Personalized Web Search by Mapping User Queries to Categories

Fang Liu
Department of Computer Science,
University of Illinois at Chicago
Chicago, IL 60607
(312) 996-4881
fliu1@cs.uic.edu

Clement Yu
Department of Computer Science,
University of Illinois at Chicago
Chicago, IL 60607
(312) 996-2318
yu@cs.uic.edu

Weiyi Meng
Department of Computer Science,
SUNY at Binghamton
Binghamton, NY 13902
(607) 777-4311
meng@cs.binghamton.edu

ABSTRACT
Current web search engines are built to serve all users, independent of the needs of any individual user. Personalization of web search is to carry out retrieval for each user incorporating his/her interests. We propose a novel technique to map a user query to a set of categories, which represent the user's search intention. This set of categories can serve as a context to disambiguate the words in the user's query. A user profile and a general profile are learned from the user's search history and a category hierarchy respectively. These two profiles are combined to map a user query into a set of categories. Several learning and combining algorithms are evaluated and found to be effective. Among the algorithms to learn a user profile, we choose the Rocchio-based method for its simplicity, efficiency and its ability to be adaptive. Experimental results indicate that our technique to personalize web search is both effective and efficient.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *User profiles and alert services*

General Terms
Algorithms, Performance, Experimentation, Design.

Keywords
Personalization, Search Engine, Category Hierarchy, Information Filtering

1. INTRODUCTION
As the amount of information on the Web increases rapidly, it creates many new challenges for Web search. When the same query is submitted by different users, a typical search engine returns the same result, regardless of who submitted the query. This may not be suitable for users with different information needs. For example, for the query "apple", some users may be interested in documents dealing with “apple” as “fruit”, while other users may want documents related to Apple computers. One way to disambiguate the words in a query is to associate a small set of categories with the query. For example, if the category "cooking" or the category "fruit" is associated with the query "apple", then the user's intention becomes clear. Current search engines such as *Google* or *Yahoo!* have hierarchies of categories to help users to specify their intentions. The use of hierarchical categories such as the Library of Congress Classification is also common among librarians.

A user may associate one or more categories to his/her query manually. For example, a user may first browse a hierarchy of categories and select one or more categories in the hierarchy before submitting his/her query. By utilizing the selected categories, a search engine is likely to return documents that are more suitable to the user. Unfortunately, a category hierarchy shown to a user is usually very large, and as a result, an ordinary user may have difficulty in finding the proper paths leading to the suitable categories. Furthermore, users are often too impatient to identify the proper categories before submitting his/her queries. An alternative to browsing is to obtain a set of categories for a user query directly by a search engine. However, categories returned from a typical search engine are often too many and independent of a particular user. In addition, many of the returned categories do not reflect the intention of the searcher. This paper studies how to supply, for each user, a small set of categories as a context for each query submitted by the user, based on his/her search history. Specifically, we provide a strategy to (1) model and gather the user's search history, (2) construct a user profile based on the search history and construct a general profile based on the *ODP* (Open Directory Project[1]) category hierarchy, (3) deduce appropriate categories for each user query based on the user's profile and the general profile, and (4) perform numerous experiments to demonstrate that our strategy of combining a user profile and a general profile (general knowledge) is both effective and efficient.

The categories obtained from the proposed method are likely to be related to the user's interest and, therefore, can provide a proper context for the user query. Consider the situation where a mobile user wants to retrieve documents using his/her PDA. Since the bandwidth is limited and the display is small, it may not be practical to transmit a large number of documents for the user to choose the relevant ones. Suppose that it is possible to show the retrieved documents on one screen. If these documents are not

CIKM’02, November 4--9, 2002, McLean, Virginia, USA

[1] RDF dumps of the Open Database are available for download from http://dmoz.org/rdf.html

relevant to the user, there is no easy way for the user to direct the search engine to retrieve relevant documents. With the use of our proposed technique, a small number of categories with respect to the user's query are shown. If none of the categories is desired, the next set of categories is provided. This is continued until the user clicks on the desired categories, usually one, to express his/her intention. As will be demonstrated by our experiments, the user usually finds the categories of interest among the first 3 categories obtained by our system. Since 3 categories can easily fit into one screen, it is likely that effective retrieval can be achieved with minimal interaction with the user. Thus, our proposed technique can be used to personalize web search.

The contribution of this paper is as follows: We provide methods to deduce a set of relevant categories for each user query based on the retrieval history of the user. The set of categories can be deduced using the user's profile only, or using the general profile only or using both profiles. We make the following comparisons:

(a) The accuracy of combining the user profile and the general profile versus that of using the user profile only.

(b) The accuracy of combining the user profile and the general profile versus that of using the general profile only.

(c) The accuracy of using the user profile only versus that of using the general profile only.

We show that the accuracy of combining both profiles is better than those using a single profile and the accuracy of using the user profile only is better than that using the general profile only, provided that there is sufficient history data.

Relationships of our work with previous researches are sketched below:

1. Many techniques are used in modern search engines to provide more contexts for user queries. *Yahoo*! (http://www.yahoo.com/), *ODP* (http://dmoz.org/) and *Google* (http://www.google.com/) return both categories and documents. *Northern Light* (http://www.northernlight.com/) and *WiseNut* (http://www.wisenut.com/) cluster their results into categories, and *Vivisimo* (http://www.vivisimo.com/) groups results dynamically into clusters. *Teoma* (http://www.teoma.com/) clusters its results and provides query refinements. A lot of research in metasearch [11][14][16][25][8][35] also investigates mapping user queries to a set of categories or collections. However, all of the above techniques return the same results for a given query, regardless of who submitted the query. This can be interpreted as having a general profile. Our experimental results indicate that using the combination of a user profile and a general profile usually yields significantly higher accuracy than using a general profile or a user profile alone.

2. Many papers on information filtering [1][5][9][28][30][31] and intelligent agent (*Syskill & Webert* [24], *WebWatcher*[2] [18], *Letizia* [21], *CiteCeer* [3], *Liza* [2]) have been published. Most of them also construct user profiles explicitly or implicitly, and recommend documents using the profiles. However, the technique we employ is different. While previous methods filter documents, our goal is to retrieve categories of interest for a user query. Furthermore, no general profile is used in information filtering.

3. Text categorization has been investigated thoroughly. A comparison of various methods is given in [34]. Four algorithms are evaluated in our paper. Categorization of web pages or collections of web pages has also been studied in [19][20][22][17]. Our utilization of a category hierarchy is derived from [22].

4. In the area of personalized web search, *WebMate* [6] uses user profiles to refine user queries, but no experimental results are given. *Watson* [4] refines queries using a local context but does not learn the user profile. I*nquirus 2* [12] uses users' preferences to choose data sources and refine queries but it does not have user profiles, and requires the users to provide their preferences of categories. In addition, only four non-topical categories are included in I*nquirus 2*. [26] learns users' profiles from their surfing histories, and re-ranks/filters documents returned by a metasearch engine based on the profiles. Our approach is different from all of the above in that we try to map user queries to a small set of categories based on user profiles and general knowledge. Furthermore, we make all three types of comparisons (a), (b) and (c) described earlier, while earlier work may be interpreted as having done only (b).

The rest of the paper is organized as follows. In Section 2, our strategy to personalize web search is introduced: how a user's search history is modeled and collected, how the collected information is used to construct a user profile and a general profile, and how the profiles can be used to deduce a set of categories which are likely to be related to the user's query. In Section 3, the construction of the two profiles using four different learning approaches, namely the Linear Least Squares Fit (LLSF) approach, the pseudo-LLSF approach (pLLSF), k-Nearest Neighbor (kNN) and Rocchio (bRocchio) is sketched. In addition, an adaptive Rocchio (aRocchio) learning approach is also given. In Section 4, methods of mapping a user query to a set of categories based on the two profiles are provided. In Section 5, experiment results are shown to compare the effectiveness of the learning algorithms and the mapping algorithms. Conclusion is given in Section 6.

2. PROBLEM

The problem is to personalize web search. We introduce a strategy in this section. First, we propose a tree model to represent a user's search history and describe how a user's search history can be collected without his/her direct involvement. In Section 2.2, a brief description of a user profile is given. A matrix representation of the user history and the user profile is described in Section 2.3. General knowledge from a category hierarchy is extracted to construct a general profile. This is given in Section 2.4. Section 2.5 sketches the deduction of the appropriate categories based on a user query and the two profiles.

2.1 User Search History

A search engine may track and record a user's search history in order to learn the user's long-term interests. We consider using the following information items to represent a user's search history: queries, relevant documents and related categories. One

[2] WebWatcher uses a collaborative-filtering method.

search record is generated for each user search session. A tree model of search records is shown in Figure 1. In this model, nodes are information items and edges are relationships between nodes. The root of a search record is a query. Each query has one or more related categories. Associated with each category is a set of documents, each of which is both relevant to the query and related to the category. Based on our experiments with users, for almost

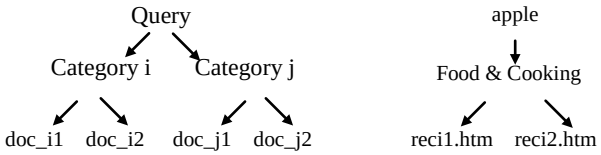

Figure 1: Model and example of a search record

all queries, each query is related to only one or two categories.

In practice, a search engine may be able to acquire the type of user's search records described above, without direct involvement by the user. Some possible scenarios are as follows:

(1) A document retrieved by a search engine can be assumed to be relevant to the user with respect to a user query if some of the following user behaviors are observed: the user clicks it and there is a reasonable duration before the next click; the user saves/prints it.

(2) A user utilizing some of the popular search engines may first select a category before submitting a query. In this way, a category related to the user query is identified. Furthermore, some search engines such as *Google* have pre-classified some results into categories; some other search engines such as *Northern Light* cluster all results into categories. When such documents are observed to be relevant (see scenario 1 above), the user query, its related categories and its relevant documents are identified.

Based on (1) and (2), a set of search records representing a user's search history can be obtained. As an example, consider the following session with the *Northern Light* search engine. A user who is interested in cooking submits a query "apple" to the search engine, and it returns the top 10 documents and 12 categories. The user clicks the 8th category "Food & cooking" and the search engine shows all documents that have been clustered into this category. Then, the user clicks two documents about cooking apples. When this search session is finished, a search record as shown in Figure 1 can be generated and saved for the user.

2.2 User Profile

User profiles are used to represent users' interests and to infer their intentions for new queries. In this paper, a **user profile** consists of a set of categories and for each category, a set of terms (keywords) with weights. Each category represents a user interest in that category. The weight of a term in a category reflects the significance of the term in representing the user's interest in that category. For example, if the term "apple" has a high weight in the category "cooking", then the occurrence of the word "apple" in a future query of the user has a tendency to indicate that the category "cooking" is of interest. A user's profile will be learned automatically from the user's search history.

2.3 Matrix Representation of User Search History and User Profile

We use matrices to represent user search histories and user profiles. Figure 2 shows an example of the matrix representations of a search history and a profile for a particular user, who is interested in the categories "COOKING" and "SOCCER". This user's search history is represented by two matrices DT (Figure 2(a)) and DC (Figure 2(b)). DT is a document-term matrix, which is constructed from the user queries and the relevant documents. (In the following discussion, we use "documents" to denote both queries and relevant documents in the matrices DT and DC). DC is a document-category matrix, which is constructed from the relationships between the categories and the documents. A user profile is represented by a category-term matrix M (Figure 2(c)). In this example, D1, D2, … are documents; lowercase words such as "football", "apple",… are terms; uppercase words such as "SOCCER", "COOKING", … are categories.

We now describe the construction of the matrices DT and DC based on the user's search records.

- Matrix $DT(m * n)$. DT is constructed from the queries (the root nodes in the tree model) and their relevant documents (the

Doc\Term	apple	recipe	pudding	football	soccer	fifa
D1	1	0	0	0	0	0
D2	0.58	0.58	0.58	0	0	0
D3	0	0	0	1	0	0
D4	0	0	0	0.58	0.58	0.58

(a) Document-Term matrix DT

Doc\Category	COOKING	SOCCER
D1	1	0
D2	1	0
D3	0	1
D4	0	1

(b) Document-Category matrix DC

Cate\Term	apple	recipe	pudding	football	soccer	fifa
COOKING	1	0.37	0.37	0	0	0
SOCCER	0	0	0	1	0.37	0.37

(c) Category-Term matrix M represents a user profile

Figure 2:Matrix representations of user search history and profile

leaf nodes in the tree model) in the user's search records. m is the number of documents in a user's search history and n is the number of distinct terms occurring in these documents. Each query or document is a row vector [29] of weighted terms in DT . If a term, say term j, occurs in the i-th query/relevant document, the weight $DT(\mathrm{i}, \mathrm{j}) > 0$; otherwise it is 0. The value of $DT(\mathrm{i}, \mathrm{j})$ is determined by the common normalized TF*IDF weight scheme [15]. Before constructing DT , a stop word list is used to remove common words. In addition, terms that appear in only one relevant document in the user's search history are removed. Furthermore, if an occurrence of a term t is more than 5 words away from any occurrence of a query term, then the occurrence of

the term t is removed. Porter stemmer [10] is also applied to each term.

- Matrix $DC(m*p)$. For each row in matrix DT, there is a corresponding row in the matrix DC. The columns of DC are the set of related categories. If a row in DT represents a query/document, then the corresponding row in the matrix DC represents the set of categories related to the query/document. More precisely, if there is an edge between the j-th category and the i-th query/document, then the entry $DC(i, j) = 1$; otherwise it is 0.

- Matrix $M(p*n)$. From DT and DC, we learn a matrix M, which represents the user profile. Each row in the matrix M is also a vector of weighted terms, representing one of the categories of the user. Thus, both categories and documents are represented in the same vector space and similarities between them can be computed. The learning methods for obtaining the user profile M will be explained in Section 3.

2.4 A Category Hierarchy

In addition to the matrices DT, DC and M as described above, we also utilize some general knowledge which is applicable to all users. The reason for using the additional information is that the knowledge acquired from a user is often limited and may not be sufficient to determine the user's intention when a new user query is encountered. For example, a new query may contain terms that have never been used by the user before, nor appeared in any of his/her previous retrieved relevant documents. The general knowledge that our system utilizes is extracted from *ODP*. Specifically, we use the first three levels of *ODP*. The categories in the first two levels (15 first level categories and 604 second level categories) are used to represent the set of all categories. The terms appearing in these three levels of categories are used to represent the categories in the first two levels. From the category hierarchy, we learn a general profile, using a process similar to that for learning the user profile. Let the three corresponding matrices related to the general knowledge be denoted by DTg, DCg and Mg (general profile).

To construct document-term matrix DTg, we generate two documents for each category in the first two levels. One document consists of all terms in the text descriptions of its subcategories. The other document consists of terms in the category's own text description. For example, in Figure 3, "Artificial intelligence" is a second level category, and has subcategories "Data mining", "Genetic algorithms", etc. Thus, for this category ("Artificial intelligence"), one document with the terms "data", "mining", "genetic" and "algorithms" and another document with the terms "artificial" and "intelligence" are generated, respectively. Note that both of the above two documents for each category are needed. On one hand, all of the terms in the text descriptions of the category and its subcategories are useful for describing the category. On the other hand, the terms in the text description of the category are likely to be more important than the terms in the text descriptions of the subcategories for describing the category. By using the above two documents, the difference in importance among the terms can be captured. As a result, a pair of rows is created in DTg for each category.

1: Computers
  2: Algorithms
… 2: Artificial intelligence
    3: Data mining
    3: Genetic algorithms
… 2: Internet

Figure 3: A category hierarchy

For each pair of rows in the matrix DTg, say row i1 and row i2, there is a corresponding pair of rows in the document-category matrix DCg, and the entries $DCg(\text{i1}, \text{j}) = DCg(\text{i2}, \text{j}) = 1$, where the j-th category represents "Artificial intelligence". In addition, if the k-th category represents the parent of the j-th category (in this case, the parent is "Computer"), the entries $DCg(\text{i1}, k)$ and $DCg(\text{i2}, k)$ are set to 0.25, indicating that this pair of documents are related to the k-th category, though to a lesser extent. All other entries in this pair of rows are set to 0. The learning method for constructing the general profile Mg will be given in Section 3.

2.5 Inference of User Search Intention

In our environment, personalization is accomplished by mapping a user query to a set of categories, which reflects the user's intention, based on the user profile and the general profile. The mapping is carried out as follows. First, the similarities between a user query and the categories representing the user's interests are computed. Next, the categories are ranked in descending order of similarities. Finally, the top three categories together with a button indicating the next three categories are shown to the user. If the user clicks on one of these top three categories, then the user's intention is explicitly shown to the system. If the user's interest is not among the top three categories, then the button can be clicked to show the next three categories.

3. ALGORITHMS TO LEARN PROFILES

Learning a user profile (matrix M) from the user's search history (matrices DT and DC) and mapping user queries to categories can be viewed as a specific multi-class text categorization task. In sections 3.1-3.3, we describe four algorithms to learn a user profile: bRocchio, LLSF, pLLSF and kNN. The last three algorithms have been shown to be among the top-performance text categorization methods in [34].

3.1 Two LLSF-based Algorithms

Given the m-by-n document-term matrix DT and the m-by-p document-category matrix DC, the Linear Least Squares Fit (LLSF) method [32] computes a p-by-n category-term matrix M such that $DT * M^T$ approximates DC with the least sum of square errors, where M^T is the transpose of M. A common technique for solving this problem is to employ the Singular Value Decomposition (SVD) [13]. DT is decomposed into the product of three matrices $U*\Sigma*V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix. After such decomposition, it is rather straightforward to compute: $M = DC^T * U * \Sigma^+ * V^T$, where Σ^+ is the inverse of Σ.

We also evaluate another variant called "pseudo-LLSF" (pLLSF), in which the dimensions of DT are reduced. Matrices Σ, U and V are replaced by Σ_k, U_k and V_k respectively, where Σ_k contains the highest k entries in the diagonal matrix Σ, U_k and V_k are obtained by retaining the first k columns of U and V respectively. Essentially, the original space is replaced by a k dimensional space. After the replacements, M is computed from these modified matrices using the same formula, i.e., $M = DC^T * U_k * \Sigma_k^{+} * V_k^T$. The basic idea is that the noise in the original document-term matrix DT is removed by the dimension reduction technique. This technique is also the key of the Latent Semantic Indexing method (LSI) [7], which has been used successfully in various applications in IR [7][9][8]. In practice, it is not easy to give a good value of k. Thus, we choose a k such that the ratio of the smallest retained singular value over the largest singular value is greater than a threshold θ, which is set to be 0.25 in this paper.

3.2 Rocchio-based Algorithm

Rocchio is originally a relevance feedback method [27]. We use a simple version of Rocchio adopted in text categorization:

$$M(i, j) = \frac{1}{N_i} \sum_{k=1}^{m} DT(k, j) * DC(k, i)$$

where m is the number of documents in DT, N_i is the number of documents that are related to the i-th category, and $M(i, j)$ is the average weight of the j-th term in all documents that are related to the i-th category. Documents that are not related to the category are not used. We call the batch-based Rocchio method bRocchio.

3.3 kNN

The k-Nearest Neighbor (kNN) does not compute a user profile. Instead, it computes the similarity between a user query and each category from DT and DC (see Section 4.1).

3.4 Adaptive Learning

The algorithms introduced above are all based on batch learning, in which the user profile is learned from the user's previous search records. Batch learning can be inefficient when the amount of accumulated search records is large. An adaptive method can be more efficient, as the user profile is modified by the new search records. LLSF-based algorithms are not suitable for adaptive learning as re-computation of the user profile M is expensive. kNN requires storing DT and DC, which is space inefficient. Furthermore, the computation of similarities using kNN can be inefficient for large amount of search records. Rocchio is efficient in both computation and storage, and is adaptive. The following formula is used:

$$M(i, j)^t = \frac{N_i^{t-1}}{N_i^t} M(i, j)^{t-1} + \frac{1}{N_i^t} \sum_k DT(k, j) * DC(k, i)$$

where M^t is the modified user profile at time t; N_i^t is the number of documents that have been accumulated from time zero to time t and these documents are related to the i-th category; the second term on right hand side of the equation is the sum of the weights of the j-th term in the documents that are related to the i-th category and obtained between time $t-1$ and time t divided by N_i^t. We call this adaptive–based Rocchio method aRocchio.

4. MAPPING QUERIES TO CATEGORIES

We examine the following 3 processes of mapping a new user query to a set of categories.

4.1 Using User Profile Only

The similarity between a query vector q and each category vector c in the user profile M is computed by the *Cosine* function [29]. As stated in Section 3, we use pLLSF, LLSF, bRocchio and aRocchio to compute M.

In the case of kNN, the algorithm first finds the k most similar documents among all document vectors in DT using the *Cosine* function. Then, among these k neighbors, a set of documents, say S, which are related to a category c can be identified using DC. Finally, the similarity between q and c is computed as the sum of the similarities between q and the documents in S. This is repeated for each category. The following formula, which is slightly modified from [34], is used:

$$Sim(q, c_j) = \sum_{d_i \in kNN} Cos(q, d_i) * DC(i, j)$$

where q is the query; c_j is the j-th category; d_i is a document among the k nearest neighbors of q and the i-th row vector in DT, $Cos(q, d_i)$ is the cosine similarity between q and d_i, and $DC(i, j) \in \{0,1\}$ denotes whether d_i is related to the j-th category. We set k=12 in this paper.

4.2 Using General Profile Only

Only pLLSF is used to compute the general profile Mg. As shown in the following section, pLLSF has the highest average accuracy; and although it is computationally expensive, Mg needs to be computed only once.

4.3 Using Both User and General Profiles

We propose 3 combining methods and compare them with the above two baseline cases. Let c^u and c^g be the category vectors for the user profile and the general profile respectively. The following computation is done for every category.

(a) Use only the user profile: $Sim(q, c) = Sim(q, c^u)$.

(b) Use only the general profile: $Sim(q, c) = Sim(q, c^g)$.

(c) Combining Method 1:
$Sim(q, c) = (Sim(q, c^u) + Sim(q, c^g)) / 2$.

(d) Combining Method 2:
$Sim(q, c) = 1 - (1 - Sim(q, c^u)) * (1 - Sim(q, c^g))$.

(e) Combining Method 3:
$Sim(q, c) = \max(Sim(q, c^u), Sim(q, c^g))$.

The combining methods are not applied to kNN, because it may produce a similarity >1 between a user query and a category. This prevents combining method 2 to be used.

The categories are ranked in descending order of the combined similarities, i.e. *Sim(q, c)*, and the top 3 categories are chosen to reflect the user's search intention. The reason that it is sufficient to use the top 3 categories only is that, for a given query, most users are interested in only one or two categories in the two-level category hierarchy.

5. EXPERIMENTS

5.1 Data Sets

Table 1: Statistics of the 7 data sets

Statistics	User 1	User 2	User 3	User 4	User 5	User 6	User 7
# of interest catetories	10	8	8	8	10	8	9
# of search records (queries)	37	50	61	26	33	29	29
avg # of related search records to one category	3.7	6.3	7.6	3.25	3.3	3.63	3.2
# of relevant documents	236	178	298	101	134	98	115
avg # of categories in one search record	1.1	1	1	1	1	1	1
# of distinct terms	7012	5550	6421	4547	4584	4538	4553

In our experiments, seven data sets were collected from seven different users are evaluated. Each user submitted a number of queries to a search engine which, in this case, is Google. For each query, the user identified the set of related categories and a list of relevant documents. The query, the set of related categories and the list of relevant documents comprised a search record as shown in Figure 1. Table 1 gives the statistics of the data sets. For example, user 1 has 10 related categories, and 37 search records with 37 queries and 236 relevant documents. As mentioned in Section 2.4, we generate a set of documents in the construction of the general profile, using the text descriptions of the categories in the first 3 levels of *ODP*. There are 619 categories in the first two levels of the hierarchy.

To evaluate our approach, we use the 10-fold cross-validation strategy [23]. For each data set, we randomly divide the search records into 10 subsets, each having approximately the same number of search records. We repeat experiments 10 times, each time using a different subset as the test set and the remaining 9 subsets as the training set. As described in Section 2.3, we construct two matrices from the search records in the training set and we call them DT_{train} and DC_{train}. Similarly, two matrices DT_{test} and DC_{test} are constructed from the test set. After the user profile M is learned from DT_{train} and DC_{train}, the set of categories is ranked with respect to each query in DT_{test} and the result is checked against DC_{test} to compute the accuracy. Finally, the average accuracy across all 10 runs is computed.

5.2 Performance Metric

In our approach, the top 3 categories are returned for each user query. The following performance metric is proposed:

$$Accuracy = (\sum_{ci \in top3} score_{ci}) / n = (\sum_{ci \in top3} \frac{1}{1 + rank_{ci} - ideal\_rank_{ci}}) / n$$

where n is the number of related categories to the query, $score_{ci}$ is the score of a related category c_i that is ranked among the top 3, $rank_{ci}$ is the rank of c_i and $ideal\_rank_{ci}$ is the highest possible rank for c_i. We compute the accuracy for each query. For example, assume that c1 and c2 are related categories to a user query, and they are ranked by the system to be the first and the third, then the accuracy should be computed in the following way: $score_{c1}$ =1/(1+1-1)=1 and $score_{c2}$ =1/(1+3-2)=0.5, so the accuracy is (1+0.5)/2=0.75.

5.3 Experimental Results

First, we investigate the effectiveness of the four batch learning algorithms based on only the user profiles. Table 2 and Figure 4 show their accuracy results. As can be seen from Figure 4, pLLSF, kNN and bRocchio have similar effectiveness and all of them perform well; their accuracy ranges from 0.768 to 0.975 with the exception of user 1. These three algorithms outperform

Table 2: pLLSF vs LLSF vs bRocchio vs kNN on average

Method	pLLSF	LLSF	bRocchio	kNN
Average	0.8236	0.7843	0.8224	0.8207

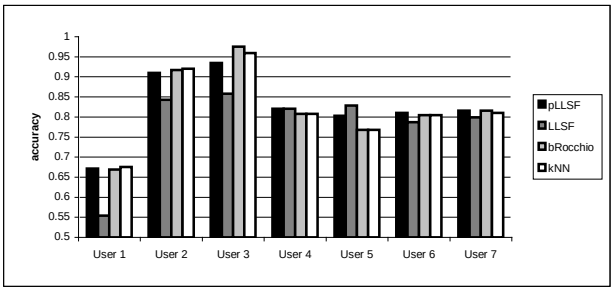

Figure 4: pLLSF vs LLSF vs bRocchio vs kNN on 7 users

LLSF as shown in Table 2. This indicates that dimension reduction with SVD is worthwhile.

We examine the effects of combining the user profile and the general profile, and compare the 3 combining methods with the 2 baselines. Since pLLSF, bRocchio and kNN have been shown to yield similar accuracy, we choose bRocchio to construct the user profile and pLLSF to construct the general profile. Another reason for choosing bRocchio is that it can be made an adaptive method. Table 3 and Figure 5 show that the 3 combining methods have approximately the same average performance, and all of them significantly outperform the two baselines. This clearly demonstrates that it is worthwhile to combine the user profile and

the general profile to yield higher accuracy than using only one of the two profiles. Another observation from Table 3 is that using

Table 3: Comparison of different mapping methods on average

Method	User	General	Comb 1	Comb 2	Comb 3
Average	0.8224	0.7048	0.8936	0.8917	0.8846

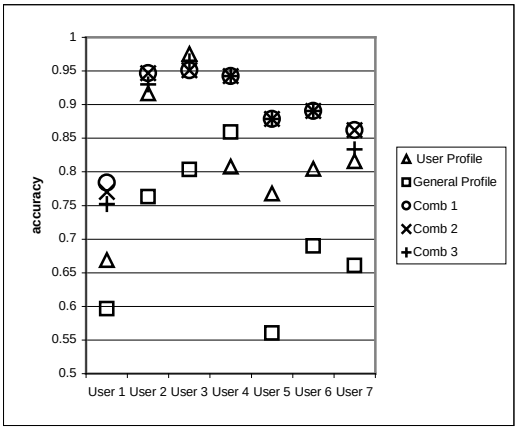

Figure 5: Comparison of different mapping methods on 7 users

the user profile alone gives better performance than using the general profile alone. This tends to imply that it is worthwhile to perform personalized search.

Finally, we examine the accuracy of the adaptive learning method aRocchio as more and more training data are given. Only combining method 1 is used as there is no significant difference among the 3 combining methods. aRocchio is experimented as follows: (1) We still use the 10-fold cross-validation strategy. The 10 subsets of each data set are numbered from 1 to 10. (2) For each user, the experiment is repeated 10 times. In the i-th run, the i-th subset is the test set. The remaining 9 subsets are used as 9 training sets. The first user profile M^1 is constructed from the training subset {i+1}. Then M^1 is modified by the training subset {i+2} to yield the next profile M^2 (see the formula in Section 3.4). This process continues until the user profile M^8 is modified by the training subset {i-1} to produce M^9. As more training subsets are given, the accuracies of using the user profile alone, using the general profile alone and using both profiles are examined. Finally, the case of using the test subset i as the training data to produce M^{10} from M^9 is carried out. The last case is of interest, as in the Internet environment, it is known that users tend to submit the same queries repeatedly. The following are some observations for the results as shown in Figure 6.

(1) When the size of training data is small, the accuracy of using the user profile alone is worse than that using the general profile alone. However, even with a small training data set, the accuracy of using both profiles is better than that using one of the two profiles only.

(2) As more training data is given, the accuracy of using the user profile increases. This also boosts the accuracy of using both profiles.

(3) When all data are employed as the training data, close to 100% accuracy is achieved.

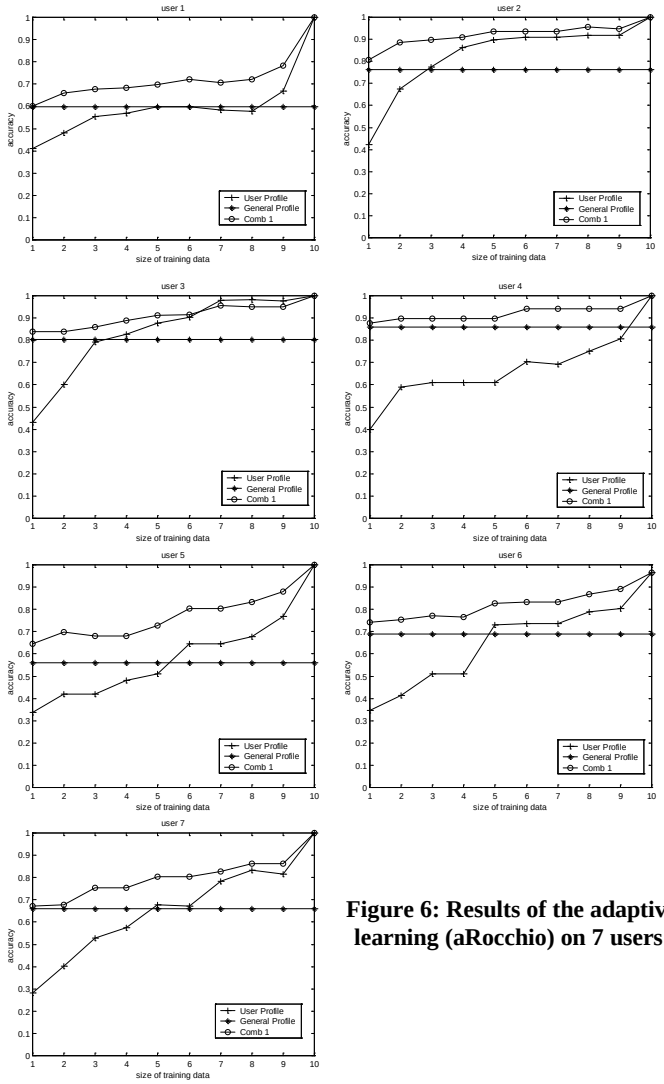

Figure 6: Results of the adaptive learning (aRocchio) on 7 users

6. CONCLUSION

We described a strategy for personalization of web search: (1) a user's search history can be collected without direct user involvement; (2) the user's profile can be constructed automatically from the user's search history; (3) the user's profile is augmented by a general profile which is extracted automatically from a common category hierarchy; and (4) the categories that are likely to be of interest to the user are deduced based on his/her query and the two profiles.

For the construction of the profiles, four batch learning algorithms (pLLSF, LLSF, kNN and bRocchio) and an adaptive algorithm (aRocchio) are evaluated. Experimental results indicate that the accuracy of using both profiles is consistently better than those using the user profile alone and using the general profile alone. The simple adaptive algorithm aRocchio is also shown to be effective and efficient.

7. ACKNOWLEDGMENTS

This work was supported in part by the following NSF grants: IIS-9902792, IIS-9902872 and EIA-9911099.

8. REFERENCES

[1] J. Allan. Incremental relevance feedback for information filtering. SIGIR, 1996

[2] M. Balabanovic and Y. Shoham. Learning information retrieval agents: Experiments with automated Web browsing. In On-line Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments, 1995.

[3] K. Bollacker, S. Lawrence, and C. Lee Giles. A system for automatic personalized tracking of scientific literature on the web. ACM DL, 1999.

[4] J. Budzik and J. K. Hammond. Watson: Anticipating and contextualizing information needs. In Proceedings of the Sixty-second Annual Meeting of the American Society for Information Science, 1999

[5] U. Çetintemel, M. J. Franklin, and C. Lee Giles. Self-Adaptive User Profiles for Large-Scale Data Delivery.ICDE, 2000

[6] L. Chen and K. Sycara. WebMate: A Personal Agent for Browsing and Searching. Autonomous Agents and Multi Agent Systems, 1998.

[7] S. Deerwester, S. T. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. JASIS, 18(2), 1990.

[8] R. Dolin, D. Agrawal, A. El Abbadi and J. Pearlman. Using Automated Classification for Summarizating and Selecting Heterogeneous Information Sources. D-Lib Magazine, 1998.

[9] W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. CACM,1992.

[10] W. Frakes, and R. Baeza-Yates. Information Retrieval: Data Structures and Algorithms. 1992.

[11] S. Gauch, G. Wang, M. Gomez. ProFusion: Intelligent Fusion from Multiple, Distributed Search Engines. Journal of Universal Computer Science, 2(9), 1996

[12] E. Glover, G. Flake, S. Lawrence, W. Birmingham, A. Kruger, C. Giles, and D. Pennock. Improving Category Specific Web Search by Learning Query Modifications. SAINT, 2001

[13] G. H. Golub and C. F. Van Loan. Matrix Computations. Third Edition, 1996

[14] L. Gravano, and H. Garcia-Molina. Generalizing GlOSS to Vector-Space Databases and Broker Hierarchies. VLDB, 1995.

[15] D. Grossman and O. Frieder. Information Retrieval: Algorithms and Heuristics. 1998.

[16] A. E. Howe and D. Dreilinger. SavvySearch: A meta-search engine that learns which search engines to query. AI Magazine, 18(2), 1997.

[17] P. Ipeirotis, L. Gravano, and M. Sahami. Probe, Count, and Classify: Categorizing Hidden Web Databases. ACM SIGMOD, 2001.

[18] Joachims, T., Freitag, D., and Mitchell, T. Webwatcher: A tour guide for the World Wide Web. IJCAI, 1997

[19] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. ICML, 1997

[20] Y. Labrou and T. Finin. Yahoo! as an ontology: using Yahoo! categories to describe documents. CIKM, 1999

[21] H. Lieberman. Letizia: An agent that assists Web browsing. IJCAI, 1995.

[22] W. Meng, W. Wang, H. Sun and C. Yu. Concept Hierarchy Based Text Database Categorization. International Journal on Knowledge and Information Systems, March 2002.

[23] T. Mitchell. Machine Learning, 1997.

[24] M. Pazzani and D. Billsus. Learning and Revising User Profiles: The identification of interesting web sites. Machine Learning, 1997.

[25] A. L. Powell, J. C. French, J. P. Callan and M. Connell. The impact of database selection on distributed searching. SIGIR, 2000.

[26] A. Pretschner and S. Gauch. Ontology based personalized search. ICTAI, 1999

[27] J. Rocchio. Relevance feedback in information retrieval. In The smart retrieval system: Experiments in automatic document processing, 1971.

[28] S. Robertson and I. Soboroff. The TREC-10 Filtering Track Final Report. TREC-10, 2001.

[29] G. Salton and M. J. McGill. Introduction to Modern Information Retrieval, 1983.

[30] D. H. Widyantoro, T. R. Ioerger and J. Yen. An adaptive algorithm for learning changes in user interests. CIKM, 1999

[31] T. W. Yan and H. Garcia-Molina. SIFT -- A Tool for Wide-Area Information Dissemination. USENIX Technical Conference, 1995.

[32] Y. Yang and C. G. Chute. An example-based mapping method for text categorization and retrieval. TOIS, 1994

[33] Y. Yang. Noise Reduction in a Statistical Approach to Text Categorization. SIGIR 1995

[34] Y. Yang and X. Liu, A re-examination of text categorization methods. SIGIR 1999.

[35] C. Yu, W. Meng, W. Wu and K. Liu. Efficient and Effective Metasearch for Text Databases Incorporating Linkages among Documents. ACM SIGMOD, 2001.