KDD-99 Workshop on Web Usage Analysis and User Profiling (to appear)

A non-invasive learning approach to building web user profiles

Philip K. Chan (pkc@cs.fit.edu)
Computer Science, Florida Institute of Technology, Melbourne, FL 32901

1 Introduction

Recently researchers have started to make web browsers more *adaptive* and *personalized*. A *personalized* web browser caters to the user's interests and an *adaptive* one learns from the users' (potentially changing) access behavior. The goal is to help the user navigate the web. Lieberman's Letizia [13] monitors the user's browsing behavior, develops a user profile, and searches for potentially interesting pages for recommendations. The user profile is developed without intervention from the user (but the details of how that is performed is not clear in [13]). While the user is reading a page, Letizia searches, in a breadth-first manner, from that location, pages that could be of interest to the user. Pazzani et al.'s Syskill & Webert [18, 19] asks the user to rank pages in a specific topic. Based on the content and ratings of pages, the system learns a user profile that predicts if pages are of interest to the user. They investigated a number of topics and a different user profile is learned for each topic. Pages are recommended from preselected web sites. Similar to Syskill & Webert, Balabanovic's Fab [2] requires the user to rank pages and learns a user profile based on the rankings and content of pages. However, Fab considers recommendations based on profiles of all the other users on the system (though Fab does not identify users of similar interests). Fab's collection agent performs an off-line best-first search of the web for interesting pages.

Our approach toward an adaptive personalized web browser does not require the user to explicitly provide information to the browser. Instead, it monitors the user's access behavior and captures his/her interests in a user profile. We developed a metric for estimating interestingness of each visited page. A user profile consists of two components: a *Web Access Graph* (WAG) and a *Page Interest Estimator* (PIE). A web access graph summarizes the web page access patterns by a user. Based on the content of web pages, a page interest estimator, learned from the user's access behavior, characterizes the interests of a user. To provide personaized on-line search, our search engine consults multiple existing search engines ([25, 26]), collates the returned records, and ranks them according to the user profile. Using similar search techniques, recommendation and prefetching of interesting pages are performed off-line at night.

In this extended abstract we focus on PIE's (Section 2), and investigate how a user's interest of a page can be approximated without asking the user (Section 2.1) and how phrases can be identified to enrich the common bag-of-words representation for documents (Section 2.2). We discuss our preliminary empirical results on data from our web site in Section 3. Finally, we conclude in Section 4 with some remarks on the challenging issues that PIE learning exhibits.

2 Page Interest Estimator (PIE)

We can identify patterns in pages that constitute the user's interest. For example, certain words or phases, are of interest to the user. Given a set of labeled (interesting or not interesting) pages, we can apply learning algorithms (for example, C4.5 [21]) to induce classifiers that predict if a page is of interest to the user. These classifiers are called *Page Interest Estimators* (PIE's). More concretely,

$$Interest \leftarrow PIE_{user}(Page)$$

and PIE can be learned:

$$PIE_{user} \leftarrow MachineLearningAlgorithm(Page, Interest).$$

In addition, PIE's are learned, stored, and used at the individual users' sites, hence privacy is maintained.

Related work in this area usually requires user involvement in providing ranking (for example, [2, 19]). However, as we will discuss in the next section (Section 2.1), the user interest of each visited page can be approximated without any user involvement. Also, how can a page be represented so that machine learning algorithms can be applied? We discuss page representation in Section 2.2.

2.1 User Interest of a Page

How do we find out if a user is interested in a page? One way is to ask the user directly (e.g., [19]). This is the common approach, but it is invasive and, in most cases, requires the user to provide some ad hoc ranking score (e.g., 0 to 10 or bad to good). Furthermore, the user may provide inconsistent rankings across pages and the process is time consuming. Another way is to monitor the user's behavior and evaluate the user's interest. This approach is non-invasive and the user is not subject to ad hoc ranking.

Throughout the course of web browsing, a user leaves behind a trail of information that can be used to model his or her interests. Four general sources of information are available: *history*, *bookmarks*, content of pages, and *access logs*.

A web browser usually maintains a *history* of the user's requests in the current session and in the past. The history of the current session allows the user to go back and forth between the pages he/she has visited. In addition, a global history maintains the timestamp of the last time each page is visited (this allows links to expire after a specified amount of time and be displayed as if they have not been visited). In Netscape's Navigator [17], the title, URL, first visit timestamp, last visit timestamp, expiration timestamp, and visit count of each visited URL is stored in the global history. We conjecture that a higher frequency and more recent visits of an URL indicate stronger user interest of that URL.

Bookmarks serve as a quick access point for interesting URL's chosen by the user. With a few mouse clicks, a user can easily jump to URL's in his/her bookmarks. It seems natural to assume that pages that are bookmarked are of strong interest to the user.

Each page usually contains links to other pages. If the page is interesting to the user, he/she is likely to visit the links referenced by the page (Lieberman [13] made a similar observation). Hence, a higher percentage of links visited from a page indicates a stronger user interest in that page. This is particularly important for index pages, which contain a lot of related links and on which the users spend less time than pages with real content.

Each entry in an *access log* corresponds to an HTTP request, which typically contains the client IP address, timestamp, access method, URL, protocol, status, and file size. A sample entry is:

maelstrom.cs.fit.edu - - [19/Jun/1998:19:02:15 -0400] "GET /toc.html HTTP/1.0" 200 2540

From these entries, time spent on each page can be calculated. The longer a user spent on a page, the likelier the user is interested in the page. If a page is not interesting, a user usually jumps to another page quickly. Experimental studies in [15, 11] confirm this observation. However, a quick jump might be caused by the short length of the page, hence the user's interest might be more appropriately approximated by the time spent on a page normalized by the page's length. We note that activities other than surfing the web (e.g., answering a phone call) can inadvertently be included in the time spent on a page. We cannot avoid this problem without a more complicated way of obtaining time statistics. However, to reduce the problem, we impose an upper limit on time (e.g., 15 minutes) spent on a page and time intervals of more than the upper limit are considered as separate sessions. Unfortunately, browsers usually have a *history*, but not an *access log* (which is essentially a more detailed history) since a history is sufficient for traversing pages in a session and maintaining timestamps for link expiration. Access logs are usually found in HTTP servers. In order to maintain an access log for the browser client, one can modify the source code of the browser (e.g., [16]). However, this requires changes to a complex piece of software and is browser-dependent. A simpler approach is to use a web proxy server that logs HTTP requests (e.g., [27]). A proxy serves as a relay between the browser client and the Web—it is usually used for security (clients behind firewalls), performance (system-wide caching of external pages), and/or filtering (blocking out undesirable sites).

Given the above four sources of information, we can devise a measure for approximating the interest of a page to a user. One simple measure is:

$$Interest(Page) = Frequency(Page),$$

where $Frequency(Page)$ is the frequency of $Page$ visited by the user. We consider the number of visits a primary indicator of interest. A more sophisticated measure uses $Frequency(Page)$ as a base and incorporates all the factors discussed above:

$$Interest(Page) = Frequency(Page) \times (1 + IsBookmark(Page) + Duration(Page) + Recency(Page) + LinkVisitPercent(Page)), \tag{1}$$

where

$$IsBookmark(Page) = \begin{cases} 1 & \text{if } page \text{ is a bookmark} \\ 0 & \text{otherwise} \end{cases},$$

$$Duration(Page) = \frac{TotalDuration(Page)/Size(Page)}{\max_{Page \in VisitedPages}(TotalDuration(Page)/Size(Page))},$$

$$Recency(Page) = \frac{Time(LastVisit) - Time(StartLog)}{Time(Now) - Time(StartLog)}, and$$

$$LinkVisitPercent(Page) = \frac{NumberOfLinksVisited(Page)}{NumberOfLinks(Page)}.$$

The maximum value of $Interest(Page)$ is $Frequency(Page) \times 5$. Based on the frequency of visits, in this measure, we value each factor equally. Some weighted scheme will likely be more appropriate after we perform some experiments to validate this model.

All the visited pages can be considered interesting to various degrees since the user accessed them. However, how do we find pages that are not interesting to the user? It is easier if the user is required to rank pages, but the user is not actively involved in our case. Furthermore, we cannot assume any page not visited on the web is of no interest to the user because he or she might not know of its existence (not to mention the staggering number of pages on the web). Since pages usually contain links to other pages and, in most cases, not all of them are followed by the user, one approach to identifying pages not interesting to the user is to consider links in visited pages that are not followed by the user. Related work is in text categorization, where documents are mapped into categories using learned models [1, 12, 28]. Usually, the documents are grouped into many categories and all the documents are known in advance. Our task, however, is to group the pages into two categories (interesting or not interesting) and cannot assume that all the pages on the web are known in advance.

2.2 Page representation

Various representations of a web page have been widely studied. Most researchers use the vector-space model pioneered by Salton [24]. In this model each document is represented by a vector of weights, each of which corresponds to a feature, a word in most cases. The word ordering information in the document is usually not retained and hence the name "bag-of-words."

However, much research focuses on single words (unigrams) as features. This is partly due to the large combination of possible multi-word phrases. Another reason is that earlier results from "syntactic" and "statistical" phrases were mixed [7]. We suspect that the ad hoc way of constructing statistical phrases might have been a problem [9]. Much of the statistical work in building multi-word features focuses on co-occurrence (e.g., [5]), that is, if two words appear frequently together, they are probably phrases. One co-occurrence metric is *mutual information.* Consider a and b are two words within a window of some size, the mutual information:

$$MI(a,b) = \log \frac{P(a,b)}{P(a)P(b)} \tag{2}$$

measures the reduction of uncertainty in knowing b's presence in the window if a's presence is known (or vice versa, the metric is symmetric). However, this metric does not consider the effect of the absence of either or both words in the window. Note that if two words always appear together or not at all, they are more likely to be a phrase than other situations. *Expected (or average) mutual information* [22, 23] (or *information gain* [21]) captures the effects of word absences.

$$EMI(A,B) = \sum_{a,\overline{a} \in A} \sum_{b,\overline{b} \in B} P(A,B) \log \frac{P(A,B)}{P(A)P(B)} \tag{3}$$

measures the expected mutual information of the four combinations of the presence and absence of a and b. Although larger $MI(a,b)$ and $MI(\overline{a},\overline{b})$ provide more evidence for "ab" to be a phrase, larger $MI(a,\overline{b})$ and $MI(\overline{a},b)$ supply more counter evidence. Therefore, we introduce *augmented expected mutual information* (AEMI) which appropriately incorporates the counter-evidence:

$$AEMI(A,B) = \sum_{(A=a,B=b),(A=\overline{a},B=\overline{b})} P(A,B)\log\frac{P(A,B)}{P(A)P(B)} - \sum_{(A=a,B=\overline{b}),(A=\overline{a},B=b)} P(A,B)\log\frac{P(A,B)}{P(A)P(B)} \tag{4}$$

In essence supporting evidence is summed, while damaging evidence is subtracted. Furthermore, we define A as the event of the first word, and B as the event in the words (of some window size W) following the first word. That is, a higher value of $AEMI$ indicates a is likely followed by b and one is less likely to be present when the other is absent. Moreover, window size W allows flexibility in the number of gaps between words in a phrase.

Using $AEMI$ with a threshold, we can find highly probable two-word phrases (bigrams) from a training corpus. n-word phrases (n-grams) can be found using the same method—event A is the first $n-1$ words and event B is a word in the following W (window size) words. However, the memory requirement of storing the necessary statistics to find n-grams is $O(s^n)$, where s is the number of unique words in the corpus; this could be prohibitive even for locating trigrams. Consider that s is 1,000 (a relatively small number) and each counter takes 1 byte, one gigabytes are needed to find trigrams! However, to reduce the combinatorial explosion, one can safely consider only a small number of bigrams with high $AEMI$ values as event A. Though this scheme is memory efficient, it requires a second pass of the corpus, which incurs disk I/O time, since the $AEMI$ values can only be calculated at the end of one pass. This scheme requires $n-1$ passes for identifying n-grams.

We propose an approximate approach to finding n-grams without requiring a lot of storage and multiple passes on the disk-resident corpus. For each bigram with $AEMI$ above some threshold T, a directed edge is inserted into a graph, whose vertices are the words. A trigram, "abc," is identified if the edges $a \to b$, $b \to c$, and $a \to c$ exist. Similarly, a quadgram, "$abcd$," is located if, in addition to the three edges for trigram "abc," the edges $a \to d$, $b \to d$, and $c \to d$ also exist. Our scheme needs $O(s^2)$ storage ($O(s^2)$ for the counters plus $O(s^2)$ for the graph, which is sparse and requires much less than s^2 storage) and a single pass on the corpus. Formally, n-gram is defined as:

$$ngram(w_1,w_2,...,w_n) = \begin{cases} edge(w_1,w_2) & \text{if } n=2 \\ ngram(w_1,w_2,...,w_{n-1}) \wedge \bigwedge_{i=1}^{n-1} edge(w_i,w_n) & \text{if } n>2 \end{cases}$$

where

$$edge(w_i,w_j) = \begin{cases} true & \text{if } AEMI(w_i,w_j) > T \\ false & \text{otherwise} \end{cases}$$

Window size W plays a role in our method of building n-grams. Given a W, our scheme allows larger gaps in shorter phrases and smaller gaps in longer phrases. This notion stems from the observation that shorter phrases might have additional intervening words but longer phrases usually do not. For instance, when W is set to 2 for building n-grams upto trigrams, we allow an additional word between the words in bigrams but none in trigrams.

3 Preliminary Experiments

Experiments were conducted on data obtained from our departmental web server. By analyzing the server access log from January to April 1999, we identified hosts that accessed at least 50 times in the first two months and also in the second two months. We use data from the first two months for training and the last two months for testing. We filtered out proxy, crawler, and our computer lab hosts, and identified "single-user" hosts, which are at dormitory rooms and a local company.

For each text web document (.html or .txt), we first extracted words, then applied a stop list to remove common articles, prepositions, and verbs, and finally stemmed the words according to Porter's stemming algorithm [20, 10]. Bigrams and trigrams were identified using our scheme described in Section 2.2. The

Table 1: Accuracy performance of PIE's learned from four algorithms and fifteen users

User	Train size	Test size	Words and Phrases C4.5	CART	BAYES	RIPPER	Words Only C4.5	CART	BAYES	RIPPER
1	102	114	67.5	73.7	58.8	73.7	69.3	73.7	57.0	73.7
2	148	162	77.2	74.7	67.3	74.7	80.2	71.6	72.8	74.1
3	106	76	80.3	77.6	54.0	78.9	82.9	84.2	57.9	78.9
4	68	96	67.7	58.3	57.3	64.6	70.8	61.5	58.3	66.7
5	52	64	60.9	64.1	59.4	67.2	60.9	59.4	54.7	65.6
6	80	62	58.1	58.1	59.7	58.1	74.2	67.7	59.7	58.1
7	86	150	54.7	65.3	54.0	65.3	60.0	65.3	53.3	65.3
8	42	70	58.6	48.6	54.3	64.3	55.7	48.6	54.3	64.3
9	44	46	65.2	69.6	65.2	69.6	71.7	69.6	54.3	69.6
10	128	80	82.5	76.2	60.0	77.5	76.2	77.5	66.2	70.0
11	38	36	75.0	80.6	72.2	69.4	83.3	75.0	58.3	69.4
12	64	116	69.0	67.2	53.5	70.7	65.5	56.0	51.7	52.6
13	46	196	83.7	84.2	54.1	84.2	82.7	84.2	55.1	84.2
14	44	112	59.8	66.1	65.2	67.0	61.6	60.7	58.0	67.0
15	76	76	80.3	80.3	64.5	80.3	80.3	80.3	63.2	80.3
Avg.	74.9	97.1	69.4	69.6	60.0	71.0	71.7	69.0	58.3	69.3

threshold T was .0025 and the window size W was 2 (i.e., the words in a bigram might not be next to each other, but those in a trigram must be adjacent to each other in the text). Two hundred and fifty Boolean features of the presence and absence of words/phrases were selected based on *expected mutual information* (Equation 3) [29] and ties were broken with preference to features with higher document frequency in interesting documents and features from longer phrases. Moreover, accessed pages were considered interesting to the user and pages not accessed were not interesting. For simplicity, we randomly picked unaccessed pages to be included in training and testing. The number of unaccessed pages is the same as accessed pages, hence the class ratio is 1:1 in the training and test sets.

We ran C4.5 [21], CART [3], naive BAYES [8], and RIPPER [6] on the data set. Table 1 has two groups of columns on the right: one for results from features with words and phrases and the other for features with words only. The two groups are for comparing the utility of adding phrases into the feature sets (in both groups, only 250 selected features were used in training and testing). For all four algorithms, based on the paired t-test with 95% confidence, the difference in accuracy is significant—adding phrases led to higher accuracy in CART, BAYES, and RIPPER, but lower accuracy in C4.5. For User 12, phrases improved the performance of CART and RIPPER by more than 10%. This provides empirical evidence that phrases can improve the accuracy performance of learned PIE's. Among the algorithms, RIPPER achieved significantly higher accuracy in the first group and C4.5 in the second group (paired t-test with 95% confidence), while BAYES was consistently less accurate than the other algorithms. The overall top performer was C4.5 with word-only features. On average, except for BAYES, 70% (significantly higher than the default 50%) accuracy can be obtained. Finally, we would like to point out that the success of our approach largely depends on the user's behavior pattern. For instance, although User 13 has one of the smallest training sets and the largest test set, many of the algorithms achieved the highest accuracy on this user. Inspection of the generated trees and rules reveals that the user has a consistent appetite in pages that contain the word "java."

4 Concluding Remarks

We discussed a non-invasive approach to estimating the user's interest of a web page and a time and space efficient method for locating multi-word phrases to enrich the common bag-of-words representation for text documents. Page Interest Estimators are learned to build web user profiles. The reported preliminary empirical results of our approach indicate that it can predict, with an average accuracy of 70% (significantly higher than the default 50%), whether a page will be visited by a user. Also, including phrases as features can improve predictive accuracy.

Most machine learning tasks (including text categorization) assume all the features are known before hand and are applicable to both the training and test sets. However, in PIE learning, words in the documents change over time, and so does the user's preferences. A monolithic classifier/regressor with a fixed set of features will not be able to adapt to these changes. We plan to investigate a multi-classifier approach with a different feature set for each classifier that is learned in a particular time period. The ensemble can then be combined via meta-learning [4].

Besides identifying phrases, other feature construction techniques can be used to enrich the bag-of-words representation. For instance, lexical knowledge from WordNet [14] can be used to group words semantically into synonym sets. Also, we can pay more attention to words in html tags like $< title >, < b >, < i >$...

In our experiments a page that is visited multiple times has only one record in the training or test sets. However, in this domain, weighting pages proportional to the number of visits during training, in an effort to emphasize the relevant features in repeatedly visited pages, could enhance performance.

References

[1] C. Apte, F. Damerau, and S. Weiss. Towards language independent automated learning of text categorization models. In *Proc. ACM SIGIR-94*, pages 23–30, 1994.

[2] M. Balabanovic. An adaptive web page recommendation service. In *Proc. 1st Intl. Conf. Autonomous Agents*, pages 378–385, 1997.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[4] P. Chan and S. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *Proc. Twelfth Intl. Conf. Machine Learning*, pages 90–98, 1995.

[5] L. Chen and K. Sycara. Webmate: A personal agent for browsing and searching. In *Proc. 2nd Intl. Conf. Autonomous Agents*, pages 132–139, 1998.

[6] W. Cohen. Fast effective rule induction. In *Proc. 12th Intl. Conf. Machine Learning*, pages 115–123, 1995.

[7] B. Croft, H. Turtle, and D. Lewis. The use of phrases and structure queries in information retrieval. In *Proc. SIGIR-91*, pages 32–45, 1991.

[8] R. Duda and P. Hart. *Pattern classification and scene analysis.* Wiley, New York, NY, 1973.

[9] J. Fagan. *Experiments in Automatic Phrase Indexing for Document Retrieval.* PhD thesis, Linguistics, Cornell Univ., Ithaca, NY, 1987.

[10] W. Frakes and R. Baeza-Yates, editors. *Information retrieval: data structures and algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[11] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: Applying collaborative filtering to usenet news. *Comm. ACM*, 40(3):77–87, 1997.

[12] D. Lewis, R. Schapire, J. Callan, and R. Papka. Training algorithms for linear text classifiers. In *Proc. ACM SIGIR-96*, pages 298–306, 1996.

[13] H. Lieberman. Letizia: An agent that assits web browsing. In *Proc. IJCAI-95*, 1995.

[14] G. Miller. WordNet: A lexical database for english. *Comm. ACM*, 38(11):39–41, 1995.

[15] M. Morita and Y. Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proc. SIGIR-94*, pages 272–281, 1994.

[16] Netscape. Netscape Mozilla. http://www.mozilla.org/.

[17] Netscape. Netscape Navigator. http://www.netscape.org/.

[18] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[19] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting web sites. In *Proc. AAAI-96*, 1996.

[20] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[21] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[22] C. Van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.

[23] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech, and Language*, 10, 1996.

[24] G. Salton. *Automatic Text Processing.* Addison-Wesley, Reading, MA, 1988.

[25] E. Selberg and O. Etzioni. Multi-service search and comparison using the metacrawler. In *Proc. WWW4*, 1995.

[26] E. Selberg and O. Etzioni. The metacrawler architecture for resource aggregration on the web. *IEEE Expert*, 12(1):8–14, 1997.

[27] Squid. Squid internet object cache. http://squid.nlanr.net/Squid/.

[28] Y. Yang. An evaluation of statistical approaches to text categorization. Technical Report CMU-CS-97-127, CMU, Pittsburgh, PA, 1997.

[29] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *Proc. Intl. Conf. Machine Learning*, 1997.