vector $\overline{\pi}$ represents a probability, the sum of its components $\sum_{i=1}^{n} \pi(i)$ must be equal to 1:

$$\sum_{i=1}^{n} \pi(i) = 1. \tag{18.5}$$

Note that this is a linear system of equations that can be easily solved using an iterative method. The algorithm starts off by initializing $\overline{\pi}^{(0)} = \overline{e}/n$, and it derives $\overline{\pi}^{(t+1)}$ from $\overline{\pi}^{(t)}$ by repeating the following iterative step:

$$\overline{\pi}^{(t+1)} \Leftarrow \alpha\overline{e}/n + (1-\alpha)P^T\overline{\pi}^{(t)}. \tag{18.6}$$

After each iteration, the entries of $\overline{\pi}^{(t+1)}$ are normalized by scaling them to sum to 1. These steps are repeated until the difference between $\overline{\pi}^{(t+1)}$ and $\overline{\pi}^{(t)}$ is a vector with magnitude less than a user-defined threshold. This approach is also referred to as the *power-iteration method*. It is important to understand that *PageRank* computation is expensive, and it cannot be computed on the fly for a user query during Web search. Rather, the *PageRank* values for *all* the known Web pages are precomputed and stored away. The stored *PageRank* value for a page is accessed only when the page is included in the search results for a particular query for use in the final ranking, as indicated by Eq. 18.2.

The *PageRank* values can be shown to be the n components of the largest left eigenvector[4] of the stochastic transition matrix P (see Exercise 5), for which the eigenvalue is 1. The largest eigenvalue of a stochastic transition matrix is always 1. The left eigenvectors of P are the same as the right eigenvectors of P^T. Interestingly, the largest *right* eigenvectors of the stochastic transition matrix P of an undirected graph can be used to construct *spectral embeddings* (cf. Sect. 19.3.4 of Chap. 19), which are used for network clustering.

[4]The left eigenvector $\overline{X}$ of P is a row vector satisfying $\overline{X}P = \lambda\overline{X}$. The right eigenvector $\overline{Y}$ is a column vector satisfying $P\overline{Y} = \lambda\overline{Y}$. For asymmetric matrices, the left and right eigenvectors are not the same. However, the eigenvalues are always the same. The unqualified term "eigenvector" refers to the right eigenvector by default.

18.4.1.1 Topic-Sensitive PageRank

Topic-sensitive PageRank is designed for cases in which it is desired to provide greater importance to some topics than others in the ranking process. While personalization is less common in large-scale commercial search engines, it is more common in smaller scale site-specific search applications. Typically, users may be more interested in certain combinations of topics than others. The knowledge of such interests may be available to a personalized search engine because of user registration. For example, a particular user may be more interested in the topic of automobiles. Therefore, it is desirable to rank pages related to automobiles higher when responding to queries by this user. This can also be viewed as the *personalization* of ranking values. How can this be achieved?

The first step is to fix a list of base topics, and determine a high-quality sample of pages from each of these topics. This can be achieved with the use of a resource such as the *Open Directory Project (ODP)*,[5] which can provide a base list of topics and sample Web pages for each topic. The *PageRank* equations are now modified, so that the teleportation is only performed on this sample set of Web documents, rather than on the entire space of Web documents. Let $\overline{e_p}$ be an n-dimensional personalization (column) vector with one entry for each page. An entry in $\overline{e_p}$ takes on the value of 1, if that page is included in the sample set, and 0 otherwise. Let the number of nonzero entries in $\overline{e_p}$ be denoted by n_p. Then, the *PageRank* Eq. 18.4 can be modified as follows:

$$\overline{\pi} = \alpha \overline{e_p}/n_p + (1 - \alpha) P^T \overline{\pi}. \tag{18.7}$$

The same power-iteration method can be used to solve the personalized *PageRank* problem. The selective teleportations bias the random walk, so that pages in the structural locality of the sampled pages will be ranked higher. As long as the sample of pages is a good representative of different (structural) localities of the Web graph, in which pages of specific topics exist, such an approach will work well. Therefore, for each of the different topics, a separate *PageRank* vector can be precomputed and stored for use during query time.

In some cases, the user is interested in specific *combinations of* topics such as sports and automobiles. Clearly, the number of possible combinations of interests can be very large, and it is not reasonably possible or necessary to prestore every personalized *PageRank* vector. In such cases, only the *PageRank* vectors for the base topics are computed. The final result for a user is defined as a weighted linear combination of the topic-specific *PageRank* vectors, where the weights are defined by the user-specified interest in the different topics.

18.4.1.2 SimRank

The notion of *SimRank* was defined to compute the structural similarity between nodes. *SimRank* determines *symmetric* similarities between nodes. In other words, the similarity between nodes i and j, is the same as that between j and i. Before discussing *SimRank*, we define a related but slightly different asymmetric ranking problem:

Given a target node i_q and a subset of nodes $S \subseteq N$ from graph $G = (N, A)$, rank the nodes in S in their order of similarity to i_q.

Such a query is very useful in recommender systems in which users and items are arranged in the form of a bipartite graph of preferences, in which nodes corresponds to users and items, and edges correspond to preferences. The node i_q may correspond to an item node,

[5] `http://www.dmoz.org`.

and the set S may correspond to user nodes. Alternatively, the node i_q may correspond to a user node, and the set S may correspond to item nodes. Recommender systems will be discussed in Sect. 18.5. Recommender systems are closely related to search, in that they also perform ranking of target objects, but while taking user preferences into account.

This problem can be viewed as a limiting case of topic-sensitive *PageRank*, in which the teleportation is performed to the *single node* i_q. Therefore, the personalized *PageRank* Eq. 18.7 can be directly adapted by using the teleportation vector $\overline{e_p} = \overline{e_q}$, that is, a vector of all 0s, except for a single 1, corresponding to the node i_q. Furthermore, the value of n_p in this case is set to 1:

$$\overline{\pi} = \alpha \overline{e_q} + (1 - \alpha) P^T \overline{\pi}. \tag{18.8}$$

The solution to the aforementioned equation will provide high ranking values to nodes in the structural locality of i_q. This definition of similarity is *asymmetric* because the similarity value assigned to node j starting from query node i is different from the similarity value assigned to node i starting from query node j. Such an *asymmetric* similarity measure is suitable for *query-centered* applications such as search engines and recommender systems, but not necessarily for arbitrary network-based data mining applications. In some applications, symmetric pairwise similarity between nodes is required. While it is possible to average the two topic-sensitive *PageRank* values in opposite directions to create a symmetric measure, the *SimRank* method provides an elegant and intuitive solution.

The *SimRank* approach is as follows. Let $In(i)$ represent the in-linking nodes of i. The *SimRank* equation is naturally defined in a recursive way, as follows:

$$\mathit{SimRank}(i, j) = \frac{C}{|In(i)| \cdot |In(j)|} \sum_{p \in In(i)} \sum_{q \in In(j)} \mathit{SimRank}(p, q). \tag{18.9}$$

Here C is a constant in $(0, 1)$ that can be viewed as a kind of decay rate of the recursion. As the boundary condition, the value of $\mathit{SimRank}(i, j)$ is set to 1 when $i = j$. When either i or j do not have in-linking nodes, the value of $\mathit{SimRank}(i, j)$ is set to 0. To compute *SimRank*, an iterative approach is used. The value of $\mathit{SimRank}(i, j)$ is initialized to 1 if $i = j$, and 0 otherwise. The algorithm subsequently updates the *SimRank* values between all node pairs iteratively using Eq. 18.9 until convergence is reached.

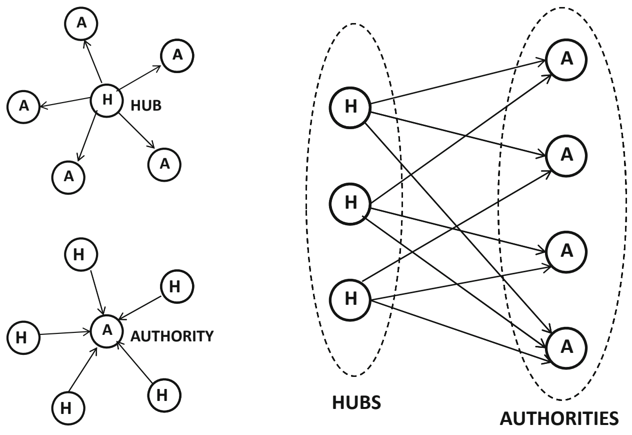
The notion of *SimRank* has an interesting intuitive interpretation in terms of random walks. Consider two random surfers walking *in lockstep* backward from node i and node j till they meet. Then the number of steps taken by each of them is a random variable $L(i, j)$. Then, $\mathit{SimRank}(i, j)$ can be shown to be equal to the expected value of $C^{L(i,j)}$. The decay constant C is used to map random walks of length l to a similarity value of C^l. Note that because $C < 1$, smaller distances will lead to higher similarity and vice versa.

Random walk-based methods are generally more robust than the shortest path distance to measure similarity between nodes. This is because random walks measures implicitly account for the *number* of paths between nodes, whereas shortest paths do not. A detailed discussion of this issue can be found in Sect. 3.5.1.2 of Chap. 3.

18.4.2 HITS

The *Hypertext Induced Topic Search (HITS)* algorithm is a *query-dependent* algorithm for ranking pages. The intuition behind the approach lies in an understanding of the typical structure of the Web that is organized into hubs and authorities.

An *authority* is a page with many in-links. Typically, it contains authoritative content on a particular subject, and, therefore, many Web users may trust that page as a resource of



(a) Hub and authority examples (b) Network organization between hubs and authorities

Figure 18.3: Illustrating hubs and authorities

knowledge on that subject. This will result in many pages linking to the authority page. A *hub* is a page with many out-links to authorities. These represent a compilation of the links on a particular topic. Thus, a hub page provides guidance to Web users about where they can find the resources on a particular topic. Examples of the typical node-centric topology of hubs and authorities in the Web graph are illustrated in Fig. 18.3a.

The main insight used by the *HITS* algorithm is that good hubs point to many good authorities. Conversely, good authority pages are pointed to by many hubs. An example of the typical organization of hubs and authorities is illustrated in Fig. 18.3b. This mutually reinforcing relationship is leveraged by the *HITS* algorithm. For any query issued by the user, the *HITS* algorithm starts with the list of relevant pages and expands them with a *hub ranking* and an *authority ranking*.

The *HITS* algorithm starts by collecting the top-r most relevant results to the search query at hand. A typical value of r is 200. This defines the *root set* R. Typically, a query to a commercial search engine or content-based evaluation is used to determine the root set. For each node in R, the algorithm determines all nodes immediately connected (either in-linking or out-linking) to R. This provides a larger *base set* S. Because the base set S can be rather large, the maximum number of in-linking nodes to any node in R that are added to S is restricted to k. A typical value of k used is around 50. Note that this still results in a rather large base set because *each* of the possibly 200 root nodes might bring 50 in-linking nodes, along with out-linking nodes.

Let $G = (S, A)$ be the subgraph of the Web graph defined on the (expanded) base set S, where A is the set of edges between nodes in the root set S. The entire analysis of the *HITS* algorithm is restricted to this subgraph. Each page (node) $i \in S$ is assigned both a *hub score* $h(i)$ and *authority score* $a(i)$. It is assumed that the hub and authority scores are normalized, so that the sum of the squares of the hub scores and the sum of the squares of the authority scores are each equal to 1. Higher values of the score indicate better quality. The hub and authority scores are related to one another in the following way:

$$h(i) = \sum_{j:(i,j)\in A} a(j) \quad \forall i \in S \tag{18.10}$$

$$a(i) = \sum_{j:(j,i)\in A} h(j) \quad \forall i \in S. \tag{18.11}$$

The basic idea is to reward hubs for pointing to good authorities and reward authorities for being pointed to by good hubs. It is easy to see that the aforementioned system of equations reinforces this mutually enhancing relationship. This is a linear system of equations that can be solved using an iterative method. The algorithm starts by initializing $h^0(i) = a^0(i) = 1/\sqrt{|S|}$. Let $h^t(i)$ and $a^t(i)$ denote the hub and authority scores of the ith node, respectively, at the end of the tth iteration. For each $t \geq 0$, the algorithm executes the following iterative steps in the $(t+1)$th iteration:

```
**for** each $i \in S$ set $a^{t+1}(i) \Leftarrow \sum_{j:(j,i)\in A} h^t(j)$;
**for** each $i \in S$ set $h^{t+1}(i) \Leftarrow \sum_{j:(i,j)\in A} a^{t+1}(j)$;
Normalize $L_2$-norm of each of hub and authority vectors to 1;
```

For hub-vector $\overline{h} = [h(1) \dots h(n)]^T$ and authority-vector $\overline{a} = [a(1) \dots a(n)]^T$, the updates can be expressed as $\overline{a} = A^T \overline{h}$ and $\overline{h} = A\overline{a}$, respectively, when the edge set A is treated as an $|S| \times |S|$ adjacency matrix. The iteration is repeated to convergence. It can be shown that the hub vector $\overline{h}$ and the authority vector $\overline{a}$ converge in directions proportional to the dominant eigenvectors of AA^T and $A^T A$ (see Exercise 6), respectively. This is because the relevant pair of updates can be shown to be equivalent to power-iteration updates of AA^T and $A^T A$, respectively.

18.5 Recommender Systems

Ever since the popularization of web-based transactions, it has become increasingly easy to collect data about user buying behaviors. This data includes information about user profiles, interests, browsing behavior, buying behavior, and ratings about various items. It is natural to leverage such data to make recommendations to customers about possible buying interests.

In the recommendation problem, the user–item pairs have *utility values* associated with them. Thus, for n users and d items, this results in an $n \times d$ matrix D of utility values. This is also referred to as the *utility-matrix.* The utility value for a user-item pair could correspond to either the buying behavior or the ratings of the user for the item. Typically, a small subset of the utility values are specified in the form of either customer buying behavior or ratings. It is desirable to use these specified values to make recommendations. The nature of the utility matrix has a significant influence on the choice of recommendation algorithm:

1. *Positive preferences only:* In this case, the specified utility matrix only contains positive preferences. For example, a specification of a "like" option on a social networking site, the browsing of an item at an online site, or the buying of a specified quantity of an item, corresponds to a positive preference. Thus, the utility matrix is sparse, with a prespecified set of positive preferences. For example, the utility matrix may contain the raw quantities of the item bought by each user, a normalized mathematical function of the quantities, or a weighted function of buying and browsing behavior. These

functions are typically specified heuristically by the analyst in an application-specific way. Entries that correspond to items not bought or browsed by the user may remain unspecified.

2. *Positive and negative preferences (ratings):* In this case, the user specifies the ratings that represent their like or dislike for the item. The incorporation of user dislike in the analysis is significant because it makes the problem more complex and often requires some changes to the underlying algorithms.

An example of a ratings-based utility matrix is illustrated in Fig. 18.4a, and an example of a positive-preference utility matrix is illustrated in Fig. 18.4b. In this case, there are six users, labeled $U_1 \ldots U_6$, and six movies with specified titles. Higher ratings indicate more positive feedback in Fig. 18.4a. The missing entries correspond to unspecified preferences in both cases. This difference significantly changes the algorithms used in the two cases. In particular, the two matrices in Fig. 18.4 have the same specified entries, but they provide very different insights. For example, the users U_1 and U_3 are very different in Fig. 18.4a because they have very different ratings for their commonly specified entries. On the other hand, these users would be considered very similar in Fig. 18.4b because these users have expressed a positive preference for the same items. The ratings-based utility provides a way for users to express negative preferences for items. For example, user U_1 does not like the movie *Gladiator* in Fig. 18.4a. There is no mechanism to specify this in the positive-preference utility matrix of Fig. 18.4b beyond a relatively ambiguous missing entry. In other words, the matrix in Fig. 18.4b is less expressive. While Fig. 18.4b provides an example of a binary matrix, it is possible for the nonzero entries to be arbitrary positive values. For example, they could correspond to the quantities of items bought by the different users.

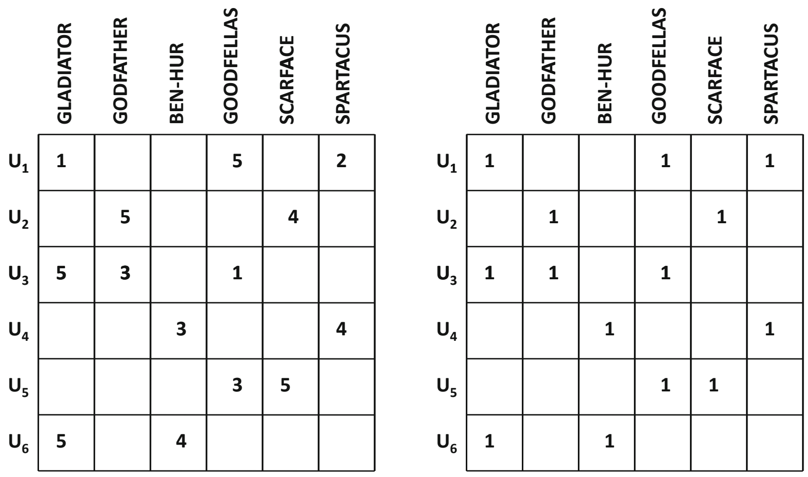
This difference has an impact on the types of algorithms that are used in the two cases. Allowing for positive and negative preferences generally makes the problem harder. From a data collection point of view, it is also harder to infer negative preferences when they are inferred from customer behavior rather than ratings. Recommendations can also be enhanced with the use of content in the user and item representations.

1. *Content-based recommendations:* In this case, the users and items are both associated with feature-based descriptions. For example, item profiles can be determined by using the text of the item description. A user might also have explicitly specified their interests in a profile. Alternatively, their profile can be inferred from their buying or browsing behavior.

2. *Collaborative filtering:* Collaborative filtering, as the name implies, is the leveraging of the user preferences in the form of ratings or buying behavior in a "collaborative" way, for the benefit of all users. Specifically, the utility matrix is used to determine either relevant users for specific items, or relevant items for specific users in the recommendation process. A key intermediate step in this approach is the determination of similar groups of items and users. The patterns in these peer groups provide the collaborative knowledge needed in the recommendation process.

The two models are not exclusive. It is often possible to combine content-based methods with collaborative filtering methods to create a combined preference score. Collaborative filtering methods are generally among the more commonly used models and will therefore be discussed in greater detail in this section.

It is important to understand that the utility matrices used in collaborative filtering algorithms are extremely large and sparse. It is not uncommon for the values of n and d in



	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			5		2
U_2		5			4	
U_3	5	3		1		
U_4			3			4
U_5				3	5	
U_6	5		4			

(a) Ratings-based utility

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			1		1
U_2		1			1	
U_3	1	1		1		
U_4			1			1
U_5				1	1	
U_6	1		1			

(b) Positive-preference utility

Figure 18.4: Examples of utility matrices.

the $n \times d$ utility matrix to exceed 10^5. The matrix is also extremely *sparse*. For example, in a movie data set, a typical user may have specified no more than 10 ratings, out of a universe of more than 10^5 movies.

At a basic level, collaborative filtering can be viewed as a missing-value estimation or matrix completion problem, in which an incomplete $n \times d$ utility matrix is specified, and it is desired to estimate the missing values. As discussed in the bibliographic notes, many methods exist in the traditional statistics literature on missing-value estimation. However, collaborative filtering problems present a particularly challenging special case in terms of data size and sparsity.

18.5.1 Content-Based Recommendations

In content-based recommendations, the user is associated with a set of documents that describe his or her interests. Multiple documents may be associated with a user corresponding to his or her specified demographic profile, specified interests at registration time, the product description of the items bought, and so on. These documents can then be aggregated into a single textual content-based profile of the user in a vector space representation.

The items are also associated with textual descriptions. When the textual descriptions of the items match the user profile, this can be viewed as an indicator of similarity. When no utility matrix is available, the content-based recommendation method uses a simple k-nearest neighbor approach. The top-k items are found that are closest to the user textual profile. The cosine similarity with tf-idf can be used, as discussed in Chap. 13.

On the other hand, when a utility matrix is available, the problem of finding the most relevant items for a particular user can be viewed as a traditional classification problem. For each user, we have a set of *training* documents representing the descriptions of the items for which that user has specified utilities. The labels represent the utility values. The descriptions of the remaining items for that user can be viewed as the test documents for classification. When the utility matrix contains numeric ratings, the class variables are numeric. The regression methods discussed in Sect. 11.5 of Chap. 11 may be used in this case. Logistic and ordered probit regression are particularly popular. In cases where only positive preferences (rather than ratings) are available in the utility matrix, all the specified utility entries correspond to positive examples for the item. The classification is then performed only on the remaining test documents. One challenge is that only a small number of positive training examples are specified, and the remaining examples are unlabeled. In such cases, specialized classification methods using only positive and unlabeled methods may be used. Refer to the bibliographic notes of Chap. 11. Content-based methods have the advantage that they do not even require a utility matrix and leverage domain-specific content information. On the other hand, content information biases the recommendation towards items described by similar keywords to what the user has seen in the past. Collaborative filtering methods work directly with the utility matrix, and can therefore avoid such biases.

18.5.2 Neighborhood-Based Methods for Collaborative Filtering

The basic idea in neighborhood-based methods is to use either user–user similarity, or item–item similarity to make recommendations from a ratings matrix.

18.5.2.1 User-Based Similarity with Ratings

In this case, the top-k similar users to each user are determined with the use of a similarity function. Thus, for the target user i, its similarity to all the other users is computed. Therefore, a similarity function needs to be defined between users. In the case of a ratings-based matrix, the similarity computation is tricky because different users may have different scales of ratings. One user may be biased towards liking most items, and another user may be biased toward not liking most of the items. Furthermore, different users may have rated different items. One measure that captures the similarity between the rating vectors of two users is the Pearson correlation coefficient. Let $\overline{X} = (x_1 \dots x_s)$ and $\overline{Y} = (y_1 \dots y_s)$ be the common (specified) ratings between a pair of users, with means $\hat{x} = \sum_{i=1}^{s} x_i/s$ and $\hat{y} = \sum_{i=1}^{s} y_i/s$, respectively. Alternatively, the mean rating of a user is computed by averaging over all her specified ratings rather than using only co-rated items by the pair of users at hand. This alternative way of computing the mean is more common, and it can significantly affect the pairwise Pearson computation. Then, the Pearson correlation coefficient between the two users is defined as follows:

$$\text{Pearson}(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{s} (x_i - \hat{x}) \cdot (y_i - \hat{y})}{\sqrt{\sum_{i=1}^{s} (x_i - \hat{x})^2} \cdot \sqrt{\sum_{i=1}^{s} (y_i - \hat{y})^2}}. \tag{18.12}$$

The Pearson coefficient is computed between the target user and all the other users. The peer group of the target user is defined as the top-k users with the highest Pearson coefficient of correlation with her. Users with very low or negative correlations are also removed from the peer group. The average ratings of each of the (specified) items of this peer group are returned as the recommended ratings. To achieve greater robustness, it is also possible to weight each rating with the Pearson correlation coefficient of its owner while computing the average. This weighted average rating can provide a prediction for the target user. The items with the highest predicted ratings are recommended to the user.

The main problem with this approach is that different users may provide ratings on different scales. One user may rate all items highly, whereas another user may rate all items negatively. The raw ratings, therefore, need to be normalized before determining the (weighted) average rating of the peer group. The normalized rating of a user is defined by subtracting her mean rating from each of her ratings. As before, the weighted average of the normalized rating of an item in the peer group is determined as a *normalized* prediction. The mean rating of the target user is then added back to the normalized rating prediction to provide a *raw* rating prediction.

18.5.2.2 Item-Based Similarity with Ratings

The main conceptual difference from the user-based approach is that peer groups are constructed in terms of *items* rather than *users*. Therefore, similarities need to be computed between items (or columns in the ratings matrix). Before computing the similarities between the columns, the ratings matrix is normalized. As in the case of user-based ratings, the average of each row in the ratings matrix is subtracted from that row. Then, the cosine similarity between the normalized ratings $\overline{U} = (u_1 \ldots u_s)$ and $\overline{V} = (v_1 \ldots v_s)$ of a pair of items (columns) defines the similarity between them:

$$\text{Cosine}(\overline{U}, \overline{V}) = \frac{\sum_{i=1}^{s} u_i \cdot v_i}{\sqrt{\sum_{i=1}^{s} u_i^2} \cdot \sqrt{\sum_{i=1}^{s} v_i^2}}. \tag{18.13}$$

This similarity is referred to as the *adjusted* cosine similarity, because the ratings are normalized before computing the similarity value.

Consider the case in which the rating of item j for user i needs to be determined. The first step is to determine the top-k most similar *items* to *item* j based on the aforementioned adjusted cosine similarity. Among the top-k matching items to item j, the ones for which user i has specified ratings are determined. The *weighted* average value of these (raw) ratings is reported as the predicted value. The weight of item r in this average is equal to the adjusted cosine similarity between item r and the target item j.

The basic idea is to leverage the user's *own* ratings in the final step of making the prediction. For example, in a movie recommendation system, the item peer group will typically be movies of a similar genre. The previous ratings history of the *same* user on such movies is a very reliable predictor of the interests of that user.

18.5.3 Graph-Based Methods

It is possible to use a random walk on the user-item graph, rather than the Pearson correlation coefficient, for defining neighborhoods. Such an approach is sometimes more effective for sparse ratings matrices. A bipartite *user-item* graph $G = (N_u \cup N_i, A)$ is constructed, where N_u is the set of nodes representing users, and N_i is the set of nodes representing items. An undirected edge exists in A between a user and an item for each nonzero entry in the utility matrix. For example, the user-item graph for both utility matrices of Fig. 18.4 is illustrated in Fig. 18.5. One can use either the personalized *PageRank* or the *SimRank* method to determine the k most similar users to a given user for user-based collaborative filtering. Similarly, one can use this method to determine the k most similar items to a given item for item-based collaborative filtering. The other steps of user-based collaborative filtering and item-based collaborative filtering remain the same.

A more general approach is to view the problem as a positive and negative *link prediction problem* on the user-item graph. In such cases, the user-item graph is augmented with positive or negative weights on edges. The normalized rating of a user for an item, after subtracting the user-mean, can be viewed as either a positive or negative weight on the edge. For example, consider the graph constructed from the ratings matrix of Fig. 18.4(a). The edge between user U_1 and the item *Gladiator* would become a negative edge because

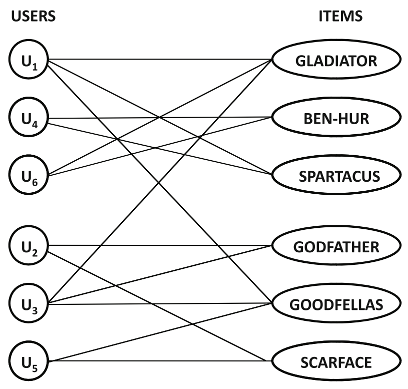


Figure 18.5: Preference graph for utility matrices of Fig. 18.4

U_1 clearly dislikes the movie *Gladiator*. The corresponding network would become a *signed* network. Therefore, the recommendation problem is that of predicting high positive weight edges between users and items in a signed network. A simpler version of the link-prediction problem with only positive links is discussed in Sect. 19.5 of Chap. 19. Refer to the bibliographic notes for link prediction methods with positive and negative links. The merit of the link prediction approach is that it can also leverage the available links between different users in a setting where they are connected by social network links. In such cases, the user-item graph no longer remains bipartite.

When users specify only positive preference values for items, the problem becomes simplified because most link prediction methods are designed for positive links. One can also use the random walks on the user-item graph to perform recommendations, rather than using it only to define neighborhoods. For example, in the case of Fig. 18.4b, the same user-item graph of Fig. 18.5 can be used in conjunction with a random-walk approach. This preference graph can be used to provide different types of recommendations:

1. The top ranking items for the user i can be determined by returning the item nodes with the largest *PageRank* in a random walk with restart at node i.

2. The top ranking users for the item j can be determined by returning the user nodes with the largest *PageRank* in a random walk with restart at node j.

The choice of the restart probability regulates the trade-off between the global popularity of the recommended item/user and the specificity of the recommendation to a particular user/item. For example, consider the case when items need to be recommended to user i. A low teleportation probability will favor the recommendation of popular items which are favored by many users. Increasing the teleportation probability will make the recommendation more specific to user i.

18.5.4 Clustering Methods

One weakness of neighborhood-based methods is the scale of the computation that needs to be performed. For *each user*, one typically has to perform computations that are proportional to *at least* the number of nonzero entries in the matrix. Furthermore, these computations need to be performed over *all* users to provide recommendations to different users.

This can be extremely slow. Therefore, a question arises, as to whether one can use clustering methods to speed up the computations. Clustering also helps address the issue of data sparsity to some extent.

Clustering methods are *exactly analogous* to neighborhood-based methods, except that the clustering is performed as a preprocessing step to define the peer groups. These peer groups are then used for making recommendations. The clusters can be defined either on users, or on items. Thus, they can be used to make either user-user similarity recommendations, or item-item similarity recommendations. For brevity, only the user-user recommendation approach is described here, although the item-item recommendation approach is exactly analogous. The clustering approach works as follows:

1. Cluster all the users into n_g groups of users using any clustering algorithm.

2. For any user i, compute the average (normalized) rating of the specified items in its cluster. Report these ratings for user i; after transforming back to the raw value.

The item–item recommendation approach is similar, except that the clustering is applied to the columns rather than the rows. The clusters define the groups of similar items (or implicitly pseudo-genres). The final step of computing the rating for a user-item combination is similar to the case of neighborhood-based methods. After the clustering has been performed, it is generally very efficient to determine all the ratings. It remains to be explained how the clustering is performed.

18.5.4.1 Adapting k-Means Clustering

To cluster the ratings matrix, it is possible to adapt many of the clustering methods discussed in Chap. 6. However, it is important to adapt these methods to sparsely specified incomplete data sets. Methods such as k-means and Expectation Maximization may be used on the normalized ratings matrix. In the case of the k-means method, there are two major differences from the description of Chap. 6:

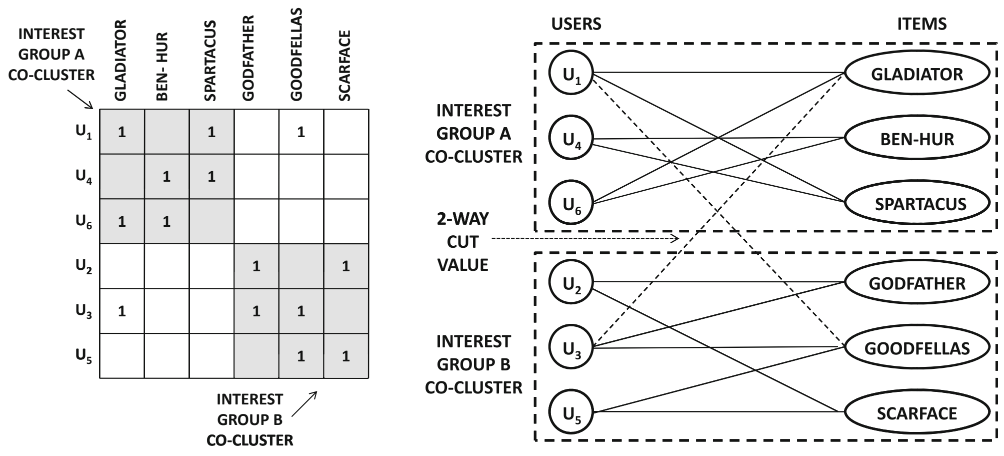
1. In an iteration of k-means, centroids are computed by averaging each dimension over the number of specified values in the cluster members. Furthermore, the centroid itself may not be fully specified.

2. The distance between a data point and a centroid is computed only over the specified dimensions in both. Furthermore, the distance is divided by the number of such dimensions in order to fairly compare different data points.

The ratings matrix should be normalized before applying the clustering method.

18.5.4.2 Adapting Co-Clustering

The co-clustering approach is described in Sect. 13.3.3.1 of Chap. 13. Co-clustering is well suited to discovery of neighborhood sets of users and items in sparse matrices. The specified entries are treated as 1s and the unspecified entries are treated as 0s for co-clustering. An example of the co-clustering approach, as applied to the utility matrix of Fig. 18.4b, is illustrated in Fig. 18.6a. In this case, only a 2-way co-clustering is shown for simplicity. The co-clustering approach cleanly partitions the users and items into groups with a clear correspondence to each other. Therefore, user-neighborhoods and item-neighborhoods are discovered simultaneously. After the neighborhoods have been defined, the aforementioned



(a) Co-cluster (b) User-item graph

Figure 18.6: Co-clustering of user-item graph

user-based methods and item-based methods can be used to make predictions for the missing entries.

The co-clustering approach also has a nice interpretation in terms of the user-item graph. Let $G = (N_u \cup N_i, A)$ denote the preference graph, where N_u is the set of nodes representing users, and N_i is the set of nodes representing items. An undirected edge exists in A for each nonzero entry of the utility matrix. Then the co-cluster is a clustering of this graph structure. The corresponding 2-way graph partition is illustrated in Fig. 18.6b. Because of this interpretation in terms of user-item graphs, the approach is able to exploit item-item and user-user similarity simultaneously. Co-clustering methods are also closely related to latent factor models such as nonnegative matrix factorization that simultaneously cluster rows and columns with the use of latent factors.

18.5.5 Latent Factor Models

The clustering methods discussed in the previous section use the aggregate properties of the data to make robust predictions. This can be achieved in a more robust way with latent factor models. This approach can be used either for ratings matrices or for positive preference utility matrices. Latent factor models have increasingly become more popular in recent years. The key idea behind latent factor models is that many dimensionality reduction and matrix factorization methods summarize the correlations across rows and columns in the form of lower dimensional vectors, or *latent factors*. Furthermore, collaborative filtering is essentially a missing data imputation problem, in which these correlations are used to make predictions. Therefore, these latent factors become hidden variables that encode the correlations in the data matrix in a concise way and can be used to make predictions. A robust estimation of the k-dimensional *dominant* latent factors is often possible even from incompletely specified data, when the value of k is much less than d. This is because the more concisely defined latent factors can be estimated accurately with the sparsely specified data matrix, as long as the number of specified entries is large enough.

The n users are represented in terms of n corresponding k-dimensional factors, denoted by the vectors $\overline{U_1} \dots \overline{U_n}$. The d items are represented by d corresponding k-dimensional factors, denoted by the vectors $\overline{I_1} \dots \overline{I_d}$. The value of k represents the reduced dimensionality of the latent representation. Then, the rating r_{ij} for user i and item j is estimated by the vector dot product of the corresponding latent factors:

$$r_{ij} \approx \overline{U_i} \cdot \overline{I_j}. \tag{18.14}$$

If this relationship is true for every entry of the ratings matrix, then it implies that the entire ratings matrix $D = [r_{ij}]_{n \times d}$ can be factorized into two matrices as follows:

$$D \approx F_{user} F_{item}^T. \tag{18.15}$$

Here F_{user} is an $n \times k$ matrix, in which the ith row represent the latent factor $\overline{U_i}$ for user i. Similarly, F_{item} is an $d \times k$ matrix, in which the jth row represents the latent factor $\overline{I_j}$ for item j. How can these factors be determined? The two key methods to use for computing these factors are singular value decomposition, and matrix factorization, which will be discussed in the sections below.

18.5.5.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is discussed in detail in Sect. 2.4.3.2 of Chap. 2. The reader is advised to revisit that section before proceeding further. Equation 2.12 of Chap. 2 approximately factorizes the data matrix D into three matrices, and is replicated here:

$$D \approx Q_k \Sigma_k P_k^T. \tag{18.16}$$

Here, Q_k is an $n \times k$ matrix, Σ_k is a $k \times k$ diagonal matrix, and P_k is a $d \times k$ matrix. The main difference from the 2-way factorization format is the diagonal matrix Σ_k. However, this matrix can be included within the user factors. Therefore, one obtains the following factor matrices:

$$F_{user} = Q_k \Sigma_k \tag{18.17}$$
$$F_{item} = P_k. \tag{18.18}$$

The discussion in Chap. 2 shows that the matrix $Q_k \Sigma_k$ defines the reduced and transformed coordinates of data points in SVD. Thus, each user has a new set of a k-dimensional coordinates in a new k-dimensional basis system P_k defined by linear combinations of items. Strictly speaking, SVD is undefined for incomplete matrices, although heuristic approximations are possible. The bibliographic notes provide pointers to methods that are designed to address this issue. Another disadvantage of SVD is its high computational complexity. For nonnegative ratings matrices, $PLSA$ may be used, because it provides a probabilistic factorization similar to SVD.

18.5.5.2 Matrix Factorization

SVD is a form of matrix factorization. Because there are many different forms of matrix factorization, it is natural to explore whether they can be used for recommendations. The reader is advised to read Sect. 6.8 of Chap. 6 for a review of matrix factorization. Equation 6.30 of that section is replicated here:

$$D \approx U \cdot V^T. \tag{18.19}$$

This factorization is already directly in the form we want. Therefore, the user and item factor matrices are defined as follows:

$$F_{user} = U \tag{18.20}$$
$$F_{item} = V. \tag{18.21}$$

The main difference from the analysis of Sect. 6.8 is in how the optimization objective function is set up for incomplete matrices. Recall that the matrices U and V are determined by optimizing the following objective function:

$$J = ||D - U \cdot V^T||^2. \tag{18.22}$$

Here, $|| \cdot ||$ represents the Frobenius norm. In this case, because the ratings matrix D is only partially specified, the optimization is performed only over the *specified entries*, rather than all the entries. Therefore, the basic form of the optimization problem remains very similar, and it is easy to use any off-the-shelf optimization solver to determine U and V. The bibliographic notes contain pointers to relevant stochastic gradient descent methods. A regularization term $\lambda(||U||^2 + ||V||^2)$ containing the squared Frobenius norms of U and V may be added to J to reduce overfitting. The regularization term is particularly important when the number of specified entries is small. The value of the parameter λ is determined using cross-validation.

This method is more convenient than *SVD* for determining the factorized matrices because the optimization objective can be set up in a seamless way for an incompletely specified matrix no matter how sparse it might be. When the ratings are nonnegative, it is also possible to use nonnegative forms of matrix factorization. As discussed in Sect. 6.8, the nonnegative version of matrix factorization provides a number of interpretability advantages. Other forms of factorization, such as probabilistic matrix factorization and maximum margin matrix factorization, are also used. Most of these variants are different in terms of minor variations in the objective function (e.g., Frobenius norm minimization, or maximum likelihood maximization) and the constraints (e.g., nonnegativity) of the underlying optimization problem. These differences often translate to variants of the same stochastic gradient descent approach.

18.6 Web Usage Mining

The usage of the Web leads to a significant amount of *log* data. There are two primary types of logs that are commonly collected:

1. *Web server logs:* These correspond to the user activity on Web servers. Typically logs are stored in standardized format, known as the *NCSA common log format*, to facilitate ease of use and analysis by different programs. A few variants of this format, such as the *NCSA combined log format*, and *extended log format*, store a few extra fields. Nevertheless, the number of variants of the basic format is relatively small. An example of a Web log entry is as follows:

```
98.206.207.157 - - [31/Jul/2013:18:09:38 -0700] "GET /productA.pdf
HTTP/1.1" 200 328177 "-" "Mozilla/5.0 (Mac OS X) AppleWebKit/536.26
(KHTML, like Gecko) Version/6.0 Mobile/10B329 Safari/8536.25"
"retailer.net"
```

2. *Query logs:* These correspond to the queries posed by a user in a search engine. Aside from the commercial search engine providers, such logs may also be available to Web site owners if the site contains search features.

These types of logs can be used with a wide variety of applications. For example, the browsing behavior of users can be extracted to make recommendations. The area of Web usage mining is too large to be covered by a section of a single chapter. Therefore, the goal of this section is to provide an overview of how the various techniques discussed in this book can be mapped to Web usage mining. The bibliographic notes contain pointers to more detailed Web mining books on this topic. One major issue with Web log applications is that logs contain data that is not cleanly separated between different users and is therefore difficult to directly use in arbitrary application settings. In other words, significant preprocessing is required.

18.6.1 Data Preprocessing

A log file is often available as a continuous sequence of entries that corresponds to the user accesses. The entries for different users are typically interleaved with one another randomly, and it is also difficult to distinguish different sessions of the same user.

Typically, client-side cookies are used to distinguish between different user sessions. However, client-side cookies are often disabled due to privacy concerns at the client end. In such cases, only the IP address is available. It is hard to distinguish between different users on the basis of IP addresses only. Other fields, such as user agents and referrers, are often used to further distinguish. In many cases, at least a subset of the users can be identified to a reasonable level of granularity. Therefore, only the subset of the logs, where the users can be identified, is used. This is often sufficient for application-specific scenarios. The bibliographic notes contain pointers to preprocessing methods for Web logs.

The preprocessing leads to a set of *sequences* in the form of page views, which are also referred to as *click streams*. In some cases, the graph of traversal patterns, as it relates to the link structure of the pages at the site, is also constructed. For query logs, similar sequences are obtained in the form of search tokens, rather than page views. Therefore, in spite of the difference in the application scenario, there is some similarity in the nature of the data that is collected. In the following, some key applications of Web log mining will be visited briefly.

18.6.2 Applications

Click-stream data lead to a number of applications of sequence data mining. In the following, a brief overview of the various applications will be provided, along with the pointers to the relevant chapters. The bibliographic notes also contain more specific pointers.

Recommendations

Users can be recommended Web pages on the basis of their browsing patterns. In this case, it is not even necessary to use the sequence information; rather, a user-pageview matrix can be constructed from the previous browsing behavior. This can be leveraged to infer the user interest in the different pages. The corresponding matrix is typically a positive preference utility matrix. Any of the recommendation algorithms in this chapter can be used to infer the pages, in which the user is most likely to be interested.

Frequent Traversal Patterns

The frequent traversal patterns at a site provide an overview of the most likely patterns of user traversals at a site. The frequent sequence mining algorithms of Chap. 15 as well as the frequent graph pattern mining algorithms of Chap. 17 may be used to determine the paths that are most popular. The Web site owner can use these results for Web site reorganization. For example, paths that are very popular should stay as continuous paths in the Web site graph. Rarely used paths and links may be reorganized, if needed. Links may be added between pairs of pages if a sequential pattern is frequently observed between that pair.

Forecasting and Anomaly Detection

The Markovian models in Chap. 15 may be used to forecast future clicks of the user. Significant deviation of these clicks from expected values may correspond to anomalies. A second kind of anomaly occurs when an entire pattern of accesses is unusual. These types of scenarios are different from the case, where a particular page view in the sequence is considered anomalous. Hidden Markov models may be used to discover such anomalous sequences. The reader is referred to Chap. 15 for a discussion of these methods.

Classification

In some cases, the sequences from a Web log may be labeled on the basis of desirable or undesirable activity. An example of a desirable activity is when a user buys a certain product after browsing a certain sequence of pages at a site. An undesirable sequence may be indicative of an intrusion attack. When labels are available, it may be possible to perform early classification of Web log sequences. The results can be used to make *online* inferences about the future behavior of Web users.

18.7 Summary

Web data is of two types. The first type of data corresponds to the documents and links available on the Web. The second type of data corresponds to patterns of user behavior such as buying behavior, ratings, and Web logs. Each of these types of data can be leveraged for different insights.

Collecting document data from the Web is often a daunting task that is typically achieved with the use of *crawlers*, or *spiders*. Crawlers may be either universal crawlers that are used by commercial search engines, or they may be preferential crawlers, in which only topics of a particular subject are collected. After the documents are collected, they are stored and indexed in search engines. Search engines use a combination of textual similarity and reputation-based ranking to create a final score. The two most common algorithms used for ranking in search engines are the *PageRank* and *HITS* algorithms. Topic-sensitive *PageRank* is often used to compute similarity between nodes.

A significant amount of data is collected on the Web, corresponding to user-item preferences. This data can be used for making recommendations. Recommendation methods can be either content-based or user preference-based. Preference-based methods include neighborhood-based techniques, clustering techniques, graph-based techniques, and latent factor-based techniques.

Web logs are another important source of data on the Web. Web logs typically result in either sequence data or graphs of traversal patterns. If the sequential portion of the data is ignored, then the logs can also be used for making recommendations. Typical applications of Web log analysis include determining frequent traversal patterns and anomalies, and identifying interesting events.

18.8 Bibliographic Notes

Two excellent resources for Web mining are the books in [127, 357]. An early description of Web search engines, starting from the crawling to the searching phase, is provided by the founders of the Google search engine [114]. The general principles of crawling may be found in [127]. There is significant work on preferential crawlers as well [127, 357]. Numerous aspects of search engine indexing and querying are described in [377].

The *PageRank* algorithm is described in [114, 412]. The *HITS* algorithm was described in [317]. A detailed description of different variations of the *PageRank* and *HITS* algorithms may be found in [127, 343, 357, 377]. The topic-sensitive *PageRank* algorithm is described in [258], and the *SimRank* algorithm is described in [289].

Recommender systems are described well in Web and data mining books [343, 357]. In addition, general background on the topic is available in journal survey articles and special issues [2, 325]. The problem of collaborative filtering can be considered a version of the missing data imputation problem. A vast literature exists on missing data analysis [364]. Item-based collaborative filtering algorithms are discussed in [170, 445]. Graph-based methods for recommendations are discussed in [210, 277, 528]. Methods for link-prediction in signed networks are discussed in [341]. The origin of latent factor models is generally credited to a number of successful entries in the Netflix prize contest [558]. However, the use of latent factor models for estimating missing entries precedes the work in the field of recommendation analysis and the Netflix prize contest by several years [23]. This work [23] shows how *SVD* may be used for approximating missing data entries by combining it with the EM algorithm. Furthermore, the works in [272, 288, 548], which were performed earlier than the Netflix prize contest, show how different forms of matrix factorization may be used for recommendations. After the *popularization* of this approach by the Netflix prize contest, other factorization-based methods were also proposed for collaborative filtering [321, 322, 323]. Related matrix factorization models may be found in [288, 440, 456]. Latent semantic models can be viewed as probabilistic versions of latent factor models, and are discussed in [272].

Web usage mining has been described well in [357]. Both Web log mining and usage mining are described in this work. A description of methods for Web log preparation may be found in [161, 477]. Methods for anomaly detection with Web logs are discussed in [5]. Surveys on Web usage mining appear in [65, 390, 425].

18.9 Exercises

1. Implement a universal crawler with the use of a breadth-first algorithm.

2. Consider the string $ababcdef$. List all 2-shingles and 3-shingles, using each alphabet as a token.

3. Discuss why it is good to add anchor text to the Web page it points to for mining purposes, but it is often misleading for the page in which it appears.

4. Perform a Google search on "*mining text data*" and "*text data mining.*" Do you get the same top-10 search results? What does this tell you about the content component of the ranking heuristic used by search engines?

5. Show that the *PageRank* computation with teleportation is an eigenvector computation on an appropriately constructed probability transition matrix.

6. Show that the hub and authority scores in *HITS* can be computed by dominant eigenvector computations on AA^T and A^TA respectively. Here, A is the adjacency matrix of the graph $G = (S, A)$, as defined in the chapter.

7. Show that the largest eigenvalue of a stochastic transition matrix is always 1.

8. Suppose that you are told that a particular transition matrix P can be diagonalized as $P = V\Lambda V^{-1}$, where Λ is diagonal. How can you use this result to efficiently determine the k-hop transition matrix which defines the probability of a transition between each pair of nodes in k hops? What would you do for the special case when $k = \infty$? Does the result hold if we allow the entries of P and V to be complex numbers?

9. Apply the *PageRank* algorithm to the graph of Fig. 18.2b, using teleportation probabilities of 0.1, 0.2, and 0.4, respectively. What is the impact on the dead-end component (probabilities) of increasing the teleportation probabilities?

10. Repeat the previous exercise, except that the restart is performed from node 1. How are steady-state probabilities affected by increasing the teleportation probability?

11. Show that the transition matrix of the graph of Fig. 18.4.1b will have more than one eigenvector with an eigenvalue of 1. Why is the eigenvector with unit eigenvalue not unique in this case?

12. Implement the neighborhood-based approach for collaborative filtering on a ratings matrix.

13. Implement the personalized *PageRank* approach for collaborative filtering on a positive-preference utility matrix.

14. Apply the *PageRank* algorithm to the example of Fig. 18.5 by setting restart probabilities to 0.1, 0.2, and 0.4, respectively.

15. Apply the personalized *PageRank* algorithm to the example of Fig. 18.5 by restarting at node *Gladiator*, and with restart probabilities of 0.1, 0.2, and 0.4, respectively. What does this tell you about the most relevant users for the movie *Gladiator* What does this tell you about the most relevant user for the movie "*Gladiator*," who has not already watched this movie? Is it possible for the most relevant user to change with teleportation probability? What is the intuitive significance of the teleportation probability from an application-specific perspective?

16. Construct the optimization formulation for the matrix factorization problem for incomplete matrices.

17. In the bipartite graph of Fig. 18.5, what is the *SimRank* value between a user node and an item node? In this light, explain the weakness of the *SimRank* model.