Mining Web Logs for Actionable Knowledge

Qiang Yang[1], Charles X. Ling[2] and Jianfeng Gao[3]

[1]Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Kowloon Hong Kong
qyang@cs.ust.hk

[2]Department of Computer Science,
University of Western Ontario,
Ontario N6A 5B7, Canada
ling@csd.wuo.ca

[3]Microsoft Research Asia, China
jfgao@microsoft.com

Abstract

Everyday, popular Web sites attract millions of visitors. These visitors leave behind vast amount of Web site traversal information in the form of Web server and query logs. By analyzing these logs, it is possible to discover various kinds of knowledge, which can be applied to improve the performance of Web services. A particularly useful kind of knowledge is knowledge that can be immediately applied to the operation of the Web site; we call this type of knowledge the actionable knowledge. In this paper, we present three examples of actionable Web log mining. The first method is to mine a Web log for Markov models that can be used for improving caching and prefetching of Web objects. A second method is to use the mined knowledge for building better, adaptive user interfaces. The new user interface can adjust as the user behavior changes with time. Finally, we present an example of applying Web query log knowledge to improving Web search for a search engine application.

1. Introduction

Today's popular Web servers are accessed by millions of Web users each day. These visitors leave behind their visiting behavior in the form of Web logs. By analyzing the Web logs recorded at these servers as well as proxy servers, it is possible to learn the behavior of the Web users themselves. It is further possible to use the learned knowledge to serve the users better.

A particularly useful kind of knowledge is knowledge that can be immediately applied to the operation of the Web site; we call this type of knowledge the actionable knowledge. Many previous approaches are only aimed to mine Web-log knowledge for human consumption, where the knowledge is presented for humans to decide how to make use of it. In contrast, we advocate an approach to mine *actionable knowledge* from Web logs. The purpose of the knowledge is for computers to consume. In this manner, the quality of the mined knowledge can be immediately put

to test and the Web services can be said to be truly self adaptive. The advantage of mining actionable knowledge is similar to that of the Semantic Web [BHL 2001].

In this chapter, we first provide an overview of the current research on Web log mining. We provide a detailed description of our own work in this area. We first describe algorithms that acquire user preferences and behavior from the Web log data and show how to design Web-page prefetching systems for retrieving new Web pages. We empirically demonstrate that the Web prefetching system can be made self-adaptive over time and can outperform some state-of-the-art Web caching systems. We then describe our work on dynamically reconfigure the user interface for individual users to cater to the users' personal interests. These reconfigured Web pages are presented to the user in the form of index pages. The computer generates these pages by consulting a data-mining algorithm for potential clusters of Web objects and determines the best balancing point according to a cost function. Finally, we present an approach to mining web query logs from a web search engine to relate the semantics of queries to their targets. The knowledge about these query logs can then be used for improving the performance of the search engine.

2. Web Log Mining for Prefetching

2.1 Data Clearning on Web Log Data

To learn from the Web logs, the first task is to perform data cleaning by breaking apart a long sequence of visits by the users into user sessions. Each user session consists of only the pages visited by a user in a row. Since we are only dealing with Web server logs, the best we could do is to take an intelligent guess as to which pages in a long sequence belong to a single user session. As we will see, a strong indicator is the time interval between two successive visits. When the time exceeds the average time it takes a person to read a Web page, there is a strong indication that the user has left the browser to do other things.

In the web logs, a user can be identified by an individual IP address, although using the IP address to identify the users is not reliable. Several users may share the same IP address, and the same user may be assigned different IP address at different times. In addition, the same user may make different sequences of visits at different times. Thus, data cleaning means to separate the visiting sequence of pages into visiting sessions. Most work in Web log mining employ a predefined time interval to find the visiting sessions. For example, one can use a two-hour time limit as the separating time interval between two consecutive visiting sessions, because people usually do not spend two hours on a single Web page.

Sometimes it is possible to obtain more information about user sessions than using a fixed time interval. By learning the grouping of web pages and web sites, one can find more meaningful session separator. The work by [LLLY 2002] provides a method to use clustering analysis to find the group of related Web servers visited by users from a Web proxy server. If a user jumps from one group of related server to another, then it is highly likely that the user ends one session and starts another. Using this information to cut and pick the web pages, more accurate user session knowledge can be obtained.

2.2 Mining Web Logs for Path Profiles

Once a Web log is organized into separate visiting sessions, it is possible to learn user profiles from the data. For example, [SKS 1997] developed a system to learn users' path profiles from the Web log data. A user visiting a sequence of Web pages often leaves a trail of the pages URL's in a Web log. A page's *successor* is the page requested immediately after that page in a URL sequence. A *point profile* contains, for any given page, the set of that page's successors in all URL sequences and the frequency with which that successor occurred. A *path profile* considers frequent subsequences from the frequently occurring paths.

Both the point and the path profiles help us to predict the next pages that are most likely to occur by consulting the set of path profiles whose prefixes match the observed sequence. For example, [PP 1999] described in their work how to find the longest-repeating subsequences from a Web log and then use these sequences to make prediction on users' next likely requests.

Similar to the path-profile-based prediction, [AZN 1999] designed a system to learn an N^{th}-order Markov model based on not only the immediately preceding Web page, but also pages that precede the last pages as well as the time between successive Web-page retrieval activities. Together these factors should give more information than the path profiles. The objective is still the same: to predict which is the next page to be most likely requested by a user. Markov models are natural candidates for Web traversal patterns, where the majority of the work now has used the Web pages as states and hyperlinks as potential transitions between the states.

One particular fact about the Web is that the user sessions follow a Zipf distribution. By this distribution, most visitors view only few pages in each session, while few visitors visit many pages in a sequence. However, experiments show that the longer paths also provide more accurate predictions. To exploit these facts, [SYZ 2000] developed an algorithm to combine path profiles with different lengths. For any given observed sequence of visits, their 'cascading' model first selects the best prediction by looking for path profiles with long lengths. When such paths do not exist in the model, the system retreats to shorter-length paths and makes predictions based on these paths. Experiments show that this algorithm can provide a good balance between prediction accuracy and coverage. In this chapter, we develop this n-gram based prediction system into a prefetching system for prefetching new Web pages, which are then stored in the Cache. When the prediction system is accurate, the prefetching system is expected to save on network latency.

2.3 Web Object Prediction

Given a web-server browsing log *L*, it is possible to train a path-based model for predicting future URL's based on a sequence of current URL accesses. This can be done on a per-user basis, or on a per-server basis. The former requires that the user-session be recognized and broken down nicely through a filtering system, and the latter takes the simplistic view that the accesses on a server is a

single long thread. We now describe how to build this model using a single sequence *L* using the latter view. An example of the log file is shown in Figure 1.

```
...
uplherc.upl.com--[01/Aug/1995:00:08:52-0400] "GET
/shuttle/resources/orbiters/endeavour-logo.gif HTTP/1.0" 200 5052

pm9.j51.com--[01/Aug/1995 :00:08:52-0400] "GET/images/xyz.html HTTP/1.0" 200
669

139.230.35.135--[01/Aug/1995 :00:08:52-0400] "GET/images/NASA-logosmall.gif
HTTP/1.0" 200 786


...
```

Figure 1: NASA log file example

We can build an n-gram prediction model based on the object-occurrence frequency. Each sub-string of length n is an n-gram. These sub-strings serve as the indices of a hash table *T* that contains the model. During its operation, the algorithm scans through all sub-strings exactly once, recording occurrence frequencies of documents' requests of the next *m* clicks after the sub-string in all sessions. The maximum occurred request (conditional probability greater than *theta*, where *theta* is a threshold set at 0.6 in our experiments) is used as the prediction of next *m* steps for the sub-string. In this case we say that the n-gram prediction has a window-size *m*. Figure 2 illustrates the prediction process.

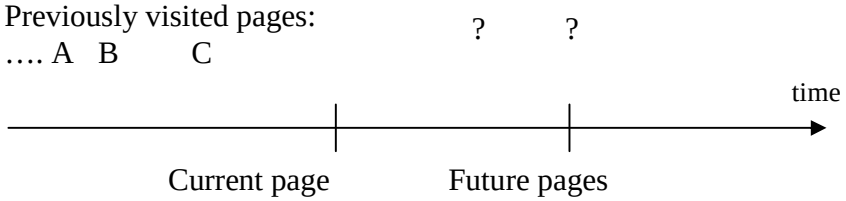


Figure 2. Illustrating Web-access prediction

We observe that many of the objects are accessed only once or twice, and a few objects are accessed a great many times. Using this well-known fact (also known as the *Zipf* distribution), we can filter out a large portion of the raw log file and obtain a compressed prediction model. In our filtering step, we removed all URL's that are accessed 10 times or less among all user requests.

As an example, consider a sequence of URL's in the server log: {A,B,C,A,B,C,A,F} Then a two-gram model will learn on the two-grams shown in Table 1, along with the predicted URL's and their conditional probabilities.

2-Gram	Prediction
A, B	{ < C, 100% > }
B, C	{ < A, 100% > }
C, A	{ < B, 50% >, < F, 50% > }

Table 1: A learned example of n-gram model

Applying n-gram prediction models has a long tradition in network systems research. [SYZ 2000] compared *n*-gram prediction models under different sized *n*, and presented a cascading algorithm for making the prediction. [SKS 1998] provided a detailed statistical analysis of web log data, pointing out the distribution of access patterns on web pages.

We now describe the prediction system in [SYZ 2000] based on the n-grams. Normally, there simultaneously exist a number of sessions on a web server. Based on their access sequences, our prediction model can predict future requests for each particular session. Different sessions will give different predictions to future objects. Since our prediction of an object comes with a probability of its arrival, we can combine these predictions to calculate the future occurrence frequency of an object. Let O_i denote a web object on the server, S_j be a session on a web server, $P_{i,j}$ be the probability predicted by a session S_j for object O_i. If $P_{i,j}=0$, it indicates that object O_i is not predicted by session S_j. Let W_i be the future frequency of requests to object O_i. A weight can be computed according to the following equation:

$$W_i = \sum_j P_{i,j}$$

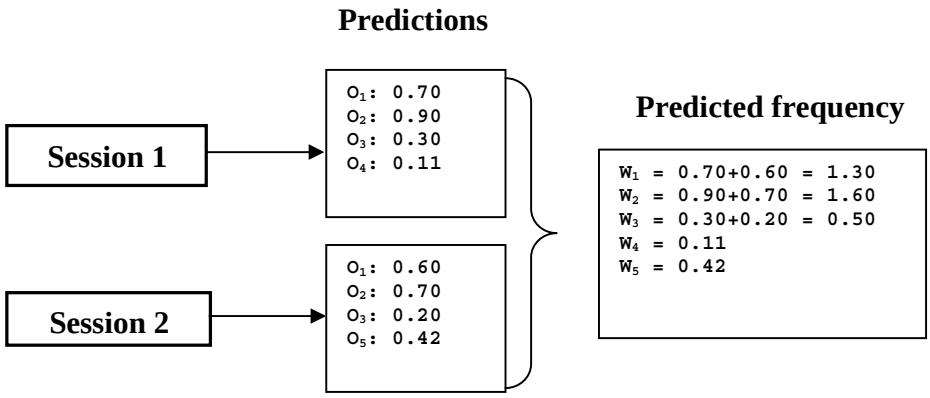


Figure 3. Prediction to frequency weight calculation

To illustrate this equation, consider the three sessions in Figure 3. Each session yields a set of predictions to web objects. Since sessions are assumed independent to each other, we use the equation to compute their weights W_i. For example, object O_1 is predicted by two sessions with a probability of 0.70 and 0.60, respectively. From the weight equation, $W_1 = 1.3$. This means that, probabilistically, object O_1 will be accessed 1.3 times in the near future.

2.4 Learning to Prefetch Web Documents

When a user requests a sequence of pages that match the left hand side of a rule, the right-hand-side can be predicted to occur with the associated confidence value as the probability of occurrence of *Oi*.

The Web prefetching system can then add all such predicted probabilities together from all rules in the prediction model as a measure of the potential future frequency estimation for *Oi*. These scores are compared and the top ranked objects are selected to be fetched into the cache if they are not already there. This prefetching method runs side by side with an existing caching system.

We proposed an integrated caching and prefetching model to further reduce the network latency perceived by users [YZ 2001]. The motivation lies in two aspects. Firstly, from Figure 6, we can see both the hit rate and byte-hit rate are growing in a log-like fashion as a function of the cache size. Our results consist with those of other researchers [CI 1997; A1999). This suggests that hit rate or byte-hit rate does not increase as much as the cache size does, especially when cache size is large. This fact naturally leads to our thought to separate part of the cache memory (e.g. 10% of its size) for prefetching. By this means, we can trade the minor hit rate loss in caching with the greater reduction of network latency in prefetching. Secondly, almost all prefetching methods require a prediction model. Since we have already embodied an n-gram model into predictive caching, this model can also serve prefetching. Therefore, a uniform prediction model is the heart of our integrated approach.

In our approach, the original cache memory is partitioned into two parts: cache-buffer and prefetching-buffer. A prefetching agent keeps pre-loading the prefetching-buffer with documents predicted to have the highest weight *W*. The prefetching stops when the prefetching-buffer is full. The original caching system behaves as before on the reduced cache-buffer except it also checks a hit in the prefetching-buffer. If a hit occurs in the prefetching-buffer, the requested object will be moved into the cache-buffer according to original replacement algorithm. Of course, one potential drawback of prefetching is that the network load may be increased. Therefore, there is a need to balance the decrease in network latency and the increase in network traffic. We next describe two experiments that show that our integrated predictive-caching and prefetching model does not suffer much from the drawback.

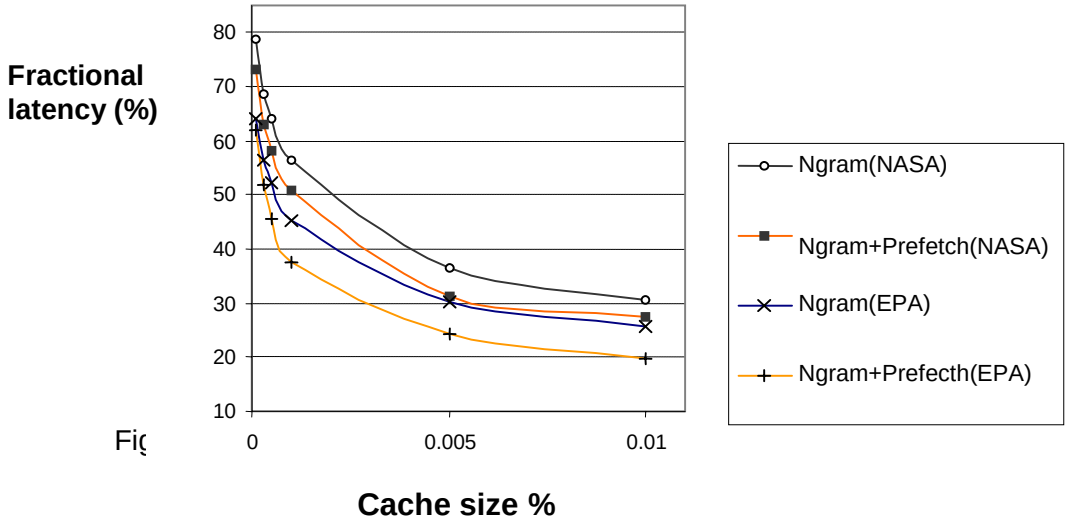
In our prefetching experiments, we again used the EPA and NASA web logs to study the prefetching impact on caching. For fair comparison, the cache memory in cache-alone system equals the total size of cache-buffer and prefetching-buffer in the integrated system. We assume that the pre-buffer has a size of 20% of the cache memory. Two metrics are used to gauge the network latency and increased network traffic:

Fractional Latency: The ratio between the observed latency with a caching system and the observed latency without a caching system.

Fractional Network Traffic: The ratio between the number of bytes that are transmitted from web servers to the proxy and the total number of bytes requested.

As can be seen from Figure 4, prefetching does reduce network latency in all cache sizes. On EPA data, when cache size is 1% (approximately 3.1MB), fractional latency has been reduced from 25.6% to 19.7%. On NASA data, when cache size is 0.001%(~ 240KB), fractional latency has been reduced from 56.4% to 50.9. However, we pay a price for the network traffic, whereby the prefetching algorithm incurs an increase in network load. For example, in NASA dataset, the fractional network traffic increases 6% when cache size is 0.01%. It is therefore important to strike for a balance the

improvement in hit rates and the network traffic. From our result, prefetching strategy better performs in a larger cache size while relatively less additional network traffic is incurred.



Fig

Figure 4. Fractional Latency Comparison

3. Web Page Clustering for Intelligent User Interfaces

Web logs can not only be used for prefetching, they can also be used to build server-side customization and transformation -- operations that converts a web site, on the server side, into one that is more convenient for users to visit and find their objectives.

Several researchers have studied the problem of creating web interfaces that can adapt to user behaviors based on web logs. Examples include *path prediction* algorithms that guess where the user wants to go next in a browsing session so that the server can either pre-send documents or short-cut browsing paths. For instance, *WebWatcher* (Armstrong et al.1995) learns to predict what links users will follow on a particular page as a function of their specified interests. A link that *WebWatcher* believes a particular user is likely to follow will be highlighted graphically and duplicated at the top of the page when it is presented.

One particularly useful application is to compose index pages for individual group of users automatically. This application corresponds to a kind of personalization on the Web. One of the first systems to do this is the *PageGather* system [PE1999], which provides users with shortcuts, which takes as input a web-server access log, where the log records the pages visited by a user at the site. Based on the visiting statistics, the system provides links on each page to visitors' eventual goals,

skipping the in-between pages. An *index page* is a page consisting of links to a set of pages that cover a particular topic (e.g., electric guitars).

Given this terminology, [PE 1999] define the *index-page synthesis problem*: given a web site and a visitor access log, create new index pages containing collections of links to related but currently unlinked pages. An access log is a document containing one entry for each page requested of the web server. Each request lists at least the origin (IP address) of the request, the URL requested, and the time of the request. *Related but unlinked* pages are pages that share a common topic but are not currently linked at the site; two pages are considered linked if there exists a link from one to the other or if there exists a page that links to both of them. In their *PageGather* algorithm, [PE 1999] presented a clustering-based method for generating the *contents* of the new web page from server access logs.

Given an access log, a useful task is to find collections of pages that tend to co-occur in visits. Clustering is a natural technique to consider for this task. In clustering, documents are represented in an N-dimensional space formed by term vectors. A cluster is a collection of documents close to each other and relatively distant from other clusters. Standard clustering algorithms *partition* the documents into a set of mutually exclusive clusters.

The *PageGather algorithm* uses cluster mining to find collections of related pages at a web site. In essence, *PageGather* takes a web server access log as input and maps it into a form ready for clustering; it then applies cluster mining to the data and produces candidate index-page contents as output. The algorithm has five basic steps:

PageGather Algorithm (Server Access Logs)
[Perkowitz and Etzioni 1999]

1. Process the access log into visits.
2. Compute the co-occurrence frequencies between pages and create a similarity matrix.
3. Create the graph corresponding to the matrix, and find maximal cliques (or connected components) in the graph.
4. Rank the clusters found, and choose which to output.
5. For each cluster, create a web page consisting of links to the documents in the cluster,
6. Present clusters to the Webmaster for evaluation.

Let N be the number of web pages at the site. This algorithm is thus $O(N^2)$ time, quadratic in the original number of web pages. We note that due to this high complexity, the algorithm is not suitable for processing large data sets that are typical of today's web access patterns. For example, everyday, MSN collects about millions of visits. Data sets of this scale must be processed with very efficient disk-based algorithms. Thus, one of our intensions is to explore more efficient clustering algorithms for synthesizing index pages.

Another drawback of the *PageGather* algorithm is that it relies on the human web masters to determine the appropriateness of the generated index pages in a final check. This will likely create a bottleneck for the workflow, especially for sites that have many web pages to be indexed. A

particularly important problem is the question of *how many index pages* to create, and *how many links* to include in each index page. We answer this question in our paper.

Having obtained the clusters, we now turn our attention to the second contribution of the paper, in building web interfaces that provide useful short cuts for people. We do this by providing index pages, which are table-of-content pages that we can put at the root of a web site. The idea of index pages is first proposed by [PE 1999], but many open issues still remained. For example, to make the knowledge actionable, it would be nice to remove the need for human users to subjectively select the links to compose the index pages. Here we would like to address the important issue of how to extend their manually evaluated index pages by a novel technique to find an optimal construction of index pages automatically.

In our own work, we improve the *PageGather* system by automating the index-page creation process completely [SYXZH 2002]. We do this by introducing a cost function, which we then use to judge the quality of the index pages created.

We first define the cost models of web browsing. The cost arising from web browsing can be summarized as a reduction of the *transition costs* between web pages. Various cost models can be used to describe the relation between costs and the number of URL's traversed; in this paper we adopt a cost model such that the cost of a browsing session is directly proportional to the number of URL's on the browsing path. Shortcutting by offering index pages should not be considered to be cost free, however. Index pages themselves tax on the users' attention by requiring that users flip through extra pages in the process of finding their destinations. Therefore, when the number of index pages increases, the transition costs should decrease while, at the same time, the *page cost* associated with the need to flip through the index pages increases.

Let *OverallCost* be the overall cost of the browsing web pages and index pages. Let *PageCost* be the cost of flipping index pages and *TransitionCost* be the cost of switching from one web page to another. Then our cost models are as follows:

$$OverallCost = PageCost + TransitionCost$$

Let *n* be the number of index pages and $N_{\max}$ be the user defined maximum number of index pages. Then we define *PageCost* to be a linear function of the number of index pages:

$$PageCost = \begin{cases} 0, n < 1 \\ \dfrac{n}{N_{\max}}, 1 \le n \end{cases}$$

For each index page P_j j=1, … n, for each session S_i in the web log, i=1, …*Sessions*, where *Sessions* is the total number of sessions in a log. Let $k_{i,j}$ be the number of URL's in P_j that also appears in S_i; $k_{i,j} - 1$ is the cost saved by including the index page P_j in session S_i.

$$TransitionCost = (TotalAccess - Sessions) - \sum_{i=1}^{Sessions} \left(\sum_{j=1}^{number\_of\_index\_pages} (k_{i,j} - 1) \right)$$

We can then normalize the cost as the following:

$$OverallCost = \frac{TransitionCost}{TotalAccess - Sessions}$$

where *TotalAccess* is the total number of transitions and Sessions is the total number of sessions in the log. This normalization function defines a cost function within the range of zero and one.

We now describe how to compose index pages in our framework. In their algorithm, Perkowitz and Etzioni first computes clusters from the web logs and then put all clusters in index pages, so that each cluster will correspond to one index page. In our experience, we have found that often each cluster will contain a large number of index pages. When hundreds of hyperlinks are included in an index page, it is very difficult for a user to find the information he/she is looking for. In addition, we feel that there should be a limited number of index pages; if the user is required to read a huge number of index pages then it might defeat the purpose of including the index pages in the first place.

Therefore, in index page construction, we will include two parameters. Let the *L* be the number of hyper links we would like to include in each index page, and let *M* be the number of index pages we wish to build. Algorithm *ConstructIndexPages* takes the parameters M and L and the clusters constructed by our RDBC algorithm, and produces M index pages as output:

Table 2. Algorithm *ConstructIndexPages*(Clusters, M, L)

```
1) For j = 1 to M
2) {
3)  Sort Clusters by the frequency count of the top L web pages;
4)  Extract the top L web pages from the first cluster and insert their hyperlinks
    into an index page;
5)  If (No cluster is left or size of each cluster < L), Stop;
6) }
```

More importantly, based on these cost functions, we can find a *minimal value* M for the *OverallCost* and its corresponding number of index pages to build. This optimal index-page construction process represents another major contribution of our work. What we do is to analyze the overall cost as a function of the number of index pages M to construct, based on a fixed value of L. We can then find empirically the best value for M so as to minimize the overall cost of user browsing effort.

Figure 5 shows the overall cost function calculated from the combination of the web page switching cost and the index page cost with the NASA data. As seen in the figure, the overall cost has a minimum value at around 3.5 index pages. This is an indication that when considering all factors, it is the best to include around 3 to 4 index pages, where each page contains L hyperlinks. The optimal number in this example is computed automatically from different statistics, rather then decided

subjectively by a human user. When in real application, the web site can be said to truly reconfigure itself based on the Web log files accumulated in a certain period of time.

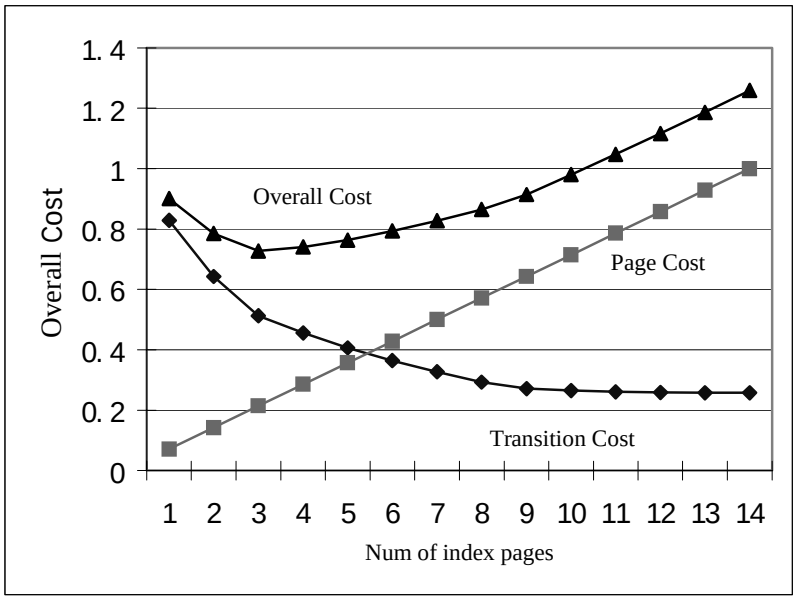


Figure 5: Cost of page construction using the clustering results on NASA's log.

4. Web Query Log Mining[1]

4.1 Web Query Logs

We now describe an application whereby we apply data mining to an Internet search engine in this paper. In particular, we apply data mining to discover useful and implicit knowledge from user logs of the Microsoft Encarta search engine (http://encarta.msn.com) to improve its performance. The user logs keep traces of users behaviors when they use the Encarta search engine. The purpose of the web-log mining is to improve Encarta search engine's performance (which is defined precisely later in the paper) by utilizing the mined knowledge as cache. Indeed, data mining is a promising approach since popular search engines like Encarta get hundreds of thousands of hits each single day, and therefore, it would be infeasible for traditional methods or human to analyze such logs.

Encarta accepts users' submission as keyword queries, and returns a list of articles for users to choose from. There are over 40,000 articles in Encarta. Users can also browse articles organized under a hierarchical structure of several levels. The log keeps track of users' queries and their clicks on the returned articles, as well as their browsing activities. The log file is quite large: a one-day log is over several hundreds megabytes in size.

[1] This section is adapted from [Ling et. Al 2003]

Our log-mining system is an integration of previous techniques (mostly on smaller problems though), but when they are carefully integrated and tuned, they are effective to solve this large-scale real-world problem. The goal is to find useful "patterns" - rules associating keywords of the queries with the target articles. If many users keying in similar queries clicked on the same article, then generalized query patterns summarizing the queries will provide a direct answer (the article) to the similar queries in the future. When a set of such patterns is used as cache of the Encarta search engine, its overall performance can be dramatically improved.

Let us assume that cache is *k* times faster than a regular search by the search engine. To obtain overall improvement with the cache, we must consider how likely a future user query is hit by the cache, and how accurately the cache predicts the article (so user does not have to revoke the regular search engine for the correct answer). The first part is measured by the **recall** (*R*) (or the hit ratio) of the cache; it measures the percent of the testing queries that are matched with the queries patterns in the cache. The second part is the **precision** (*P*), which measures, among queries hit by the cache, the percent of the testing queries that are correctly predicted (that is, user's click on the article returned from the query is the same as what query patterns predict). Thus, assuming a unit time is needed by a regular search engine, the overall time when the search engine is equipped with the cache is:

$$P\,R / k \;+\; R\,(1 - P)(1+1/k) + (1 - R)(1+1/k)$$

The first term is for the user query correctly matched with a pattern (with the probability *PR* the cache returns the correct result within time *1/k*), the second term for an incorrectly matched query (so the user requests another search from the regular search engine), and the third term for no match (so the regular search engine is invoked). The overall time is equal to:

$$1 + 1/k - P\,R$$

Clearly, the larger the *k*, and the larger the product of *P* and *R*, the less the overall time of the search engine equipped with cache. Thus, our first goal of mining user logs of Encarta is to improve the product of the precision and recall of the cache.

A major advantage of our caching technique over the previous ones is that our cache consists of generalized query patterns so that each pattern covers many different user queries. They can even match with new queries that were not asked previously. Most previous caching approaches keep the frequent user queries, or invert lists directly (See [Saraiva et al, 2001] and the references), limiting the recall of cache with a fix size dramatically. Some previous work on semantic caching (such as [Chidlovskii et al, 1999]) can compose answers of new queries by applying Boolean operations to the previous answers. However, they do not allow generalization of the previous queries as we do.
The second goal of the research is to discover comprehensible descriptions on the topics in which users are mostly interested. The web editors can study those topics, and add more articles on popular topics. Our generalized query patterns are production-rule based, and are easy to comprehend. Therefore, we need to build a user-friendly interface, providing different and clear views of the query patterns discovered.

Past work on web-log mining has been done. However, most has focused on mining to change the web structure for easier browsing [Craven, et al, 2000; Sundaresan and Yi, 2000], predicting browsing behaviors for pre-fetching [Zaine, et al, 1998; Boyan, 1996], or predicting user preference for active advertising [Pei, et al, 2000; Perkowitz, 1997]. Some work has been done on mining to improve search engine's performance. The most notable example is the Google search engine [Google], in which statistical information is mined from the linkage structure among web pages for weighting and ranking of the web pages. However, when users click on web pages returned, they access to the web pages directly. Therefore, Google does not collect users' feedback on the search results (unless when users click on the cached pages on the Google server). Thus, Google's improvement on the search engine is largely syntactic and static based on linkage relations among web pages. Our method is semantic and dynamic, since it mines from the users actual clicks (feedback) on the search results, and it dynamically reflects the actual usage of the search engine. See Section 4 for more reviews of other work and its relation to ours.

4.2. Mining Generalized Query Patterns

In this section, we describe in details how data mining is used to improve the Encarta search engine's performance. The Encarta website (http://encarta.msn.com) consists of a search engine, with 41,942 well-versed articles (written by experts of various domains) as answers to the user queries, and a well-organized hierarchy of articles for browsing (but one cannot use the search engine and the hierarchy interchangeably). The users' activities are recorded in the raw IIS (Internet Information Services) log files, which keep a lot of information about each user's access to the web server. The log files are large: one-day log files are about 700 megabytes in size.

A log pre-processing program is written to extract user queries and their corresponding articles clicked. The search engine is a keyword-based search engine, and therefore, almost all queries are simply lists of keyword(s). The result of the log pre-processing program is as follows:

$$\text{keyword}_1, \text{keyword}_2, \ldots; \text{article}_1, \text{article}_2, \ldots$$

Each line indicates that a user query contains keyword_1, keyword_2, and so on, and the corresponding articles that the user clicks are article_1, article_2, etc.

We find that most queries have a very small number of keywords: 52.5% of queries contain only one keyword, 32.5% two, 10% three, and 5% four and above. Also most users clicked on only one article for their queries: only 0.0619% of user session clicked on two or more articles for a query. We simply convert each of those queries into several queries, each of which contains the same keyword(s), and one article that the user clicked. This would introduce an inherit error rate, since the same query is linked to different articles as answers.

One inherent difficulty in our work is that user queries are mostly short, consisting one or two keywords. On the other hand, user clicks of the articles can vary due to various user intentions. Our data-mining algorithm should be able to deal with noise in the data, and to find rules associating queries with the most relevant articles.

From two-month Encarta's user log, we obtained 4.8 million queries, each of which is a list of keyword(s) and one article clicked. Our data mining starts from this query list, and produces generalized query patterns, which summarize similar queries answered to the same article. Again, those queries patterns will act as the cache of the Encarta search engine.

4.3 A Bottom-up Generalization Algorithm

A query pattern represents a set of user queries with the same or similar intention, and thus is associated with an article as the answer. It is difficult to capture the "similar intention" in natural language, and therefore, we use both syntactic constraints (such as stemming of keywords; see later) as well as semantic constraints (such as generalized concepts and synonyms; see later) in our definition of queries with "similar intentions".

The simplest form of patterns consists of some keyword(s), and possibly, a "don't care" keyword, in the format of:

article ID $\leftarrow$ keyword$_1$, …, keyword$_i$, [any]; accuracy; coverage

where "any" represents any keyword(s), and is placed in […] to distinguish it from user-inputted keywords. If a pattern contains a generalized term placed in [ ], it is called a *generalized pattern*; otherwise (if it contains user-inputted keywords only) it is called a *simple pattern*.

Accuracy and coverage are two numbers for evaluating query patterns. (Note that the precision and recall mentioned earlier are measures for testing queries on the whole set of patterns, while accuracy and coverage here are for each individual query pattern.) *Coverage* is the number of original user queries that are correctly covered (or predicted) by this pattern, and *accuracy* is the percentage of queries that correctly covered. If *error* is the number of queries incorrectly covered by a pattern (when users clicked a different article), then *accuracy = coverage/(coverage + error)*. If the accuracy and coverage of all query patterns are known, and the distributions of testing set on those patterns are known, then the overall precision and recall of the pattern set (cache) on the test set can be easily derived.

A bottom-up generalization algorithm is designed to find generalized patterns with [any], and the overall process is described below. First, it groups the queries clicked by the same article together, and sorts them by the article frequency, so that queries of the most clicked articles are mined first. From the queries of the same article, two are chosen according to a greedy heuristic (see later) if they have some keyword(s) in common. Then a tentative pattern is created with those common keyword(s), and the [any] keyword inserted. Its accuracy and coverage can then be calculated. If the accuracy or the coverage of this pattern is below some thresholds, it is discarded; else the pattern replaces all queries correctly covered by it, and the process repeats, until no new patterns can be formed. Then the generalized patterns (with the generalized concept [any]) and the simple patterns (the remaining user queries) are returned, and they are all associated with that article. This process is applied to all articles. In the end, all patterns of all articles are sorted according to their coverage, and the top *t*

patterns with highest coverage from all articles are outputted as the cache. The cache size *t* was chosen as 5,000 in this paper, since a query distribution analysis shows that when *t* is overly larger than 5,000, the overall benefit of larger caching is reduced.

Note that the relation between query patterns and articles is a many-to-many one. On one hand, the same query pattern can associate with more than one article (since general keywords such as "animal" can be the keyword of several articles). Therefore, several articles can be returned if the user query matches with several query patterns with different articles. On the other hand, the same article can be associated with many query patterns (different keywords can associate with the same article).

To choose which two queries for generalization, a greedy heuristic is used. Every pair of queries is tried, and the pair with the maximum coverage while the accuracy is above some pre-set threshold is chosen at each step for generalization.

Experiments (not shown here) indicate that this algorithm does not produce satisfactory results: only a few useful patterns are produced. The first reason is that the generalization step of introducing [any] into patterns is often too bold: such patterns cover too many queries incorrectly, and thus have a low accuracy. Second, many keywords have slightly different spellings (sometimes with a minor spelling error) and thus are not regarded as the same, preventing possible generalization. Third, many keywords are synonyms; but since they are spelled differently, they cannot be treated as the same for possible generalization. Fourth, one-keyword queries are not paired for generalization, since it would inevitably produce an overly generalized pattern "article ID ← [any]". However, most (52.5%) of the queries in the user logs are one-word queries. We want generalization to happen more gradually. This would also alleviate the first problem mentioned above.

In the following four subsections we provide solutions to the four problems mentioned above. Most improvements have been published previously (see Section on Relation to Previous Work), but when they are carefully integrated and tuned, they are shown to be effective to solve this large-scale real-world problem.

4.4 Improvement 1: A Hierarchy over Keywords

The first problem mentioned earlier is over generalization. That is, any two different keywords from two queries will be replaced by [any]. To provide more graded generalization, a hierarchical structure with the "is-a" relation would be useful. With this hierarchy, the generalization of two keywords would be the lowest concept that is an ancestor of both keywords in the hierarchy.
It is a tedious job to construct such a hierarchy over tens of thousands of keywords in the user queries. We used WordNet [Miller, 1990] to generate such a hierarchy automatically. A problem occurred is that most keywords have different senses or meanings, which in turn, have different parents in the hierarchy. We adopt the first, or the most frequently used meaning of each keyword in the WordNet. Previous research [Ng et al, 1996; Lin, 1997] found that the most frequently used meanings are accurate enough compared with more sophisticated methods.

For example, if we use "<" to represent the is-a relation, then WordNet would generalize "alphabet" as:

alphabet < character set < list < database < information < message < communication < social relation < relation < abstraction

Similarly, "symbol" would be generalized as:

symbol < signal < communication < social relation < relation < abstraction

Then "communication" would be the least general concept of "alphabet" and "symbol" in the hierarchy.

A problem that allows for gradual generalization using WordNet is that it introduces too many possibilities of generalization from two user queries. A keyword in one query may generalize with any other keyword in the other query using WordNet, introducing an explosive number of combinations. A heuristic is further introduced: *two queries can be generalized only when they have the same number of keywords, and only one keyword in the two queries is different*. That is, the two queries must be in the format of:

$keyword_1$, …, $keyword_i$, $keyword_j$
$keyword_1$, …, $keyword_i$, $keyword_k$
where $keyword_j \neq keyword_k$. A potential pattern is then produced:
$keyword_1$, …, $keyword_i$, [$concept_1$]

where $concept_1$ is the lowest concept which is an ancestor of $keyword_j$ and $keyword_k$ in the hierarchy. With the introduction of the concept hierarchy, more interesting patterns can be discovered. For example, from

Greek, alphabet
Greek, symbol
A generalized query pattern:
article ID ← Greek, [communication]

is produced. "[communication]" would also cover other keywords such as "document", "letter", and "myth", which might be contained in future user queries.

Obviously, when the generalized query patterns are cached for the search engine Encarta, WordNet API calls must be available for testing whether or not a keyword in a new user query is covered by generalized concepts. This can be optimized to have little impact on the speed of the cache.

4.5 Improvement 2: Flexible Generalizations

Due to the introduction of the hierarchy, pairs of one-keyword queries may now be generalized. As to which two queries are paired for generalization, the same greedy heuristic is used: *the pair that produces a pattern with the maximum coverage while the accuracy is above a threshold is chosen*. All queries covered by the new pattern are removed, and the process repeats until no new patterns can be generated.
As discussed in the Introduction, the goal of producing generalized query patterns is to have the product of the recall and precision of patterns as large as possible. Therefore, a further generalization is applied here. After a least general keyword in the WordNet is obtained from a pair of two keywords, it is further generalized by "climbing" up in the hierarchy of the WordNet until just before the accuracy of the pattern falls under the fixed threshold. This would produce patterns with a maximum recall while maintaining the adequate accuracy.

This improvement produces many useful generalized queries, such as:
"Democracy, of, [American_state]" from

- Democracy, of, California
- Democracy, of, Texes
- Democracy, of, Pennsylvania;

"[weapon_system]" from

- Gun
- Pistol
- Bullet
- firearm

"[European], unification" from

- German, unification
- Italian, unification

"[rank], amendment" from

- first, amendment
- fourteenth, amendment
- fifth, amendment

4.6 Improvement 3: Morphology Conversion

The second reason preventing useful generalization of the two queries is that minor differences in the spelling of the same keyword are regarded as different keywords. For example, "book" and "books"

are regarded as completely different keywords. Misspelled words are also treated as completely different words. This would prevent many possible generalizations since all other keywords except one in two queries must be exactly the same.

A simple morphology analysis using WordNet is designed to convert words into their “original” forms (i.e., stemming). For example, “books” to “book”, “studied” to “study”. We apply the conversion in our program before generalization is taken place. Currently we are working on spelling correction on the keywords in user queries.

With this improvement, we see user queries such as:

- populations, Glasgow
- population, Edinburgh

which cannot be generalized (because both keywords in the two queries are different) can now be generalized, and produce a generalized query pattern “population, [UK_region]”

4.7 Improvement 4: Synonym Conversion

A similar problem is that many keywords are synonyms: they have very similar meanings but with different spellings. For example, “Internet” and “WWW”, “penalty” and “punishment”, and “computer” and “data-processor” are all synonyms. Since any generalization of two queries requires all keywords except one must be the same, such synonyms should be recognized to allow more possible generalizations.

WordNet is again used for the synonym conversion. A dictionary of keywords appearing in user queries is dynamically created when the user logs are scanned. Before a new keyword is inserted into the dictionary, its synonyms are checked to see if any of them is already in the dictionary. If it is, the synonym in the dictionary replaces this keyword. This would reduce the size of the dictionary by about 27%. Even though the reduction is not huge, we found that the reduction happens on keywords that are very frequently used in the queries.

With this improvement, more useful generalized queries can be produced, which would not be possible previously. For example:

“DNA, [investigation]” from

- DNA, testing
- gene, analysis
- genetic, inquiry

4.8 Implementations

The pseudo-code of our log mining algorithm for Encarta with all improvements is presented in Table 3. The program is written in Visual C++.

Several further possible generalization methods are currently under investigation. For example, queries with different number of keywords, queries with two different keywords in the pair, and so on.

```
QuerySet = IIS log files                    /* Original user log of Encarta */
QuerySet = Filtering(QuerySet)              /* Producing list of user keywords and
click */
QuerySet = Stemming(QuerySet)               /* Improvement 3 (morphology
conversion) */
QuerySet = Synonym(QuerySet)                /* Improvement 4 (synonym conversion)
*/
QuerySet = sort(QuerySet)             /* Cluster queries by article ID */
For each article ID do
  Repeat
    Max_pattern = " "
    For each pair of queries with the same length and only one keyword different
      Temp_pattern = generalize(pair)       /* use WordNet to generalize the
keyword */
      While accuracy(climb_up(Temp_pattern)) > threshold do          /*
Improvement 2 */
```

Table 3. Implementation of the algorithm

4.8 Simulation Experiments

We have constructed an off-line mining system as described in the previous section. From the two-month user log files of a total size about 22 GB, it takes a few hours to mine all the patterns. Most of the time was spent on error calculation of each generalized query pattern, since it requires scanning all of the rest of the queries to see if it covers any "negative examples". The efficiency of the code would be improved since no attempt is made yet to optimize it. In the end, 5,000 query patterns are chosen whose accuracy is above 0.75 (a threshold we chose for the experiments).

We run several realistic simulations to see how our generalized query patterns help to improve the Encarta search engine, compared to a baseline cache which consists of the same number of the most frequent (ungeneralized) user queries. We partition the two-month log files into two batches: a training batch which is about 80% of all logs from the beginning, and a testing batch which is the last 20% of the logs. We then run our mining algorithm on the training batch, and test the patterns produced on the testing batch (last 20% of the queries unused in the training), simulating the actual deployment of the cache in the search engine after data mining has been conducted.
Again, recall and precision of the cache are calculated to measure the overall speed of the search engine with cache of the generalized query patterns. The overall saving would be larger if the product of recall and precision is larger, and if the *k* (the speed difference between regular search and cache) is larger.

4.9 Analyses of the Results

Table 4 presents our experiment results. As we can see, the recall (or hit ratio) of the cache does improve dramatically when more improvements in the mining algorithms are incorporated, and is much higher than the baseline cache, which consists of the same number of most frequent original user queries. On the other hand, the precision is virtually the same for different versions. This is actually within our expectation since patterns are chosen according to their coverage when their accuracy is above a fixed threshold (0.75). Overall, the best version produces the highest product of precision and recall. As we can see, the major improvement in recall comes from the generalization with hierarchy from the WordNet (improvements 1 and 2), and from synonym conversion (improvement 4).

If *k* is equal to 100, which is quite common as the speed up of using the cache vs the regular search engine, then the search engine with cache of the generalized query patterns would have an overall user search time of only 30% of the search engine without the patterns according to the formula $1 + 1/k - P\ R$ (see Introduction). That is, the search engine with the cache of the best generalized patterns is 3.3 times faster overall then the one without using the cache, and 3.1 times faster than the baseline cache of the most frequent user queries.

	Recall	Precision	Overall search time with cache, if no cache is 100%.
Baseline cache	7.31%	82.3%	95%
I: with hierarchy	41.7%	81.7%	67%
II: I + morphology analysis	52.8%	90.6%	53%
III: II + synonyms	80.4%	88.3%	30%

Table 4: Results of our log-mining algorithm for the Encarta search engine.

4.10 Relation to Previous Work

Many techniques used in this paper have been used (mostly on smaller problems though). We demonstrate that when those techniques are carefully integrated and tuned, they can solve this large-scale real-world problem.

Pattern learning discussed in this paper is similar to rule learning from labeled examples, a well studied area in machine learning. Many methods have been developed, such as C4.5 rule, the AQ family, and RIPPER. C4.5 rule [Quinlan 1993] is a state-of-art method for learning production rule. It first builds a decision tree, and then extracts rules from the decision tree. Rules are then pruned by deleting conditions that do not affect the predictive accuracy. However, C4.5 rule is limited to produce rules using only the original features (that is, only "dropping variables" is used in generalization), while our method also introduces generalized concepts in the hierarchy.

The AQ family [Michalski, 1983] generates production rules directly from data. It uses the "generate and remove" strategy, which is adopted in our algorithm. However, AQ is a top-down induction algorithm, starting from a most general rule and making it more specific by adding more conditions. AQ does not utilize concept hierarchy as we use in our generalization process. It seems difficult to incorporate concept hierarchy in the top-down learning strategy, as a large number of concepts in the hierarchy must be tried for making a rule more specific. The difference between our mining algorithm and RIPPER [Cohen, 1995] are also similar.

Concept hierarchy has been used in various machine learning and data mining systems. One usage is as background knowledge, as in [Han, Cai and Cercone, 1993]. Concept hierarchies have also been used in various algorithms for characteristic rule mining [Han, 1995, 1996; Srikant, 1995], multiple-level association mining [Han 1995], and classification [Kamber 1997]. What makes our work different is that our concept hierarchy is much larger and more general; it is generated automatically from WordNet over tens of thousands of keywords.

The improvements over the simple generalization algorithm are similar to approaches to the term mismatch problem in information retrieval. These approaches, called dimensionality reduction in [Fabio, 2000], aim to increase the chance that a query and a document refer to the same concept using different terms. This can be achieved by reducing the number of possible ways in which a concept is expressed; in other word, reducing the "vocabulary" used to represent the concepts. A number of techniques have been proposed. The most important ones are manual thesauri [Aitchison and Gilchrist, 1987], stemming and conflation [Frakes, 1992], clustering or automatic thesauri [Rasmussen 1992, Srinivasan 1992], and Latent Semantic Indexing [Deerwester et al., 1990]. These techniques propose different strategies of term replacement. The strategies can be characterized by (1) semantic considerations (manual thesauri), (2) morphological rules (stemming and conflation), and (3) term similarity or co-occurrence (clustering or

Latent Semantic Indexing). In our work, we used strategies similar to (1) and (2). The manual thesauri we used are WordNet. We also dealt with synonymous, but unlike strategy (3) which clusters similar terms based on co-occurrence, we used clusters of synonymous provided by WordNet directly. Our training data are user logs, which do not contain full text information of processed documents, which are necessary for co-occurrence estimation.

5. Conclusions

The purpose of this chapter is to advocate the discovery of actionable knowledge from Web logs. Actionable knowledge is particularly attractive for Web applications because they can be consumed by machines rather than human developers. Furthermore, the effectiveness of the knowledge can be immediately put to test, making the merits of the type of knowledge and methods for discovering the knowledge under more objective scrutiny than before. In this chapter, we presented two examples of actionable Web log mining. The first method is to mine a Web log for Markov models that can be used for improving caching and prefetching of Web objects. A second method is to use the mined knowledge for building better, adaptive user interfaces. A third application is to use the mined knowledge from a query web log to improve the search performance of an Internet Search Engine. In our future work, we will further explore other types of actionable knowledge in Web applications, including the extraction of content knowledge and knowledge integration from multiple Web sites.

References

[AWY 1999] Aggarwal, C. Wolf, J. L. and Yu, P. S. (1999) Caching on the World Wide Web. In IEEE Transactions on Knowledge and Data Engineering, volume 11, pages 94-107, 1999.

[Aitchison and Gilchrist, 1987] J. Aitchison, and A. Gilchrist, Thesaurus Construction. A practical manual. ASLIB, London, 2nd edition, 1987.

[AZN 1999] Albrecht, D., Zukerman, I., and Nicholson, A. (1999). Pre-sending Documents on the WWW: A Comparative Study. In Proceedings of the 1999 International Conference on Artificial Intelligence, *IJCAI99*, pp. 1274-1279, Sweden.

[AFJM 1995] Armstrong R., D. Freitag, T. Joachims, and T. Mitchell. (1995) *Webwatcher*: A learning apprentice for the World Wide Web. In *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, pages 6-12, Stanford University, 1995. AAAI Press.

[AFCDJ 1999] M. Arlitt, R. Friedrich L. Cherkasova, J. Dilley, and T. Jin. (1999) Evaluating content management techniques for web proxy caches. In *HP Technical report*, Palo Alto, Apr. 1999.

[BHL 2001] Berners-Lee, T., Hendler, J. and Lassila, O. (2001) The Semantic Web. Scientific American, 284(5):34--43, 2001.

[BP 1998] Brin, S. and Page, L. (1998) The Anatomy of a Large-Scale Hypertextual Web Search Engine Proceedings of the 7th WWW, Brisbane, 1998

[Boyan, 1996] J. Boyan, D. Freitag, and T. Joachims, A Machine Learning Architecture for Optimizing Web Search Engines, Proc. of AAAI Workshop on Internet-Based Information Systems, Portland, Oregon, 1996.

[Chidlovskii et al, 1999] B. Chidlovskii, C. Roncancio, and M.L. Schneider. Semantic cache mechanism for heterogeneous web querying. Computer Networks, 31(11-16): 1347-1360, 1999. Proc. of the 8th Int. World Wide Web Conf. 1999.

[CI 1997] Cao, P. and Irani, S. (1997) Cost-aware WWW proxy caching algorithms. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, pp. 193--206, December 1997.

[Cohen, 1995] W.W. Cohen, Fast Effective Rule Induction, Proc. of International Conference on Machine Learning, 1995.

[Craven 2000] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to Construct Knowledge Bases from the World Wide Web, Artificial Intelligence, 118(1/2):69-113. 2000.

[Deerwester et al., 1990] S. Deerwester, S.T. Dumais, T. Landauer, and Harshman, Indexing by Latent Semantic Analysis, Journal of the American Society for Information Science, 41(6):391-407, 1990.

[Fabio, 2000] C. Fabio, Exploiting the Similarity of Non-Matching Terms at Retrieval Time. Information Retrieval, 2, 25-45 (2000).

[Fayyad, et al, 1996] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996.

[Frakes, 1992] W.B. Frakes, Stemming Algorithm, In: W.B. Frakes and R. Baeza-Yates, Eds., Information Retrieval: data structures and algorithms, Prentice Hall, Englewood Cliffs, New Jersey, USA, ch.8, 1992.

[E 1996] Etzioni, O. (1996) The World Wide Web: Quagmire or Gold Mine? Communications of the ACM 39(11), November 1996, pp. 65-68.

[Google] http://www.google.com.

[JFM 1997] Joachims, T., Freitag, D.and Mitchell, T. (1997). WebWatcher: A Tour Guide for the World Wide Web. In Proceedings of The 15th International Conference on Artificial Intelligence (IJCAI 97), Nagoya, Japan; 1997

[Han, 1995, 1996] J. Han, Mining Knowledge at Multiple Concept Levels, Proc. of 4th Int. Conf. on Information and Knowledge Management, Maryland, 1995.

[Han, Cai and Cercone, 1993] J. Han, Y. Cai and N. Cercone, Data-Driven Discovery of Quantitative Rules in Relational Databases, IEEE Tran. On Knowledge and Data Engineering, 5(1):29-40. 1993.

[Kamber 1997] M. Kamber, L. Winstone, W. Gong, S. Cheng, J. Han. Generalization and Decision Tree Induction: Efficient Classification in Data Mining, Proc. of 1997 Int. Workshop on Research Issues on Data Engineering, Birmingham, England, 1997.

[Ling et. Al 2003] C.X. Ling, J. Gao, H. Zhamg, W. Qian, H. Zhang. Improving Encarta Search Engine Performance by Mining User Logs. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*. 2003

[Lin, 1997] D. Lin, Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. Proceedings of ACL/EACL-97, pp. 64-71, 1997.

[LMY 2001] Liu, B., Ma, Y. and Yu, P. (2001) Discovering unexpected information from your competitors' Web sites. Proceedings of the Seventh ACM Knowledge Discovery and Data Mining Conference. Pages 144-153

[LLLY 2002] Lou, W., Liu, G., Lu, H. and Yang, Q. (2002) Cut-and-Pick Transactions for Proxy Log Mining. In Proceedings of the 2002 Conference on Extending Database Technology. March 24-28 2002, Prague.

[Ng et al, 1996] H. T. Ng and H. B. Lee, Integrating Multiple Knowledge Sources to Disambiguate Word Sense: An Exemplar-Based Approach. Proceedings of 34th Annual Meeting of the Association for Computational Linguistics, pp. 44-47, 1996.

[Michalski, 1983] R. S. Michalski, A Theory and Methodology of Inductive Learning, Artificial Intelligence, 20(2):111-161, 1983.

[Miller, 1990] G. Miller, WordNet: an online lexicon database, International Journal of Lexicography, 3(4):235-312. 1990.

[PE 2001] Perkowitz , M. and Etzioni, O. (2001) Adaptive Web Sites: Concept and Case Study *Artificial Intelligence*, 118(1-2), 2000, PP. 245-275.

[PMB 1996] Pazzani, M.J., Muramatsu, J. and Billsus, D. (1996) Syskill & Webert: Identifying Interesting Web Sites. Proceedings of the American Association of Artificial Intelligence. 1996: 54-61

[PP 1999] Pitkow, J. and Pirolli, P. (1999) Mining Longest Repeating Subsequences to Predict World Wide Web Surfing. In *Second USENIX Symposium on Internet Technologies and Systems*, Boulder, C0, 1999.

[Pei, et al, 2000] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu, Mining Access Patterns efficiently from Web Logs, Proc. 2000 Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Kyoto, Japan, 2000.

[Perkowitz, 1997] M. Perkowitz and O. Etzion, Adaptive Sites: Automatically Learning from User Access Patterns. The Sixth International WWW Conference, Santa Clara, California, USA, 1997.

[Quinlan 1993] J. R. Quinlan, C4.5: Programs for Machine Learning, San Mateo: Morgan Kaufmann, 1993.

[Rasmussen, 1992] E. Rasmussen. Clustering Algorithm. Chapter 16 in: W.B. Frakes and R. Baeza-Yates, editors, Information Retrieval: data structures and algorithms, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1992.

[SCDT 2000] Srivastava, J., Cooley, R., Deshpande, M. and Tan, P. (2000) Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. In ACM SIGKDD Explorations, (1) 2, 2000.

[SKS 1998] Schechter, S., Krishnan, M. and Smith, M.D. (1998) Using Path Profiles to Predict HTTP Requests. In Proc. *7th International World Wide Web Conference*, Brisbane, Qld., Australia, April 1998, pp. 457--467.

[SP 2000] Spiliopoulo, M. and Pohle, C. (2000) Data Mining for Measuring and Improving the Success of Web Sites. *Data Mining and Knowledge Discovery Journal*. Kluwer. pp. 85-114. 2000.

[Srikant, 1995] R. Srikant and R. Agrawal, Mining Generalized Association Rules, Proc. 1995 Int. Conf. Very Large Data Bases, Zurich, Switzerland, 1995.

[Srinivasan, 1992] P. Srinivasan. Thesaurus Construction, In: W.B. Frakes and R. Baeza-Yates, Eds., Information Retrieval: data structures and algorithms, Prentice Hall, Englewood Cliffs, New Jersey, USA, ch.9, 1992.

[Sundaresan and Yi, 2000] N. Sundaresan and J. Yi, Mining the Web for Relations, The Ninth International WWW Conference, Amsterdam, The Netherlands, 2000.

[Zaine et al, 1998] O. R. Zaine, M. Xin, and J. Han, Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs, Proc. Advances in Digital Libraries Conf. (ADL98), Santa Barbara, CA, April 1998, pp. 19-29.

[Saraiva et al, 2001] P.C. Saraiva, E.S. Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-Preserving Two-Level Caching for Scalable Search Engines. ACM SIGIR 24th Int. Conference on Information Retrieval, 51-58, 2001

[SYX 2002] Su, Z., Yang. Q., Zhang, H., Xu, X. and Hu, Y and S.P. Ma. (2002) Correlation-based Web-Document Clustering for Web Interface Design. *International Journal Knowledge and Information Systems*. (2002) 4:141-167. Springer-Verlag London Ltd. 2002

[SUZ 2000] Su, Z., Yang, Q., and Zhang, H. (2000) A Prediction System for Multimedia Pre-fetching in Internet. In *Proc. 2000 Int'l ACM Conf. on Multimedia*, Los Angeles, California, *2000.*

[YZ 2001] Yang, Q. and Zhang, H. (2001) Integrating Web Prefetching and Caching Using Prediction Models. ***World Wide Web Journal***. Kluwer Academic Publishers. Vol. 4 No. 4,. Pages 299—321.

[YZL 2001] Yang, Q., Zhang, H. and Li, I.T. (2001) Mining Web Logs for Prediction Models in WWW Caching and Prefetching . In the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'01, August 26 - 29, 2001 San Francisco, California, USA.

[ZXH 1998] Zaïane, O., Xin, M. and Han, J. (1998) Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs, in Proc. *ADL'98* (Advances in Digital Libraries), Santa Barbara, April 1998