Personalized Web Search For Improving Retrieval Effectiveness

Fang Liu,[1] Clement Yu,[1] Weiyi Meng[2]
[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607, {fliu1, yu}@cs.uic.edu, (312) 996-2318
[2] Department of Computer Science
SUNY at Binghamton, NY 13902, meng@cs.binghamton.edu, (607) 777-4311

ABSTRACT [*+]

Current web search engines are built to serve all users, independent of the special 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 learn user profiles from users' search histories. The user profiles are then used to improve retrieval effectiveness in web search. 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, which represent the user's search intention and serve as a context to disambiguate the words in the user's query. Web search is conducted based on both the user query and the set of categories. Several profile learning and category mapping algorithms and a fusion algorithm are provided and evaluated. Experimental results indicate that our technique to personalize web search is both effective and efficient.

Index Terms—Category Hierarchy, Information Filtering, Personalization, Retrieval Effectiveness, Search Engine

[*] A preliminary version of this paper (but not containing the part on how the categories can be used to improve retrieval effectiveness) has been published in CIKM02.

[+] This work is supported in part by the following NSF grants: IIS-9902792, IIS-9902872, EIA-9911099, IIS-0208434, IIS-0208574 and the Army Research Office: 2-5-30267.

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 some 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 as a context for the query, 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 still independent of a particular user and many of the returned categories do not reflect the intention of the searcher. To solve these problems, we propose a two-step strategy to improve retrieval effectiveness. In the first step, the system automatically deduces, for each user, a small set of categories for each query submitted by the user, based on his/her search history. In the second step, the system uses the set of categories to augment the query to conduct the web search. 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) improve web search effectiveness by using these categories as a context for each query. Numerous experiments are performed to demonstrate that our strategy of personalized web search is both effective and efficient.

A scenario in which our proposed personalized search can be beneficially utilized is as follows. 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. Even if it is possible to show some of the retrieved documents on one screen, there is no easy way for the user to direct the search engine to retrieve relevant documents if the initially retrieved documents are irrelevant. In contrast, 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 contributions of this paper are as follows:

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

(1) We provide methods to deduce a set of related 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 and show that:

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

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

(c) The accuracy of using the user profile only is higher than that of using the general profile only.

(2) We propose two modes, one semi-automatic and another completely automatic, to personalize web search based on both the query and its context (the set of related categories). We show that both personalization modes can improve retrieval effectiveness.

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 and distributed retrieval [Gauch96, Grav95, Howe97, Powell00, Dolin98, Yu01, Xu99, Fuhr99] 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 [Allan96, Ceti00, Foltz92, Robe01, Widy99, Yan95] and intelligent agent (Syskill & Webert [Pazz97], WebWatcher [Joac97], Letizia [Lieb95], CiteCeer [Boll99], Liza [Bala95]) 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 determine the categories which are likely to be the intention of the user. The determined categories are used as a context for the user query to improve retrieval effectiveness. Furthermore, no general profile was used in information filtering in previous papers.

(3) Text categorization has been investigated thoroughly. A comparison of various methods is given in [Yang99]. Four algorithms are evaluated in our paper. Categorization of web pages or collections of web pages has also been studied in [Koller97, Labrou99, Meng02, Ipei01]. Our utilization of a category hierarchy is similar to that from [Meng02].

(4) In the area of personalized web search, WebMate [Chen98] uses user profiles to refine user queries, but no experimental results are given. Watson [Budz99] refines queries using a local context but does not learn the user profile. Inquirus 2 [Glover01] 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 Inquirus 2. [Pret99] 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 each user query to a small set of categories based on the user's profile and the general profile and we retrieve web pages by merging multiple lists of web pages from multiple query submissions. Furthermore, we make all three types of comparisons (a),

(b) and (c) described under item 1 of our contribution, while earlier works may be interpreted as having done only (b). Among all these related works, [Pret99] is the most similar one to ours. Additional differences between our work and that in [Pret99] are as follows:

a. The user profiles in the two approaches are different. In our approach, a category in a user profile is a weighted term vector, in which a high weight of a term indicates that the term is of high significance in that category for the user, and a low weight of the same term in another category indicates that the term is not important in that category. In other words, we utilize the weights of terms in different categories to identify the categories of interest to the user. In [Pret99], no association of terms with categories is used in a user profile. The difference in the two approaches may yield substantial difference in identifying categories of interest. As an example, suppose there is a user who is interested in both "COOKING" and "COMPUTER", and has previously used "apple" in retrieving relevant documents in the category "COOKING", but has not used the same word in retrieving relevant documents in the category "COMPUTER". As a consequence, the user profile should have a high weight for the word "apple " in the category "COOKING", but the word has a low or zero weight in the category "COMPUTER". Using this user profile, when the user who wants to find some information about "apple cooking" submits a query containing the word "apple", the category "COOKING" will be predicted for this user.
b. [Pret99] reported an 8% improvement of retrieval effectiveness, and we get a 12%-13% improvement for automatic mode and 25.6% improvement for semi-automatic mode, although the tested collections are different.

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, how the profiles can be used to deduce a set of categories which are likely to be related to the user's query and how web searches are conducted using the set of categories. In Section 3, the constructions 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) are 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. Section 5 gives the methods to conduct personalized web searches to improve retrieval effectiveness. In Section 6, experimental results are shown to report the efficiency of our technique and compare the effectiveness of the learning algorithms, the mapping algorithms and the merging (fusion) algorithms. Conclusion is given in Section 7.

2. PROBLEM

The problem is to personalize web search for improving retrieval effectiveness. Our strategy includes two steps. The first step is to map a user query to a set of categories, which represent the user's search intension and serve as a context for the query. The second step is to utilize both the query and its context to retrieve web pages. In order to accomplish the first step, a user profile and a general profile are constructed. We propose a tree model in Section 2.1 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 for the purpose of constructing the 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. The last Section sketches the utilization of the categories to improve web search.

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

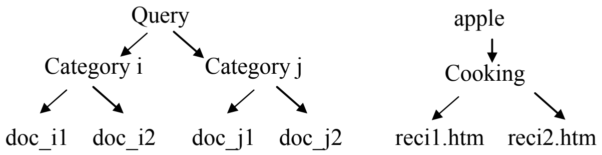

Figure 1: A Model and an example of a search record

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 documents into categories; some other search engines such as Northern Light cluster all retrieved documents 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

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

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, … D4 are documents; lowercase words such as "football" and "apple" are terms; uppercase words such as "SOCCER" and "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 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 relevant document is a row vector [Salton83] 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 [Gros98]. 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 each query term, then the occurrence of the term t is removed. Porter stemmer [Frby92] 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. Since a row in DT represents a query/document, the corresponding row in the matrix DC indicates 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 in the tree model of a search record, 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, which represents a category of interest to the user, is a vector of weighted terms. Thus, both categories and documents are represented in the same vector space of terms and similarities between them can be computed. The learning methods for obtaining 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 previously 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 has the terms "data", "mining", "genetic" and "algorithms" and another document has the terms "artificial" and "intelligence". Each term in the former document, though important in characterizing the category, is of lower significance than each term in the latter document. This is reflected by the fact that there are more terms in the former document than the latter document.

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

Figure 3: An example of the 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 $DCg(\mathrm{i1},\mathrm{j}) = DCg(\mathrm{i2},\mathrm{j}) = 1$, 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 $DCg(\mathrm{i1},k)$ and $DCg(\mathrm{i2},k)$ 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 method to construct the general profile Mg will be given in Section 3.

2.5 Inference of User Search Intention

In our environment, the first step of personalized search is accomplished by mapping a user query to a set of categories, which reflects the user's intention and serves as a context for the query, 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, which when pressed, will indicate the next three categories are shown to the user. If the user clicks on one or more 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.

A user may have new interests. Our use of the general profile, which has interests for all users, is likely to be helpful. A user may have changing interests. We intend to keep the most recent search records. Thus, the user profile of a user reflects his/her most recent interests.

2.6 Improving Retrieval Effectiveness Using Categories

Our goal is to improve retrieval effectiveness. To accomplish it, we propose the following modes of retrieval:

(1) The user query is submitted to a search engine (in this paper Google Web Directory[2]) without specifying any category. In fact, this is not a mode of personalized search and will be considered as the *baseline* mode in our experiment.

(2) As discussed before, our system determines the three categories which are most likely to match the interests of the user with the given user query. From these three categories, the user can either pick the ones which are most suitable or he/she can decide to see the next 3 categories. The process continues until the desired categories are chosen by the user. As shown in Section 6.3.1, the user usually finds the desired categories within the first three categories presented by the system. Let us call this the *semi-automatic* mode.

(3) In the *automatic* mode, the system automatically picks the top category or the top 2 categories or the top 3 categories without consulting the user. Thus, the two-step personalization of web search can be accomplished automatically, without the involvement of users.

In the last two modes, the user query is initially submitted without specifying any category. Then, the query is submitted by specifying each of the chosen categories as a context. The multiple lists of returned documents are merged using a weighted voting-based merging algorithm (see Section 5.1).

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 [Yang99].

[2] Google Web Directory (http://directory.google.com/) is a Google version of ODP. All web pages in Google Web Directory have been pre-classified into the category hierarchy, and Google Web Directory supports searching by specifying a category.

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 [Yang94] 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). 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. [Golub96] gives a solution based on such a decomposition: $M = DC^T * U * \sum^+ * V^T$, where Σ^+ is the pseudo-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., $M = DC^T * U_k * \sum_k{}^+ * V_k{}^T$. 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) [Deer90], which has been used successfully in various applications in Information Retrieval (IR) [Deer90, Foltz92, Dolin98]. 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 [Rocc71]. 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 matrix representing the user profile, N_i is the number of documents that are related to the i-th category, m is the number of documents in DT, $DT(k,j)$ is the weight of the j-th term in the k-th document, $DC(k,i)$ is a binary value denoting whether the k-th document is related to the i-th category. Clearly, $M(i,j)$ is the average weight of the j-th term in all documents that are related to the i-th category and documents that are not related to the category are not contributing to $M(i,j)$. We call the batch-based Rocchio method bRocchio.

3.3 kNN

The k-Nearest Neighbor (kNN) method does not compute a user profile. Instead, it computes the similarity between a user query and each category directly 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. The kNN method 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's method can be made adaptive as follows:

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

where M^t is the modified user profile at time t; N_i^t is the number of documents, which are related to the i-th category and have been accumulated from time zero to time t; 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. For example, suppose at time $t-1$, the value of $M(i,j)^{t-1}$ is 0.5, N_i^{t-1} is 10; between time $t-1$ and t, we collect a number of new documents, among which are 5 documents that are related to the i-th category; and the sum of the weights of the j-th term in these 5 document is 1. Then, N_i^t is 10+5=15 and $M(i,j)^t = \frac{10}{15} * 0.5 + \frac{1}{15} * 1 = 0.4$. We call this adaptive–based Rocchio method aRocchio.

4. MAPPING QUERIES TO RELATED 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 [Salton83]. As stated in Section 3, we use pLLSF, LLSF, bRocchio and aRocchio to compute M .

The kNN method 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 [Yang99], 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 will be shown in Section 6, 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.

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

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

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

(4) Combining Method 2: $Sim(q,c) = 1 - \left(1 - Sim(q,c^u)\right) * \left(1 - Sim(q,c^g)\right)$.

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

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.

5. IMPROVING RETRIEVAL EFFECTIVNESS

Our system maps each user query to a set of categories, and returns the top three categories. In this section, we provide methods to improve retrieval effectiveness using categories as a context of the user query. Three modes of retrieval have been briefly introduced in Section 2.6. In the three modes of process, the user query is submitted to the search engine (in this case Google Web Directory) multiple times. In the first mode, it is submitted to the search engine without specifying any category. Let the list of documents retrieved be *DOC-WO-C* (documents retrieved without specifying categories). Let its cardinality be *MO*. In the second and third modes, the query is submitted by specifying a set of categories which is obtained either semi-automatically or completely automatically. Let the list of documents retrieved by specifying the top i category be *DOC-W-Ci*. Let its cardinality be *MWi*. *MO* is usually larger than *MWi*. As a consequence, a fair comparison between retrieval using the specified categories and that of not specifying any category is not possible. Our solution is as follows. We will merge the retrieved lists of documents *DOC-WO-C* and *DOC-W-Ci* in such a way that the resulting set has exactly the same cardinality as *DOC-WO-C*.

5.1 Algorithm

Our algorithm to merge multiple lists of retrieved documents, *DOC-WO-C* and *DOC-W-Ci*, is by modifying a voting-based merging scheme [Mont02]. The original merging scheme is as follows:

> Each retrieved list has the same number of documents, say *N*. The i-th ranked document in a list gets (*N* - i +1) votes. Thus, the first document in a list gets *N* votes and later documents in the list get fewer votes. If a document appears in multiple lists, it gets the sum of the votes of that document appearing in the lists. In other words, if a document appears in multiple lists, it usually gets more votes than a document

appearing in a single list. Documents in the merged list are ranked in descending order of votes. No document relevance scores are required. It has been shown in [Mont02] that this way of merging is both effective and efficient.

Our modification of the above scheme is a weighted voting-based merging algorithm:

(1) Let *MM* be the number of documents in the longest list. In our case, the longest list is *DOC-WO-C* and *MM=MO*.

(2) Each list, say the j-th list, has a weight *Wj* associated with it. The number of votes assigned to the i-th ranked document in the j-th list is *Wj* * (*MM* - i + 1). The weight *Wj* is dependent on the rank of the category, say *C*, the similarity of the category with respect to the query and the number of documents in the list. It is given by:

Wj = *rank-C* * square-root(*sim-C*) * *num-C*, where

- *rank-C* =1, if the rank of C with respect to the query is 1;

  0.5, if the rank is 2;

  0.25, if the rank is 3.
- *sim-C* is $Sim(q,C)$ as given in Section 4.3.
- *num-C* is the number of retrieved documents in the list (either *MWi* or *MO*).
- *rank-C* is 1 and *sim-C* is 1 in the semi-automatic mode in which the category is selected by the user. If the list of documents is obtained by not specifying any category, then *rank-C* is 0.5; *sim-C* is 0.1, which is approximately the average similarity of the top ranked categories for the queries.

The following scenarios explain the motivation that weights are assigned as introduced above:

- Suppose the top ranked category has a similarity much higher than 0.1, then assuming that each list has the same number of documents, the weight associated with *DOC-W-C1* is much higher than that associated with *DOC-WO-C*. This is consistent to the notion that a category, if it obtains high similarity, receives high confidence. This implies that documents in *DOC-W-C1* gets higher votes than those in *DOC-WO-C*. Conversely, if a top-ranked category receives low similarity, then the documents in *DOC-W-C1* get low votes. Consider the extreme situation where none of the query terms appears in either the user profile or the general profile. In that case, the similarities between the query and all the categories will be all zeros. This means that the weight *Wj* = 0. As a consequence, only the list of documents *DOC-WO-C* is retrieved.

- Documents retrieved using higher ranked categories get more votes than those retrieved using lower ranked categories. This explains the relative values assigned to rank-C as well as sim-C.
- If a query is submitted to a wrong category, then the number of documents retrieved is usually very few, which is an indication that the confidence of using the category is low.

After all votes of the documents are counted, the documents are arranged in descending order of votes and the top *MO* documents (the same number of documents as *DOC-WO-C*) are retained. In case several documents from multiple lists get the same number of votes, the document from the list with the highest weight *Wj* will be ranked ahead of the other documents with the same number of votes. For example, suppose we want to merge two lists of returned documents for a given query, one of which is obtained by not specifying any category and the other one is obtained by specifying the top 1 category. Let the two lists be:

DOC-WO-C: {d1, d2, d3, d4, d5, d6, d7, d8, d9, d10}

DOC-W-C1: {d1, d5, d6, d8, d9, d11, d12, d13, d14, d15}

Each underlined document is relevant to the query. Let the similarity between the query and the top category be 0.1. Thus, the weight of the DOC-WO-C list is 0.5 * sqrt(0.1) * 10, the weight of the DOC-W-C1 list is 1 * sqrt(0.1) * 10, and votes for document d1 to d15 are respectively {15, 4.5, 4, 3.5, 12, 10.5, 2, 8.5, 7, 0.5, 5, 4, 3, 2, 1} * sqrt(0.1). Finally, we get the top 10 documents of the merged list and it is:

{d1, d5, d6, d8, d9, d11, d2, d12, d3, d4}

In this example, the merged list has 1 more relevant document than each of the two original lists. It is also clear that the merged list is more effective than the *DOC-WO-C* list.

Other merging algorithms such as those in [Dwork01] can be employed. In fact, we experiment with the best algorithm, *MC4*, in [Dwork01]. It yields 1-2% improvement over the algorithm [Mont02] reported here, but is less efficient. Due to limited space, *MC4* is not presented here.

6. EXPERIMENTS

6.1 Data Sets

In our experiments, seven data sets were collected from seven different users in two phases. In the first phase, each user submitted a number of queries to a search engine which, in this case, is Google Web Directory. For each query, the user identified the set of related categories and a list of relevant documents, as well as provided a statement, which describes the semantics of the query (similar to the "Description" part of a "Topic" in TREC[Voor01]). Each query (not containing the statement), the set of related categories and the list of relevant documents comprise a search record. In the second phase, each query is submitted in 3 different modes to the Google Web Directory as described in Section 5. For each submission, at most top 10

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

documents are examined by the user. The relevance of each returned documents is judged as either relevant or irrelevant.

Table 1 gives the statistics of the data sets. For example, user 1 has 10 interest 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 to map a user query to a set of categories, we use the 10-fold cross-validation strategy [Mitch97]. 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. This can also be considered as a simulation of users' changing interests, as both the training set and the test set change. 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. The average accuracy across all 10 runs is computed. This is a measurement of performance of mapping queries to categories. In addition, each query in DT_{test} with the set of ranked categories is used to conduct the three modes of retrieval. A standard effectiveness measure will be used.

6.2 Performance Measures

6.2.1 Accuracy of Mapping User Queries to Categories

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

$\text{Accuracy} = (\sum_{ci \in top3} \text{score}_{\text{ci}})/n = (\sum \frac{1}{1 + rank_{\text{ci}} - ideal\_rank_{ci}})/n$ 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, $rank_{\text{ci}}$ 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. If neither $c1$ or $c2$ are among the top 3, the accuracy will be 0. For each data set, we compute the average accuracy of all queries.

6.2.2 Measure of Web Page Retrieval

The measure of effectiveness is essentially the "Precision at 11 standard recall levels" as used in TREC evaluation [TREC10]. It is briefly described as follows:

- For each query, for each list of retrieved documents up to the top 10 documents, all relevant documents are identified. (In practice, a number higher than 10 may be desirable. However, we have a limited amount of human resources to perform manual judgment of relevant documents. Furthermore, most users in the Web environment examine no more than 10 documents per query.)
- The union of all relevant documents in all these lists is assumed to be the set of relevant documents of the query.
- For each value of recall (the percentage of relevant documents retrieved) among all the recall points {0.0, 0.1, ..., 1.0}, the precision (the number of relevant document retrieved divided by the number of retrieved documents) is computed.
- Finally, the precision, averaged over all recall points, is computed.

For each data set and for each mode of retrieval, we obtain a single precision value by averaging the precision values for all queries.

The measure of efficiency is the average wall clock time for processing a user query.

6.3 Experimental Results

6.3.1 Results of Mapping User Queries to Categories

First, we investigate the effectiveness of the four batch learning algorithms based on only the user profiles. Figure 4 and Table 2 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 LLSF as shown in Table 2. This indicates that dimension reduction with SVD is worthwhile.

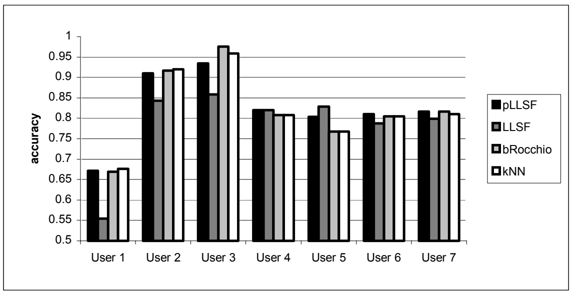

Figure 4: pLLSF vs LLSF vs bRocchio vs kNN on 7 users

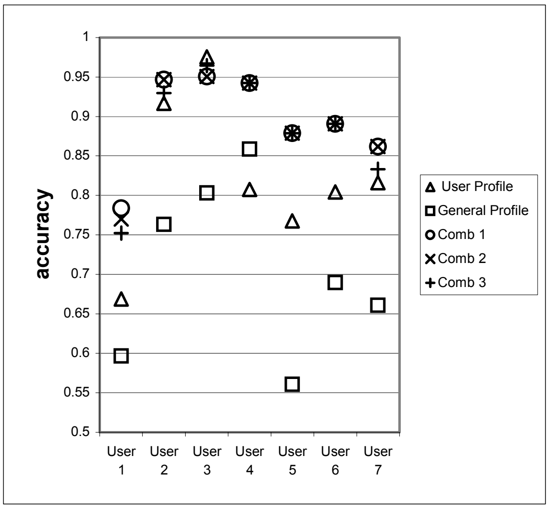

Figure 5: Comparison of different mapping methods on 7 users

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

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

We examine the effects of combining the user profile and the general profile, and compare the 3 combining methods with the 2 baselines (one using the general profile only and the other using the user profile only). 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. Figure 5 and Table 3 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 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 (i.e. the window size of user search history becomes bigger and biger). 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, Table 4-6.

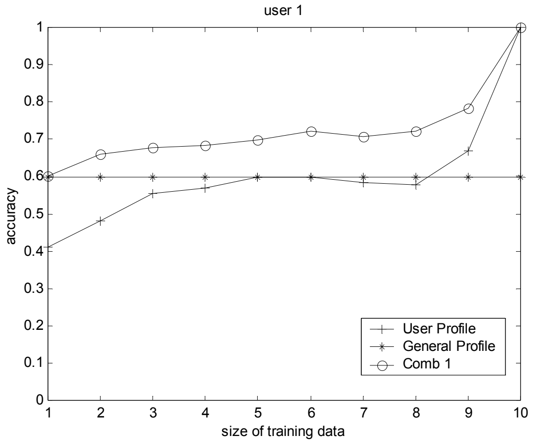

Figure 6: Results of the adaptive learning (aRocchio) on user 1

(All results for other users are similar)

Table 4: Results of the Adaptive Learning (aRocchio)
Using Only User Profiles

Size	User 1	User 2	User 3	User 4	User 5	User 6	User 7
1	0.410	0.423	0.432	0.397	0.338	0.345	0.282
2	0.480	0.673	0.601	0.590	0.419	0.414	0.402
3	0.554	0.773	0.792	0.609	0.419	0.512	0.529
4	0.568	0.863	0.828	0.609	0.480	0.512	0.575
5	0.597	0.897	0.877	0.609	0.510	0.730	0.678
6	0.599	0.910	0.904	0.705	0.647	0.736	0.672
7	0.583	0.910	0.981	0.692	0.647	0.736	0.782
8	0.579	0.917	0.984	0.750	0.677	0.787	0.833
9	**0.669**	**0.917**	**0.975**	**0.808**	**0.768**	**0.805**	**0.816**
10	1.000	1.000	1.000	1.000	1.000	0.966	1.000

Table 6: Results of the Adaptive Learning (aRocchio)
Using Both Profiles – Method *Comb1*

Size	User 1	User 2	User 3	User 4	User 5	User 6	User 7
1	0.601	0.807	0.839	0.878	0.647	0.741	0.672
2	0.660	0.887	0.839	0.897	0.697	0.753	0.678
3	0.678	0.897	0.858	0.897	0.682	0.770	0.753
4	0.685	0.910	0.888	0.897	0.682	0.764	0.753
5	0.698	0.937	0.913	0.897	0.727	0.828	0.805
6	0.722	0.937	0.915	0.942	0.803	0.833	0.805
7	0.707	0.937	0.956	0.942	0.803	0.833	0.828
8	0.721	0.957	0.952	0.942	0.833	0.868	0.862
9	**0.784**	**0.947**	**0.951**	**0.942**	**0.879**	**0.891**	**0.862**
10	1.000	1.000	1.000	1.000	1.000	0.966	1.000

Table 5: Results of the pLLSF method Using Only the General Profile

Size *	User 1	User 2	User 3	User 4	User 5	User 6	User 7
	0.597	**0.763**	**0.803**	**0.859**	**0.561**	**0.690**	**0.661**

*Results using only the general profile are independent of the size of user history

(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.

6.3.2 Results of Retrieval Effectiveness and Efficiency

A comparison of the three modes of retrieval is conducted. The following experiments are carried out.

1. *Base*: the average precision of the queries submitted by the users without specifying the categories.
2. *Semi*: the average precision when the top categories are determined automatically, the correct categories are identified by the user, and the retrieved lists are merged as described above.
3. *Auto1*: the average precision when the top category determined automatically for each query is used for retrieval and the two retrieved lists of documents, *DOC-WO-C* and *DOC-W-C1* are merged as described above.
4. *Auto2*: same as (3) except that the top 2 categories are used and the three retrieved lists, *DOC-WO-C*, *DOC-W-C1* and *DOC-W-C2* are merged.
5. *Auto3*: same as (4) except that the top 3 categories are used.

Based on the categories obtained by the first step (we use the results of the *Comb1* method as shown in Figure 5 and Table 3), we examine the improvement in retrieval effectiveness using our weighted voting-based merging algorithm. The results (Precision at 11 standard recall levels and the improvement of the two modes of personalization to the baseline) for the 7 users are given in Table 7 and Figure 7-8. We have the following observations from the results:

Table 7: Precision at 11 standard recall levels on 7 users

User\Mode	*Base*	*Semi*	*Auto1*	*Auto2*	*Auto3*	Accuracy*
1	0.4552	0.5962 (+30.31%)	0.4898 (+7.36%)	0.4871 (+7.30%)	0.4869 (+7.30%)	0.7838
2	0.5564	0.5966 (+7.02%)	0.583 (+4.08%)	0.5929 (+6.06%)	0.5932 (+6.06%)	0.9467
3	0.3582	0.4561 (+27.23%)	0.4321 (+20.26%)	0.4381 (+22.23%)	0.4381 (+22.23%)	0.9508
4	0.5416	0.6374 (+17.17%)	0.6074 (+12.11%)	0.6085 (+12.14%)	0.6085 (+12.14%)	0.9423
5	0.4021	0.5405 (+34.34%)	0.4695 (+16.38%)	0.4679 (+16.34%)	0.4679 (+16.34%)	0.8788
6	0.3151	0.4743 (+50.55%)	0.3834 (+21.57%)	0.3827 (+21.55%)	0.3826 (+21.54%)	0.8621
7	0.4401	0.5486 (+24.27%)	0.4761 (+8.22%)	0.483 (+9.27%)	0.4724 (+7.23%)	0.8908
Avg. P +	**0.4384**	**0.55 (25.6%)**	**0.4916 (12.1%)**	**0.4943 (12.8%)**	**0.4928 (12.4%)**	

* Accuracy: accuracy of mapping user queries to categories, which is the result of the "Comb1" method shown in Figure 5.
+ Avg. P: is the average Precision at 11 standard recall levels for all the seven users, and is plotted in Figure 8.
The entries in the Table are the average precision values and the percentage of improvement over the baseline.

- The improvement in retrieval effectiveness due to the *semi-automatic* mode is about 25.6%.

However, even though all categories are identified by the user to be relevant, the returned documents are not all relevant.

- The improvement in retrieval effectiveness using any one of the three *automatic* methods, namely *Auto1*, *Auto2* and *Auto3* yields about the same result, which is in the range 12%-13%. Since *Auto1* only needs to combine results from two lists, it is more efficient than *Auto2* and *Auto3*. Thus, *Auto1* is preferred.

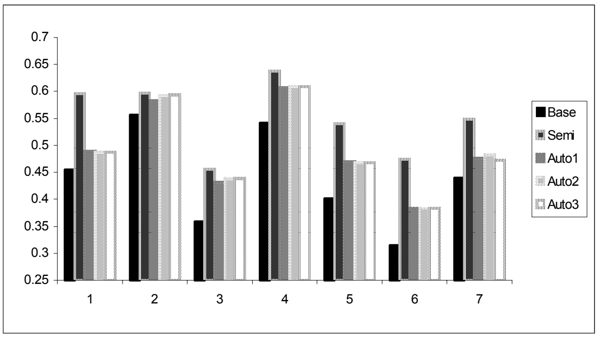

Figure 7: Precision at 11 standard recall levels on 7 users

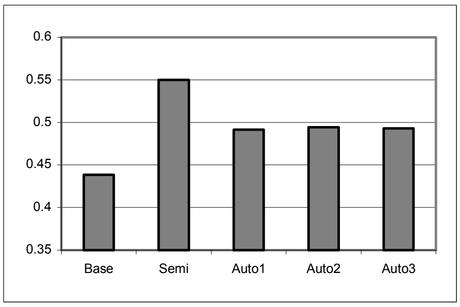

Figure 8: Precision at 11 standard recall levels on average

- In all the above cases, a significant improvement in retrieval effectiveness is established when personalized search is used.

Next, we examine the efficiency of our technique. Table 8 shows that the average times for processing a query in seconds. Each of the times reported in the table consists of:

Table 8: Average wall clock time for processing a query (second)

	Base	*Auto1*	*Auto2*	*Auto3*
Avg. Query Time	0.361	0.702	1.014	1.304

(a) the time to map the user query to a set of categories;

(b) the time for the search engine, Google Directory, to retrieve the documents;

(c) the time for our system to extract lists of documents from the search engine result pages; and

(d) the time to merge the multiple lists of documents into a final list of documents.

99% of the time is spent on step (b) and (c). Thus, the portion of our algorithm which consists of step (a) and (d) is efficient.

7. 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 and is augmented by a general profile which is extracted automatically from a common category hierarchy; (3) the categories that are likely to be of interest to the user are deduced based on his/her query and the two profiles; and (4) these categories are used as a context of the query to improve retrieval effectiveness of web search.

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. For the web search, the weighted voting-based merging algorithm is used to merge retrieval results. The semi-automatic and automatic modes of utilizing categories determined by our system are shown to improve retrieval effectiveness by 25.6% and around 12% respectively. We also show that our technique is efficient (at most 0.082 second/query).

It should be noted that the experimental results reported here include 7 users, a few hundred queries and the identifying of a limited number of relevant documents. There is also room for obtaining higher levels of improvement than reported here, as we choose reasonable (but not exhaustive) values for a number of parameters (e.g. the weight associated with each list of retrieved documents). Future research in the area consists of much larger scale of experiments as well as optimization of parameters.

REFERENCES

[Allan96] James Allan. Incremental relevance feedback for information filtering. Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, p.270-278, New York, 1996, ACM Press.

[Bala95] Marko Balabanovic and Yoav 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, p.13-18, Stanford University.

[Boll99] Kurt Bollacker, Steve Lawrence, and C. Lee Giles. A system for automatic personalized tracking of scientific literature on the web. In Proceedings of the 4th ACM Conference on Digital Libraries, p.105-113, New York, 1999. ACM Press.

[Budz99] Jay Budzik and Kristian Hammond. Watson: Anticipating and contextualizing information needs. In Proceedings of the Sixty-second Annual Meeting of the American Society for Information Science, 1999, Information Today, Inc.

[Ceti00] Ugur Cetintemel, Michael J. Franklin, and C. Lee Giles. Self-Adaptive User Profiles for Large-Scale Data Delivery. Proceedings of 16th the International Conference on Data Engineering (ICDE), 2000, p.622-633, IEEE Computer Society.

[Chen98] Liren Chen and Katia Sycara. WebMate: A Personal Agent for Browsing and Searching. Proceedings of the 2nd International Conference on Autonomous Agents and Multi Agent Systems, p.132-139, 1998

[Deer90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer and Richard Harshman. Indexing by latent semantic analysis. Journal of the American Society for Information Science (JASIS), 41(6), p.391-407, 1990.

[Dolin98] Ron Dolin, Divyakant Agrawal, Amr El Abbadi and J. Pearlman. Using Automated Classification for Summarizating and Selecting Heterogeneous Information Sources. D-Lib Magazine, 1998.

[Dwork01] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In Proceedings of the Tenth International World Wide Web Conference, p.613-622, 2001.

[Foltz92] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: An analysis of information filtering methods. Communications of the ACM, 35(12), p.51-60, 1992.

[Frby92] William B. Frakes and Ricardo Baeza-Yates. Information Retrieval: Data Structures and Algorithms. Prentice Hall, 1992.

[Fuhr99] Norbert Fuhr, A Decision-Theoretic Approach to Database Selection in Networked IR. ACM Transactions on Information Systems (TOIS), 17(3), p229-249, 1999.

[Gauch96] Susan Gauch, Guijun Wang, and Mario Gomez. ProFusion: Intelligent Fusion from Multiple, Distributed Search Engines. Journal of Universal Computer Science, 2(9), p.637-649, 1996

[Glover01] Eric J. Glover, Gary W. Flake, Steve Lawrence, William P. Birmingham, Andries Kruger, C. Lee Giles, and David M. Pennock. Improving Category Specific Web Search by Learning Query Modifications. SAINT, p.23-34, 2001

[Golub96] Gene H. Golub, and Charles F. Van Loan. Matrix Computations. Third Edition, 1996

[Grav95] Luis Gravano, and Hector Garcia-Molina. Generalizing GlOSS to Vector-Space Databases and Broker Hierarchies. Proceedings of the 21st International Conference on Very Large Databases (VLDB), p.78-89, 1995.

[Gros98] David A. Grossman, and Ophir Frieder. Information Retrieval: Algorithms and Heuristics. 1998.

[Howe97] Adele E. Howe, and Daniel Dreilinger. SavvySearch: A meta-search engine that learns which search engines to query. AI Magazine, 18(2), p.19-25 , 1997.

[Ipei01] Panagiotis Ipeirotis, Luis Gravano, and Mehran Sahami. Probe, Count, and Classify: Categorizing Hidden Web Databases. ACM SIGMOD, p.67-78, 2001.

[Joac97] Thorsten Joachims, Dayne Freitag, and Tom Mitchell. Webwatcher: A tour guide for the World Wide Web. Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI), p.770-777, 1997

[Koller97] Daphne Koller, and Mehran Sahami. Hierarchically classifying documents using very few words. . Proceedings of the 14th International Conference on Machine Learning (ICML), p.170-178, 1997

[Labrou99] Yannis Labrou, and Tim Finin. Yahoo! as an ontology: using Yahoo! categories to describe documents. Proceedings of the 8th ACM International Conference on Information and Knowledge Management (CIKM), p.180-187, 1999

[Lieb95] Henry Lieberman. Letizia: An agent that assists Web browsing. Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), p.924-929, 1995.

[Meng02] Weiyi Meng, Wenxian Wang, Hongyu Sun and Clement Yu. Concept Hierarchy Based Text Database Categorization. International Journal on Knowledge and Information Systems, p.132-150, March 2002.

[Mitch97] Tom Mitchell. Machine Learning, McGraw Hill, 1997.

[Mont02] Mark Montague and Javed A. Aslam. Condorcet Fusion for Improved Retrieval. Proceedings of the 11th ACM International Conference on Information and Knowledge Management (CIKM), p.538-548, 2002

[Pazz97] Michael Pazzani, and Daniel Billsus. Learning and Revising User Profiles: The identification of interesting web sites. Machine Learning, 27, p.313-331, 1997.

[Powell00] Allison L. Powell, James C. French, James P. Callan, Margaret E. Connell, and Charles L. Viles. The impact of database selection on distributed searching. Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, p.232-239, 2000.

[Pret99] Alexander Pretschner, and Susan Gauch. Ontology based personalized search. Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), p.391-198, 1999

[Rocc71] J. Rocchio. Relevance feedback in information retrieval. In The smart retrieval system: Experiments in automatic document processing, p.313-323, 1971.

[Robe01] Stephen E. Robertson and Ian Soboroff. The TREC-10 Filtering Track Report. Text REtrieval Conference (TREC-10), 2001.
http://trec.nist.gov/pubs/trec10/papers/filtering_track.pdf

[Salton83] Gerard Salton, and Michael McGill. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.

[Voor01] Ellen M. Voorhees: Overview of TREC 2001. Text REtrieval Conference (TREC-10), 2001. http://trec.nist.gov/pubs/trec10/papers/overview_10.pdf

[TREC10] Ellen M. Voorhees, and Donna Harman, editors. Common Evaluation Measures. Text REtrieval Conference (TREC-10), p.A-14, 2001.

[Widy99] Dwi H. Widyantoro, Thomas R. Ioerger, John Yen. An adaptive algorithm for learning changes in user interests. Proceedings of the 8th ACM International Conference on Information and Knowledge Management (CIKM), p.405-412, 1999.

[Xu99] Jinxi Xu and W. Bruce Croft. Cluster-based language models for distributed retrieval. Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR), p.254-261, 1999.

[Yan95] Tak W. Yan, and Hector Garcia-Molina. SIFT -- A Tool for Wide-Area Information Dissemination. Proceedings of the 1995 USENIX Technical Conference, p.177-186, 1995.

[Yang94] Yiming Yang, and Christopher G. Chute. An example-based mapping method for text categorization and retrieval. ACM Transaction on Information Systems (TOIS), 12(3), p.252-277, 1994

[Yang95] Yiming Yang. Noise Reduction in a Statistical Approach to Text Categorization. Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR), p.256-263, 1995

[Yang99] Yiming Yang, and Xin Liu, A re-examination of text categorization methods. Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR), p.42-49, 1999.

[Yu01] Clement T. Yu, Weiyi Meng, Wensheng Wu and King-Lup Liu. Efficient and Effective Metasearch for Text Databases Incorporating Linkages among Documents. ACM SIGMOD, p.187-198, 2001.