Flexible User Profiles for Large Scale Data Delivery*

Uğur Çetintemel
Institute for Advanced Computer Studies
Computer Science Department
University of Maryland
ugur@cs.umd.edu

Michael J. Franklin
Institute for Advanced Computer Studies
Computer Science Department
University of Maryland
franklin@cs.umd.edu

C. Lee Giles
Institute for Advanced Computer Studies
University of Maryland
and
NEC Research Institute
giles@research.nj.nec.com

Abstract

Push-based data delivery requires knowledge of user interests for making scheduling, bandwidth allocation, and routing decisions. Such information is maintained as user profiles. We propose a new incremental algorithm for constructing user profiles based on monitoring and user feedback. In contrast to earlier approaches, which typically represent profiles as a single weighted interest vector, we represent user-profiles using multiple interest clusters, whose number, size, and elements change adaptively based on user access behavior. This flexible approach allows the profile to more accurately represent complex user interests. The approach can be tuned to trade off profile complexity and effectiveness, making it suitable for use in large-scale information filtering applications such as push-based WWW page dissemination. We evaluate the method by experimentally investigating its ability to categorize WWW pages taken from Yahoo! categories. Our results show that the method can provide high retrieval effectiveness with modest profile sizes and can effectively adapt to changes in users' interests.

1 Introduction

Publish/subscribe models and other forms of push-based data delivery have been gaining popularity as ways to relieve Internet users of the burden of having to continuously hunt for new information. These techniques deliver data items to users according to a pre-arranged plan, so that they don't have to make specific requests for items of interest.

In order to effectively target the right information to the right people, push-based systems rely upon *user profiles* that indicate the general information types (but not necessarily the specific data items) that a user is interested in receiving. From the user's viewpoint, profiles provide a means of *passively* retrieving relevant information. A user can submit a profile to a push-based system once, and then continuously receive items that are relevant to him or her in a timely fashion. From a systems point of view, profiles fulfill a role similar to that of

*Franklin and Çetintemel's work has been partially supported by the NSF under grant IRI-9501353, and by Rome Labs agreement number F30602-97-2-0241 under ARPA order number F078.

queries in database or information retrieval systems; in fact, profiles are a form of continuously executing query. The system depends on these profiles in order to achieve more *efficient* and *targeted* information dissemination.

1.1 Profile Quality

The quality of user profiles is a key to making a push-based system work. From the user's point of view, there are two potential problems. One is the *precision* problem. If a high proportion of the items that the system sends to a user are irrelevant, then the system becomes more of an annoyance than a help. Conversely, if the system fails to provide the user with enough relevant information, then the benefit of push-based delivery is largely lost, because the user will still have to actively hunt for information. This latter problem is known as the *recall* problem. Both problems can translate to unhappy users, which can ultimately render the system worthless.

One contributing factor to profile quality is the language used to describe the profiles. For unstructured or semi-structured items such as web pages, it is notoriously difficult to formulate boolean (or relational) queries that return result sets of manageable size. Such queries typically suffer from the problems of either returning too many results, or returning no results at all. Furthermore, it has been shown that as the size of the data set grows, formulating effective queries with such languages becomes even harder [Tur94].

For text-based data items, profiles based on natural language techniques from Information Retrieval (IR) have been shown to be reasonably effective at representing user information needs. Even assuming a good profile representation, however, with existing approaches it is still quite likely that a user's profile will not provide adequate precision or recall. There are three main reasons for this. The first is that existing approaches represent user interests in terms of a single profile vector or multiple *independent* profile vectors (e.g, SIFT [YGM95], MyExcite [Cha99]). Single vectors, as we will demonstrate, are insufficient for adequately modeling interests, leading to low effectiveness values. Using multiple vectors, on the other hand, helps achieve better effectiveness. In existing systems, however, these multiple vectors are typically treated independently by the system, resulting in redundant storage and processing of overlapping subscriptions and overly broad specifications of information needs. Second, existing systems typically require users to *explicitly* specify their profiles, often as a set of (possibly weighted) keywords or categories. It is difficult for a user to exactly and correctly specify their information needs to such a system. Third, state-of-the-art large-scale information filtering systems are typically built on the assumption that users change their interests only infrequently (e.g,. [Poi99, YGM95, Cha99]). If the profile does not keep up with the user's information needs, then precision and recall problems will quickly arise.

1.2 Maintaining Profiles

Many existing publish/subscribe-based systems require users to manually reflect their interest changes to the profile. This approach places the burden of identifying and making profile changes on the users. In contrast, other systems use a more *automatic* approach based on a technique called *relevance feedback* [Roc71, SB90]. In this latter approach, users provide feedback to the system about the data items that they have been sent (typically a binary indication of whether or not the item was considered useful). The system then uses this feedback to adjust the user's profile.

In this paper, we present a novel feedback-based approach for learning, representing, and maintaining user profiles. The algorithm is intended to support targeted dissemination of loosely structured documents such as web pages to large numbers of users. As such, it works well in an incremental fashion, where web pages are presented to users individually or in small batches.

A key feature that distinguishes our algorithm from previous work is that it uses a *multi-modal* representation of user profiles; i.e., a profile is represented as a collection of clusters of user interests rather than as a single entity. The algorithm automatically and dynamically adjusts the content and the number of clusters used to represent a profile based on feedback that it receives incrementally (i.e., one data item at-a-time). Even in this incremental mode, our approach provides more accurate results than a batch version of a more traditional approach. This flexibility also allows the algorithm to adapt profiles to cope with changes in user interests over time. Finally, as we will demonstrate, our multi-modal representation of profiles combined with its incremental nature allows it to be tuned to trade off effectiveness (i.e., accuracy) and efficiency, which makes it suitable for use in large-scale information dissemination systems.

The main contributions of the paper can be summarized as follows: First, we propose a new adaptive feedback-based algorithm that uses multiple *inter-related* profile vectors to represent user interests (as opposed to using a single, or multiple but *independent* vectors, e.g., SIFT). Second, the algorithm we propose can be parameterized to adjust the degree of its tendency to generate more or less complicated profiles. This flexibility enables it to trade off effectiveness and efficiency which, in turn, enables it to be tuned based on the requirements/characteristics of the target application and environment. Third, we evaluate our approach by using a detailed experimental framework based on the WWW pages obtained from the Yahoo! topic hierarchy [Yah99], analyzing the effectiveness, efficiency, and adaptability issues involved and comparing it to other algorithms that are representatives of the existing related approaches.

The remainder of the paper is organized as follows: In Section 2 we give an overview of the main issues related to user profile construction for push-based data delivery, focusing on the relevance feedback technique that we utilize in our algorithm. In Section 3 we present MM, the incremental profile construction and maintenance algorithm that we have developed. We discuss the experimental environment and workloads we used to test the ability of the algorithm to recognize relevant web pages in Section 4. The results of experiments based on Yahoo! categories are presented in Section 5. In Section 6 we discuss previous related work. Section 7 presents our conclusions.

2 User Profile Construction

Effective profile management requires techniques for representing data items and profiles, assessing the relevance of the profiles to data items, and updating the profiles based on user feedback. In this section, we briefly discuss these issues in the context of a push-based data dissemination environment.

2.1 Vector Space Model for Text and Profile Representation

Unlike databases, in which all correct systems must provide the same answer to a given query on a given database, information filtering systems can differ widely in the quality of filtering they provide. As such, comparing filtering approaches requires more than simply measuring the efficiency of the system. Rather, the *effectiveness* of the filtering is a primary metric for comparing such systems. Effectiveness is typically measured using *recall* and *precision*. Recall is the ratio of the number of relevant documents returned to the user vs. the total number of relevant documents that exist in the collection. Precision is the percentage of the documents returned to the user that are actually relevant. These two metrics are somewhat contradictory. For example, to achieve perfect recall, a system could simply return all the documents in the collection. Such an approach, however, would have terrible precision. In Section 4, we discuss these metrics further.

Our algorithm is based on the Vector Space Model (VSM) [Sal89]. In VSM, text-based documents are represented as vectors in a high-dimensional vector space where the value of dimensions are based on the words occurring in that document. Documents describing similar topics are likely to be close to each other, as they possibly include common words. A *profile* can also be represented as a vector (or a collection of vectors), which can be derived from the previously judged document vectors. In general, a profile vector should have a position close (in vector space) to those of relevant document vectors. If a new document is close to the profile, then it will also be close to other documents which are known to be relevant; thus, it will also be likely to be relevant.

In VSM, each document is represented as a vector of *term* and *weight* pairs. If there are *n* distinct terms in a document *d*, then *d* will be represented by a vector $V = ((t_1, w_1), (t_2, w_2), ..., (t_n, w_n))$. In general, a term is a word that exists in the document, and its weight is a measure of the relative importance of the term in the document. The standard process for computing the vector representation of a document includes *stop-list removal* and *stemming* [FBY92]. The weight of a term is commonly calculated by its *tf·idf* (*term frequency-inverse document frequency*) value: $w_{t,d} = tf_{t,d} * log_2(N/df_t)$ where $w_{t,d}$ is the weight of term *t* in document *d*, $tf_{t,d}$ is the frequency of term *t* in document *d* (i.e., *term frequency*), df_t is the number of documents that contain term *t* (i.e., *document frequency*), and N is the total number of documents in the collection. *Length normalization* is used to cope with documents of differing lengths. This is accomplished by computing the length of the vector and dividing the weights of its terms by this value. The angle between two vectors has been exploited as an effective measure of content similarity, and many systems use the *cosine similarity* measure to compute the similarity among document and profile representations. It is based on the inner (dot) product of two vectors, and can be formulated as:

$$cosine(v_1, v_2) = \frac{v_1 \cdot v_2}{|v_1||v_2|} = \frac{\sum_t w_{t,v_1} \cdot w_{t,v_2}}{\sqrt{\sum_t w_{t,v_1}^2} \cdot \sqrt{\sum_t w_{t,v_2}^2}}$$

2.2 Relevance Feedback

Relevance feedback is an effective information retrieval technique that can be used to form query (or profile) vectors based on document contents [Sal89]. The main idea is to use the documents that have already been evaluated by the user, emphasizing the terms that occur in relevant ones while deemphasizing those occurring in non-relevant ones in future formulations of the same query. More formally,

$$Q_{i+1} = \alpha Q_i + \beta \sum_{d \in R} v_d - \gamma \sum_{d \in NR} v_d,$$

where Q_i is the initial query vector, Q_{i+1} is the modified query vector, v_d is a vector representation of document d, α, β, and γ are the feedback parameters to be set, and R and *NR* represent the sets of relevant and non-relevant documents respectively. Several relevance feedback schemes have been proposed, which mainly differ in the way they set the parameters α, β, and γ. Among those, Rocchio relevance feedback [Roc71] is a well-known, effective scheme which instantiates the feedback parameters as $\alpha = 1$, $\beta = 2$, and $\gamma = 0.5$.

Traditional relevance feedback assumes that the document collection is fixed and that all the documents relevant to the query are available at the time of query reformulation. This is referred to as a batch relevance feedback approach. Batch approaches are not suitable for an information filtering environment, where there is a continual stream of documents and a relatively fixed query (or profile). Thus, an incremental approach is needed. Purely incremental feedback can update a query (or profile) for each individual document judgment that is received by a system. It is also possible to combine such judgments into groups and incorporate each group

Notation	Description
d	document
v_d	vector representation of d
f_d	user feedback for d ($\in \{-1, 1\}$)
P	user profile
p_i	profile vector

Notation	Description
$temperature_{p_i}$	temperature of p_i (initially set to 0)
$strength_{p_i}$	strength of p_i (initially set to 1)
δ	threshold value ($\in$ [0, 1])
λ	adaptability value ($\in$ [0, 1])
p_{active}	active profile vector

Figure 1: Basic notation

using a single update. Allan [All96] studied the effect of group size on the effectiveness of incremental relevance feedback (in a non-filtering environment). He showed that effectiveness increases with the group size, and that the highest effectiveness was obtained using all the judgments at once (i.e., in batch mode).

3 Multi-Modal Incremental User Profile Construction

State-of-the-art information filtering systems typically use a *single* vector to represent the information needs of a user. Some systems (e.g. SIFT), allow the use of multiple *independent* vectors (essentially multiple profiles for a single person). As we will show in Section 5, using a single vector may limit the effectiveness of the profile. Using multiple profiles can help solve this problem, but requires that the decision on how many profiles to maintain be made in a fairly static fashion, thereby reducing the ability of the system to adapt to changes in users' interests.

To address these problems, we have developed MM, a new algorithm for automatically constructing and maintaining user profiles based on user feedback. MM represents user profiles as a set of multiple vectors, size and elements of which change *adaptively*. In this section we describe MM in detail.

3.1 Multi-Modal Profile Structure

MM represents a user profile P as a set of *profile vectors* $\{p_1, p_2, ..., p_n\}$ where

$$p_i = ((t_{i_1}, w_{i_1}), (t_{i_2}, w_{i_2}), ..., (t_{i_m}, w_{i_m})),\ i = 1, 2, ..., n$$

In fact, the number of profile vectors is not constant; it changes in time based on the feedback pattern obtained from user. Individually, each profile vector represents only a portion of a user's information needs; e.g., a relevant concept. Collectively, however, the profile vectors model the user comprehensively.

MM also maintains two auxiliary integer variables $strength_{p_i}$ and $temperature_{p_i}, \forall p_i \in P$. The former is an indication of the *confidence* that we have for the corresponding vector as a representative of a portion of the user's information needs. In other words, the vectors which represent more stable information needs have more strength. If the strength of a profile vector drops below a certain value (1 in our case), we lose our confidence in the ability of that vector to represent a relevant concept, so we remove the vector as it is likely to degrade the modeling effectiveness of the overall profile. The strength of a vector is adjusted using its temperature value. The temperature keeps track of the *pattern* of the feedbacks received. Initially, it is set to 0, indicating a stable condition. Any non-zero temperature value indicates an unstable condition with negative values causing strength decrease and positive values causing strength increase.

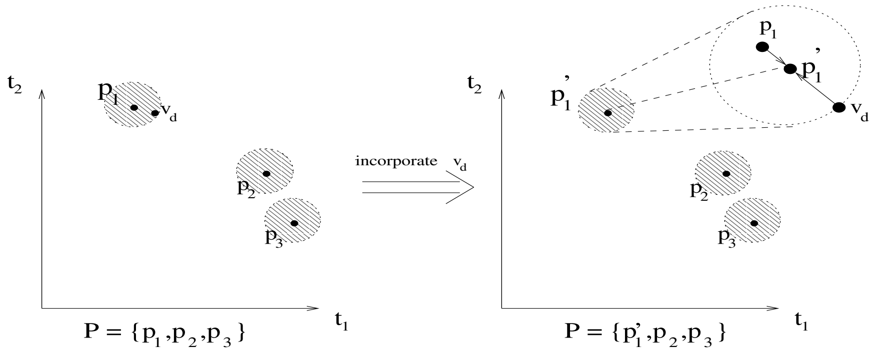

Figure 2: Incorporating a document vector into a profile vector

3.2 The MM Algorithm

The general structure of MM is based on the traditional incremental clustering algorithm applied to document clustering [FBY92]. To simplify the presentation, we first describe the fundamentals of MM by describing the procedure for incrementally clustering document vectors. The basic idea is to maintain clusters of document vectors. Each cluster is represented by using a single *representative* vector. The first document is assigned as the representative for the first cluster. When a new document is processed, its similarity with the representatives of all the existing clusters is calculated. If the similarity of the closest cluster is greater than a threshold (specified by δ), then the document vector is added into that cluster and the cluster representative is recomputed by moving it towards the new document (the impact of the new document is controlled by a parameter λ). Otherwise, if the similarity is smaller than δ, the document vector is used to initiate a new cluster. The size of the space covered by each cluster is the same and is defined by δ. Therefore, the larger the number of clusters used, the bigger the portion of the covered vector space is (assuming that the clusters are not completely overlapping).

3.3 Updating Vector Strengths

MM builds upon this basic incremental clustering algorithm with structures and procedures specifically designed for multi-modal profile construction in a filtering environment. Each profile vector basically corresponds to a cluster representative. Each relevant document vector can either create its own profile vector or be incorporated into an existing profile vector. Non-relevant document vectors, on the other hand, cannot create their own clusters as a profile represents relevant concepts only. They can, however, be incorporated into other profile vectors. Two similar profile vectors may be merged into a single one, in order to avoid redundancy and decrease profile size. A profile vector may also be deleted (i.e., removed from the profile) if MM deems it to be no longer representing a concept relevant to the user.

3.4 The Main Loop

The pseudocode for MM is shown in Appendix A (refer to Figure 1 for notation). The algorithm works as follows: When a new relevance judgment is received, MM first checks if the user profile P is empty and the user feedback for the document (i.e., f_d) is positive. If so, it inserts the document vector, v_d, into P, setting the strength to 1 and the temperature to 0. The feedback is simply ignored when the profile is empty and the feedback is negative. This is because the user profile represents the concepts that are *relevant* to the user, and not the *irrelevant* ones.

If P is non-empty, then the profile vector which is most similar to v_d is identified, and becomes the

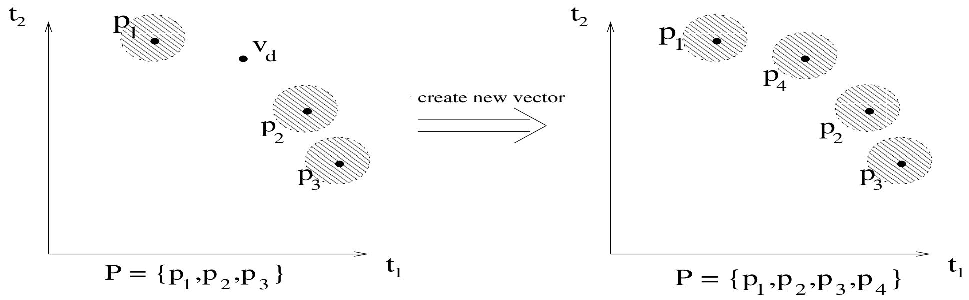

Figure 3: Creating a new profile vector

active profile vector, p_{active}, for the profile. The document vector will have all its further interactions with p_{active}. The relevance of the profile to the document is computed as the similarity between p_{active} and v_d (i.e., cosine(p_{active}, v_d)). If this relevance value is higher than a certain threshold (specified by δ), then the effect of the feedback is reflected to the profile by incorporating v_d into p_{active}. This incorporation of the document vector into the active profile vector is done by by moving p_{active} linearly towards v_d according to parameter λ.

An example of incorporating a new document vector into a profile vector is shown in Figure 2.[1] In the example, P is initially $\{p_1, p_2, p_3\}$. If a document vector falls within the shaded *similarity* circle for a profile vector, then it is deemed *similar enough* (i.e., eligible for incorporation in that profile vector). In this example, the document vector v_d falls inside the circle of p_1, which becomes the active profile vector p_{active} for p_1. It is possible that the document vector falls inside two or more such circles, however there can only be a single active vector at any time; namely the one most similar to the document vector.[2] The document vector v_d is then incorporated into p_1 by pushing p_1 closer to v_d. Note that the size of the profile does not change as a result of an incorporation operation.

On the other hand, if the relevance value of v_d is lower than δ (i.e., it falls outside of all similarity circles), then v_d is inserted into P as a new element and the size of the profile increases. This case is illustrated in Figure 3. The strength and the temperature of the new profile vector are set to 1 and 0, respectively.

When a document vector is incorporated into p_{active}, then the strength of p_{active} is updated.[3] In order to understand the operation of this procedure, it is necessary to understand what the temperature of a profile vector represents. The temperature is an integer value that keeps track of the feedback pattern as observed by that vector. The temperature of a profile vector is initialized to 0 at the time of creation. As long as a profile vector does not receive a negative feedback, its temperature remains unchanged; i.e., 0.

The first time a vector receives a negative feedback[4], it enters an *uncertainty period* which continues as long as its temperature remains non-zero. A vector may exit its uncertainty period either when it is deleted from the profile (its strength becomes less than 1), or when its temperature again becomes 0. The latter case occurs when the vector receives more positive feedbacks than negative feedbacks during its uncertainty period. Informally, the uncertainty period implies that we are *uncertain* about keeping this vector as part of the profile, therefore we continue to observe the feedback pattern on that vector.

[1]For ease of illustration we show the vectors in a hypothetical 2-D vector-space. Note that this illustration shows relevance as the Euclidean distance among vectors, which is not the way relevance is defined in MM. Regardless of the relevance measure, however, the underlying principles remain the same.

[2]In case of two or more vectors having the same similarity, one of them must be selected; e.g., randomly or based on strength.

[3]The pseudocode for the procedure to update the strength of the active profile vector is shown in Appendix A.

[4]When we talk about "a profile vector receiving feedback", we implicitly state that the mentioned vector is the active profile vector and that a document vector is incorporated in it.

At each update step, based on the current temperature value of the vector, the strength is updated using a simple exponential decay function. The parameter c (see Appendix A) is a positive constant that controls the decay rate. It is set to 0.5 in all our experiments. If the temperature is non-zero (i.e., during an uncertainty period), the strength of the vector decreases or increases exponentially if its temperature is negative or positive, respectively. If the temperature is zero, then the strength is incremented to reflect the incorporation of the new feedback into the vector. We illustrate this update process with two examples:

Example 3.1 Let the strength of the vector be S, its temperature be 0, and let us assume that the vector received a feedback sequence of [-1, -1, 1, 1, 1]. After receiving the initial negative feedback, the vector enters an uncertainty period and its temperature takes the sequence of values [-1, -2, 2, 1, 0] after each respective feedback. Meanwhile the corresponding strength values after each feedback are [Se^{-c}, Se^{-3c}, Se^{-c}, S, S]. Initially the strength decreases after each negative feedback, but then (assuming that it is still above 1) it regains its original value after receiving two positive feedbacks. The final positive feedback ends the uncertainty period and, once again the vector is able to gain further strength when it receives positive feedback. Note that the vector cannot gain more strength than it lost during this period. □

Example 3.2 Consider a case where the user had a shift in interest and began to provide negative feedback for a concept he or she used to be interested in. The vector (partially) representing that concept will then receive consecutive negative feedbacks and its strength will decrease exponentially with decreasing temperature. Unless it receives positive feedback, its strength will eventually drop below 1 and it will be removed from the profile. Note that during the uncertainty period, the temperature value does not lose the count of the difference between the number of negative and positive feedbacks received. Either because of noise in input data (e.g., due to the well-known limitations of the keyword-based vector space approach) or an inconsistent behavior on part of the user, it is always possible for a vector to get a positive/negative feedback which it should not ideally receive. Let us assume that the sequence of feedbacks received by a vector with relevant parameters set as in the previous example be [-1, -1, -1, 1, -1, -1]. The corresponding temperature and strength values after each feedback will respectively be [-1, -2, -3, 3, -3, -4] and [Se^{-c}, Se^{-3c}, Se^{-6c}, Se^{-3c}, Se^{-6c}, Se^{-10c}]. The fourth feedback is inconsistent with the general pattern, but the algorithm is able to restore its effect quickly. In fact, this procedure is designed to provide fast response to interest shifts with tolerance for noise and/or inconsistent feedback. □

3.5 Adjusting the Profile Size

If the strength of the active profile vector drops below 1 (remember that the strength of a vector is initialized to 1 when created), then it is assumed that the vector does not represent a concept relevant to the user anymore. Thus, it is immediately removed from the profile. Recall that the active profile vector was moved towards the document vector, meaning that its position in the vector space was changed. If two profile vectors come close (enough) to each other, intuitively it means that they represent similar (if not the same) concept(s) in vector space. Therefore, a single vector may possibly be sufficient to represent the concept(s) that are represented by the two vectors. This deletion ability allows MM to avoid multiple profile vectors redundantly representing similar concepts, thus allowing a more compact representation.

The merge procedure (presented in Appendix A) checks whether a merge is possible between p_{active} and the other profile vectors. The profile vector closest to p_{active} is identified and if the similarity between them is greater than δ, they are merged. The merge is done in a way very similar to that in incorporating a document vector to p_{active} (see Figure 2). The two vectors being merged are pushed towards each other linearly, however, using their relative strengths rather than a constant value this time. The vector having more strength stays closer to its original position, and the other vector moves more. This is the effect we would like to achieve, because the strength of a vector is an indication of its stability. Notice that we only consider the vector pairs containing p_{active} for merging. This is because the only profile vector which moved to a new position is p_{active} and the inter-vector distances for the other vectors are the same as before. After the merge, however, the similarity between the combined vector and another profile vector may be larger than δ, requiring another merge. We do not consider this situation and choose to allow only a single merge operation in a single iteration for efficiency reasons. The other potential merge operations, if any, are accomplished lazily in future iterations if necessary.

3.6 Discussion

There are several parameters that control the way MM behaves and thus have to be set properly. One of them is the *threshold parameter*, $\delta \in [0, 1]$, which is mainly used to decide whether a new document vector should be incorporated into an existing profile vector or create a new profile vector. This parameter can be used to control the number of profile vectors. If δ is set to 1, then all (distinct) relevant documents will form their own profile vectors, achieving a very fine granularity user model, but exploding the profile size. At the other extreme, if it is set to 0, all vectors will be incorporated into a single profile vector and the number of profile vectors will always be 1. In this case, the overhead of profile management is extremely low, however the effectiveness of the profile is limited. In this paper, we are actually interested in intermediate values which will provide an optimal effectiveness/efficiency tradeoff for a given application.

The other important parameter is the *adaptability parameter*, $\lambda \in [0, 1]$. As mentioned before, it controls the rate with which the active profile vector is pushed towards the document vector. In other words, it decides how fast the user profile should adapt itself based on feedback. If it is set to 1, active profile vectors will be replaced by document vectors at each feedback, resulting in maximum adaptation (i.e., memoryless mode). If set to 0, on the other hand, active vectors will not change with feedback, and virtually no adaptation will take place.

MM also includes new operations and structures designed specifically for filtering environments. The new operators we propose, as we will show, control the number of vectors that form the profile and allow for fast adaptation when there is a shift in user interests.

4 Experimental Environment

Filtering systems developers typically rely upon the technique of *user simulation* in order to understand and quantify the effectiveness of their solutions (e.g., [All96, LMMP96]). Due to the absence of WWW-oriented filtering workloads in standard suites such as TREC, we devised a benchmark using categories of web pages obtained from Yahoo! [Yah99]. In this section, we describe our approach for evaluating the MM algorithm.

4.1 Document Collection

The first problem that needs to be addressed when designing a filtering study is the identification of a suitable document collection. In our study, we used web pages referenced from the top two levels of the Yahoo! category hierarchy (which is formed by human editors). We chose ten top-level Yahoo! categories and ten sub-categories (i.e., second-level categories) for each of the selected top-level categories. In the remainder of this paper, we denote a top-level category i by C_i, and a second-level category j categorized under C_i by C_{ij}, where $i, j \in [0, 1, ..., 9]$.

To obtain the actual text of the web pages, we followed the hyperlinks at the second-level of the Yahoo! hierarchy and gathered the corresponding pages. We ignored cross-links, other pages belonging to Yahoo!, and the pages that resided on non-USA domains. We then converted these documents to their vector representations by removing the HTML tags and other non-words, followed by stop-list removal and stemming.[5] This process is illustrated in Figure 4.

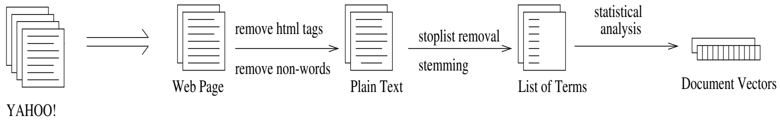

Figure 4: Formation of document vectors

The resulting document collection we used consists of 900 documents which are uniformly distributed over all categories. As in any study of a learning algorithm, the input must be divided into a *training set*, which is presented to the system for learning purposes, and a *testing set*, which is then used to evaluate the trained system. In our studies we used $2/3$ of the documents for training. At that point, the profiles were frozen and their quality was measured using the remaining $1/3$ of the documents as a testing set.

For each document vector we only keep the 100 highest-weighted terms as: (1) our early experiments revealed that it is safe to use the highest-weighted 80-120 terms for representing documents without any degradation in effectiveness[6], and (2) it is important to keep the vector size as small as possible due to storage and computational requirements, particularly in a large-scale information filtering environment.

4.2 User Simulation

We simulate the behavior of a typical user by assuming that the user is interested in a subset of our Yahoo! categories and gives feedback correspondingly. In other words, a simulated user is assigned a synthetic profile (*SP*) consisting of a subset of the categories, and gives positive feedback to a document only if that document is classified under a category that appears in the synthetic profile (i.e., $f_d = +1$ if $category_d \in$ *SP*). All the other documents are given negative feedback (i.e., $fd = -1$). In our experiments, either *SP* $\subset \{C_0, C_1, ..., C_9\}$, or *SP* $\subset \{C_{00}, C_{01}, ..., C_{99}\}$; i.e., the synthetic profile is defined completely either by top-level categories or by second-level categories.

4.3 Methodology and Performance Metrics

We chose to base our evaluation methodology on the one used in the *routing track* of the TREC [VH96] benchmark. The idea is to have the system score and then rank-order a collection of documents based on their likelihood of relevance to a particular profile (or query). We chose this approach over an alternative based on the TREC *filtering track*, which uses a binary filtering function that simply accepts or rejects each new document as it arrives. The reason for this decision is that evaluations of this latter approach indicate that results obtained using it are heavily dependent on the specific filtering function (such as the setting of relevance cutoffs, etc.) used [Hul97]. This function must be optimized separately for each technique being evaluated. Thus, an evenhanded comparison of learning techniques using this latter benchmark is problematic, calling into question the quality of the filtering function used for each of the competitors. By using the rank-based measure provided

[5]We used a stop-list of 429 stop-words and a Porter stemmer (see [FBY92]).

[6]In fact, effectiveness is sometimes slightly higher with smaller vectors, due to the well-known problem of *overfitting* [Mit97] that arises in machine learning.

by the *routing* benchmark, however, this problem is avoided. Any approach that can accurately rank documents according to their relevance, can also be used as the basis for an accurate filtering function, by defining a ranking cutoff[7] for relevance, below which, any documents will be considered irrelevant.

The experiments are executed as follows. Each run is begun by randomly selecting categories to form a *synthetic* user profile of desired complexity. The user profile is initialized to be empty.[8] We then present the documents in our training set one by one to the system along with the corresponding feedback values obtained from the synthetic profile as defined in Section 4.2. After training is completed, the profile algorithm is disabled (i.e., the profiles generated are 'frozen') and the system is evaluated by having the profile it generated score and rank-order a set of test documents that it has not yet seen, based on their likelihood of relevance.

The main effectiveness metric used in the experiments is a variant of *non-interpolated average precision* ($niap$), which is a rank-based metric used in TREC. This metric integrates precision and recall values into a single effectiveness measure. Given a ranking of documents in their predicted likelihood of relevance, $niap$ is defined as follows: Starting from the highest ranked document, the actual relevant documents are counted. If the ith relevant document has rank r_i, then $niap = \frac{\sum \frac{i}{r_i}}{T}$, where T is total number of relevant documents in the test collection. For example, assume that there are 3 relevant documents in the test collection and the filtering system assigns the ranks 2, 4, and 6 to these documents, then $niap = \frac{1/2+2/4+3/6}{3} = 0.5$. In other words, $niap$ computes the mean of the precision values at each relevant document's position in the ranked list. With the example system, which operates at an $niap$ of 0.5, (on average) half of the documents it deems relevant are in fact relevant to the user, while the other half are not. Higher $niap$ values imply better use of system resources (e.g., bandwidth) and higher user satisfaction. In the remainder of the paper, we use the term "precision" to mean $niap$, unless otherwise specified.

In addition to precision figures, we also measure the size of a user profile in terms of the number of vectors constituting the profile. This metric is important because it dictates the storage requirements for profile management. Such requirements can become a serious concern in a large-scale filtering environment. As with document vectors, we represent each profile vector with 100 (term, weight) pairs due to the reasons discussed previously. The storage benefits for profile vectors, however, are far more important than for document vectors as the latter are typically only retained for a short duration, while profile vectors are stored and maintained for long periods of time. Profile size also has implications on filtering efficiency. Larger number of profile vectors typically indicate higher filtering times. The filtering cost, however, is not linearly proportional to the number of vectors in the profile since well-known indexing techniques are applicable.

5 Performance Experiments

5.1 Algorithms Studied

In this section, we present the results of our experiments using MM for profiling WWW page interests. We also present results for two other algorithms, namely (purely) Incremental Rocchio (RI) and Group Rocchio (RG). RG is the incremental relevance feedback algorithm studied by Allan [All96] (see Section 2.2) that uses a group of judged documents for updating the profile using relevance feedback. RI is a special case of RG where the group

[7] The ranking cutoff parameter should not be confused with the threshold parameter δ that controls the size of the profile. Typically, these two parameters will be set differently. See [Cal98] for a recent study on learning ranking cutoffs.

[8] It is also possible to initiate the experiments with initialized profiles. Such an approach is more realistic in a real-world implementation as it could reduce the training time significantly. It, however, introduces an additional variable into the study, namely, the quality of the initial profile.

size is set to 1. We adopted the Rocchio based formula used by Allan and calculated the weights of the terms forming the profile vector for RI and RG as $w(t)_{i+1} = w(t)_i + 2w_{t,R} - \frac{1}{2}w_{t,NR}$, where $w(t)_i$ and t, $w(t)_{i+1}$ are the current and the updated (after feedback) weights of t in the profile, and $w_{t,R}$ and $w_{t,NR}$ are the weights of t in relevant and non-relevant documents, respectively, as defined using the formulas given by Allan:

$$w_{t,R} = \frac{1}{|R|}\sum_{d \in R} bel_{t,d}, \quad \text{and} \quad w_{t,NR} = \frac{1}{|\text{NR}|}\sum_{d \in NR} bel_{t,d}$$

where $bel_{t,d}$ is given by: $bel_{t,d} = 0.4 + 0.6 \cdot tfbel_{t,d} \cdot idf_t$ with

$$tfbel_{t,d} = \frac{tf_{t,d}}{tf_{t,d} + 0.5 + 1.5 \cdot len_d/avglen}, \quad idf_t = log(\frac{N+0.5}{docf_t})/log(N+1)$$

where len_d is the length of document d, and $avglen$ is the average length of documents in the collection.[9] We use this weight calculation method for all of the learning algorithms studied here.

In order to determine appropriate training times, we investigated the learning rate of MM and observed that its effectiveness increases rapidly, and levels off somewhat after seeing about 200 documents, but continues to increase. After training with 400-500 documents, however, we observed no significant increase in effectiveness. The RI and RG approaches stabilize slightly faster. Unless otherwise stated, all the results presented here are taken after training with 500 documents.

In all the experiments we present, we fix λ at 0.2. We conducted numerous preliminary experiments utilizing all levels of our experimental parameters and different profile schemes. We observed that λ, if set in range [0.1,0.3] in general had good performance for almost all cases, and found little difference in terms of the precision values obtained for different settings of λ in this range. We decided, therefore to fix λ for the experiments described in this paper.

The test runs for all of the algorithms begin with an initially empty profile. The training set is then presented to the system, followed by the test set, during which the experimental results are obtained. The results presented in the following graphs are the average of at least four runs with (different) user interest categories randomly chosen according to the specification of the workload under study.

We begin by investigating the retrieval effectiveness of the three profile learning techniques using interest categories drawn from the top-level Yahoo! categories. Figure 5 shows the precision results for three different interest ranges, covering of 10%, 20%, and 30% of the documents (i.e., 1, 2, and 3 top-level categories out of the 10 in the database). For each interest range, the precision results are shown for (from left to right) incremental Rocchio (RI), group Rocchio (RG), and MM. As can be seen from the figure, the results are consistent across all three interest sizes: MM provides the highest precision, followed by RG, followed by RI.

5.2 Top-Level Retrieval Effectiveness

Comparing the two Rocchio implementations, the results show that as described in Section 2.2, the effectiveness of relevance feedback increases with the group size. In fact, we also ran these experiments using a batch version of Rocchio, in which all 500 training documents were presented to the algorithm at once. This approach, which represents a (non-incremental) best case for Rocchio, had precision values of roughly 3-4% more (in absolute terms) than RG in the three cases studied here. Thus, our fully incremental algorithm, which is presented with only a single document at a time, significantly outperforms even the batch Rocchio approach.

[9] We computed the values for the collection-based parameters (e.g., $avglen$, $docf_t$) by statistical analysis of the document collection. In a real filtering setting, however, this information must be collected incrementally over time.

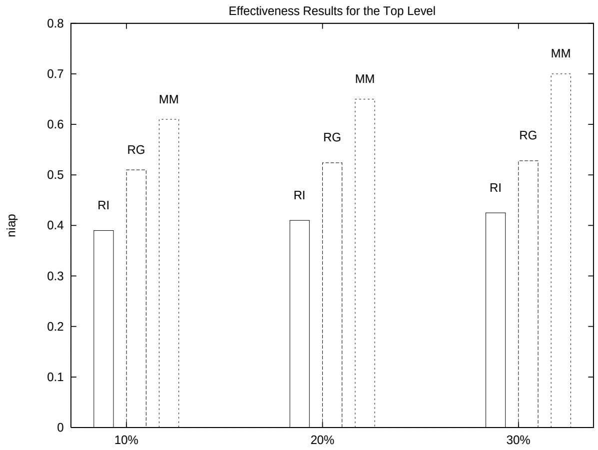

Figure 5: Precision values for the top-level: $\delta = 0.15$

Experiment	RG	MM
1.10	+30.1%	+56.1%
1.20	+27.4%	+58.7%
1.30	+24.5%	+64.2%

Figure 6: Percentage increase in precision over incremental Rocchio (RI)

Comparing the results for the three interest ranges in Figure 5, it can be seen that the benefit of MM over the others grows as the number of categories in the profile is increased. These results are shown in Table 6. This behavior demonstrates a fundamental benefit of the multi-modal approach. As more categories are added, the number of profile vectors maintained by MM can be increased, allowing the profile to automatically adjust in order to model the increasingly disparate interests of the (simulated) user. In contrast, since the Rocchio algorithms maintain only a single profile vector, the documents from the different categories must be lumped together, resulting in a less accurate model of the user's interests. [10]

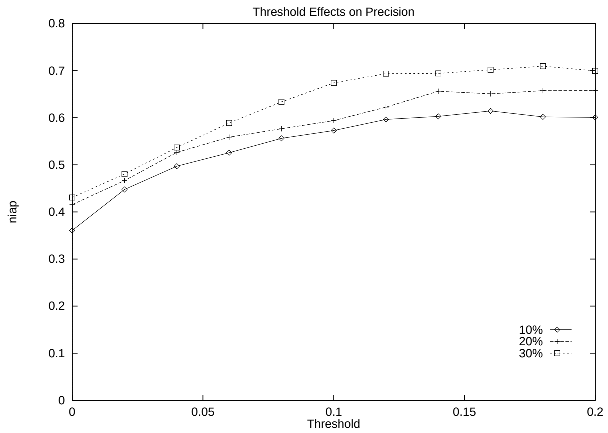

Figure 7: Effects of threshold on precision for the top-Level

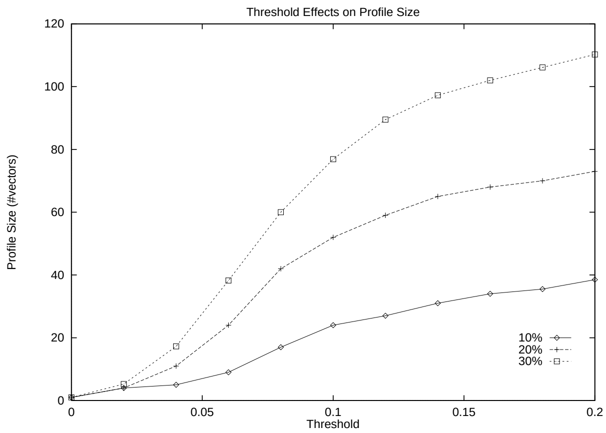

Figure 8: Effects of threshold on profile size for the top-Level

5.3 Threshold Effects

The importance of maintaining multiple vectors for this workload is demonstrated in Figure 7, which shows the precision obtained by MM, as the threshold parameter δ is increased. Recall that δ determines the tendency for MM to create new vectors. When $\delta = 0$, a single vector is maintained, and MM performs similarly to RI. As δ is increased, MM becomes more likely to create additional vectors. At an extreme value of $\delta = 1$ (not shown),

[10] In fact, there is a slight improvement in precision for RI and RG as categories are added. This improvement can be attributed to the increase in the percentage of relevant documents in the testing set, which raises the probability of a "lucky" guess.

MM maintains a separate vector for each relevant document presented to it. This case is similar to the "nearest relevant neighbor" (NRN) method, which was studied in [FD92].

Keeping vectors for all relevant documents, however, is not practical for an information filtering environment as the number of documents presented to such a system grows monotonically over the life of the system. Fortunately, as is indicated in Figure 7, such a high setting for δ is not necessary. Beyond a value of approximately 0.15 (the default value used in these experiments), the precision obtained by the algorithm begins to level out. In fact, with high thresholds, MM will be susceptible to *over-fitting*, which can negatively impact effectiveness. Over-fitting is particularly a problem in noisy environments (e.g., such as the WWW). A final argument for not retaining vectors for all documents is adaptability. As is discussed in Section 5.5, a key benefit of MM is its ability to adjust to changes in user's interests. An approach that maintains vectors for all (or most) relevant documents ever seen would adapt much more slowly.

Figure 8 shows the number of vectors maintained by MM for each of the three workloads as δ is increased. The important fact to notice here is that for a given threshold, MM keeps more vectors as the number of relevant categories is increased. That is, as the percentage of relevant documents increases, more vectors are needed to represent the concepts exemplified by those documents. Taken together, Figures 7 and 8 demonstrate how the δ parameter allows the MM algorithm to be tuned to trade off precision for profile size. MM is capable of spanning the range of algorithms from single-vectored Rocchio, to a vector-per-document approach such as NRN. Unlike either of those extreme algorithms, however, it allows for middle-ground solutions that provide good precision while maintaining moderate storage requirements and good adaptability.

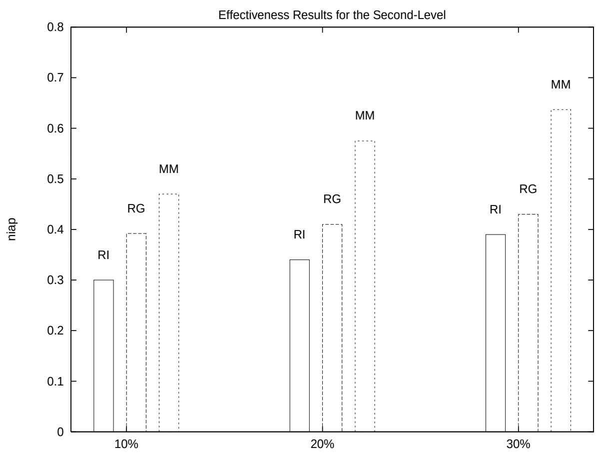

Figure 9: Precision values for the second-Level: $\delta = 0.15$

%relevant docs.	RI	RG	MM
10	-23.2%	-23.3%	-22.9%
20	-17.5%	-21.8%	-11.5%
30	-14.1%	-18.5%	-9.0%

Figure 10: Percentage decrease in precision when second-level is used

5.4 Profile Complexity

We also compared the algorithms using categories from the second-level. This workload is likely to generate more complex profiles as the relevant documents are chosen from a wider, more disparate group of topics. The results are shown in Figure 9. Qualitatively, the results are similar to what was seen in the previous experiment with top-level categories. MM is the best, followed by RG, followed by RI. All of the approaches have slightly lower precision here, however. The decrease in precision compared to the top-level case for each of the algorithms is shown in Figure 10. As can be seen in the table, MM suffers the least of the three algorithms, while RG has the highest relative drop. Again, the flexibility of the MM approach allows it to adapt to the more complex workload. In this case, MM maintains a small number of additional vectors (3 or 4) for each of the workload sizes compared to the corresponding workload size in the top-level case discussed previously.

5.5 Interest Changes

As stated in the introduction, an important requirement for an incremental profile generation algorithm is that it must be able to recognize and adapt to changes in users' interests. In this section we evaluate the alternative learning techniques in this light. We examine four types of changes: complete and partial changes in the categories of interest, addition of a new category to an existing profile, and deletion of a category from an existing profile. As before, we compare the MM, RI, and RG algorithms. In addition, we also measure a version of MM, called MM-No Decay (MMND), in which the decay function (i.e., the removal of vectors) is disabled. Recall that the Rocchio techniques have an implicit type of decay in which the old vector is augmented with information about new documents (see Section 2.2).

While we studied many different scenarios, due to space considerations, we only show results from a single, representative case here. For all experiments shown, MM was run using the default values of $\delta = 0.15$ and $\lambda = 0.2$, and RG was run with a group size of 100 documents.[11] All of the results shown in this section were obtained using the 20% top-level category workload (i.e., relevant documents are chosen from two top-level categories). To see how quickly the learning techniques adapt, we initially trained them using 200 documents. At that point, the synthetic profile is changed instantaneously, and we measure how quickly (if at all) the precision values obtained by the various techniques recover. In each of the graphs that follow, we plot the precision as documents are presented to the system.

5.5.1 Shifting Interests

In these experiments, we study the adaptability of the techniques when the number of categories in which the user is interested remains constant, but the particular categories are changed. We show two cases. Figure 11 shows the effectiveness of the learning techniques in the case where one of the categories of interest is changed (after the 200th document has been seen) while the other remains fixed. That is, before the shift: *SP* $= \{C_i, C_j\}$; while after the shift *SP* $= \{C_i, C_k\}$. In this case, as before, MM (as well as MMND) initially achieves higher

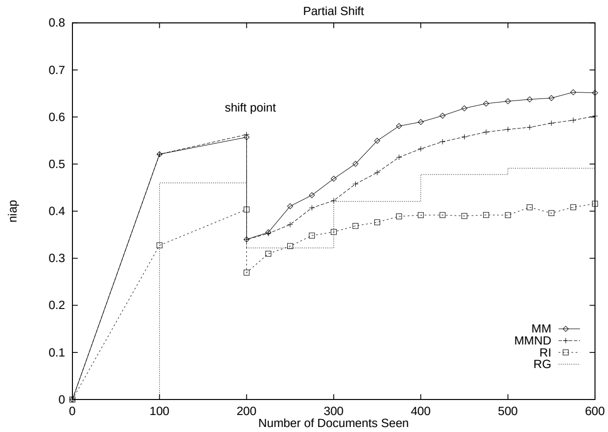

Figure 11: Partially changing interests

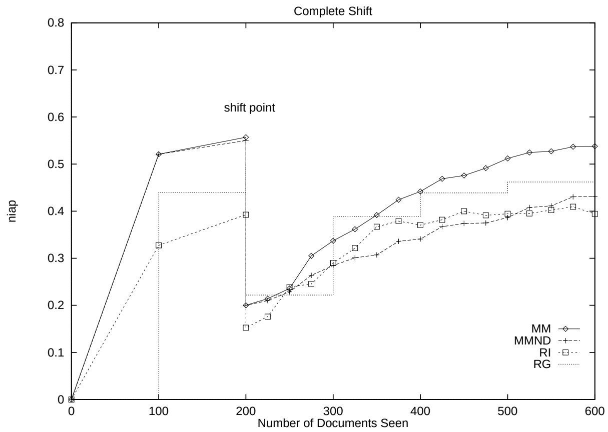

Figure 12: Completely changing interests

[11] A group size of 100 requires somewhat more space than MM in this case, which requires at most 66 vectors here. At a group size of 66 the precision of RG is within 2% of the values shown in these experiments.

precision than the Rocchio techniques. After the shift, however, MM, and RG recover (i.e., regain the precision that they had at the shift point) fastest, followed by MMND and RI. In fact, MM recovers slightly earlier than RG here, but only because RG waits to collect an entire group before changing the profile. Recall that larger groups improve the effectiveness for Rocchio in the static case. In the dynamic case, however, larger groups result in longer periods of lower effectiveness, which could have an impact on user satisfaction in an information filtering environment. RI adjusts reasonably well here, but its effectiveness remains well below that of the MM approaches throughout the entire document sequence.

Comparing MM and MMND, it can be seen that MM recovers much faster than MMND (they have similar precision at the shift point). With decay, negative feedback accelerates the removal of the vectors representing the concepts of the dropped category. Without decay, these vectors remain in place longer, impacting the precision results. Thus, decay significantly improves the effectiveness of MM when users interests change dynamically (which we would expect to be the normal situation for many applications). Fortunately, the fact that both MM and MMND have similar precision up to the shift point shows that using decay does not harm the precision of MM in periods of static interests.

Figure 12 shows the effectiveness of the algorithms for a more complete shift. In this case, the user's interests are completely changed at the shift point. More formally, before the shift: *SP* $= \{C_i, C_j\}$; and after the shift *SP* $= \{C_k, C_l\}$. While this case is less likely to happen than the partial shift, we use it to investigate the behavior of the algorithms in an extreme case. Here, all the past relevance judgments are invalid; each algorithm has to realize this and forget all those judgments in order to recover.

Comparing Figure 12 to the partial change case shown in Figure 11, it can be seen that the MM approaches take longer to recover in the more extreme case than they did in the partial case. MM with decay recovers its original precision somewhat more slowly than RG does here, but it is important to note that even before it is fully recovered (i.e., after the 400th document), its precision is superior to that of RG. Without decay, however, MMND recovers much more slowly than in the previous case, and in fact, has lower precision than RG throughout the entire test range shown here. In this case, the vectors existing prior to the shift point provide no valuable information, and thus need to be destroyed as quickly as possible. Without a decay function, these old vectors can impact effectiveness for a long time. RI adjusts reasonably quickly in this case, but its effectiveness remains below that of MM throughout the entire document sequence.

The previous cases represented fairly dramatic shifts in user interests while keeping the number of categories in the interest range constant. Such user behavior is not expected to be likely in practice, but is useful for demonstrating the tradeoffs of the different algorithms in the absence of effects caused by changes in the size of the relevant document set. In this section, we briefly look the case of more gradually changing interests.

5.5.2 Adding and Deleting Interests

Figure 13 shows the case where a new top-level category is added to an existing set of interests (originally containing a single category), for example, if a user becomes interested in a new hobby. That is, before the shift: *SP* $= \{C_i\}$; and after the shift *SP* $= \{C_i, C_j\}$. In this case, since there is only an extension of the user interests, the previous (positive) relevance judgments remain valid. For this reason, there is no difference in the effectiveness and recovery time of MM and MMND. This result extends our earlier result, by validating that the decay function has no negative impact on the effectiveness of MM even for changing interests, as long as existing interests are not dropped. RI and RG again show reasonable recovery behavior but overall lower effectiveness. RG dominates RI except for one small region (between the 200th and 300th documents), during which RG is

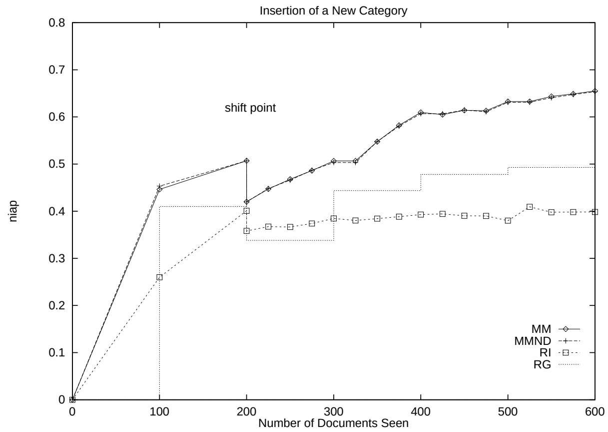

Figure 13: Adding interests

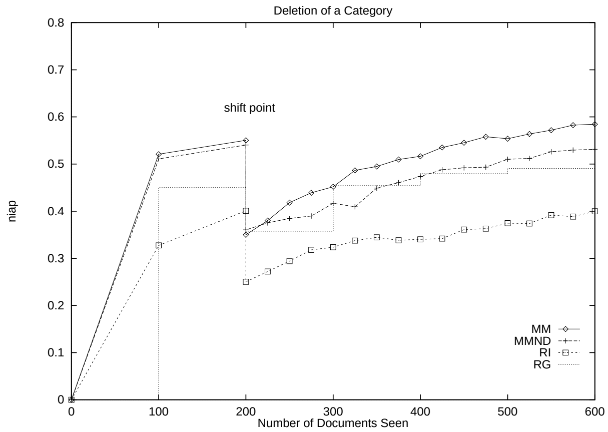

Figure 14: Deleting interests

collecting a group of relevance judgments before applying them to its vector.

Figure 14 shows the complementary case to the previous one, in which a user is initially interested in two top-level categories and then one of them is made irrelevant. For example, a student's interest in a category for a school project can drop suddenly after that project is finished, people can lose interest in a hot news story after it has become tiresome. More formally, before the shift: $SP = \{C_i, C_j\}$; and after the shift $SP = \{C_i\}$. In this case, the results confirm what we have seen before. In the presence of dropped interests, the decay function of MM speeds its recovery. The precision of RG recovers quickly, but remains below (or in one case, equal to) that of MM with decay. Thus, across all the scenarios shown, MM provides the highest effectiveness; it has higher precision for static situations, but also recovers fast enough to preserve its advantage when the user's interests change.

6 Related Work

There has been a huge volume of research on text-based profile construction in information retrieval community, especially in the framework of TREC routing and filtering tasks [VH96]. In the routing task, the system is given a set of documents relevant to a topic and is asked to rank-order an unseen set of documents according to their estimated relevance based on the constructed profile. In the filtering task, the system is required to make a binary decision as to which document is relevant or non-relevant. The main emphasis of TREC tasks, however, has always been on the effectiveness of the participating systems, rather than on their efficiency. Most of the techniques used for these tasks require batch processing of previously judged documents, imposing relatively high storage and computation costs, and thus, making them inappropriate for large-scale filtering environments. Furthermore, the documents used in these benchmarks differ from the wide variability of typical web pages, making it difficult to extrapolate results from these benchmarks to performance on the WWW.

Publish/subscribe protocols have been a subject of increasing interest in the database and data management communities. Recent projects such as the C3 project at Stanford [CAW98], the CQ (continuous queries) project at OGI [LP98], and the Grand Central project at IBM Almaden [IBM97] all contain a user profile management component. To date, however, these projects have not emphasized learning-based acquisition and maintenance of profiles.

The machine learning community, of course, has also shown great interest in different aspects of profile generation, especially in the framework of personalized information filtering (e.g., [SM93, Ste92]). For reasons of brevity, we only discuss previous research which is directly relevant to our work and the database community.

The work most closely related to ours is that of Allan, who studied the utility of relevance feedback for information filtering environments [All96]. He investigated the case where only a few judged documents are available each time, and showed that highly effective results can be achieved using relatively few judged documents. He also addressed the significant issues of reducing storage requirements and coping with shifts in user interests that arise in an information filtering environment. The main difference of our work is the introduction of a parametric approach that adaptively changes the number of vectors used to represent profiles.

Previously, Aalbersberg evaluated the effectiveness of incremental relevance feedback [Aal92], however, from the standpoint of an information retrieval environment. Lam et al. addressed the issue of shifts in user interests [LMMP96], using a two-level approach which combines reinforcement and Bayesian learning. Unlike our work, which adopts a quite general definition of user profiles, their work uses a fixed number of categories to define user interests.

More recently, Balabanovic conducted a similar study where he evaluated a gradient descent approach for text recommendation systems [Bal98]. Unlike the way our algorithm represents profiles as a collection of vectors, Balabanovic used category preferences (i.e., favoring a topic over another) to represent user interests. He evaluated his technique against the classical Rocchio (which we have also used in this study) and obtained comparable results to Rocchio.

Foltz and Dumais used Latent Semantic Indexing (LSI) to derive a reduced dimensional vector space [FD92] and constructed a profile vector from each document judged as relevant by the user. The relevance of a document to the profile is then computed based on its cosine similarity to the closest profile vector. Notice that their approach of having a separate profile vector for each document of interest is a special case of MM , namely, when threshold δ is set to 1. Note that it is straightforward to generalize the approaches we describe here to use an LSI space rather than the regular (keyword) vector space.

The approach taken by SIFT, which uses the publish/subscribe model for wide-area information dissemination [YGM95], requires users to explicitly submit their profiles and update those profiles using relevance feedback. Our approach differs from that of SIFT in its use of a set of *inter-related* profile vectors whose contents and cardinality change based on user feedback, and its ability to construct profiles completely automatically[12].

7 Conclusions

Push-based data dissemination depends upon knowledge of user interests for making scheduling, bandwidth allocation, and routing decisions. Such information is maintained as user profiles. We have proposed a novel, incremental algorithm for constructing user profiles based on monitoring and user feedback. In our approach, a user-profile is represented as multiple vectors whose size and elements change adaptively based on user access behavior. We developed a set of workloads based on the Yahoo! WWW categories and used them to analyze the multi-modal approach and compared it to approaches that have been shown to have good effectiveness in other recent, related work.

Our experimental results showed that the "multi-modal" approach has several advantages: 1) It is capable of providing significantly higher accuracy than a single-modal approach; 2) It automatically and dynamically adjusts the number of vectors used to represent a profile based on feedback that is provided to it incrementally (i.e., one document at-a-time). This flexibility allows the algorithm to adapt profiles to cope with changes in

[12]The advantages of automatic profile construction have been shown experimentally using human subjects in [FD92]. These advantages are further validated by extensive experimentation with INQUERY [BCC94].

user interests over time; 3) Even in this incremental mode, the approach provides more accurate results than a batch version of the more traditional approach, which is in current use today; 4) The multi-modal representation of profiles combined with its incremental nature allows it to be tuned to trade off effectiveness and efficiency, which makes it suitable for use in large-scale information dissemination systems.

In terms of future work, there is much to be done. Push technology in general, and publish/subscribe systems in particular are becoming increasingly popular. As these systems scale to larger and more diverse user populations, efficient techniques for managing and updating large numbers of user profiles will become more important. Also, profiling techniques must be extended to cope with multi-media data types beyond textual ones. These and other issues provide a host of interesting research opportunities.

Acknowledgments

We would like to thank Douglas Oard for many helpful references and discussions. We also would like to thank Björn T. Jónsson for his useful comments.

References

[Aal92] I. J. Aalbersberg. Incremental relevance feedback. In *Proc. ACM SIGIR Conf.*, pages 11–22, 1992.

[All96] J. Allan. Incremental relevance feedback for information filtering. In *Proc. ACM SIGIR Conf.*, Zurich, Switzerland, August 1996.

[Bal98] M. Balabanovic. Exploring versus exploiting when learning user models for text recommendation. *User Modeling and User-Adapted Interaction*, 8(1), 1998.

[BCC94] J. Broglio, J. Callan, and W. B. Croft. Inquery system overview. In *Proc. of the TIPSTER Text Program*, San Francisco, 1994.

[Cal98] J. Callan. Learning while filtering documents. In *Proc. ACM SIGIR Conf.*, Melbourne, 1998.

[CAW98] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Intl. Conf. on Data Engineering*, Orlando, 1998.

[Cha99] My Excite Channel. `http://my.excite.com/`, 1999.

[FBY92] W. B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[FD92] P. W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35:51–60, 12 1992.

[Hul97] D. A. Hull. The TREC-6 filtering track: Description and analysis. In *The Sixth Text REtrieval Conf. (TREC-6)*, NIST, Gaithersburg, 1997.

[IBM97] IBM. Information on the fast track. *IBM Research Magazine*, 35, 1997.

[LMMP96] W. Lam, S. Mukhopadhyay, J. Mostafa, and M. Palakal. Detection of shifts in user interests for personalized information filtering. In *Proc. ACM SIGIR Conf.*, Zurich, August 1996.

[LP98] L. Liu and C. Pu. CQ: A personalized update monitoring toolkit. In *ACM SIGMOD Conf.*, Seattle, 1998.

[Mit97] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Poi99] PointCast. `http://www.pointcast.com/`, 1999.

[Roc71] Jr. J.J. Rocchio. Relevance feedback in information retrieval. In *The Smart System – Experiments in Automatic Document Processing*, pages 313–323. Prentice Hall, 1971.

[Sal89] G. Salton. *Automatic Text Processing*. Addison Wesley, 1989.

[SB90] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.

[SM93] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In *Proc. of the Ninth Conf. on AI for Applications*, pages 345–352. IEEE Computer Society, 1993.

[Ste92] C. Stevens. *Knowledge-Based Assistance for Accessing Large, Poorly Structured Information Spaces*. PhD thesis, University of Colorado, Department of Computer Science, Boulder, 1992.

[Tur94] H. Turtle. Natural language vs. boolean query evaluation: A comparison of retrieval performance. In *Proc. ACM SIGIR Conf.*, 1994.

[VH96] E. M. Voorhees and D. Harman. Overview of the fifth Text REtrieval Conference (TREC-4). In *The Fifth Text REtrieval Conf.. (TREC-5)*, NIST, Gaithersburg, 1996.

[Yah99] Yahoo. `http://www.yahoo.com/`, 1999.

[YGM95] T. W. Yan and H. Garcia-Molina. SIFT- a tool for wide-area information dissemination. In *Proceedings of the 1995 USENIX Technical Conference*, pages 177–186, 1995.

A Pseudocode for MM and Auxiliary Procedures

MM (P, v_d, f_d)
    if $(P == \{\}$ and $f_d > 0)$
      $strength_{v_d} = 1$; $temperature_{v_d} = 0$; $P = \{v_d\}$;
    else
      let $p_{active} = p_i$ s.t. cosine$(p_i, v_d) \geq$ cosine$(p_j, v_d)\ \forall p_i \in P,\ i \neq j$;
      $relevance_{d,P}$ = cosine(p_{active}, v_d);
      if $(relevance_{d,P} \geq \delta)$ **then**
        $p_{active} = (1 - \lambda) \times p_{active} + \lambda \times f_d \times v_d$;
        update_strength(p_{active}, f_d);
        if $(strength_{p_{active}} < 1)$ delete(p_{active}, P);
        else merge(p_{active}, P);
      else if $(f_d > 0)$
        $strength_{v_d} = 1$; $temperature_{v_d} = 0$; $P = P \cup \{v_d\}$;

update_strength(p, f_d)
  if $(f_d == 1)$
    if $(temperature_p == 0)$ $strength_p = strength_p + 1$;
    else
      if $(temperature_p < 0)$ $temperature_p = -1 \times temperature_p$;
      else
        $temperature_p = temperature_p - 1$;
      $strength_p = strength_p \times$ exp$(c \times temperature_p)$;
  else
    if $(temperature_p > 0)$ $temperature_p = -1 \times temperature_p$;
    else $temperature_p = temperature_p - 1$;
    $strength_p = strength_p \times$ exp$(c \times temperature_p)$;

merge(p_{active}, P)
    let $p_{closest} = p_i$ s.t. cosine$(p_i, p_{active}) \geq$ cosine$(p_j, p)\ \forall p_i \in P,\ i \neq j,\ and\ p_i \neq p$;
    if (cosine$(p, p_{closest}) \geq \delta)$
      $\mu = strength_{p_{closest}} / (strength_p + strength_{p_{closest}})$;
      $p = (1 - \mu) \times p + \mu \times p_{closest}$;
      $strength_p = strength_p + strength_{p_{closest}}$;
      $temperature_p = 0$;