1000-9825/2002/13(06)1040-10 ©2002 Journal of Software 软 件 学 报 Vol.13, No.6

Prediction for Visiting Path on Web*

HAN Jing[1], ZHANG Hong-jiang[2], CAI Qing-sheng[1]

[1](*Department of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei* 230027, *China*);

[2](*Media-Computing Group*, *Microsoft Research Asia*, *Beijing* 100080, *China*)

E-mail: hanjing@mail.ustc.edu.cn

http://www.ustc.edu.cn

Received December 20, 2000; accepted September 24, 2001

Abstract: When Internet users are browsing the Web, they always have to wait for the next page to come up after selecting the link to the page, even though the Web server is sometimes idle. To reduce the perceived response time, the server is required to predict users' next HTTP request and pre-process the Web page with remaining CPU resources. This paper introduces various methods to predict users' next HTTP request using classified Web page information, user profiles, and Web logs. Sixteen algorithms and methods are analyzed and compared in this paper. Experimental results show that some methods produce highly probable results with the user of user profiles and classified Web page information.

Key words: Web page; http request; prediction; user profile

When the first World Wide Web site came online, the entirety of the content handled by Hypertext Transfer Protocol (HTTP) servers was static. Due to factors such as network speed, server load, and the dynamic nature of the content, after users click a link, they always have to spend a certain amount of time waiting for the next page to come up. Some current efforts to resolve this issue are: caching at the server and client to reduce the latencies of user requests, putting an additional layer of cache — proxy between the client and server, increasing the speed of dynamic content generators, etc. However, there is a limit to the amount by which these techniques can decrease the perceived response time, as it requires the server to wait for a user's request before it starts the process. Moreover, Webmasters often use the following rule of thumb to ensure that HTTP server performance will not degrade when network traffic is heaviest — provide twice the server capacity required to handle a site's average load. As a result the server spends half of its CPU cycles idle waiting for a request[1]. If a Web server can predict the next HTTP request from a user in its idle state, it can then pre-process (pre-generate and pre-fetch) dynamic content before a user makes the request. This will reduce the latency of requests that are predicted correctly. Since every prediction costs CPU resources on the Web server, this paper tries to decrease the cost in the following ways: use simple algorithms for prediction, and move as much work (such as studying the logs) as possible to midnight when the server is not busy. On the other hand, pre-fetching will increase the load on the Web server and occupy some bandwidth, so accuracy is very important. To avoid this side effect, server will not pre-fetch during network rush

* This research was done in Microsoft Research Asia, Beijing 100080. Supported partly by the National Natural Science Foundation of China under Grant No.70171052 (国家自然科学基金)

HAN Jing was born in 1974. She is a Ph.D. candidate at the Department of Computer Science, USTC. Her current research interests are machine learning, agent, artificial life and CSPs. **ZHANG Hong-jiang** was born in 1960. He is the Vice-Dean of Microsoft Research China, leader of Media-Computing group in Microsoft Research China. He is the senior member of IEEE and ACM. He has published 2 books, more than 70 papers and owns 14 patents. **CAI Qing-sheng** was born in 1938. He is a professor and Ph.D. supervisor in the AI Laboratory of USTC. His research interests include artificial intelligence, machine learning and knowledge discovery.

hours.

There have been several attempts at predicting HTTP requests. One attempt is to focus on pre-fetching images that are referenced in HTML documents (http://shika.aist-nara.ac.jp/products/wcol/). Stuart[1] predicts the user’s next request entirely based on logs of paths the user has visited and point profiles, but ignores some useful information about the Web page content and user’s personal preference. Other related pieces of work are recommendation systems, collaborative filtering and user clustering.

Recommendation systems are somewhat like prediction systems, but totally different in nature. A recommendation system finds the best choice for the current user while a prediction system finds the most probable choice. The former will affect the user’s browsing behavior but the latter will not. When a recommendation system predicts the next best link, it will tell the user to follow it. In some cases, the user is willing to accept the recommendation although he/she in fact has had different ideas in mind. Therefore, it can be concluded that the accuracy of prediction system is lower than the recommendation system. Prediction is more difficult. There are many examples of recommendation systems: WebWatcher by CMU (http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-6/web-agent/www/project-home.html), Syskill & Webert by UCI (http://www.ics.uci.edu/~pazzani/Publications/Publications.html), Letizia and Let’s Browse by MIT (http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia.html), Fab by Stanford (http://rsv.ricoh.com/~marko/), Softbots by Washington University (http://www.cs.washington.edu/research/projects/softbots/www/projects.html), WBI by IBM (http://www.almaden.ibm.com/cs/wbi/).

Most of the above systems use Collaborative Filtering[2] techniques. There are two kind of systems: memory-based and model-based. A memory-based system uses past records to predict the active user’s vote. Two kinds of methods are used for the prediction: one is called correlation, and the other is called vector similarity. A model-based system tries to build a prediction model, such as a Clustering Model or a Bayesian Network Model. A clustering model provided by Microsoft Research[3] to cluster users in our experiments. It turns out that 70% of the users are clustered into one group, so this model is not suitable for this problem.

In this paper we focus on using classified Web page information and user preference to make a prediction. The cost of classifying Web pages is one-off and it is not necessary to classify for every prediction, so this cost can be ignored in the cost of prediction. The problem is that it is not easy to get users’ profiles if they are visiting a general Web site. Only for professional Web sites are users required to provide their profiles. To solve this problem, we try to use the Web logs to rebuild the user vector in Section 2.4.2. The experimental result in section 4 shows that the prediction accuracy will not vary much.

Given some classified Web pages, some users’ profiles and their Web logs, we try to predict a new user’s highly probable next HTTP request. This paper analyzes 16 simple algorithms and methods. We then compare these 16 algorithms and methods and find that link-based algorithms and combined methods are the most efficient.

The problem is defined in Section 1. The 16 algorithms and methods for prediction are analyzed in Section 2. The experimental setup, results and analysis are presented in Section 3. Conclusions and future work are presented in Section 4.

1 Problem Definition

A smart Web server should have the ability to assist users while they are browsing. To implement this assistance ability, we can put a proxy between the Web server and the user. A Web server with a proxy that can provide assistance to users is called a smart server. In this paper, the prediction functionality is implemented in a proxy on the server’s side (See Fig.1.). We classify Web pages into n basic types according to the nature of their contents. On the other hand, user profile is known to a certain degree. The proxy holds the classified information of

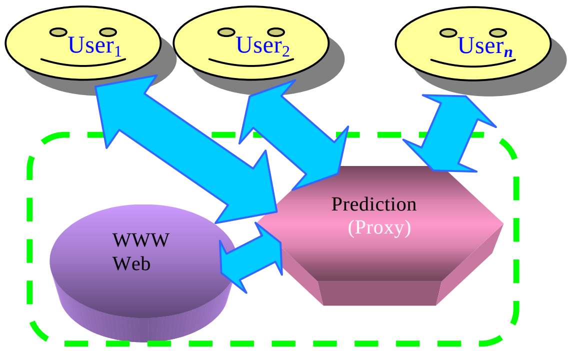


Fig.1 Architecture of smart Web server

Web pages, the current user's profile and the Web logs. After studying past user profiles and the corresponding Web logs, the proxy is able to build a prediction model to predict other new users' browsing paths.

The problem can be defined as the following:

Given:

1. User profiles for their preference;
2. Classified information of Web pages;
3. Some Web logs;

Goal:

Through the study of some training data (past user profiles and their Web logs), predict a new user's next HTTP request (see Fig.2.).

In this paper, we build a mathematical model for this problem using vectors.

a) Web site: A directional graph. Each node *p* of the graph represents a Web page (see Fig.3). There is a directional edge $l(p_1,p_2)$ from node p_1 to node p_2 if there exists a hyperlink from page p_1 to page p_2.

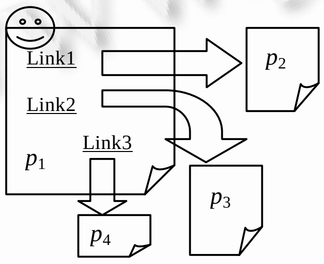


Fig.2 User is reading page p_1, next HTTP request: Link1? Link2? Link3?

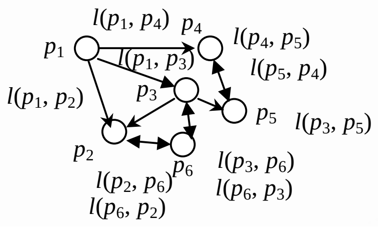
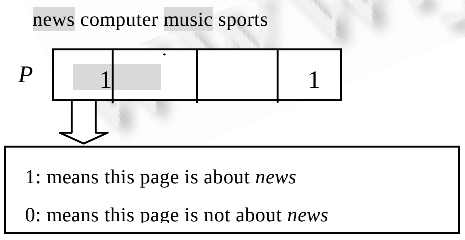


Fig.3 Graph for a simple Web site

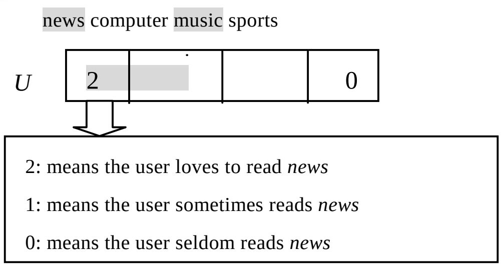
b) Web page: Suppose there are n basic types of contents. So a page can be described as an *n*-dimension vector $P=(r_1,r_2,\ldots,r_n)$, with each bit (r_i=0/1) of the vector representing the corresponding type (See Fig.4). For example, suppose there are 4 basic types of contents: news, computer, music, and sports. If the content of a page is about news and computer, it can be represented as a vector of *P*='1100'.



1: means this page is about *news*

0: means this page is not about *news*

Fig.4 Page vector



2: means the user loves to read *news*

1: means the user sometimes reads *news*

0: means the user seldom reads *news*

Fig.5 User vector

c) User: Suppose there are m (m is always equal to n) basic types of user preferences. A user can be described as an m-dimensional vector $U=(m_1,m_2,\ldots,m_m)$, with each bit (m_i=0/1/2) of the vector representing the degree of interest in the corresponding type of content (See Fig.5). For example (using the above assumption), if the user loves to read news and is willing to read music pages, but has no interest in computer and sports, he can be represented as a vector U=‘2010’.

d) Hyperlink: $l(p_1,p_2)$ is a hyperlink from page p_1 to page p_2, like the Web page vector, we can define a link vector L to represent the hyperlink. (We call p_1 ‘link-from page’ and p_2 ‘link-to page’ in this paper.) However, there are always only a few words underlined that describe a hyperlink. Furthermore, words underlined that describe a hyperlink are always very simple, such as ‘Next page’, ‘>> More’, ‘Return’, ‘Important’, ‘What’s new’, etc. It is too difficult to know what exact type of link is from those underlined words. So retrieving the link vector from underlined words is not as simple as retrieving the page vector from page content. Even though we can retrieve the link vector from the vector of page that is linked to, some effect of link vector in prediction will be lost since it is the same as page vector. Because vectors of two different links with same link-to page, $l_1(p_1,p_3)$ and $l_2(p_2,p_3)$, should be different. But the link vectors will be the same if they are retrieved by the link-to page p_3. So we must find other ways to retrieve the link vector which can fit the prediction problem. In Section 2, we will discuss this problem.

e) Visited (u,p), Visited (u,l): In the Web logs, if the user u has ever visited page p or link l, the function Visited(u,p) or Visited(u,l) will return 1; otherwise 0.

2 Algorithms and Methods

The basic idea of the prediction algorithm is: compute a score for each link on the current page being read, and select the link with the maximum score. The score for each link is computed based on the current user’s profile, the current user’s history (Web logs for pages the user has visited during this time of browse), the classified information of the current page (vector of the current page), and some training data (past user profiles and their Web logs). We can define a function φ for computing a score for a link:

$$\varphi(U_x,L_m(P_i,P_j),W_x,M):\{U\}\times\{L\}\times\{W\}\times\{M\}\rightarrow[0,1].$$

U_x is the current user vector. P_i is the current page being read. $L_{\mathrm{m}}\in\vartheta_i$, ϑ_i is a set containing vectors of all links on page p_i. $P_j\in\Gamma_i$, Γ_i is a set containing link-to-page vectors for all links from page p_i. W_x is the history vector for user U_x, which are Web logs of this time of browse. M is a knowledge model which is extracted from the training data. $\varphi(U_x,L_m(P_i,P_j),W_x,M)$ returns a score for link L_{m} which is from current page p_i to p_j for user U_x. The prediction result can be defined as

$$\xi:\ \{U\}\times[1,n]\rightarrow\{l\},$$

where n is the total number of pages. $\xi(U_x,i)$ always returns a link on page p_{i} depending on scores computed by the φ function:

$$\xi(U_x,i)=l_y,\ \text{such that}\ \forall L_u\in\vartheta_i,\ \varphi(U_x,L_y,W_x,M)\geq\varphi(U_x,L_u,W_x,M).$$

In the following section, we will introduce 16 prediction algorithms and methods. We classify them into five groups: Simple Algorithms, Page-Based Algorithms, Link-Based Algorithms, User-Based Algorithms, and Combined Methods. The first 4 groups are called pure algorithms. All the algorithms and methods will compute the score for each link using different methods and/or select a link in different ways, that is to say, function φ and (or) ξ are different. In section 2.6, we will analyze the computational complexity for each algorithm and method.

2.1 Simple algorithms (for comparison purpose only)

2.1.1 Random (RD)

Score for each link is randomly produced and the system chooses the link with the maximum score as the prediction result.

$\varphi_{\text{-RD}}(U_x,L_m(P_i,P_j),W_{\mathrm{x}},M)$=random(0,1), where random(0,1) returns a value between 0 and 1 at random.

Apparently, this algorithm is bad in performance, but easy to implement.

2.1.2 Similarity between user and page (SS)

Another obvious way is to compute the similarity between the current user vector U_x and the link-to-page vector P_j by Cosine (U_x,P_j), and then choose the link whose link-to-page vector resembles the current user vector the most.

$$\varphi_{\text{-}SS}(U_x,L_m(P_i,P_j),W_x,M)=Cosine(U_x,P_j),$$

where $Cosine(V_1,V_2)=V_1\cdot V_2/|V_1||V_2|=\sum_j (v_{1j}\, v_{2j})/\sqrt{\sum_k V_{1k}^2}\sqrt{\sum_k V_{2k}^2}$.

This algorithm seems reasonable since a user who likes sports will most likely click the link to a sports page. However, the experiment shows that is not the case. Because the user can never give a clear profile for his/her real preference, and pages always contain more than one kind of content, the user's choice will be affected.

2.2 Page-Based algorithms

In order to predict the next HTTP request, that is to say, predict which page the user will read next, we can focus on linked-to pages for each link in the current page. In this section we will introduce some algorithms based on linked-to pages.

2.2.1 Visited page counter (PC)

Users will read very attractive pages no matter what kind of content these pages have. A user who is bored with music may read sports pages to learn how many gold medals the Chinese team has won in the Olympic Games. So the more popular the page is, the greater reading probability is. Therefore, we can count how many people have read each page and select a link to a popular page as the prediction result.

$$Pop\,(P_j,M)=\varphi_{\text{-}PC}(U_x,L_m(P_i,P_j),W_x,M)=\Sigma_{U\,k\in M}\,Visited(u_k,p_j)$$

Pop function is always used in combination with other algorithms since it uses only one kind of information. Used alone, this algorithm is bad in performance, but it can be used as a weight (WGT) to improve the performance of other algorithms.

2.2.2 Page vector rebuild-group page by grouped users (PV)

It is tedious to classify pages and retrieve page vectors. Neither by man nor by computer can pages be classified perfectly. Since the system has user profiles, vectors of users who have read page *p* can be used to rebuild vector *P*:

$$P'=\Sigma_{U\,k\in M}U_k\times Visited(u_k,\,p).$$

We can then use algorithm 3.1.2 to make the prediction:

$$\varphi_{\text{-}PV}(U_x,i,P_j,L_m,W_x,M)=Cosine(U_x,P_j')=Cosine(U_x,\Sigma_{U\,k\in M}U_k\times Visited(u_k,\,p_i)),$$

or if we consider the popularity of the page:

$$\varphi_{\text{-}PV}(U_x,i,P_j,M)=Cosine(U_x,P_j')\times Pop(P_j,M).$$

2.2.3 Page neighborhood (PN)

If page p_1 is similar to p_2, the user who has read p_1 will probably also read p_2. This idea inspires another algorithm: compute the similarity between each linked-to page and pages the current user has read, and then select the one with the most similarity as the prediction result.

$$\varphi_{\text{-}PN}(U_x,L_m(P_i,P_j),W_x,M)=\Sigma_{P\,k\in Wx}Cosine(P_k,P_j)\times Visited\,(u_x,p_k).$$

2.3 Link-Based algorithms

Unfortunately, the underlined words designating a link do not really tell the user what kind of content the next page contains, not to mention these words can sometimes be misleading. Since all we want to know is which link on the current page the user will follow, the prediction can be made by considering the information of links. This approach is more straight-forward and accurate than considering linked-to pages.

2.3.1 Visited link counter (LC)

Obviously, we can count the number of hits for each link just as what we did in Section 2.2.1.

$$Pop(L_m,M)=\varphi_{\text{-}LC}(U_x,L_m(P_i,P_j),W_x,M)=\Sigma_{U\,k\in M}\,Visited(u_k,l_m).$$

2.3.2 Link vector-group link by grouped user (LV)

Recall in the *mathematical model* of Section 1 we discussed the difficulty in retrieving a link vector. It is not a good idea to assign the value of from-to page vector to the link vector. Since user vectors are available, we can get the link vector from vectors of users who have clicked the link before:

$$L=\sum_{U\,k\in M} U_k\times Visited\,(u_k,l).$$

Then we can compute the similarity between the current user vector and each link vector, and then select the link with the most similarity as the prediction result.

$$\varphi_{\text{-}LV}(U_x,L_m(P_i,P_j),W_x,M)=Cosine(U_x,L_m)=Cosine(U_x,\textstyle\sum_{U\,k\in M}U_k\times Visited(u_k,l_m)),$$

or if we consider the popularity of the page:

$$\varphi_{\text{-}LV}(U_x,L_m(P_i,\,P_j),W_x,M)=Cosine(U_x,L_m)\times Pop\,(P_j,\,M).$$

The experiment shows this algorithm is the most efficient among singular algorithms.

2.3.3 Link neighborhood (LN)

Similar to the *Page Neighborhood Algorithm*, if link l_1 is similar to l_2, and the user has clicked l_1, he/she will probably also click l_2. Therefore, we can first get the link vectors using user vectors as mentioned in Section 2.3.2, and then compute the similarity between each link and links the current user has clicked.

$$\varphi_{\text{-}LN}(U_x,L_m(P_i,P_j),W_x,M)=\sum_{L\,k\in Wx}Cosine(L_k,L_m)\times Visited(u_x,l_k).$$

2.4 User-Based algorithms

As mentioned above, user profiles do not always reflect what the user really wants while he/she is browsing the Web. One reason is that it is impossible to get an exact profile and the user is tired of giving a detailed profile before browsing. The other reason is that there are more factors other than personal preference which affects the user’s browsing pattern, and the user’s browsing pattern will change. In this part, we will introduce some user-based algorithms.

2.4.1 User mapping (UM)

Given the user vector, the user’s browsing behavior is not always in accordance with the preference described by this vector on all Web sites. Here we try to use the past Web logs to map the user vector to another space which is more related to the current Web site. We first extract a matrix from the past Web logs *M*, and then get a new vector by mapping:

$$Matrix=\sum_{U\,k,P\,j\,\in M} U_k\cdot P_j\times Visited\,(u_k,\,p_j),$$
$$U'=U\cdot Matrix.$$

We then take new vector *U′* instead of *U* to make prediction using page-based algorithms and link-based algorithms. Experiments show that accuracy increases in page-based algorithms but decreases in link-based algorithms.

2.4.2 User vector rebuild-group user by grouped pages/links (UVP/UVL)

Since the user profile does not always reflect what the user really wants, we can rebuild the user vector with his/her visiting history *W* and then use it to make prediction. A user vector built from page vectors is called UVP, and a user vector built from link vectors is called *UVL*.

$$U'=\sum_{Pi\in W}P_i\times Visited(u,p_i)\quad \text{------ UVP}$$
$$\text{or } U'=\sum_{Li\in M}L_i\times Visited(u,l_i)\quad \text{------ UVL}$$
$$\varphi_{\text{-}UVP}(U_x,L_m(P_i,P_j),W_x,M)=Cosine(U_x,P_j)\times Pop(P_j,M)$$

2.4.3 User neighborhood (UNP/UNL)

If user vector u_1 is similar to u_2, user u_2 will probably select link *l* or page *p* if u_1 has selected *l* or *p*. Based on this assumption, we can calculate

$$\varphi_{\text{-}UN}(U_x,L_m(P_i,P_j),W_x,M)=\sum_{U\,k,P_j\in M}Cosine(U_x,U_k\,)\times Visited(U_k,P_j)\quad \text{----- UNP}$$

or $\varphi_{\text{-}UN}(U_x,L_m(P_i,P_j),W_x,M)=\sum_{Uk,L\,j\in M}Cosine(U_x,U_k)\times Visited(U_k,L_j)$ ----- UNL

Formulae *UNP*/*UNL* take page/link popularity into account although there is no pop function present, because they calculate the sum, not the average.

2.4.4 User vector online learning (UVO)

As mentioned above, user preference is dynamic. It is not wise to treat user preference as a static vector. We can update the user vector online by considering pages the user has read (not just clicked), only pages the user has read for more than t seconds (suppose $t = 5$, t=10, etc.). The vector is updated after he/she has read a page p:

$$U'=U+P\times Read(u,p).$$

Then we can use page-based or link-based algorithms to make the prediction.

2.5 Combined methods

Up till now, we have introduced 13 pure algorithms. In this part, we will introduce some combined methods with higher prediction accuracy.

2.5.1 Top two (T2)

A simple way to increase prediction accuracy is to first calculate the scores by some pure algorithms (e.g. PV), and then select the top two score links as the prediction result. It can obviously increase the accuracy. But the tradeoff is that the server will have to pre-fetch two pages; the server's workload will increase.

2.5.2 Hybrid (HB)

Since PV, LV and LC algorithms have relatively good performances, we want to find a way to mix them together. First, we use PV, LV and LC to calculate scores for link L_m respectively, and then compute the final score of link L_m as follows:

$$\varphi_{\text{-}HB}(U_x,L_m(P_i,P_j),W_x,M)=c_1\times\varphi_{\text{-}PV}(U_x,L_m(P_i,P_j),W_x,M)+c_2\times\varphi_{\text{-}LV}(U_x,L_m(P_i,P_j),W_x,M)+$$
$$c_3\times\varphi_{\text{-}LC}(U_x,L_m(P_i,P_j),W_x,M),$$

c_1, c_2 and c_3 are the respective weights for each algorithm. The better the algorithm, the more weight. In this case, $c_2>c_3>c_1$.

2.5.3 Agreed (AD)

In order to increase accuracy, we tighten the restrictions on prediction: a prediction is final if and only if several algorithms reach the same result: $\xi_{\text{-}PV}=\xi_{\text{-}LV}=\xi_{\text{-}LC}$. For example, we use *PV*, *LV* and *LC* to calculate scores respectively, and if link l gets the highest scores in all cases, l becomes the predicted link. Otherwise, no prediction will be made. So,

$$\xi_{\text{-}AD}(U_x,i)=\begin{cases} l_y & \text{if } \forall L_u\in\vartheta_i, \varphi_{-PV}(U_x,L_y,W_x,M)\ge\varphi_{-PV}(U_x,L_u,W_x,M) \\ & \text{and } \varphi_{-PV}(U_x,L_y,W_x,M)\ge\varphi_{-LV}(U_x,L_u,W_x,M) \text{ and } \varphi_{-LC}(U_x,L_y,W_x,M)\ge\varphi_{-LC}(U_x,L_u,W_x,M) \\ \text{no prediction,} & \text{otherwise}\end{cases}.$$

2.6 Computational complexity

In this section we will analyze the computational complexity for computing the score for a link (see Table 1). Since some computing tasks can be arranged to carry out off-line outside rush hours, such as creating the *UM Matrix*, rebuilding page vectors, and building link vectors, we can ignore these computations in Table 1. Generally speaking, n is smaller than 50, so all the computational complexities in table 1 are small.

Table 1 Computational complexity for computing score for a link in 16 algorithms and methods

Algorithm	RD	SS	PC	PV	PN	LC	LV	LN
Complexity	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(w_n)$	$O(1)$	$O(n)$	$O(w_n)$
Algorithm	UM	UVL	UVP	UN	UVO	T2	HB	AD
Complexity	$O(n)$	$O(n)$	$O(n)$	$O(s_n)$	$O(n)$	$O(2n)$	$O(2n)$	$O(2n)$

A page or user vector contains *n* bits for *n* basic types. *w* is the length of user history. *s* is the number of other users who have also followed this link.

3 Experiments and Results

The previous section has provided 16 algorithms and methods for predicting a user's browsing path on the Web. In this section we present experimental results on the data of 49 users' Web logs for visiting www.east.net.cn inside Microsoft Research China. We compare performances of these algorithms and methods, and find that some of them produce highly probable predictions.

3.1 Experimental setup

The predictability of HTTP requests was measured using training and testing data consistent with the rules of cross-validation. We used separate data sets to construct and test the prediction model *M*. We downloaded a mirror Web site of www.east.net.cn (accessed in July 26, 1999), removed multi-frame pages, retrieved Web page vectors, and extracted the directional graph from the structure of the Web site. The mirror Web site contained 300 pages and 800 links. We then collected 49 users' 3800 Web logs and their profiles. The Web logs and profiles of these 49 users were split into two groups, and 8 randomly selected users' Web logs and their profiles were used as training data. We tested 100 random partitions in the experiment and calculated the average performance. Both the page vector and the user vector contained 30 bits, 1 bit each for: sports, travel, game, music, films, cartoon, politics, economy, news, military affairs, law, science and technology, computer, internet, zoology, education, books, company, finance and investment, advertisement, literature, painting and calligraphy, design, character, traditional culture, foods, shopping, life, medical treatment, and others. For each bit in the user vector, 0 means dislike, 1 means occasional interest, and 2 means like. For each bit in the page vector, 1 means the page is related to this type of content, 0 otherwise.

3.2 Results and analysis

During each test, we respectively reported the percentage rate of accuracy and recall, as well as the average among 100 random partitions:

$$\text{Accuracy}=\text{Total Correct predictions}/\text{Total predictions}\times 100\%,$$

$$\text{Recall}=\text{Total predictions}/\text{Total HTTP Requests}\times 100\%.$$

3.2.1 Pure algorithms

Figure 6 shows the average *accuracy* of pure algorithms. The *recall* is 100% for pure algorithms since they make a prediction for each request. From this chart, we can draw two conclusions: (1) Accuracy of pure algorithms is not high, (2) Link-based algorithms have the better performance among pure algorithms; the best one is Link Vector (LV, 30.4%).

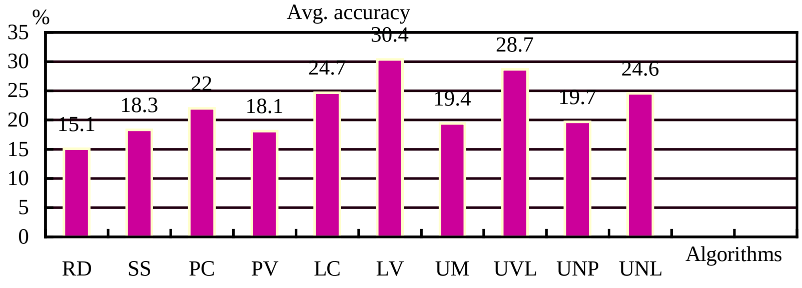


Fig.6 Avg. accuracy of pure Algorithms, recall is 100%

3.2.2 Combined methods

From Table 2, we can see: (1) Combined methods can increase the accuracy, especially AD (79.1%), better than the algorithm provided by Stuart[1]. The reason is that Stuart' prediction is entirely based on visited paths, but ignores other useful information such as Web page content types and user preferences. (2) The tradeoff of high accuracy is low recall. Although AD has high accuracy, its recall is low (17.4%).

Experiments also show that for the HB method, the following way seems effective for getting the values of c_1, c_2, and c_3: first compute the accuracy A_{PV}, A_{LV} and A_{LC} of PV, LV, and LC respectively with training data, and then assign $c_1=A_{PV}$, $c_2=A_{LV}$, $c_3=A_{LC}$.

For the AD method, we compare these three combinations: PV-LV-LC, LV-LC, and PV-LV. LV-LC and PV-LV are variations of AD: PV-LV means the system will make a prediction only if PV and LV produce the same result; LV-LC means the system will make a prediction only if LV and LC produce the same result. Fig. 7 shows their respective results. We can draw the following conclusions: (1) Recall: PV-LV-LC<LV-LC<PV-LV; (2) Accuracy: PV-LV-LC>LV-LC>PV-LV; (3) Higher accuracy, lower recall.

3.2.3 User vector built from page

We find that it is very difficult to get user profiles because users are impatient while they are filling out profile forms and they are unable to give a perfect profile even though they want to. So maybe we can retrieve the user vector without any user profiles. One way is to build the user vector from page vectors, as mentioned in Section 2.2.4. (UVP / UVL). In this experiment (see Fig. 8), we use page vectors to rebuild the user vector and test SS, PV, LV, UM, T2, HB, and AD with the new user vector U'. We find that the accuracy decreases a little in LV, UM and T2, but increases a little in SS, PV, HB, and AD. Therefore, in the future work and experiments we can build the user vector from page vectors, which makes users happy because they no longer have to fill out a form before browsing.

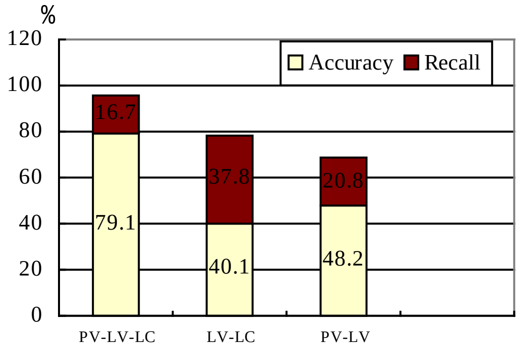


Fig.7 Avg. accuracy and recall of 3 kinds of AD

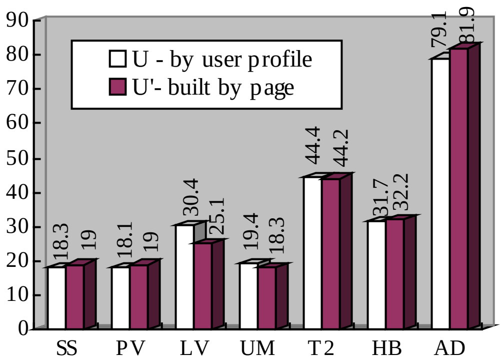


Fig.8 Comparison of avg. accuracy for U and U′

4 Conclusions and Future Work

Given that most servers spend a significant portion of CPU cycles idle, we should take advantage of this predictability and use these free cycles to hide the latency of page generation and other pre-processes as much as possible. Web applications can pre-process predicted pages for their clients before the pages are even requested by the user. This is very helpful in reducing the waiting time for the user. The core of this is prediction. In the above sections, we have analyzed and compared 16 algorithms and methods for predicting the user's next HTTP request. LV is the best one among pure algorithms, but is still not good enough (accuracy is 31%). Combined methods increase the accuracy, especially for AD, whose accuracy is 80%, which is higher than that of the Stuart method whose prediction is entirely based on logs of paths visited. Although recall of AD is low, it will not do harm to the server. With no prediction the server remains idle during the idle CPU cycles, but wrong prediction will increase the server load. To best utilize a limited number of idle CPU cycles, we can use a heuristics of predicting pages with the AD method in the idle CPU cycles. There are a lot of enhancements to be done in the future: (1) Improvement on prediction**.** For existing algorithms, we can consider the improvements on pop functions, AD+TT, bit value of vector from 0/1 to –1/1, better weight setting in HB, inverse user frequency, etc. The user's choice is not only affected by his/her personal preference (long term/short term, changeable/static), but also by the link's attractiveness (underlined words and their positions on the screen) and the time of browse (weekday/weekend, daytime/night). We can also try other methods to make the prediction, such as Decision Tree, Naive-Bayesian, Bayesian Network, Reinforcement Learning, Alife Techniques, even adaptive agent for user, adaptive agent for page. Another consideration is the selection of training samples: we can find good samples from experiments or samples produced by experts. (2) Improvement on experiment design**.** As mentioned above, in the future experiment, the user need not give a profile, and we can get the user profile from pages he/she has read. We also plan to test data from other Web sites such as MS.COM. (3) Improvement on pre-fetching. The most important thing for prediction is accuracy. To increase accuracy, one way is to use agreed algorithms. Other possible ways are: Top1+Top2, threshold (needs learning) for prediction made, combination of prediction and recommendation, etc.

Acknowledgement Lie Gu has helped us to implement a browser for experiment. Machine Learning and Applied Statistics Group (MLAS) of Microsoft Research Redmond provided an executive program for clustering method. Thanks to 48 staff and interns in Microsoft Research Asia who joined in the experiment.

References:

[1] Schechter, S., Krishnan, M., Smith, M.D. Using path profiles to predict HTTP requests. In: Proceedings of the 7th International World Wide Web Conference. Brisbane: Elsevier, 1998. http://www7.scu.edu.au/programme/posters/1839/com1839.htm.

[2] Breese, J., Heckerman, D., Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. Technical Report, MSR-TR-98-12, Microsoft Research, 1998.

[3] Cadez, I., Heckerman, D., Meek, C., *et al*. Visualization of navigation patterns on a Web site using model based clustering. Technical Report, MSR-TR-00-18, Microsoft Research, 2000.

网站浏览路径预测

韩 靖[1], 张宏江[2], 蔡庆生[1]

[1](中国科学技术大学 计算机科学技术系,安徽 合肥 230027);

[2](微软亚洲研究院 多媒体计算组,北京 100080)

摘要: 用户浏览因特网网站点击某个链接后,该链接的网页往往需要一段时间才能传送到用户端.为了缩短用户的等待时间,利用网站服务器的空闲CPU时间,对用户下一个HTTP请求进行预测,以便预处理下一个网页是一种可能途径.使用已分类的网页信息、用户配置文件和网站日志进行预测,介绍并且分析了16种预测算法.实验结果证明某些算法是比较有效可信的.

关键词: 网页;HTTP 请求;预测;用户配置文件

中图法分类号: TP393 文献标识码: A