Self-Adaptive User Profiles for Large-Scale Data Delivery*

Uğur Çetintemel
Dept. of Computer Science
University of Maryland
ugur@cs.umd.edu

Michael J. Franklin
Computer Science Division, EECS
University of California, Berkeley
franklin@cs.berkeley.edu

C. Lee Giles
NEC Research Institute and UMIACS
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 as multiple interest vectors, 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. Although there has been significant research on user profiles, our approach is unique in that it can be tuned to trade off profile complexity and quality. This feature, together with its incremental nature, makes our method 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 filtering 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 prearranged plan, so that they do not 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. For users, profiles are 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 queries in database or information retrieval systems; in fact, profiles are a form of continuously executing query.

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 large 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, the difficulty of formulating effective queries grows with the size of the data set [23].

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, it is still quite likely that a user's profile will not provide adequate precision or recall with existing approaches. 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 vec-

*The work of Çetintemel and Franklin has been partially supported by Rome Labs agreement number F30602-97-2-0241 under DARPA order number F078, by the NSF under grant IRI-9501353, and by research grants from Intel and NEC.

tors (e.g, SIFT [27], MyExcite [10]). Single vectors, as we will demonstrate, are insufficient for adequately modeling interests. Using multiple independent vectors, on the other hand, results 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 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,. [19, 27, 10]). If the profile does not keep up with the user's information needs, then precision and recall problems will quickly arise.

1.2. Basic approach and contributions

In this paper, we present a novel approach for representing, learning, 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 approach from previous work on user profile construction is that it uses a *multi-modal* representation of user profiles; i.e., a profile is represented as a collection of (*inter-related*) 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. Our algorithm is based on *relevance feedback* [20, 22]: 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. This technique frees the user from the burden of explicitly specifying the profile, and manually identifying and making profile changes, yielding higher quality profiles [11]. Our algorithm is incremental; it receives user feedback one at-a-time and modifies the user profile accordingly. This incremental nature also allows the algorithm to adapt profiles to cope with changes in user interests over time.

Our approach to user profile construction utilizes a single-pass, non-hierarchical clustering algorithm (see [12] for a nice overview of clustering algorithms for information retrieval). Clustering-based approaches have gained a lot of attention lately and have been used in a variety of applications such as data mining (e.g., [28, 13]) and searching the WWW (e.g., [17]). These approaches typically rely on expensive batch processing techniques that require all data to be stored and available, which is clearly impractical in our target environment and application.

The main contributions of this paper can be summarized as follows: First, we propose a new self-adaptive profiling approach that represents user interests as a dynamic set of inter-related profile vectors. We demonstrate that our approach can be parameterized to adjust its tendency to generate more or less complicated profiles. This flexibility enables our approach to trade off effectiveness and efficiency, which, in turn, enables it to be tuned based on the requirements/characteristics of the target environment. Although there has been significant research on user profiles (see Section 6), our approach is unique in that it enables such a quality vs. efficiency tradeoff. Second, we describe how to efficiently implement this approach by extending an incremental clustering algorithm with structures and operators specifically designed for filtering environments. Third, we evaluate our approach by using a detailed experimental framework based on WWW pages obtained from the Yahoo! topic hierarchy [25], 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 relevance feedback. In Section 3, we describe our approach for profile construction and maintenance. We discuss the experimental environment and workloads used to test the ability of the algorithm to recognize relevant web pages in Section 4, and present the results of experiments based on Yahoo! categories in Section 5. We discuss previous related work in Section 6, and present our conclusions in Section 7.

2. Background

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. The vector space model

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 to 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 exam-

ple, to achieve perfect recall, a system could simply return all the documents in the collection. Such an approach, however, would have terrible precision.

Our algorithm is based on the Vector Space Model (VSM) [21]. In the 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 the documents. 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 to those of relevant document vectors (in the vector space). 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 the 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, a non-negative value, 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* [12]. 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 [12]. The cosine similarity between two vectors, v_1 and v_2, is based on the inner (dot) product of v_1 and v_2, 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 vectors based on document contents [21]. 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 [20] 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 the system. It is also possible to combine such judgments into groups and incorporate each group using a single update. Allan [2] studied the effect of group size on the effectiveness of incremental relevance feedback (in a non-filtering environment). He showed that effectiveness increases with group size, and that the highest effectiveness was obtained using all the judgments at once (i.e., in batch mode).

3. The multi-modal approach

We developed Multi-Modal (MM), a new approach for automatically constructing and maintaining user profiles based on user feedback. MM represents user profiles as a set of vectors, the number, size, and elements of which change *adaptively*. Specifically, MM represents a user profile P as a set of *profile vectors* $p_1, p_2, ..., p_n$ where $p_i = ((t_{i_1}, w_{i_1}), (t_{i_2}, w_{i_2}), ..., (t_{i_m}, w_{i_m})),\ i = 1, 2, ..., n.$ The number of profile vectors, n, and the size of a profile vector, m, change over 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.

3.1. Overview

In order to simplify the presentation, we first describe the fundamentals of MM by describing a procedure for incre-

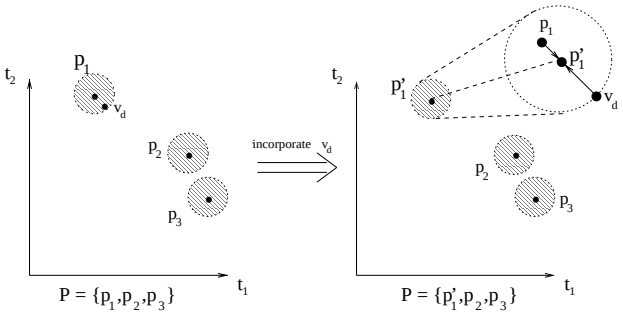

Figure 1: Incorporating a document vector into a profile vector

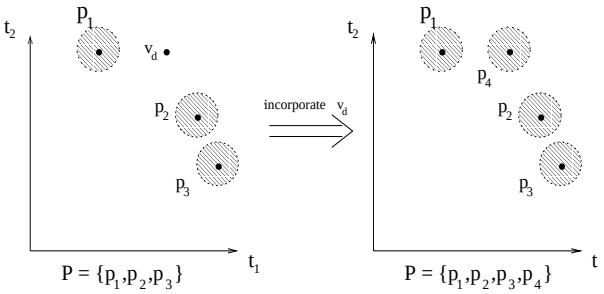

Figure 2: Creating a new profile vector from a document vector

mentally clustering document vectors. The basic idea is to maintain clusters of document vectors, where each cluster is stored as a single *representative* vector. The first document vector is assigned as the first cluster. When a new document is processed, its similarity with 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 incorporated into that cluster and the cluster representative is repositioned (the influence 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.

MM builds upon this basic incremental clustering algorithm with structures and operations specifically designed for multi-modal profile construction in a filtering environment. In MM, each relevant document vector can either create a new 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 only the relevant concepts. However, they can 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. This deletion operation allows for fast adaptation when there is a shift in user interests.

Notation	Description
d	document
v_d	vector representation of d
f_d	indication of user feedback for d ($\in \{-1, 1\}$)
P	user profile
p_i	profile vector
δ	threshold value ($\in [0.0, 1.0]$)
λ	adaptability value ($\in [0.0, 1.0]$)

Table 1: Basic Notation

In the rest of this section, we describe MM in detail (see [8] for a pseudocode of MM, and refer to Table 1 for notation).

3.2. Constructing the profile

When a new relevance judgment, f_d, for a document, d, is received, MM first identifies the profile vector that is most similar to v_d. This profile vector, if it exists, becomes the *active profile vector*, p_{act}, of the profile (with respect to v_d). The document vector will have all its further interactions with p_{act}. The relevance of the profile to the document is computed as the similarity between p_{act} and v_d (i.e., cosine(p_{act}, v_d)). If this similarity value is larger than a certain threshold, δ, then the effect of the feedback is reflected to the profile by incorporating v_d into p_{act}. The incorporation of v_d into p_{act} is accomplished by moving p_{act} linearly according to parameter λ. If f_d is positive, then p_{act} is moved towards v_d; otherwise, p_{act} is moved away from v_d. Formally,

$$p_{act} = (1 - \lambda) \times p_{act} + \lambda \times f_d \times v_d$$

An example of incorporating a new document vector into a profile vector is shown in Figure 1.[1] If a document vector falls within the shaded *similarity circle* for a profile vector, then it is deemed similar enough and is eligible for incorporation in that profile vector. In this example, the document vector v_d falls inside the circle of p_1, and p_1 becomes the active profile vector with respect to v_d. It is possible that the document vector falls inside multiple 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

[1] For illustration purposes, 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 can be selected randomly.

p_1 closer to v_d. Note that the size of the profile (i.e., the number of profile vectors) does not change as a result of an incorporation operation.

On the other hand, if the similarity between v_d and p_{act} is smaller than δ (i.e., v_d falls outside of all similarity circles), then v_d is inserted into P as a new profile vector, and the size of the profile increases by one. This case is illustrated in Figure 2.

If P is empty when the feedback is received, MM checks the sign of f_d. If f_d is negative, then the feedback is simply ignored (because the profile represents only the relevant concepts); otherwise, v_d is inserted into P as a new profile vector.

3.3. Adjusting the profile size

When a new document vector is incorporated into p_{act}, the position of p_{act} in the vector space changes. If two profile vectors come close (enough) to each other in the vector space, they represent similar (if not the same) concept(s). Therefore, a single vector may potentially be sufficient to represent the concept(s) that are represented by the two vectors. The merge operation allows MM to avoid multiple profile vectors redundantly representing similar concepts, thereby allowing a more compact representation.

The merge operation checks whether a merge is possible between p_{act} and the other profile vectors. The profile vector closest to p_{act}, p_c, is identified and if the similarity between p_{act} and p_c is larger than δ, p_{act} and p_c are merged. The merge of the two vectors is performed in a way similar to that of incorporating a document vector to p_{act}; p_{act} is pushed towards p_c linearly, and p_c is removed from the profile. Recall that the parameter λ defines how much p_{act} moves in the case of incorporating a document vector into p_{act}. In the merge case, however, the ratio of the strength (see Section 3.4) of p_c to the sum of the strengths of p_{act} and p_c is used instead of λ. Note that MM uses the same similarity threshold, which defines the area of the similarity circles, for all profile vectors (including the ones that are created as a result of a merge operation).

We need to consider only the vector pairs containing p_{act} for merging. This is because the only profile vector that can move to a new position (after any single feedback step) is p_{act} and the inter-vector distances for the other vectors do not change. 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. Other merge operations, if any, are accomplished lazily in future iterations.

3.4. Deleting the profile vectors

In addition to the merge operation described above, MM uses a deletion operation that also reduces the profile size. However, the primary reason for using the deletion operation is to effectively adapt to shifting interest patterns. Consider a profile vector that represents concepts that the user is not interested in anymore. In addition to the fact that the vector is stored redundantly, the existence of the vector is more likely to degrade the overall modeling effectiveness of the profile than to improve it. MM identifies such vectors and removes them from the profile.

MM utilizes negative feedbacks as indications of possible interest changes. If a document whose vector representation is similar to an existing profile vector has received negative feedback from the user, then MM records this event. If such an event occurs *sufficient* times for the same profile vector, then MM removes that vector from the profile.

In order to implement this operation, MM maintains simple statistics about the feedbacks for the documents that were incorporated into each profile vector. Each profile vector is given a *strength* value (1.0 in our case) when the vector is created. This value is updated each time a document vector is incorporated into the profile vector. The basic idea is as follows: If the feedback for the document vector is positive, then the strength of the profile vector is increased; otherwise, the strength is decreased. Strength modifications are performed using a simple exponential decay function where a positive constant, c, controls the decay rate. If the strength of a profile vector drops below a certain *deletion threshold* value, then the vector is removed from the profile. When two profile vectors are merged, the resulting vector obtains a strength value that is equal to the sum of the strengths of the merged vectors. The details of the deletion operation, along with illustrative examples, can be found in [8].

3.5. Discussion

There are two important parameters that control the way MM behaves and thus have to be set properly. One of them is the *threshold parameter*, $\delta \in [0.0, 1.0]$, which is mainly used to decide whether a new document vector should be incorporated into an existing profile vector or should create a new profile vector. This parameter can be used to control the number of profile vectors. If δ is set to 1.0, 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 δ is set to 0.0, all vectors will be incorporated into a single profile vector and the number of profile vectors will always be one. In this case, the overhead of profile management is extremely low, however the effectiveness of the profile is limited. In this paper, we are interested in intermediate values which will provide an optimal effectiveness/efficiency tradeoff for a given application.

The other important parameter is the *adaptability parameter*, $\lambda \in [0.0, 1.0]$. As mentioned before, λ controls the rate at which the active profile vector is moved when incorporat-

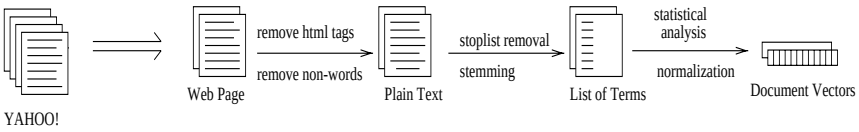

Figure 3: Converting web pages to document vectors

ing a new document vector into the existing profile. In other words, λ decides how fast the user profile adapts itself based on feedback. If λ is set to 1.0, active profile vectors will be replaced by document vectors at each feedback, resulting in maximum adaptation (i.e., memoryless mode). If λ is set to 0.0, on the other hand, active vectors will not change with feedback, and virtually no adaptation will take place.

4. Experimental environment

In this section, we describe our experimental environment and methodology for evaluating the MM algorithm. More details can be found in [8].

4.1. Document collection

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! [25]. More specifically, 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 selected top-level category. 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 $i, j \in [0, 1, ..., 9]$. We picked 900 pages uniformly distributed over all categories and converted them to their vector representations using the process illustrated in Figure 3. For each document vector we keep only the 100 highest-weighted terms. As in any learning study, we used a subset of the documents as the *training set*, which is used for learning purposes, and the remaining documents as the *test set*.

4.2. User simulation

Filtering systems developers typically rely upon the technique of *user simulation* in order to understand and quantify the effectiveness of their solutions (e.g., [2, 15]). 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. Specifically, a simulated user is assigned a synthetic profile (*SP*) consisting of a subset of the categories. The simulated user gives positive feedback to a document only if that document is classified under a category that appears in her synthetic profile; all the other documents are given negative feedback. More formally,

$$f_d = \begin{cases} +1 & \text{if } cat_d \in SP \\ -1 & \text{otherwise} \end{cases}$$

where cat_d is the category to which document d belongs. In our experiments, the synthetic profile is defined completely either by top-level categories or by second-level categories; i.e., either $SP \subset \{C_0, C_1, ..., C_9\}$, or $SP \subset \{C_{00}, C_{01}, ..., C_{99}\}$.

4.3. Methodology and performance metrics

We chose to base our evaluation methodology on the one used in the *routing track* of the TREC benchmark suite [24]. 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. The experiments are executed as follows. Each run starts by randomly selecting categories to form a synthetic user profile of desired complexity. The user profile is initialized to be empty.[3] We then present the documents in our training set 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 three 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

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

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 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 (at most) 100 term and weight pairs. 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 numbers of profile vectors typically indicate higher filtering times. The filtering cost, however, is not linearly proportional to the number of vectors 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 [2] (see Section 2.2), which uses a group of judged documents for updating the profile using relevance feedback. RI is a special case of RG where the group 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 $w(t)_{i+1}$ are the current and the updated (after feedback) weights of term 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:

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

$$w_{t,NR} = \frac{1}{|\mathrm{NR}|}\sum_{d \in NR} bel_{t,d}$$

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

$$tfbel_{t,d} = \frac{tf_{t,d}}{tf_{t,d} + 0.5 + 1.5 \cdot len_d/avglen}$$

$$idf_t = log(\frac{N + 0.5}{df_t})/log(N + 1)$$

where len_d is the length of document d, and $avglen$ is the average length of documents in the collection.[4] 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 using various settings of our experimental parameters. We observed that λ, if set in the range [0.1,0.3] 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. In all the experiments, the decay constant, c, is set to 0.5, and the deletion threshold is set to 1.0.

Experiments 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.

5.2. Top-level filtering effectiveness

We begin by investigating the filtering effectiveness of the three profile learning techniques using interest categories drawn from the top-level Yahoo! categories. Figure 4 shows the precision results for three different interest ranges, covering 10%, 20%, and 30% of the documents (i.e., one, two, and three 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) with a group size of 10 documents, 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.

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

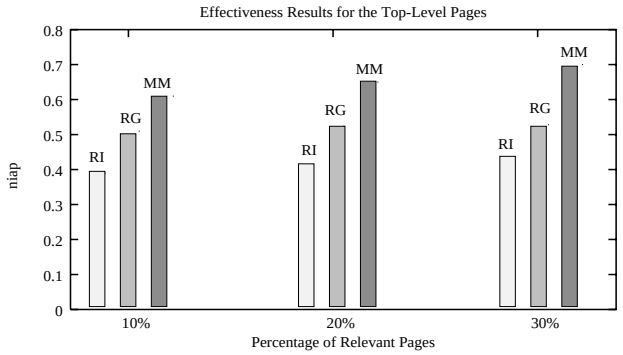

Figure 4: Precision values, top-level categories ($\delta = 0.15$, RG Group Size $= 10$)

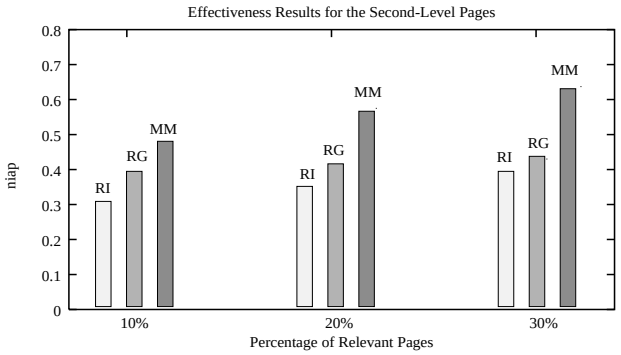

Figure 5: Precision values, second-level categories ($\delta = 0.15$, RG Group Size $= 10$)

Comparing the two Rocchio implementations, the results show that (as mentioned 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.

Comparing the results for the three interest ranges in Figure 4, it can be seen that the benefit of MM over the others grows as the number of categories in the profile is increased. 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 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.[5]

5.3. 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 5. 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. In terms of the percentage decrease in precision compared to corresponding top-level case, MM suffers the least of the three algorithms, while RG has the highest relative drop. Again, the flexibility of MM allows it to adapt to the more complex workload. In this case, MM maintains a small number of additional vectors (i.e., three or four) for each of the workload sizes compared to the corresponding workload size in the top-level case.

5.4. Threshold effects

The importance of maintaining multiple vectors is demonstrated in Figure 6, which shows the precision obtained by MM for the top-level categories using 10%, 20%, and 30% interest ranges, as the threshold parameter δ is increased. Recall that δ determines the tendency for MM to create new vectors. When $\delta = 0.0$, 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 $\delta = 1.0$ (not shown), 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 [11].

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 6, 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* [18], which can negatively impact effectiveness. Over-fitting is particularly a problem in noisy environments such as the WWW. A final argument for not retaining vectors for all documents is adaptability. As 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

[5] 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 test set, which raises the probability of a page being identified as relevant by the single-vector-per-profile approaches.

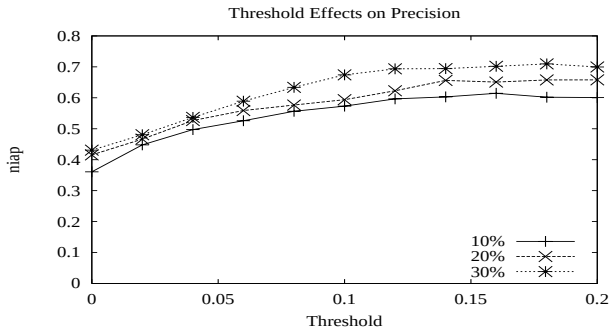

Figure 6: Precision vs. Threshold (top-level categories)

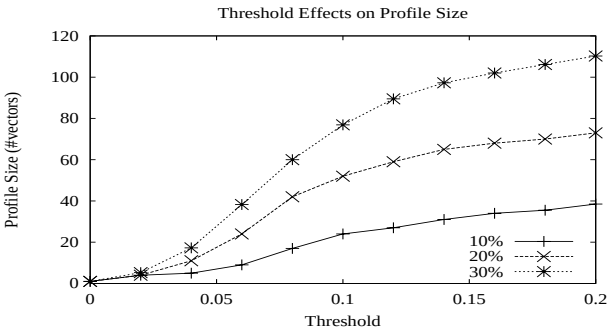

Figure 7: Profile Size vs. Threshold (top-level categories)

documents ever seen would adapt much more slowly.

Figure 7 shows the number of vectors maintained by MM for each of the three workloads as δ is increased. Notice 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 6 and 7 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.

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: partial and complete changes in the categories of interest, addition of a new category to a profile, and deletion of an existing category from a 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 deletion operation) 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 $\lambda = 0.2$. In order to make a fair comparison in terms of the storage used by RG and MM, RG was run with a group size of 100 documents.[6] 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 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.

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 8 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 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 re-

[6] 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.

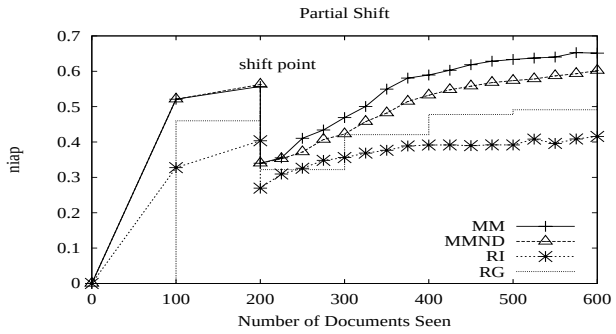

Figure 8: Partially changing interests ($\delta = 0.15$, RG Group Size $= 100$)

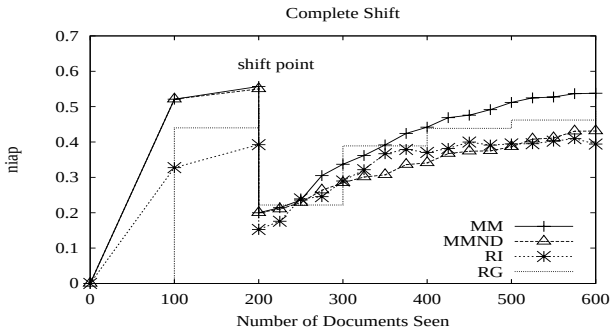

Figure 9: Completely changing interests ($\delta = 0.15$, RG Group Size $= 100$)

covers 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 9 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 9 to the partial change case shown in Figure 8, 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 influence effectiveness for a long time. RI adjusts reasonably quickly in this case, but its effectiveness remains below that of MM throughout the entire experiment.

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. We now briefly look at the case of more gradually changing interests.

Adding and deleting interests. Figure 10 shows the case where a new top-level category is added to an existing set of interests (originally containing a single category); for example, when 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 that the decay function has no negative influence during periods of static interests by validating that the decay function has no negative impact even for changing interests as long as existing interests are not dropped. RI and RG again demonstrate 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 collecting a group of relevance judgments before applying them to its vector.

Figure 11 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 recovery. The precision of RG recovers quickly, but remains below (or in one case, equal to) that of MM with decay.

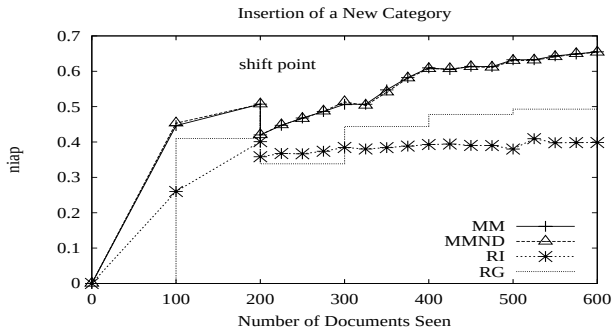

Figure 10: Adding new interests ($\delta = 0.15$, RG Group Size $= 100$)

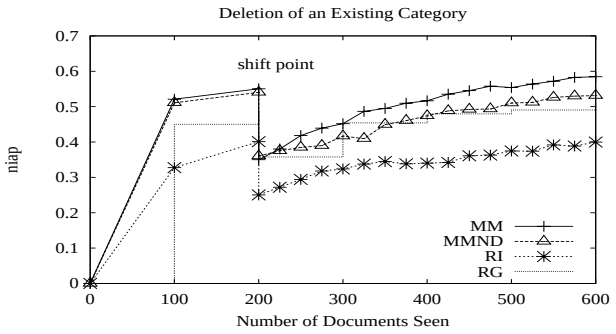

Figure 11: Deleting interests ($\delta = 0.15$, RG Group Size $= 100$)

The results presented in this section revealed that, 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

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 [9], the CQ (continuous queries) project at OGI [16], the Grand Central project at IBM Almaden [14], and the DBIS project at Maryland and Brown [4] all contain a user profile management component. To date, however, these projects have not emphasized learning-based acquisition and maintenance of profiles.

There has been significant research on text-based profile construction in information retrieval community (e.g., [5, 1, 7, 3]), especially in the framework of TREC [24]. The main emphasis of TREC, 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 do not exhibit the wide variability typical of web pages, making it difficult to extrapolate results from these benchmarks to performance on the WWW.

The work most closely related to ours is that of Allan, who studied the utility of relevance feedback for information filtering environments [2]. 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.

Foltz and Dumais used Latent Semantic Indexing (LSI) to derive a reduced dimensional vector space [11] 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 1.0. Notice 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 [27, 26], 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. The advantages of automatic profile construction were shown experimentally using human subjects in [11]. These advantages were further validated by extensive experimentation with INQUERY [6].

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 proposed a novel, incremental approach for constructing user profiles based on monitoring and user feedback. In our approach, a user profile is represented as a set of vectors whose size and elements change adaptively based on user feedback. We developed a set of workloads based on the Yahoo! WWW categories and used them to analyze our 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 uni-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, allowing the algorithm to adapt to changes in 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 widely used 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.

Information Dissemination 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 even more important. Also, profiling techniques must be extended to cope with multi-media and to take advantage of more structured data types as enabled by XML and other emerging standards. These issues provide a host of future 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

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

[2] J. Allan. Incremental relevance feedback for information filtering. In *Proc. of the ACM SIGIR Conf.*, pages 270–278, Zurich, 1996.

[3] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *Proc. of the ACM SIGIR Conf.*, pages 37–45, Melbourne, 1998.

[4] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. DBIS toolkit - adaptible middleware for large-scale data delivery. In *Proc. of the ACM SIGMOD Conf.*, pages 544–546, Philadelphia, 1999.

[5] N. J. Belkin and W. B. Croft. Information filtering and information retrieval: Two sides of the same coin. *Communications of the ACM*, 35(12):29–38, Dec. 1992.

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

[7] J. Callan. Document filtering with inference networks. In *Proc. of the ACM SIGIR Conf.*, pages 262–269, Zurich, 1996.

[8] U. Cetintemel, M. J. Franklin, and C. L. Giles. Flexible user profiles for large-scale data delivery. Technical Report CS-TR-4005 (UMIACS-TR-99-18), University of Maryland, 1999.

[9] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proc. of the IEEE Conf. on Data Engineering*, pages 4–13, Orlando, 1998.

[10] Excite. `http://my.excite.com/`, 1999.

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

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

[13] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. of the ACM SIGMOD Conf.*, pages 73–84, Seattle, 1998.

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

[15] W. Lam, S. Mukhopadhyay, J. Mostafa, and M. Palakal. Detection of shifts in user interests for personalized information filtering. In *Proc. of the ACM SIGIR Conf.*, pages 317–325, Zurich, 1996.

[16] L. Liu and C. Pu. CQ: A personalized update monitoring toolkit. In *Proc. of the ACM SIGMOD Conf.*, pages 547–549, Seattle, 1998.

[17] M. Mechkour, D. Harper, and G. Muresan. The webcluster project. using clustering for mediating access to the world wide web. In *Proc. of the ACM SIGIR Conf.*, pages 357–358, Melbourne, 1998.

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

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

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

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

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

[23] H. Turtle. Natural language vs. boolean query evaluation: A comparison of retrieval performance. In *Proc. of the ACM SIGIR Conf.*, pages 212–220, 1994.

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

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

[26] T. Yan and H. Garcia-Molina. Efficient dissemination of information on the Internet. *IEEE Data Engineering Bulletin*, 19(3):48–54, 1996.

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

[28] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. of the ACM SIGMOD Conf.*, pages 103–114, Montreal, 1996.