

Chapter 2

Data Structures and Mathematical Algorithms

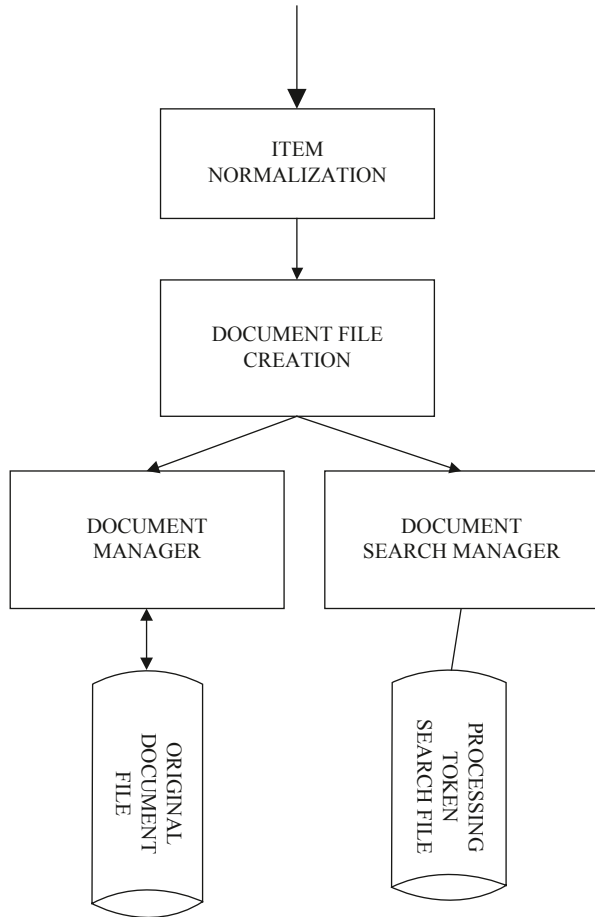
2.1 Data Structures

2.1.1 Introduction to Data Structures

There are usually two major data structures in any information system. One structure stores and manages the received items in their normalized form and is the version that is displayed to the user. The process supporting this structure is called the “document manager.” The other major data structure contains the processing tokens and associated data (e.g., index) to support search. Figure 2.1 shows the document file creation process which is a combination of the ingest and indexing processes. The results of a search are references to the items that satisfy the search statement, which are passed to the document manager for retrieval. This chapter focuses on data structures used to support the search function. It does not address the document management function nor the data structures and other related theory associated with the parsing of queries.

The Ingest and Indexing processes are described in Chaps. 3 and 4, but some of the lower level data structures to support the indices are described in this chapter. The most common data structure encountered in both data base and information systems is the inverted file system (discussed in Sect. 2.1.2). It minimizes secondary storage access when multiple search terms are applied across the total database. All commercial and most academic systems use inversion as the searchable data structure. A variant of the searchable data structure is the N-gram structure that breaks processing tokens into smaller string units (which is why it is sometimes discussed under stemming) and uses the token fragments for search. N-grams have demonstrated improved efficiencies and conceptual manipulations over full word inversion. PAT trees and arrays view the text of an item as a single long stream versus a juxtaposition of words. Around this paradigm search algorithms are defined based upon text strings. Signature files are based upon the idea of fast elimination of non-relevant items reducing the searchable items to a manageable subset. The subset can be returned to the user for review or other search algorithms may be applied to it to eliminate any false hits that passed the signature filter.

Fig. 2.1 Major data structures



The XML data structure is the most common structure used in sharing information between systems and frequently how it is stored within a system. It is how items are received by the Ingest process and it is typically used if items are exported to other applications and systems. Given the commonality of XML there has been TREC conference experiments on how to optimize search systems whose data structure is XML.

The hypertext data structure is the basis behind URL references on the internet. But more importantly is the logical expansion of the definition of an item when hypertext references are used and its potential impact on searches. The latest Internet search systems have started to make use of hypertext links to expand what information is indexed associated with items. Most commonly it is used when indexing multimedia objects but there is a natural extension to textual items.

There are some mathematical notions that are frequently used in information retrieval systems. Bayesian mathematics has a variety of uses in information re-

trieval. Another important concept comes from Communications systems and Information Theory based upon the work of Claude Shannon and is the basis behind most of the commonly used weighting algorithms. Hidden Markov models are used in both searching and also are a technical base behind multimedia information item processing. Latent Semantic Indexing is one of the few techniques that has been used commercially to create concept indices. Neural networks and Support Vector Machines are the most common learning algorithms used to automatically construct search structures from user examples used for example in Categorization.

2.1.2 Inverted File Structure

The most common data structure used in both database management and Information Retrieval Systems is the inverted file structure. Inverted file structures are composed of three basic files: the document file, the inversion lists (sometimes called posting files) and the dictionary. The name “inverted file” comes from its underlying methodology of storing an inversion of the documents: inversion of the documents from the perspective that instead of having a set of documents with words in them, you create a set of words that has the list of documents they are found in. Each document in the system is given a unique numerical identifier. It is that identifier that is stored in the inversion list. The way to locate the inversion list for a particular word is via the Dictionary. The Dictionary is typically a sorted list of all unique words (processing tokens) in the system and a pointer to the location of its inversion list (see Fig. 2.2). Dictionaries can also store other information used in query optimization such as the length of inversion lists. Additional information may be used from the item to increase precision and provide a more optimum inversion list file structure. For example, if zoning is used, the dictionary may be partitioned by zone. There could be a dictionary and set of inversion lists for the “Abstract” zone in an item and another dictionary and set of inversion lists for the “Main Body” zone. This increases the overhead when a user wants to search the complete item versus restricting the search to a specific zone. Another typical optimization occurs when the inversion list only contains one or two entries. Those entries can be stored as part of the dictionary. The inversion list contains the document identifier for each document in which the word is found. To support proximity, contiguous word

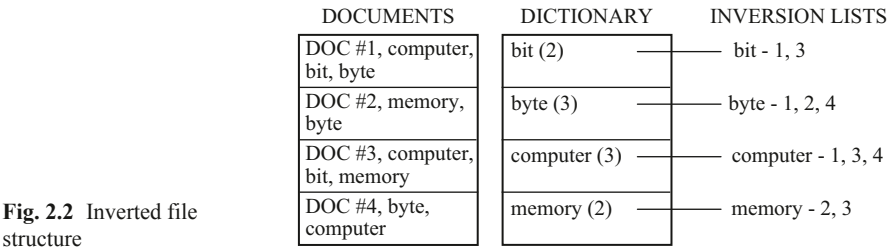


Fig. 2.2 Inverted file structure

phrases and term weighting algorithms, all occurrences of a word are stored in the inversion list along with the word position. Thus if the word “bit” was the tenth, twelfth and eighteenth word in document #1, then the inversion list would appear:

bit—1(10), 1(12), 1(18)

Weights can also be stored in inversion lists. Words with special characteristics are frequently stored in their own dictionaries to allow for optimum internal representation and manipulation (e.g., dates which require date ranging and numbers).

When a search is performed, the inversion lists for the terms in the query are located and the appropriate logic is applied between inversion lists. The result is a final hit list of items that satisfy the query. For systems that support ranking, the list is reorganized into ranked order. The document numbers are used to retrieve the documents from the Document File. Using the inversion lists in Fig. 2.2, the query (bit AND computer) would use the Dictionary to find the inversion lists for “bit” and “computer.” These two lists would be logically ANDed: (1,3) AND (1,3,4) resulting in the final Hit list containing (1,3).

Rather than using a dictionary to point to the inversion list, B-trees can be used. The inversion lists may be at the leaf level or referenced in higher level pointers. Fig. 2.3 shows how the words in Fig. 2.1 would appear. A B-tree of order m is defined as:

- A root node with between 2 and 2m keys
- All other internal nodes have between m and 2m keys
- All keys are kept in order from smaller to larger
- All leaves are at the same level or differ by at most one level.

Cutting and Pedersen described use of B-trees as an efficient inverted file storage mechanism for data that undergoes heavy updates (Cutting-90).

The nature of information systems is that items are seldom if ever modified once they are produced. Most commercial systems take advantage of this fact by allowing document files and their associated inversion lists to grow to a certain maximum size and then to freeze them, starting a new structure. Each of these databases of document file, dictionary, inversion lists is archived and made available for a user’s query. This has the advantage that for queries only interested in more recent information; only the latest databases need to be searched. Since older items are seldom

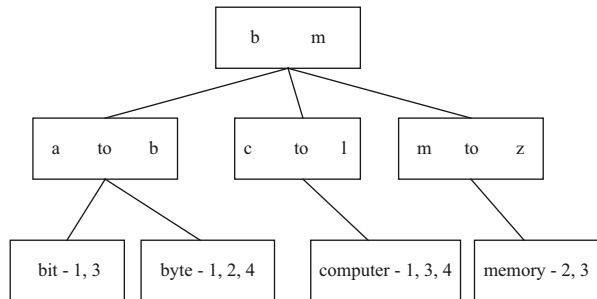


Fig. 2.3 B-tree inversion lists

deleted or modified, the archived databases may be permanently backed-up, thus saving on operations overhead. Starting a new inverted database has significant overhead in adding new words and inversion lists until the frequently found words are added to the dictionary and inversion lists. Previous knowledge of archived databases can be used to establish an existing dictionary and inversion structure at the start of a new database, thus saving significant overhead during the initial adding of new documents. Other more scalable inversion list techniques are discussed in Chap. 8.

Inversion lists structures are used because they provide optimum performance in searching large databases. The optimality comes from the minimization of data flow in resolving a query. Only data directly related to the query are retrieved from secondary storage. Also there are many techniques that can be used to optimize the resolution of the query based upon information maintained in the dictionary.

Inversion list file structures are well suited to store concepts and their relationships. Each inversion list can be thought of as representing a particular concept. Words are typically used to define an inversion list but in Chap. 3 when categorization and entities are discussed, the inversion lists can easily be extended to include those as additional index for an item. The individual word may not be representative of a concept but by use of a proximity search the user can combine words all within a proximity (e.g., in the same sentence) and thus get closer to a concept. The inversion list is then a concordance of all of the items that contain that concept. Finer resolution of concepts can additionally be maintained by storing locations with an item and weights of the item in the inversion lists. With this information, relationships between concepts can be determined as part of search algorithms. Location of concepts is made easy by their listing in the dictionary and inversion lists. For Natural Language Processing algorithms, other structures may be more appropriate or required in addition to inversion lists for maintaining the required semantic and syntactic information.

2.1.3 *N-Gram Data Structures*

N-Grams can be viewed as a special technique for conflation (stemming) and as a unique data structure in information systems. N-Grams are a fixed length consecutive series of “n” characters. Unlike stemming that generally tries to determine the stem of a word that represents the semantic meaning of the word, n-grams do not care about semantics. Instead they are algorithmically based upon a fixed number of characters. The searchable data structure is transformed into overlapping n-grams, which are then used to create the searchable database. Examples of bigrams, trigrams and pentagrams are given in Fig. 2.4 for the word phrase “sea colony.”

For n-grams, with n greater than two, some systems allow interword symbols to be part of the n-gram set usually excluding the single character with interword symbol option. The symbol # is used to represent the interword symbol which is anyone of a set of symbols (e.g., blank, period, semicolon, colon, etc.). Each of the n-grams

Fig. 2.4 Bigrams, trigrams and pentagrams for “sea colony”

se ea co ol lo on ny	Bigrams (no interword symbols)
sea col olo lon ony	Trigrams (no interword symbols)
#se sea ea# #co col olo lon ony ny#	Trigrams (with interword symbol #)
#sea# #colo colon olony lony#	Pentagrams (with interword symbol #)

created becomes separate processing tokens and are searchable. It is possible that the same n-gram can be created multiple times from a single word.

2.1.3.1 History

The first use of n-grams dates to World War II when it was used by cryptographers. Fletcher Pratt states that “with the backing of bigram and trigram tables any cryptographer can dismember a simple substitution cipher” (Pratt-42). Use of bigrams was described by Adamson as a method for conflating terms (Adamson-74). It does not follow the normal definition of stemming because what is produced by creating n-grams are word fragments versus semantically meaningful word stems. It is this characteristic of mapping longer words into shorter n-gram fragments that seems more appropriately classified as a data structure process than a stemming process.

Another major use of n-grams (in particular trigrams) is in spelling error detection and correction (Angell-83, McIlroy-82, Morris-75, Peterson-80, Thorelli-62, Wang-77, and Zamora-81). Most approaches look at the statistics on probability of occurrence of n-grams (trigrams in most approaches) in the English vocabulary and indicate any word that contains non-existent to seldom used n-grams as a potential erroneous word. Damerau specified four categories of spelling errors (Damerau-64) as shown in Fig. 2.5.

Using the classification scheme, Zamora showed trigram analysis provided a viable data structure for identifying misspellings and transposed characters. This impacts information systems as a possible basis for identifying potential input errors for correction as a procedure within the normalization process (see Chap. 1). Frequency of occurrence of n-gram patterns also can be used for identifying the language of an item (Damashek-95, Cohen-95).

Error Category	Example
Single Character Insertion	comp <u>u</u> uter
Single Character Deletion	comp <u>t</u> er
Single Character Substitution	comp <u>i</u> ter
Transposition of two adjacent characters	comp <u>ut</u> er

Fig. 2.5 Categories of spelling errors

In information retrieval, trigrams have been used for text compression and to manipulate the length of index terms (Schek-78, Schuegraf-76). Some implementations used a variety of different n-grams as index elements for inverted file systems. They have also been the core data structure to encode profiles for the Logicon LMDS system (Yochum-95) used for Selective Dissemination of Information. For retrospective search, the Acquaintance System uses n-grams to store the searchable document file (Damashek-95, Huffman-95) for retrospective search of large textual databases.

2.1.3.2 N-Gram Data Structure

As shown in Fig. 2.4, an n-gram is a data structure that ignores words and treats the input as a continuous data, optionally limiting its processing by interword symbols. The data structure consists of fixed length overlapping symbol segments that define the searchable processing tokens. These tokens have logical linkages to all the items in which the tokens are found. Inversion lists, document vectors (described in Chap. 4) and other proprietary data structures are used to store the linkage data structure and are used in the search process. In some cases just the least frequently occurring n-gram is kept as part of a first pass search process (Yochum-85).

The choice of the fixed length word fragment size has been studied in many contexts. Yochum investigated the impacts of different values for “n.” Other researchers investigated n-gram data structures using an inverted file system for $n = 2$ to $n = 26$. Trigrams (n-grams of length 3) were determined to be the optimal length, trading off information versus size of data structure. The Acquaintance System uses longer n-grams, ignoring word boundaries. The advantage of n-grams is that they place a finite limit on the number of searchable tokens.

$$\text{MaxSeg}_n = (\lambda)^n$$

The maximum number of unique n-grams that can be generated, MaxSeg, can be calculated as a function of n which is the length of the n-grams, and λ which is the number of processable symbols from the alphabet (i.e., non-interword symbols).

Although there is a savings in the number of unique processing tokens and implementation techniques allow for fast processing on minimally sized machines, false hits can occur under some architectures. For example, a system that uses trigrams and does not include interword symbols or the character position of the n-gram in an item finds an item containing “retain detail” when searching for “retail” (i.e., all of the trigrams associated with “retail” are created in the processing of “retain detail”). Inclusion of interword symbols would not have helped in this example. Inclusion of character position of the n-gram would have discovered that the n-grams “ret,” “eta,” “tai,” “ail” that define “retail” are not all consecutively starting within one character of each other. The longer the n-gram, the less likely this type error is to occur because of more information in the word fragment. But the longer the n-gram, the more it provides the same result as full word data structures since

most words are included within a single n-gram. Another disadvantage of n-grams is the increased size of inversion lists (or other data structures) that store the linkage data structure. In effect, use of n-grams expands the number of processing tokens by a significant factor. The average word in the English language is between six and seven characters in length. Use of trigrams increases the number of processing tokens by a factor of five if interword symbols are not included. Thus the inversion lists increase by a factor of five.

Because of the processing token bounds of n-gram data structures, optimized performance techniques can be applied in mapping items to an n-gram searchable structure and in query processing. There is no semantic meaning in a particular n-gram since it is a fragment of processing token and may not represent a concept. Thus n-grams are a poor representation of concepts and their relationships. But the juxtaposition of n-grams can be used to equate to standard word indexing, achieving the same levels of recall and within 85% precision levels with a significant improvement in performance (Adams-92). Vector representations of the n-grams from an item can be used to calculate the similarity between items. N-grams can be very useful when the items in the database are not typical textual items. For example a database of software programs would be far more searchable using n-grams as the tokenization data structure.

2.1.4 PAT Data Structure

Using n-grams with interword symbols included between valid processing tokens equates to a continuous text input data structure that is being indexed in contiguous “n” character tokens. A different view of addressing a continuous text input data structure comes from PAT trees and PAT arrays. The input stream is transformed into a searchable data structure consisting of substrings. The original concepts of PAT tree data structures were described as Patricia trees (Frakes-92) and have gained new momentum as a possible structure for searching text and images and applications in genetic databases. The name PAT is short for PATRICIA Trees (PATRICIA stands for Practical Algorithm To Retrieve Information Coded In Alphanumeric.)

In creation of PAT trees each position in the input string is the anchor point for a sub-string that starts at that point and includes all new text up to the end of the input. All substrings are unique. This view of text lends itself to many different search processing structures. It fits within the general architectures of hardware text search machines and parallel processors. A substring can start at any point in the text and can be uniquely indexed by its starting location and length. If all strings are to the end of the input, only the starting location is needed since the length is the difference from the location and the total length of the item. It is possible to have a substring go beyond the length of the input stream by adding additional null characters. These substrings are called sistring (semi-infinite string). Figure 2.6 shows some possible sistrings for an input text.

Fig. 2.6 Examples of sistrings

Text	Economics for Warsaw is complex.
sistring 1	Economics for Warsaw is complex.
sistring 2	conomics for Warsaw is complex.
sistring 5	omics for Warsaw is complex.
sistring 10	for Warsaw is complex.
sistring 20	w is complex.
sistring 30	ex.

A PAT tree is an unbalanced, binary digital tree defined by the sistrings. The individual bits of the sistrings decide the branching patterns with zeros branching left and ones branching right. PAT trees also allow each node in the tree to specify which bit is used to determine the branching via bit position or the number of bits to skip from the parent node. This is useful in skipping over levels that do not require branching.

The key values are stored at the leaf nodes (bottom nodes) in the PAT Tree. For a text input of size “n” there are “n” leaf nodes and “n – 1” at most higher level nodes. It is possible to place additional constraints on sistrings for the leaf nodes. We may be interested in limiting our searches to word boundaries. Thus we could limit our sistrings to those that are immediately after an interword symbol. Figure 2.7 gives an example of the sistrings used in generating a PAT tree. The example only goes down 9 levels. It shows the minimum binary prefixes that uniquely identify each row. If the binary representations of “h” is (100), “o” is (110), “m” is (001) and “e” is (101) then the word “home” produces the input 100110001101.... Using the sistrings, the full PAT binary tree is shown in Fig. 2.8. A more compact tree where skip (reduced PAT tree) values are in the intermediate nodes is shown in Fig. 2.9. In the compact tree, if only one branch of a tree is being extended by the sistrings, you can skip comparisons on those levels because the values are not optional (i.e., cannot be a 1 or a 0—but just one of those values) and thus there are not branches that you could take. The value in the intermediate nodes (indicated by rectangles) is the number of bits to skip until the next bit to compare that causes differences between similar terms. This final version saves space, but requires one additional comparison whenever you encounter a 1 and zero optional level to validate there were no errors in the positions that were jumped over. In the example provided it is at the leaf level but could occur at any level within the tree (in an oval). In the

INPUT	0110111101101110
sistring 1	01101111...
sistring 2	1101111...
sistring 3	10111....
sistring 4	0111.....
sistring 5	1111....
sistring 6	1110.....
sistring 7	110110...
sistring 8	10110...
sistring 9	011011110...

Fig. 2.7 Sistrings for input “0110111101101110”

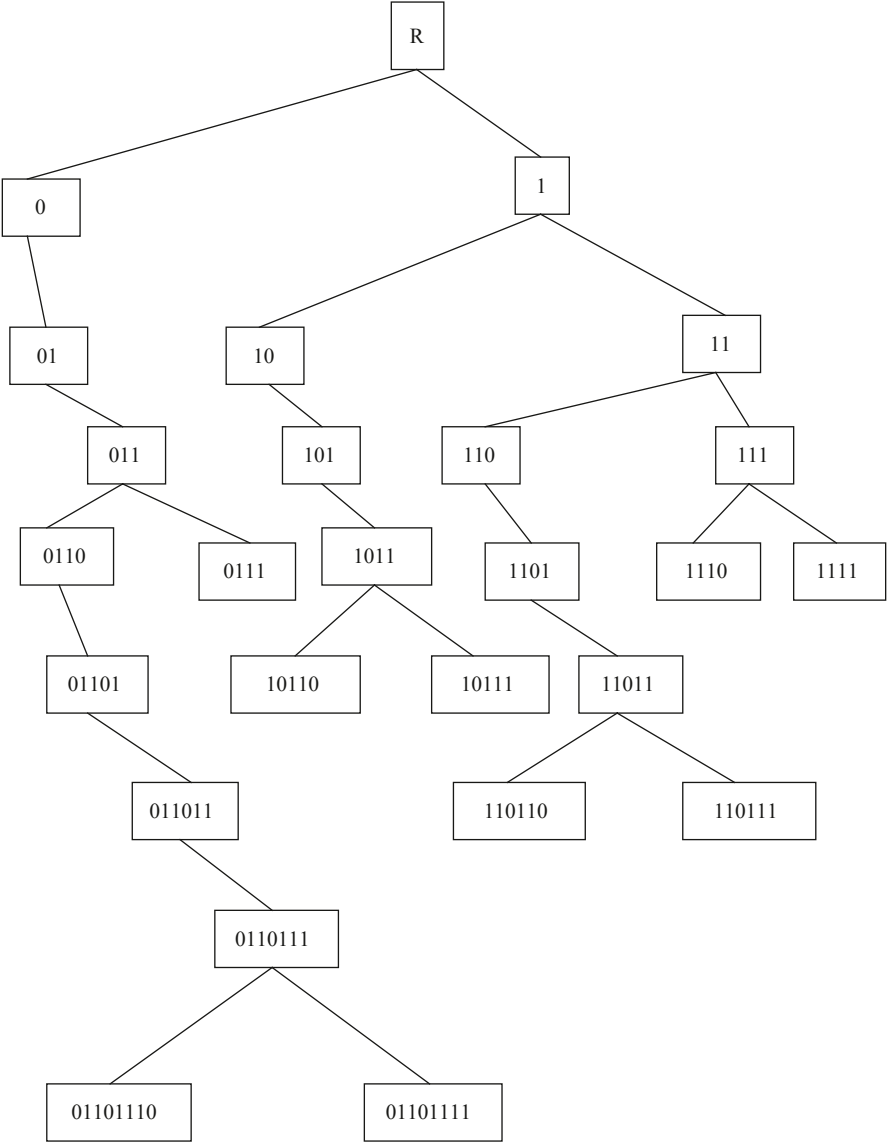


Fig. 2.8 PAT Binary tree for input “0110111101101110”

reduced PAT tree the node that has “111” in it could have alternatively been shown as a circle with a skip of 1 position.

To search, the search terms are also represented by their binary representation and the PAT trees for the sistrings are traveled down based upon the values in the search term to look for match(es).

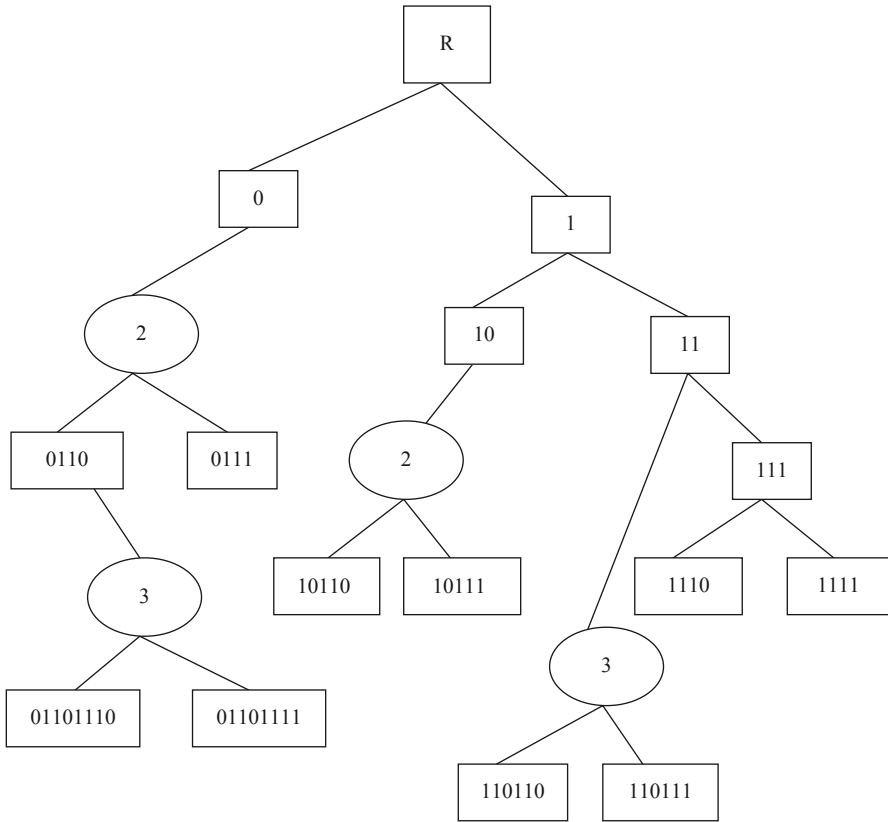


Fig. 2.9 Reduced PAT tree for “0110111101101110”

As noted in Chap. 1, one of the most common classes of searches is prefix searches. PAT trees are ideally constructed for this purpose because each sub-tree contains all the sistrings for the prefix defined up to that node in the tree structure. Thus all the leaf nodes after the prefix node define the sistrings that satisfy the prefix search criteria. This logically sorted order of PAT trees also facilitates range searches since it is easy to determine the sub-trees constrained by the range values. If the total input stream is used in defining the PAT tree, then suffix, imbedded string, and fixed length masked searches (see Sect. 2.1.5) are all easy because the given characters uniquely define the path from the root node to where the existence of sistrings need to be validated. Fuzzy searches are very difficult because large number of possible sub-trees could match the search term.

A detailed discussion on searching PAT trees and their representation as an array is provided by Gonnet, Baeza-Yates and Snider (Gonnet-92). In their comparison to Signature and Inversion files, they concluded that PAT arrays have more accuracy than Signature files and provide the ability to string searches that are inefficient in inverted files (e.g., suffix searches, approximate string searches, longest repetition).

Pat Trees (and arrays) provide an alternative structure if string searching is the goal. They store the text in an alternative structure supporting string manipulation. The structure does not have facilities to store more abstract concepts and their relationships associated with an item. The structure has interesting potential applications, and was the original structure used in the BrightPlanet (<http://www.brightplanet.com>) system that searches the deep web (discussed in Chap. 3). Additionally PAT trees have been used to index Chinese since they do not have word separators (see Chap. 3).

2.1.5 Signature File Structure

The goal of a signature file structure is to provide a fast test to eliminate the majority of items that are not related to a query. The items that satisfy the test can either be evaluated by another search algorithm to eliminate additional false hits or delivered to the user to review. The text of the items is represented in a highly compressed form that facilitates the fast test. Because file structure is highly compressed and unordered, it requires significantly less space than an inverted file structure and new items can be concatenated to the end of the structure versus the significant inversion list update. Since items are seldom deleted from information data bases, it is typical to leave deleted items in place and mark them as deleted. Signature file search is a linear scan of the compressed version of items producing a response time linear with respect to file size.

The surrogate signature search file is created via superimposed coding (Faloutsos-85). The coding is based upon words in the item. The words are mapped into a “word signature.” A word signature is a fixed length code with a fixed number of bits set to “1.” The bit positions that are set to one are determined via a hash function of the word. The word signatures are ORed together to create the signature of an item. To avoid signatures being too dense with “1”s, a maximum number of words is specified and an item is partitioned into blocks of that size. In Fig. 2.10 the block size is set at five words, the code length is 16 bits and the number of bits that are allowed to be “1” for each word is five.

TEXT: Computer Science graduate students study (assume block size is five words)		
<u>WORD</u>	<u>Signature</u>	
Computer	0001	0110 0000 0110
Science	1001	0000 1110 0000
graduate	1000	0101 0100 0010
students	0000	0111 1000 0100
study	0000	0110 0110 0100
Block Signature	1001	0111 1110 0110

Fig. 2.10 Superimposed coding

The words in a query are mapped to their signature. Search is accomplished by template matching on the bit positions specified by the words in the query.

The signature file can be stored as a signature with each row representing a signature block. Associated with each row is a pointer to the original text block. A design objective of a signature file system is trading off the size of the data structure versus the density of the final created signatures. Longer code lengths reduce the probability of collision in hashing the words (i.e., two different words hashing to the same value). Fewer bits per code reduce the effect of a code word pattern being in the final block signature even though the word is not in the item. For example, if the signature for the word “hard” is 1000 0111 0010 0000, it incorrectly matches the block signature in Fig. 2.10 (false hit). In a study by Faloutsos and Christodoulakis (Faloutsos-87) it was shown that if compression is applied to the final data structure, the optimum number of bits per word is one. This then takes on the appearance of a binary coded vector for each item, where each position in the vector represents the existence of a word in the item. This approach requires the maximum code length but ensures that there are not any false hits unless two words hash to the same value.

Search of the signature matrix requires $O(N)$ search time. To reduce the search time the signature matrix is partitioned horizontally. One of the earliest techniques hashes the block signature to a specific slot. If a query has less than the number of words in a block it maps to a number of possible slots rather than just one. The number of slots decreases exponentially as the number of terms increases (Gustafson-71). Another approach maps the signatures into an index sequential file, where, for example, the first “n” bits of the signature is used as the index to the block of signatures that will be compared sequentially to the query (Lee-89). Other techniques are two level signatures (Sacks-Davis-83) and use of B-tree structures with similar signatures clustered at leaf nodes (Deppisch-86).

Another implementation approach takes advantage of the fact that searches are performed on the columns of the signature matrix, ignoring those columns that are not indicated by hashing of any of the search terms. Thus the signature matrix may be stored in column order versus row order (Faloutsos-88, Lin-88, Roberts-79), called vertical partitioning. This is in effect storing the signature matrix using an inverted file structure. The major overhead comes from updates, since new “1”s have to be added to each inverted column representing a signature in the new item.

Signature files provide a practical solution for storing and locating information in a number of different situations. Faloutsos summarizes the environments that signature files have been applied as medium size databases, databases with low frequency of terms, WORM devices, parallel processing machines, and distributed environments (Faloutsos-92).

One of the first steps in ingesting items is to detect duplicate and near duplicate items (see Chap. 3). One way of representing the text in items is via signatures which could be used to detect near duplicates.

2.1.6 *Hypertext and XML Data Structures*

The advent of the Internet and its exponential growth and wide acceptance as a new global information network has introduced new mechanisms for representing information. This structure is called hypertext and differs from traditional information storage data structures in format and use. The hypertext is stored in Hypertext Markup Language (HTML) and eXtensible Markup Language (XML). HTML is an evolving standard as new requirements for display of items on the Internet are identified and implemented. Both of these languages provide detailed descriptions for subsets of text similar to the zoning discussed previously. These subsets can be used the same way zoning is used to increase search accuracy and improve display of hit results.

In addition to using the HTML or XML to define zones, it also can be used to identify metadata to be extracted and associated with that item. For example there could be a date field or a source field. HTML also contains display information such as “bolding”. That information is also useful to indicate the importance of a word used in ranking (ordering) the hits from a search. Over the past few years a new standard called XHTML has been introduced that merges the XML data description with the HTML presentation.

2.1.6.1 **Definition of Hypertext Structure**

The Hypertext data structure is used extensively in the Internet environment and requires electronic media storage for the item. Hypertext allows one item to reference another item via an imbedded pointer. Each separate item is called a node and the reference pointer is called a link. The referenced item can be of the same or a different data type than the original (e.g., a textual item references a photograph). Each node is displayed by a viewer that is defined for the file type associated with the node.

For example, Hypertext Markup Language (HTML) defines the internal structure for information exchange across the World Wide Web on the Internet. A document is composed of the text of the item along with HTML tags that describe how to display the document. Tags are formatting or structural keywords contained between less-than, greater than symbols (e.g., <title>, meaning display prominently). The HTML tag associated with hypertext linkages is where “a” and “/a” are an anchor start tag and anchor end tag denoting the text that the user can activate. “href” is the hypertext reference containing either a file name if the referenced item is on this node or an address (Uniform Resource Locator—URL) and a file name if it is on another node. “#NAME” defines a destination point other than the top of the item to go to. The URL has three components: the access method the client used to retrieve the item, the Internet address of the server where the item is stored, and the address of the item at the server (i.e., the file including the directory it is in). For example, the URL for the HTML specification appears:

<http://info.cern.ch/hypertext/WWW/MarkUp/HTML.html>

Fig. 2.11 Example of segment of HTML

```
<CENTER>
<IMG SRC="/images/home_iglo.jpg" WIDTH=468 HEIGHT=107
  BORDER=0 ALT="WELCOME TO NETSCAPE"><BR>
<P>
<DL>
<A HREF="/comprod/mirror/index.html">
<DD>
  The beta testing is over: please read our report <A
  HREF="http://www.charm.net/doc/charm/report/theme.html"> and
  your can find more references at
  HREF="http://www.charm.net/doc/charm/results/tests.html">
```

“HTTP” stands for the Hypertext Transfer Protocol which is the access protocol used to retrieve the item in HTML. Other Internet protocols are used for other activities such as file transfer (ftp://), remote logon (telnet://) and collaborative newsgroups (news://). The destination point is found in “info.cern.ch” which is the name of the “info” machine at CERN with “ch” being Switzerland, and “/hypertext/WWW/MarkUP/HTML.html” defines where to find the file HTML.html. Figure 2.11 shows an example of a segment of a HTML document. Most of the formatting tags indicated by < > are not described, being out of the scope of this text, but detailed descriptions can be found in the hundreds of books available on HTML. The are the previously described hypertext linkages.

An item can have many hypertext linkages. Thus, from any item there are multiple paths that can be followed in addition to skipping over the linkages to continue sequential reading of the item. This is similar to the decision a reader makes upon reaching a footnote, whether to continue reading or skip to the footnote. Hypertext is sometimes called a “generalized footnote.” But that can be misleading because quite often the link is to a major extension of the current item.

In a conventional item the physical and logical structure are closely related. The item is sequential with imbedded citations to other distinct items or locations in the item. From the author’s perspective, the substantive semantics lie in the sequential presentation of the information. Hypertext is a non-sequential directed graph structure, where each node contains its own information. The author assumes the reader can follow the linked data as easily as following the sequential presentation. A node may have several outgoing links, each of which is then associated with some smaller part of the node called an anchor. When an anchor is activated, the associated link is followed to the destination node, thus navigating the hypertext network. There is text that the reader sees that is associated with the anchor (anchor text). This takes on importance in Information retrieval because it is quite often used as index text for the anchor when it is pointing to a multimedia file versus just another textual page. The organizational and reference structure of a conventional item is fixed at printing time while hypertext nodes and links can be changed dynamically. New linkages can be added and the information at a node can change without modification to the item referencing it.

Conventional items are read sequentially by a user. In a hypertext environment, the user “navigates” through the node network by following links. This is the defining capability that allows hypertext to manage loosely structured information. Each

thread through different nodes could represent a different concept with additional detail. In a small and familiar network the navigation works well, but in a large information space, it is possible for the user to become disoriented.

Quite often hypertext references are used to include information that is other than text (e.g., graphics, audio, photograph, video) in a text item. During the ingest process described in Chap. 3, the system can easily identify different multimedia modalities to assist in directing those items to the appropriate ingest and indexing software. The multiple different uses for hypertext references are evolving as more experience is gained with them. When the hypertext is logically part of the item, such as in a graphic, the referenced file is usually resident at the same physical location. When other items created by other users are referenced, they frequently are located at other physical sites. When items are deleted or moved, there is no mechanism to update other items that reference them. Linkage integrity is a major issue in use of hypertext linkages.

Dynamic HTML became available with Navigator 4.0 and Internet Explorer 4.0. It is a collective term for a combination of the latest HTML tags and options, style sheets and programming that will let you create WEB pages that are more animated and responsive to user interaction. Some of the features supported are an object-oriented view of a WEB page and its elements, cascading style sheets, programming that can address most page elements add dynamic fonts. Object oriented views are defined by the Document Object Model—DOM (Micorsoft calls this the Dynamic HTML Object Model while Netscape calls it the HTML Object Model). For example every heading on a page can be named and given attributes of text style and color that can be manipulated by name in a small “program” or script included on the page. A style sheet describes the default style characteristics (page layout, font, text size, etc) of a document or portion of a document. Dynamic HTML allows the specification of style sheets in a cascading fashion (linking style sheets to predefined levels of precedence within the same set of pages). As a result of a user interaction, a new style sheet can be applied changing the appearance of the display. Layering is the use of alternative style sheets to vary the content of a page by providing content layers that overlay and superimpose existing content sections. The existing HTML programming capabilities are being expanded to address the additional data structures.

HTML prior to version 5 was based upon SGML (Standard Generalized Mark-up Language) and was a very simplified subset of it. With the increasing use of XML to define data structures it became sensible to define a new HTML structure that could work well with XML data and provide Internet displays of the XML data. This lead to XHTML (extensible hypertext mark-up language). Since it works with XML it also inherits the “well formed” structural constraints associated with XML. This makes the automated processing easier versus the more complex parsers needed for HTML based upon SGML. The other advantage is XHTML documents could include XML structures from other XML based languages. At this point changes from HTML to XHTML have been kept to a minimum primarily to adhere to the rules of XML. Since Internet Explorer has not accepted XHTML there remains major resistance to its general usage. In July 2009 W3C announced that they will stop work on expanding XHTML and focus on HTML 5 that combines HTML and XHTML.

2.1.6.2 Hypertext History

Although information sciences is just starting to address the impact of the hypertext data structure, the concept of hypertext has been around for over 50 years. In 1945 an article written by Vannevar Bush in 1933 was published describing the Memex (memory extender) system (Bush-67). It was a microfilm based system that would allow the user to store much of the information from the scientific explosion of the 1940s on microfilm and retrieve it at multiple readers at the user's desk via individual links. The term "hypertext" came from Ted Nelson in 1965 (Nelson-74). Nelson's vision of all the world's literature being interlinked via hypertext references is part of his Xanadu System. The lack of cost effective computers with sufficient speed and memory to implement hypertext effectively was one of the main inhibitors to its development. One of the first commercial uses of a hypertext system was the mainframe system, Hypertext Editing System, developed at Brown University by Andres van Dam and later sold to Houston Manned Spacecraft Center where it was used for Apollo mission documentation (van Dam-88). Other systems such as the Aspen system at MIT, the KMS system at Carnegie Mellon, the Hyperties system at the University of Maryland and the Notecards system developed at Xerox PARC advanced the hypertext concepts providing hypertext (and hypermedia) systems. HyperCard, delivered with Macintosh computers, was the first widespread hypertext production product. It had a simple metalanguage (HyperTalk) that facilitated authoring hypertext items. It also provided a large number of graphical user interface elements (e.g., buttons, hands,) that facilitated the production of sophisticated items.

Hypertext became more available in the early 1990s via its use in CD-ROMs for a variety of educational and entertainment products. Its current high level of popularity originated with it being part of the specification of the World Wide Web by the CERN (the European Center for Nuclear Physics Research) in Geneva, Switzerland. The Mosaic browser, freely available from CERN on the Internet, gave everyone who had access the ability to receive and display hypertext documents.

2.1.7 XML

The eXtensible Markup Language (XML) is also becoming a standard encoding structure for documents on the WEB and as a data exchange format for Web services applications (e.g., used for web services). Its first recommendation (1.0) was issued on February 10, 1998. It is a middle ground between the simplicities but lack of flexibility of HTML and the complexity but richness of SGML (ISO 8879). Its objective is extending HTML with semantic information and removing the display specification from the data specification. The logical data structure within XML is defined by a Data Type Description (DTD) and is not constrained to the 70 defined tags and 50 attributes in the single DTD for HTML. The original DTD did not allow for complex definition of data types within the data structure so it was

expanded to other ways of defining XML structures called schemas. The DTD is a very restricted version of an XML schema. Some of the other more common schemas are Schema W3C and RELAX NG. The user can create any tags needed to describe and manipulate their structure. The W3C (World Wide Web Consortium) is redeveloping HTML as a suite of XML tags. The following is a simple example of XML tagging:

```
<company>Widgets Inc.</company>
<city>Boston</city>
<state>Mass</state>
<product>widgets</product>
```

The W3C is also developing a Resource Description Format (RDF) for representing properties of WEB resources such as images, documents and relationships between them. This will include the Platform for Internet Content Selection (PICS) for attaching labels to material for filtering (e.g., unsuitable for children).

Hypertext links for XML were being defined in the Xlink (XML Linking Language) but work stopped in this area. Xpoint (XML Pointer language) specifications. This would allow for distinction for different types of links to locations within a document and external to the document. This would allow an application to know if a link is just a repositioning reference within an item or link to another document that is an extension of the existing document. This would help in determining what needs to be retrieved to define the total item to be indexed. But the standards committees could not get a critical mass following interested in implementing this concept.

Finally XML will include an XML Style Sheet Linking definition to define how to display items on a particular style sheet and handle cascading style sheets. This will allow designers to limit what is displayed to the user (saving on display screen space) and allow expansion to the whole item if desired. Cascading Style Sheets provide an easy way to dynamically manage the output display of XML to the user.

2.2 Mathematical Algorithms

2.2.1 Introduction

There are a number of mathematical concepts that form the basis behind a lot of the weighted indexing techniques used in creating the indices for information retrieval systems. The goal of this section is to provide a brief introduction to the important mathematical concepts. If the student wants to use the concepts in either research or applications they are developing then significant additional reading on the concepts is required. The two most important theories are the Bayesian theory and Shannon's Information theory. Bayesian models are a conditional model associated with probabilities that estimates the probability of one event

given another event takes place. This directly maps into the probability that a document is relevant given a specific query. It additionally can be used to define clustering relationships used in automatic creation of taxonomies associated with search results and item databases. Shannon's information model describes the "information value" given the frequency of occurrence of an event. In this case it can be related to how many items contain a particular word and how that affects its importance (if a word is found in every item in the database it does not have much search value).

Hidden Markov Models are the basis behind the transformation of audio into transcribed text that is one approach to indexing audio and video. In addition it is frequently used in the optical character processing of text in images to computer recognized text. It also has been proposed as a basis behind indexing and search for textual items. Latent semantic indexing is one of the best mathematical techniques to explain how a "concept" index is created and it has been used commercially to create concept indices. It is technique that allows for automatic mapping of millions of words used to create items into a small number (e.g. 300) concept vectors that represent the vocabulary of the language. The concept vectors are then like a meta-language used to express both the items and the queries.

In addition to the algorithms used in creating the index, there is a need in information retrieval for learning algorithms that allow the system to learn what is of interest to a user and then be able to use the dynamically created and updated algorithms to automatically analyze new items to see if they satisfy the existing criteria. This is used in techniques often labeled as "Categorization". The two primary techniques used for the learning algorithms are neural networks and support vector machines.

The goal of this section is to introduce the mathematical basis behind the algorithms used in information retrieval. To really understand the details on how the algorithms are used in information retrieval you should take courses in probability and machine learning.

2.2.2 *Bayesian Mathematics*

The earliest mathematical foundation for information retrieval dates back to the early 1700s when Thomas Bayes developed a theorem that relates the conditional and marginal probabilities of two random events—called Baye's Theorem. It can be used to compute the posterior probability (probability assigned "after" relevant evidence is considered) of random events. For example, it allows to consider the symptoms of a patient and use that information to determine the probability of what is causing the illness. Bayes' theorem relates the conditional and marginal probabilities of events A and B , where B cannot equal zero:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

$P(A)$ is called the prior or marginal probability of A . It is called “prior” because it does not take into account any information about B . $P(A|B)$ is the conditional probability of A , given B . It is sometimes named the posterior probability because the probability depends upon the probability of B . $P(B|A)$ is the conditional probability of B given A . $P(B)$ is the prior or marginal probability of B , and normalizes the result.

Putting the terms into words given our example helps in understanding the formula:

- The probability of a patient having the flu given the patient has a high temperature is equal to the probability that if you have a high temperature you have the flu times the probability you will have the flu. This is then normalized by dividing times the probability that you have a high temperature.

To relate Bayesian Theory to information retrieval you need only to consider the search process. A user provides a query, consisting of words, which represent the user’s preconceived attempt to describe the semantics needed in an item to be retrieved for it to be relevant. Since each user submits these terms to reflect their own idea of what is important, they imply a preference ordering (ranking) among all of the documents in the database. Applying this to Bayes’s Theorem you have:

$$P(\text{An item is relevant/Query}) = \frac{P(\text{Query/Relevant item}) P(\text{An item is relevant})}{P(\text{Query})}$$

The major issues with using this to determine which items are most relevant to the query are Bayes Theorem assumes independence (i.e., each term is independent of every other term), and how to get the probability for some of the terms in the above formula. These issues will be discussed in Chap. 4 on indexing.

A Bayesian network is a directed acyclic graph in which each node represents a random variable and the arcs between the nodes represent a probabilistic dependence between the node and its parents (Howard-81, Pearl-88). Figure 2.12 shows the basic weighting approach for index terms or associations between query terms and index terms.

The nodes C_1 and C_2 represent “the item contains concept C_i ” and the F nodes represent “the item has feature (e.g., words) F_{ij} .” The network could also be inter-

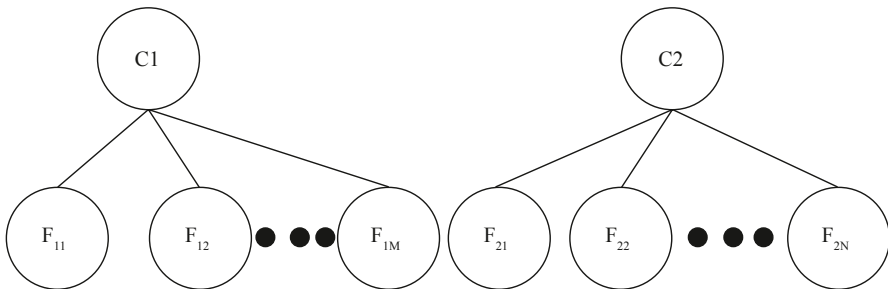


Fig. 2.12 Two-level Bayesian network

puted as C representing concepts in a query and F representing concepts in an item. The goal is to calculate the probability of C_i given F_{ij} . To perform that calculation two sets of probabilities are needed:

1. The prior probability $P(C_i)$ that an item is relevant to concept C
2. The conditional probability $P(F_{ij}/C_i)$ that the features F_{ij} where $j = 1, m$ are present in an item given that the item contains topic C_i .

The automatic indexing task is to calculate the posterior probability $P(C_i/F_{i1}, \dots, F_{im})$, the probability that the item contains concept C_i given the presence of features F_{ij} . The Bayes inference formula that is used is:

$$P(C_i/F_{i1}, \dots, F_{im}) = P(C_i) P(F_{i1}, \dots, F_{im}/C_i) / P(F_{i1}, \dots, F_{im}).$$

If the goal is to provide ranking as the result of a search by the posteriors, the Bayes rule can be simplified to a linear decision rule:

$$g(C_i/F_{i1}, \dots, F_{im}) = \sum_k I(F_{ik}) w(F_{ik}, C_i)$$

where $I(F_{ik})$ is an indicator variable that equals 1 only if F_{ik} is present in the item (equals zero otherwise) and w is a coefficient corresponding to a specific feature/concept pair. A careful choice of w produces a ranking in decreasing order that is equivalent to the order produced by the posterior probabilities. Interpreting the coefficients, w , as weights corresponding to each feature (e.g., index term) and the function g as the sum of the weights of the features, the result of applying the formula is a set of term weights (Fung-95).

2.2.3 Shannon's Theory of Information

In the late 1940s Claude Shannon, a research mathematician at Bell Telephone Laboratories, invented a mathematical theory of communication to be used in the design of telephone systems. The issues to be resolved were how to design telephone systems to carry the maximum amount of information and how to correct for noise on the lines. He approached the problem by defining a simple abstraction of human communication called the channel. Shannon's communication channel consisted of a sender (a source of information), a transmission medium (with noise), and a receiver (whose goal is to reconstruct the sender's messages). In order to analyze the sending of the information through the channel, he defined the concept of the amount of information in a message. In this concept he considered redundant information versus unique information. In this approach a message is very informative (has a high information value) if the chance of its occurrence is small because the loss of the message means the information will be lost. If, in contrast, a message is very predictable, then it has a small amount of information—one is not surprised to receive it and its loss is not as critical because it will be sent again.

Of less importance to information retrieval Shannon also defined the entropy rate that measured the production rate of information production and a measure of

the channel capacity to carry information. He showed that if the amount of information you want to send exceeds the capacity you will lose information. If the amount of information you want to send is less than the capacity you can encode the information in a way that it will be received without errors.

Shannon adapted his theory to analyze ordinary human (written) language. He showed that it is quite redundant, using more symbols and words than necessary to convey messages. Presumably, this redundancy is used by us to improve our ability to recognize messages reliably and to communicate different types of information. The formula for the information value of an event is:

$$\text{Info}_k = -\log(p_k)$$

This lead to the interpretation of Shannon's theory that the information value of a word is inversely proportional to how often it is used. A word that is found in every document has no information value because it will always be there. But a word that is found in few documents has high information value when you want to retrieve documents with that word in it. This theory is the basis for the "inverse document formula" (IDF) weighting formula used in many informational retrieval weighting algorithms. It is also used in many other ways such as by the Autonomy product in how it does concept searches—by applying this as a factor on the words it finds when it creates taxonomy for them. This will be discussed in detail in Chap. 4 on Indexing.

2.2.4 *Latent Semantic Indexing*

Latent Semantic Indexing (LSI) was created to support information retrieval and solve the problem of the mismatch between a user's vocabulary and that of the author. Its assumption is that there is an underlying or "latent" structure represented by interrelationships between words (Deerwester-90, Dempster-77, Dumais-95, Gildea-99, Hofmann-99). LSI starts with a "vector/matrix view of a set of documents. Just consider a vector where every position represents one word in a language. Thus it will be a vector that will have millions of positions. A document can be represented by the vector by placing a "weight" in each word location as to the weight of that word in describing the semantics of the document. If you place the vector for each document in the database in rows you will have a matrix representing your documents.

Latent Semantic Indexing uses singular-value decomposition to model the associative relationships between terms similar to eigenvector decomposition and factor analysis (see Cullum-85). This is a form of factor analysis. In SVD, a rectangular matrix is decomposed into the product of three other matrices. One matrix describes the original row entities as vectors of derived orthogonal factor values, another matrix describes the original column entities in the same way, and the final matrix is a diagonal matrix containing scaling values such that when the three components are matrix-multiplied, the original matrix is reconstructed. There is a mathematical

proof that any matrix can be so decomposed perfectly, using no more factors than the smallest dimension of the original matrix.

When fewer than the necessary number of factors is used, the reconstructed matrix is a least-squares best fit which minimizes the differences between the original and reduced matrix. One can reduce the dimensionality of the solution simply by deleting coefficients in the diagonal matrix, ordinarily starting with the smallest. Values. By having the values are sorted this will be the bottom rows of the matrix.

Mathematically, the rectangular matrix can be decomposed into the product of three matrices. Let X be a $m \times n$ matrix such that:

$$X = T_0 \cdot S_0 \cdot D_0'$$

where T_0 and D_0 have orthogonal columns and are $m \times r$ and $r \times n$ matrices, S_0 is an $r \times r$ diagonal matrix and r is the rank of matrix X . This is the singular value decomposition of X . The k largest singular values of S_0 are kept along with their corresponding columns/rows in T_0 and D_0 matrices, the resulting matrix:

$$\bar{X} = T_n \cdot S_n \cdot D_n'$$

is the unique matrix of rank k that is closest in least squares sense to the original X . The matrix \bar{X} , containing the first k independent linear components of the original X represents the major associations with noise eliminated.

If you consider X to be the term-document matrix (e.g., all possible terms being represented by columns and each item being represented by a row), then truncated singular value decomposition can be applied to reduce the dimensionality caused by all terms to a significantly smaller dimensionality that is an approximation of the original X :

$$X = U \cdot SV \cdot V'$$

where $u_1 \dots u_k$ and $v^1 \dots v^k$ are left and right singular vectors and $sv_1 \dots sv_k$ are singular values. A threshold is used against the full SV diagonal matrix to determine the cutoff on values to be used for query and document representation (i.e., the dimensionality reduction). Hofmann has modified the standard LSI approach using additional formalism via Probabilistic Latent Semantic Analysis (Hofmann-99). Chapter 4 will relate this specifically to informational retrieval indexing with examples.

It is instructive to show how to calculate the different matrices. An example of how to calculate the three matrices follows (an online calculator for SVD is available at <http://www.bluebit.gr/matrix-calculator/>):

Perform Single Value Decomposition on the given matrix A such that $A = USV^T$

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Step 1: Calculate $A^T A$.

$$A^T A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 2 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 7 & 3 & 5 \\ 3 & 10 & 5 \\ 5 & 5 & 7 \end{bmatrix}$$

Step 2: Find the determinant such that $|A^T A - CI| = 0$ —where I is the identity matrix and C is a scalar—to obtain the *Eigenvalues* and *singular values* which will be used to construct the S matrix.

$$A^T A - CI = \begin{bmatrix} 7 & 3 & 5 \\ 3 & 10 & 5 \\ 5 & 5 & 7 \end{bmatrix} - \left(C * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 7-c & 3 & 5 \\ 3 & 10-c & 5 \\ 5 & 5 & 7-c \end{bmatrix}$$

$$\begin{aligned} |A^T A - CI| &= (7-c)[(10-c)(7-c) - (5*5)] - 3[3(7-c) - (5*5)] \\ &\quad + 5[(3*5) - 5(10-c)] \\ &= (7-c)(70 - 10c - 7c + c^2 - 25) - 3(21 - 3c - 25) \\ &\quad + 5(15 - 50 + 5c) \\ &= (7-c)(c^2 - 17c + 45) - 3(-3c - 4) + 5(5c - 35) \\ &= 7c^2 - 119c + 315 - c^3 + 17c^2 - 45c + 9c + 12 + 25c - 175 \\ &= -c^3 + 24c^2 - 130c + 152 = 0 \end{aligned}$$

$$\left. \begin{array}{l} c_1 = 16.801 \\ c_2 = 5.577 \\ c_3 = 1.622 \end{array} \right\} \text{Eigenvalues}$$

$$|c_1| > |c_2| > |c_3|$$

The singular values would be:

$$s_1 = \sqrt{16.801} = 4.0989$$

$$s_2 = \sqrt{5.577} = 2.3616$$

$$s_3 = \sqrt{1.622} = 1.2736$$

$$S = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix} = \begin{bmatrix} 4.0989 & 0 & 0 \\ 0 & 2.3616 & 0 \\ 0 & 0 & 1.2736 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} 0.244 & 0 & 0 \\ 0 & 0.4234 & 0 \\ 0 & 0 & 0.7852 \end{bmatrix}$$

Step 3: Compute the *Eigenvectors* by evaluating $(A^T A - c_i I) X_i = 0$ —where c_i corresponds to each of the Eigenvalues that were computed in the previous step.

Calculating the Eigenvector for the Eigenvalue $c_1 = 16.801$

$$\begin{aligned} A^T A - c_1 I &= \begin{bmatrix} 7 - 16.801 & 3 & 5 \\ 3 & 10 - 16.801 & 5 \\ 5 & 5 & 7 - 16.801 \end{bmatrix} \\ &= \begin{bmatrix} -9.801 & 3 & 5 \\ 3 & -6.801 & 5 \\ 5 & 5 & -9.801 \end{bmatrix} \end{aligned}$$

$$(A^T A - c_1 I)X_1 = \begin{bmatrix} -9.801 & 3 & 5 \\ 3 & -6.801 & 5 \\ 5 & 5 & -9.801 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$-9.801x_1 + 3x_2 + 5x_3 = 0 \quad (1)$$

$$3x_1 - 6.801x_2 + 5x_3 = 0 \quad (2)$$

$$5x_1 + 5x_2 - 9.801x_3 = 0 \quad (3)$$

By subtracting Eq. (2) from Eq. (1) we get:

$$-12.801x_1 + 9.801x_2 = 0 \rightarrow x_1 = (-9.801/-12.801)x_2 \rightarrow x_1 = 0.7656x_2$$

x_1	-1
x_2	-1.3061
x_3	-1.1765

The Eigenvector for $c_1 = \begin{bmatrix} -1 \\ -1.3061 \\ -1.1765 \end{bmatrix}$

Normalize the vector by the length

$$L = \sqrt{(-1)^2 + (-1.3061)^2 + (-1.1765)^2} = \sqrt{4.0901} = 2.0224$$

The normalized Eigenvector for $c_1 = \begin{bmatrix} -0.4945 \\ -0.6458 \\ -0.5817 \end{bmatrix}$

Using similar approach for calculating the Eigenvector for the Eigenvalue $c_2 = 5.577$ you get

The Eigenvector for $c_2 = \begin{bmatrix} 1 \\ -1.1083 \\ 0.3805 \end{bmatrix}$

Normalize the vector by the length

$$L = \sqrt{(1)^2 + (-1.1083)^2 + (0.3805)^2} = \sqrt{2.3731} = 1.5405$$

The normalized Eigenvector for $c_2 = \begin{bmatrix} 0.6491 \\ -0.7194 \\ 0.247 \end{bmatrix}$

And calculating the Eigenvector for the Eigenvalue $c_3 = 1.622$

The Eigenvector for $c_3 = \begin{bmatrix} -1 \\ -0.4422 \\ 1.3408 \end{bmatrix}$

Normalize the vector by the length

$$L = \sqrt{(-1)^2 + (-0.4422)^2 + (1.3408)^2} = \sqrt{2.9932} = 1.7301$$

The normalized Eigenvector for $c_3 = \begin{bmatrix} -0.5780 \\ -0.2556 \\ 0.775 \end{bmatrix}$

Step 4: Construct the V matrix by using the calculated Eigenvectors as columns in V.

$$V^T = \begin{bmatrix} 0.4945 & 0.6491 & 0.5780 \\ 0.6458 & -0.7194 & 0.2556 \\ -0.5817 & -0.247 & 0.775 \end{bmatrix}$$

Step 5: Calculate the U matrix such that $U = AVS^{-1}$.

$$U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} -0.4945 & 0.6491 & -0.5780 \\ -0.6458 & -0.7194 & -0.2556 \\ -0.5817 & 0.247 & 0.775 \end{bmatrix}$$

$$\times \begin{bmatrix} 0.244 & 0 & 0 \\ 0 & 0.4234 & 0 \\ 0 & 0 & 0.7852 \end{bmatrix}$$

$$U = \begin{bmatrix} 0.4202 & 0.0748 & -0.0461 \\ 0.2995 & -0.2 & 0.4078 \\ 0.1207 & 0.2748 & -0.4539 \\ 0.1576 & -0.3046 & -0.2007 \\ 0.1207 & 0.2748 & -0.4539 \\ 0.2626 & 0.3794 & 0.1546 \\ 0.4202 & 0.0748 & -0.0461 \\ 0.4202 & 0.0748 & -0.0461 \\ 0.2626 & 0.3794 & 0.1546 \\ 0.3152 & -0.6092 & -0.4014 \\ 0.2995 & -0.2 & 0.4078 \end{bmatrix}$$

2.2.5 Hidden Markov Models

Hidden Markov Models (HMM) have been applied for the last 20 years to solving problems in speech recognition and to a lesser extent in the areas locating named entities (Bikel-97), optical character recognition (Bazzi-98) and topic identification (Kubala-97). More recently HMMs have been applied more generally to information retrieval search with good results. One of the first comprehensive and practi-

cal descriptions of Hidden Markov Models was written by Dr. Lawrence Rabiner (Rabiner-89).

A HMM can best be understood by first defining a discrete Markov process. The easiest way to understand it is by an example. Let's take the example of a three state Markov Model of the Stock Market. The states will be one of the following that is observed at the closing of the market:

State 1 (S1): market decreased

State 2 (S2): market did not change

State 3 (S3): market increased in value

The movement between states can be defined by a state transition matrix with state transitions (this assumes you can go from any state to any other state):

$$A = \{a_{i,j}\} = \begin{bmatrix} 0.5 & 0.3 & 0.4 \\ 0.1 & 0.6 & 0.3 \\ 0.6 & 0.7 & 0.5 \end{bmatrix}$$

Given that the market fell on one day (State 1), the matrix suggests that the probability of the market not changing the next day is 0.1. This then allows questions such as the probability that the market will increase for the next 4 days then fall. This would be equivalent to the sequence of SEQ = {S3, S3, S3, S3, S1}. In order to simplify our model, let's assume that instead of the current state being dependent upon all the previous states, let's assume it is only dependent upon the last state (discrete, first order, Markov chain.) This would then be calculated by the formula:

$$\begin{aligned} P(\text{SEQ}) &= P[S3, S3, S3, S3, S1] \\ &= P[S3] * P[S3/S3] * P[S3/S3] * P[S3/S3] * P[S1/S3] \\ &= S3(\text{init}) * a_{3,3} * a_{3,3} * a_{3,3} * a_{1,3} \\ &= (1.0) * (.5) * (.5) * (.5) * (.4) \\ &= .05 \end{aligned}$$

In the equation we also assume the probability of the initial state of S3 is $S3(\text{init}) = 1$. The Fig. 2.13 depicts the model. The directed lines indicate the state transition probabilities $a_{i,j}$. There is also an implicit loop from every state back to itself. In the example every state corresponded to an observable event (change in the market).

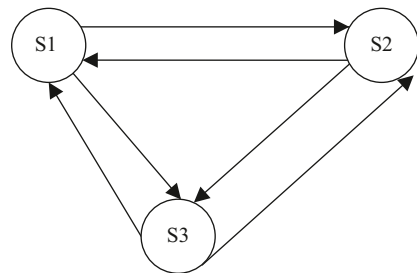


Fig. 2.13 Diagram of Markov model

When trying to apply this model to less precise world problems such as in speech recognition, this model was too restrictive to be applicable. To add more flexibility a probability function was allowed to be associated with the state. The result is called the Hidden Markov Model. It gets its name from the fact that there are two stochastic processes with the underlying stochastic process not being observable (hidden), but can only be analyzed by observations which originate from another stochastic process. Thus the system will have as input a series of results, but it will not know the number of states that were associated with generating the results nor the probability of the states. So part of the HMM process is in determining which model of states best explains the results that are being observed.

A more formal definition of a discrete Hidden Markov Model is summarized by consists of the following:

1. $S = \{s_0, \dots, s_{n-1}\}$ as a finite set of states where s_0 always denotes the initial state. Typically the states are interconnected such that any state can be reached from any other state.
2. $V = \{v_0, \dots, v_{m-1}\}$ is a finite set of output symbols. This will correspond to the physical output from the system being modeled.
3. $A = S \times S$ a transition probability matrix where a_{ij} represents the probability of transitioning from state i to state j such that $\sum_{j=0}^{n-1} a_{i,j} = 1$ for all $i = 0, \dots, n-1$. Every value in the matrix is a positive value between 0 and 1. For the case where every state can be reached from every other state every value in the matrix will be non-zero.
4. $B = S \times V$ is an output probability matrix where element $b_{j,k}$ is a function determining the probability and $\sum_{k=0}^{m-1} b_{j,k} = 1$ for all $j = 0, \dots, n-1$.
5. The initial state distribution.

The HMM will generate an output symbol at every state transition. The transition probability is the probability of the next state given the current state. The output probability is the probability that a given output is generated upon arriving at the next state.

Given the HMM definition, it can be used as both a generator of possible sequences of outputs and their probabilities (as shown in example above), or given a particular out sequence it can model its generation by an appropriate HMM model. The complete specification of a HMM requires specification of the states, the output symbols and three probability measures for the state transitions, output probability functions and the initial states. The distributions are frequently called A , B , and π , and the following notation is used to define the model:

$$\lambda = (A, B, \pi).$$

One of the primary problems associated with HMM is how to efficiently calculate the probability of a sequence of observed outputs given the HMM model. This can best be looked at as how to score a particular model given a series of outputs. Or another way to approach it is how to determine which of a number of competing models should be selected given an observed set of outputs. This is in effect uncov-

ering the hidden part of the model. The typical approach is to apply an “optimality criterion” to select the states. But there are many such algorithms to choose from. Once you have selected the model that you expect corresponds to the output, then there is the issue of determining which set of state sequences best explains the output. The final issue is how best to tune the λ model to maximize the probability of the output sequence given λ . This is called the training sequence and is crucial to allow the models to adapt to the particular problem being solved. More details can be found in Rabiner’s paper (Rabiner-89).

2.2.6 Neural Networks

An artificial neural network is based upon biological neural networks and is generally simplified to a directed multilevel network of that uses weighted additive values coupled with non-linear transfer functions and a final output layer. One of the first neural networks created was the Perceptron network created by Frank Rosenblatt in 1958. It had an analogy to how the visual system works. Thus, the first input layer was called the “retina” that distributed inputs to the second layer composed of association units that combined the inputs with weights and triggered a step function that would send the results to the final output layer. The output layer would do the final combination of the inputs and output the results. This model was a simple approximation of the neurons in the human system. But the use of a step function, where a functions value increases in steps versus is continuous and each step would be a different category, made the mathematics very difficult to allow the system to train itself based upon inputs. By 1969 the problems with this model were documented by papers by Marvin Minsky and Seymour Papert. The mathematical approach was revived in 1986 by Rumelhart, Hinton and Williams when they expanded the concept to include a multilayer model that used nonlinear transfer functions in lieu of the step functions.

There are many different types and approaches to neural networks. One of the more common approaches continues with the Perceptron multilayer network which is presented below. The simplest network is a three layer feed forward network that has an input layer, middle layer (often called hidden layer) and an output layer. Figure 2.14 shows the network. In the Input Function (IF), normalizes the

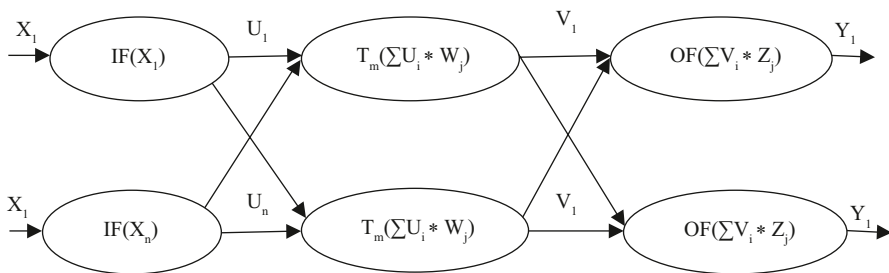


Fig. 2.14 Neural network

input values by subtracting the median and dividing by the interquartile range and presents the resultant value U_i to the middle layer. The interquartile range (IQR) is a measure of the variability of the distribution and is less sensitive to errors, being equal to the difference between the third and first quartiles. If you divide the sorted list into four parts, the quartiles are the three values from the list that separate each section—the median is the second quartile. Every value goes to very function in the middle layer. Each value is multiplied by a weight W and then summed creating a new value that then has the transfer function T applied to it producing the output V_i . The V values are then multiplied by a weight Z and summed. The summed value has the Output Transfer function (OF) applied to it producing the final output from the network, Y . This is a feed forward network because none of the values are fed back to previous layers. All neural networks have an Input and Output layer. The number of middle layers can vary. But in general only one middle layer is needed for most problems.

Training is a critical aspect of a neural network. In the training process a set of known data is used that the ideal outputs (Y_i) are known. In the training process the objective is to modify the weight values (W and Z) to match the output as closely as possible. This leads to some of the problems that have to be monitored in the training process. Additional middle layers may help improve the results although as noted above usually one or two middle layers are sufficient. It may be useful to not feed all of the outputs from one layer into all of the nodes at the next layer (the number of nodes at one layer can be different than the previous layer—in the above example they appear to be the same). The biggest issue is to be careful that the solution is not a local maximum versus a more general global maximum that will apply as new inputs are processed causing over fitting of the solution.

Selecting the number of nodes (neurons) at each layer is very complex. If too few are selected it will be difficult to model complex problems. If too many are selected the computation time increases exponentially and the result can more likely be overfitted to the training data. For this reason two sets of test data are used. The first for the training and the second to validate that the system has not been overfitted to just the original data set.

Trying to find the optimum weights is also a very difficult problem. There can be hundreds of weights that need to be estimated. But the estimation is not linear to produce the desired outputs. In the process of finding the weights there will be many cases of local minima and maxima that need to be avoided. To avoid local minima the easiest technique is to try a number of random starting points in the estimation and choose the one that works best. A more sophisticated technique uses widely separated random values and then gradually reduces the widely separated to closer values to produce the weight. By starting with widely varying values the system is more likely to avoid a particular minima that drives to a local solution.

In a typical training scenario Backward propagation is used. The current set of weights will produce a set of outputs. These outputs are then used with the known expected outputs to calculate the error difference. The errors are then averaged across the outputs and then is propagated back through the network in reverse direction where the adjustments to the weights are made to minimize the error.

2.2.7 Support Vector Machines

Support Vector Machines (SVM) is recently becoming the technical base for learning systems. SVMs are a type of machine learning algorithms used to classify items. A Support Vector Machine (SVM) assigns an item to a category by constructing an N -dimensional hyperplane that optimally separates the data into two categories. The SVM approach maps the set of attributes that make up the vector representing an item into a set of features. The features are then used to determine the hyperplane that distinguishes between the two categories an item could go into. One of the challenges is to find an optimal feature representation. The goal of SVM is to find the optimal hyperplane that separates clusters of vector in such a way that items with one category of the target variable are on one side of the plane and items with the other category are on the other side of the plane. The vectors near the hyperplane are the *support vectors*. The optimal hyperplane will have the maximum distance from the support vectors of each category to the plane that classifies them. This will reduce the errors in miss classifying a new item.

To understand the SVM process lets take a simple two dimensional example. Let's assume we have a number of items that are discussing biology and Physics. Let's assume that we have one feature on the X axis and another feature on the Y axis. Figure 2.15a, b shows the graphical layout of each category with circles being Biology and squares being Physics. The SVM process tries to determine a 1-dimensional hyperplane (i.e., a line) that maximally separates the two groups of items. This is sometimes referred to as maximizing the “fatness” and gives the best classification since it has the maximum difference to help in determining which class an items is assigned to. The diagram shows two options—one being a vertical line and the other a line at an angle. It's obvious by observation that the hyperplane

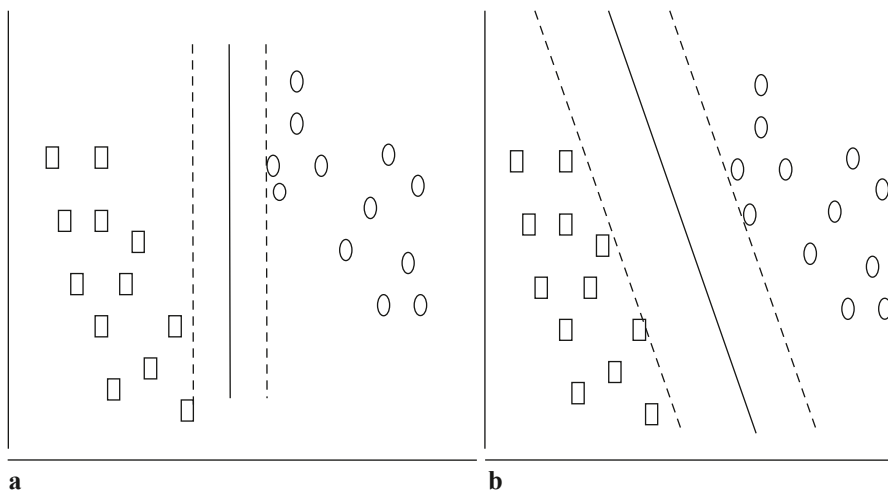


Fig. 2.15 a Vertical separator. b Optimal separator

for the diagonal line is better in that it has the maximum distance between items in each group and the hyperplane. The dashed lines in each figure are showing the specific items (support vectors) from each group that are closest to the hyperplane. The distance between the dashed lines is called the margin and the goal is to find the hyperplane that maximizes the margin. The specific items that are closest to the dashed lines are called the support vectors because they drive the size of the margin. Even though they appear as points in the diagram they are called support vectors because each point defines a vector from the original to that point. As the hyperplane changes, the support vectors (items) that drive the margin change. The Support Vector Machine finds the hyperplane that has support vectors that maximize the margin.

In the example we took the simplest case of a two dimension set of items. This can easily expand to a multidimensional case with a multidimensional hyperplane. The more complex case is when the items are not separated by a plane but some sort of non-linear region (e.g. a curved line). In this case SVM uses a kernel function that maps the items into a different space where they can now be separated by a hyperplane. In some cases additional dimensionality needs to be added in the kernel mapping process. SVM models can be related to neural networks. A SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network.

In addition to the use of mapping to higher dimensionality for the non-linear problem, the real world problem of trying to categorize items based upon text is never statistically pure. There will always be exceptions that come from the variances of language. This is referred to as problems due to the high dimensionality (i.e., lots of unique processing tokens) of text categorization. The approach to solving this is called soft margin classification. In this case instead of trying to raise the dimensionality to account for the data points that are categorized in the wrong category, we ignore them. The way to handle them is to introduce slack variables and by adjusting them minimize the impact by moving those points. The goal is to tradeoff moving points to fit within the current “fat”.

2.3 Summary

Data structures provide the implementation basis of search techniques in Information Retrieval Systems. They may be searching the text directly, as in use of signature and possibly PAT trees, or providing the structure to hold the searchable data structure created by processing the text in items. The most important data structure to understand is the inverted file system. It has the greatest applicability in information systems. The use of n-grams has also found successes in a limited number of commercial systems. Even though n-grams have demonstrated successes in finding information, it is not a structure that lends itself to representing the concepts in an item. There is no association of an n-gram with a semantic unit (e.g., a word or word stem). Judging the relative importance (ranking) of items is much harder to accomplish under this data structure and the algorithmic options are very limited.

PAT and Signature data file structures have found successful implementations in certain bounded search domains. Both of these techniques encounter significant problems in handling very large databases of textual items. The Hypertext data structure is the newest structure to be considered from an Information Retrieval System perspective. It certainly can be mathematically mapped to linked lists and networks. But the model of how dependencies between items as hyperlinks are resolved is just being considered. The future high usage of this structure in information systems makes its understanding important in finding relevant information on the Internet. Marchionini and Shneiderman believe that hypertext will be used in conjunction with full text search tools (Marchionini-88).

Information retrieval algorithms from basic indexing to learning algorithms for categorization are based upon a number of mathematical models. A general understanding of the models and how they apply to information retrieval provide a foundation for develop of new algorithms. Baysean conditional probabilities, Shannon's Information theory and Latent Semantic Indexing are useful in different approaches to defining the ranked index for items. Hidden Marjkov Models can be used for indices but have greater application in multimedia indexing. Neural networks and Support vector Machines provide a foundation for categorization algorithms and learning how to filter items based upon training examples provided by the users.

2.4 Exercises

1. Describe the similarities and differences between term stemming algorithms and n-grams. Describe how they affect precision and recall.
2. a. Compare advantages and disadvantages of Porter Stemming algorithm, Dictionary stemming algorithm and Success Variety stemming algorithm.
 b. Create the symbol tree for the following words (bag, barn, boss, bot any, box, bottle, botch and both). Using successor variety and the Peak and Plateau algorithm, determine if there are any stems for the above set of words.
 c. If there are stems created explain if they make any sense as a stem and why.
3. a. Create the PATRICIA Tree and Reduced PATRICIA for the following binary input. Take it to 9 levels of sistrings: 01110011100111111010
 b. Given the query 111000 show how it would be executed against each tree with the number of decisions.
4. Assuming a term is on the average 6 characters long, calculate the size of the inversion lists for each of the sources in Table 1.1, Distribution of words in TREC Database. Assume that 30% of the words in any item are unique. What is the impact on the calculation if the system has to provide proximity versus no proximity. Assume 4 bytes is needed for the unique number assigned to each item.
5. Describe how a bigram data structure would be used to search for the search term "computer science" (NOTE: the search term is a contiguous word phrase). What are the possible sources of errors that could cause non-relevant items to be retrieved?

6. Perform Single value decomposition on the following matrix:

$$A = \begin{vmatrix} 1 & 3 & 1 \\ 1 & 0 & 2 \\ 0 & 2 & 3 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \\ 0 & 1 & 3 \end{vmatrix}$$



<http://www.springer.com/978-1-4419-7715-1>

Information Retrieval Architecture and Algorithms

Kowalski, G.

2011, XII, 305 p., Hardcover

ISBN: 978-1-4419-7715-1