DISSERTATION

AGENTS FOR PERSONALIZED CLIENT-SIDE INFORMATION GATHERING FROM THE WEB

Submitted by

Gabriel L. Somlo

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2005

COLORADO STATE UNIVERSITY

June 15, 2005

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY GABRIEL L. SOMLO ENTITLED AGENTS FOR PERSONALIZED CLIENT-SIDE INFORMATION GATHERING FROM THE WEB BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Committee Member

Committee Member

Committee Member

Advisor

Department Head

ABSTRACT OF DISSERTATION

AGENTS FOR PERSONALIZED CLIENT-SIDE INFORMATION GATHERING FROM THE WEB

We present the design, implementation, and evaluation of a personalized Web information gathering agent, intended to address several shortcomings of today's centralized search engines. The potential privacy issues are addressed by a standalone client-side implementation, placing the agent under its users' administration. For personalization, we build on current text filtering and machine learning research, enabling the agent to adapt to its users' dynamic information needs. We also explore the tradeoff between performance and user friendliness, which arises due to the limited resources available to a client-side implementation.

As a key improvement over existing Web agents, we separate the treatment of *relevance prediction* from that of *document gathering*, and approach each problem using the most appropriate tools. For relevance prediction, we compare two main classes of text filtering algorithms: TF-IDF (for *term frequency, inverse document frequency*), which measures term-count distributions within and across documents, and Bayesian, which learns individual term contributions to the overall probability of relevance. Several versions of these algorithms are implemented to assess how performance is impacted by factors such as the amount of training, availability of negative feedback, and availability of topic-labeled training samples. For document gathering, we offload the brute-force

work to a large centralized search engine (e.g., Google), and focus on higher-level techniques, including generation of search queries from the agent's user profile, change detection between subsequent document versions, and tracking persistent user interests over time.

We approach the problem of evaluating Web information agents from two perspectives. We use benchmark datasets for speed, convenience, and statistical significance. We also conduct user studies to assess how the aggregate system behaves in a live situation, with limited training from real users.

Our main conclusions are that it is possible to build high-performance, lightweight text filters, especially when the operating environment facilitates users providing negative feedback; fast and efficient methods exist to detect whether a document changes in a relevant way or is made redundant by a previous document; and that relevant search engine queries can easily be extracted from a TF-IDF profile and used to supplement the incoming document stream for significantly improved recall.

Gabriel L. Somlo
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
Fall 2005

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The World Wide Web (WWW or Web) is perhaps one of the most visible applications of the Internet, to the point where many people equate it with the Internet itself. Created by Tim Berners-Lee in 1990 [11], the Web is a hypertext space consisting of a huge amount of documents that reference each other via unique addresses named Universal Resource Locators (URLs). Typically, Web documents are retrieved via their URL, and most browsers allow users to collect and organize URLs for future reference (i.e., "bookmarks").

In the early days of the Web, new documents were found by following URLs in documents whose address was already known; if found interesting or relevant, they could be added to the user's bookmark list. However, with the exponential growth of the Web during the 1990s, this method of finding information became overly limiting due to scaling issues. In spite of the fact that HTTP, the underlying protocol of the Web [46], still uses URLs to reference documents, users today prefer to reach Web resources based on content, a method made possible by search engines.

Search engines allow users to describe content of interest via queries containing a few relevant keywords. Using Information Retrieval (IR) techniques (and, more recently, other optimizations based on, e.g., Web topology), search engines respond with long lists of URLs ordered decreasingly by their expected relevance to the query.

Query (i.e., content) based access via a search engine quickly became *the* standard way to locate documents of interest on the Web, and remains so to this day. However, weaknesses in this method became apparent in the late 1990s. Search engines that were popular during that time (e.g., AltaVista[2], Lycos[90], Yahoo[144]) became technically unable to cover the entirety of the Web in a useful amount of time [79], and were also falling victim to commercialism, both from without (commercial pages "doctored" to score high when certain keywords were being searched) and from within (search pages cluttered with ads, pay-for-placement links returned without being clearly marked as such, etc.). This also brought to light the divergence between the interests of users on one hand, whose goal is to find information relevant to their interests in privacy, and the search engine operators on the other hand, who, for reasons of expediency and scalability gear their systems toward the average user modeled from a very large population (numbering in the millions or more). Also, users' private data may be viewed as a valuable commercial resource, and exploited in a way which does not necessarily benefit the users. Doing a "good enough" job for the largest possible number of users is advantageous for the search engine operator but happens at the expense of addressing users' personal information needs once they begin to diverge from the mainstream.

Modeling such large user populations also leads to problems with query ambiguity. Viewing queries as a content-based addressing mechanism for the Web, collisions will inevitably occur when many concepts are identified by short lists of keywords. The result is either a noisy list of returned documents (documents containing the concept of interest are intermingled with documents containing other concepts matching the same query), or the less popular concept being drowned out by the mainstream one.

Another problem is that search engines operate in a stateless fashion: users submit a query, and immediately receive the list of results currently available from the search engine. There is no obvious temporal component to this interaction, i.e., long-term user

interests can only be satisfied by repeatedly visiting the search engine to see if/what new information has become available. Search engine operators and online newspapers (such as the New York Times) have recently began to address this issue, by providing services that alert users when the top hits matching a query change [56], but the effort expended on maintaining a user profile for true personalization is typically limited.

1.1 Techniques Addressing Parts of the Problem

One of the proposed solutions, mainly intended to address search engines' inability to completely index the Web, is Meta-search [123, 44, 77]. The idea is to offer users a clean, simple interface where they may type in their queries, and then, to turn around and submit these queries to a set of "real" search engines. Results gathered from all search engines are then integrated and presented to the user in a uniform way. Meta-search was a great way to deal with the shortcomings of early search engines, but became less widely used with the advent of the latest generation of search engines, such as Google. Today, meta-search engines are successfully used by specialized search services, such as comparison shopping sites, although general-purpose sites such as Dogpile[40] are still viable (as shown by Search Engine Watch[143]).

Researchers from the Machine Learning and Agents communities proposed solutions for tailoring Web search to the needs of individual users. Web information agents (e.g., Letizia[84], WebWatcher[5], WebMate[29], and Copernic[36]) are designed to learn the interests of a user (i.e., build a *user profile*), and, once a sufficiently good profile is learned, assist users with their information gathering tasks. These systems address the problem of centralized services' excessive user abstraction, lack of personalization, and potential for privacy abuse by being implemented as stand-alone systems placed under the users' direct control.

Existing Web agents typically lack a distinction between the task of *gathering* Web

documents, and that of *predicting their relevance*. The first task (locating documents) is addressed by *focused crawlers* [25, 67, 100], which support document collection on behalf of "mini" search engines dedicated to a specific topic of interest (a form of personalization). The second task (discriminating between relevant and non-relevant documents) has been studied extensively by the IR community, in the form of text filtering [20, 71, 117, 148]. Although the main focus of text filtering is to deal with a different type of document source (dynamic incoming stream, as opposed to fixed, random-access collection or corpus), there are important lessons in how filtering systems learn to make binary predictions on document relevance with respect to a topic of interest.

Continuous query systems [28, 87, 138], originated from the database research community, are a good example of systems that address the temporal (stateful) component of users' need for information. Users register their information needs with the continuous query system, which then notifies them asynchronously once relevant information becomes available.

1.2 Our Solution and Lessons Learned

The goal of our research is to facilitate the construction of Web agents that help individual users find documents relevant to their own personal interests, while preserving user privacy, and without sacrificing ease of use. Personalization is achieved through the use of machine learning techniques borrowed from both agents and text filtering research. User privacy is protected mainly by a design that allows the entire agent to be placed under the end users' control, with no requirement for pooled or centralized resources. User friendliness is accomplished by reducing the amount of work required from users to train their Web agent, which sometimes happens at the expense of the agent's performance. A typical example of the tradeoff involved is whether or not to require users to provide negative feedback (i.e., examples of *non*-relevant documents) when train-

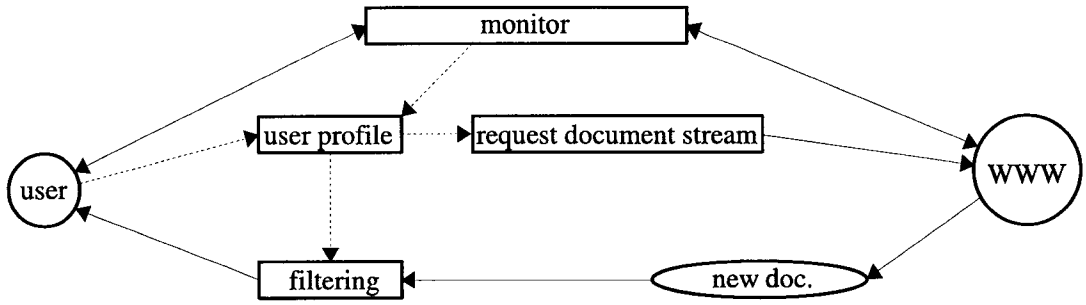


Figure 1.1: SurfAgent: Typical architecture of a Web information gathering agent

ing the agent's machine learning component. Negative feedback often leads to better performing algorithms, but may be inconvenient for users to provide. One of our key contributions is to assess the tradeoff between performance and user convenience of the various building blocks of a personalized Web information agent.

Another key contribution of this dissertation is that our Web agent design explicitly approaches document relevance prediction *separately* from document gathering, which is an improvement over current practice. We accomplish relevance assessment by adapting, implementing, and fine-tuning algorithms for text filtering. SurfAgent[130, 129, 132] is our first practical implementation of a Web agent, and is mainly used as a testbed for our filtering experiments. With an architecture outlined in Figure 1.1, SurfAgent operates as an HTTP proxy with access to a user's entire Web browsing activity. Training occurs only at the user's initiative, and consists of relevant/non-relevant judgments on downloaded Web documents. This feedback leads to the construction of a user profile, which may subsequently be used to request and filter more Web documents. Using SurfAgent as a testbed, we implemented and experimented with various algorithms for learning user profiles and filtering new documents. Our experiments used simulated user interaction based on pre-judged benchmark document collections to study the tradeoff between optimizing filtering performance, reducing resource use (e.g., storage), and limiting the required amount of user-provided training for optimized user friendliness. Our main results include an adaptive mechanism for learn-

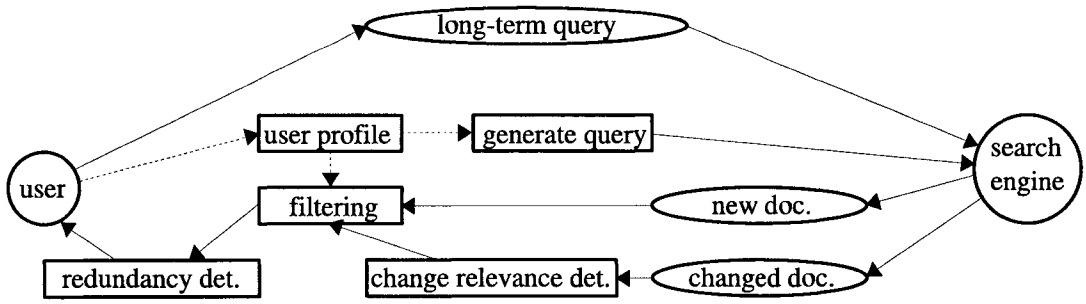


Figure 1.2: QueryTracker: Document gathering and long-term monitoring of user interests

ing TF-IDF dissemination thresholds, and a thorough comparison of TF-IDF and Bayes filtering algorithms with respect to their applicability to personalized Web agents.

Document gathering services are essential in ultimately deciding the usefulness of a Web agent. With our second testbed agent, QueryTracker[131, 133], we implemented a document gathering technique where a query is constructed from the user profile and submitted to a search engine. By repeating this process periodically, we offer the additional feature of tracking user interests over time. QueryTracker (see Figure 1.2) benefits from the filtering techniques developed and perfected using SurfAgent. Implemented as a Web service to support experimentation, QueryTracker is intended to be deployed as a stand-alone system where users register queries describing their interests. These queries are repeatedly submitted to a search engine, and returned documents are filtered for relevance. After learning a user profile for each registered query, the agent automatically uses it to construct additional queries intended to be a better reflection of the users' *real* information needs (personalization). Change/redundancy detection is employed to eliminate duplicate (aliased or mirrored) documents, as well as to detect when an old document was modified with relevant information. By monitoring queries over time, and by using change detection on previously seen documents, QueryTracker addresses the lack of statefulness (temporal component) of current search engines. Our results based on QueryTracker include comparative evaluations of change relevance and redundancy

detection algorithms, query generation from user profiles, and a study on the interaction between filtering and query generation in a real-world situation.

We provide a detailed evaluation for each of our agents' building blocks, both by simulating user interaction with benchmark data, and by collecting real-world data from user studies. Our filtering techniques are evaluated by simulating SurfAgent's interaction with hypothetical users simulated using standard IR benchmarks. Our document gathering technique (and its interaction with the filtering subsystem) is evaluated by collecting live usage data from QueryTracker.

Our research shows that it is possible to build high-performance, lightweight text filters that also reduce inconvenience to their users. Topics of interest may be learned automatically, thus further reducing the users' effort in training the system. However, negative feedback may not be as big an inconvenience as we originally believed, at least not with the proper usage scenario (e.g., QueryTracker). Existing algorithms for detecting semantic redundancy are more than adequate at eliminating mirrored and aliased Web content; however, we also demonstrate a much cheaper method which takes advantage of the underlying filtering algorithm, with promising results. Generating search engine queries from user profiles learned by our agent improves performance over simply monitoring original user queries.

1.3 Chapter Organization

This dissertation begins by reviewing fundamental concepts of IR and text filtering (Chapter 2), and their application to the Web (search engines and Web agents, respectively, in Chapter 3). The experiments described in this dissertation were performed over the course of several years. They are presented in a logical sequence (filtering, change/redundancy detection, and, finally, query generation), but bits and pieces of each were conducted at different points in time, which may explain some inconsistencies in

data sets used, experiment methodology, etc. Our experiments on text filtering from the perspective of a Web agent, including learning user profiles, multiple topics, and the tradeoff between performance and convenience is presented in Chapter 4. Techniques for assessing the relevance of changes between subsequent document versions and for detecting redundancy are explored in Chapter 5. One of the possible ways an agent may use to gather documents is by automatically generating and submitting a search engine query based on the learned user profile; considerations on how to extract such queries from the profile are presented in Chapter 6. A user study designed to evaluate the interaction of filtering and query generation in a live deployment is presented in Chapter 7. Finally, Chapter 8 offers a summary of our key contributions, ideas for future work, and conclusions.

Chapter 2

Information Retrieval Techniques

Our interest in IR stems from its being a crucial component of the typical search engine model. Just like libraries, search engines collect a huge number of documents by periodically crawling the Web. Between subsequent crawls, these documents are treated as a static, random-access collection, on which traditional IR is performed in response to user-supplied queries. Today, Google's [55] success is owed to augmenting the traditional search engine model with topology-derived knowledge and massive parallelization, but the basic model still applies.

Relevance feedback [118] was an early attempt at providing personalization to users of large, centralized IR systems. Relevance feedback consists of the iterative refinement of queries, by having users point out relevant documents returned during previous iterations.

Text filtering, a relatively recent offshoot of IR, takes further steps toward the personalization of document retrieval in two major ways: first, it builds on the concept of relevance feedback by maintaining complex, long-running queries (sometimes referred to as *user profiles*). Secondly, filtering does not require random access to collected documents, and instead limits itself to sequentially accessing documents which arrive one at a time from an incoming stream. This technique is interesting because it parallels the behavior of client-side Web information agents which, due to limited resources, cannot

afford to collect large numbers of documents and must examine them one at a time as they are encountered.

Finally, we review evaluation techniques for IR and filtering systems. Recall and precision are the standard metrics by which IR systems are traditionally judged. We examine various other metrics (some which are derivatives of recall and precision, and some alternatives which are not), as well as existing benchmarks used for a more or less uniform and unbiased comparison of performance between unrelated IR systems.

2.1 Classic Models of Information Retrieval

In general, text retrieval methods operate on large, fixed collections of documents (*corpora*), from which they attempt to select useful items that best match (are most *relevant* to) a user's need for information, as described by a set of keywords (*query*). The typical application would be retrieval of documents from a library or database. The success of an information retrieval task is traditionally assessed using two metrics: *precision*, measuring the relevance of the retrieved items, and *recall*, measuring the completeness of the list of retrieved items. Ideally, all retrieved documents should be highly relevant with respect to the query (maximizing precision), and no relevant document should be omitted from the list of results (maximizing recall). In practice, these two goals seem to have an adverse effect on each other: high precision usually results in failure to retrieve some documents that are still relevant, and high recall results in the inclusion of documents of little relevance.

According to Gudivada et al.[59], methods for text retrieval and indexing can be based on the processing of single or multiple terms. Single-term methods are generally based on counting the frequency of occurrence of individual words in a text document. Multiple-term methods can either be extensions of single-term methods to groups of co-occurring words (phrases), or based on Natural Language Processing (NLP) techniques.

Since NLP methods applied to IR problems have been faced with mixed success [128], we will restrict our further analysis to term-frequency (*bag-of-words*, or TF) methods, such as the Vector Model, Latent Semantic Indexing (LSI) – based on Singular Value Decomposition (SVD), and the Probabilistic Model.

2.1.1 Vector Model: TF-IDF

The vector model proposed by Salton et al.[121, 119] is currently the most widely used text retrieval method. Each text document is represented as a vector in a multi-dimensional space, with each dimension corresponding to a distinct word (*term*). Each document vector is built such that its projection along each dimension represents the importance of the corresponding term in describing the content of the document.

Let us consider a corpus consisting of D documents. To each term t, we attach a number DF_t, which is equal to the number of documents in the corpus that contain it (referred to as the *document frequency* of term t). Each document d in the corpus will be described by a TF-IDF vector containing one weight for each term t in the corpus, defined as follows:

$$w_{d,t} = TF_{d,t} \cdot \log \frac{D}{DF_t}$$

where $TF_{d,t}$ is the number of occurrences of term t in document d, referred to as *Term Frequency*, and the logarithm is a factor referred to as the *Inverse Document Frequency* of term t. The intent behind the above formula is to emphasize terms that have a high term frequency within a document as good indicators of the document's content, while also de-emphasizing terms that are very common across all documents and therefore poor indicators of a particular document's content (hence the acronym TF-IDF for "Term Frequency – Inverse Document Frequency").

Note that if t does not occur in a document d at all, its term frequency is $TF_{d,t} = 0$, which causes its corresponding weight in the document vector to also be zero ($w_{d,t} = 0$).

In practice, each document contains a small subset of all the terms that occur across the corpus, and therefore document vectors are usually sparse.

A good predictor of the similarity between two documents is the amount of overlapping terms they contain. It is reasonable to expect that if two documents are on the same topic, they will share many terms used to describe that topic. Because the individual components of a TF-IDF vector are also affected by the document size (longer documents will have higher term frequencies than shorter ones), simply computing the Euclidean distance between two document vectors is not sufficient in order to assess their similarity. Instead, the cosine of the angle between the directions of the two vectors is used:

$$similarity(W_{d_1}, W_{d_2}) = \cos(\angle(W_{d_1}, W_{d_2})) = \frac{W_{d_1} \cdot W_{d_2}}{|W_{d_1}| \cdot |W_{d_2}|} = \frac{\sum_t w_{d_1,t} \cdot w_{d_2,t}}{\sqrt{\sum_t w_{d_1,t}^2 \cdot \sum_t w_{d_2,t}^2}}$$

where W_{d_1} and W_{d_2} are the document vectors corresponding to documents d_1 and d_2 respectively, and t is an iterator over all terms that appear across the corpus from which the documents originated. The intuition behind cosine similarity is that collinear document vectors represent similar term distributions in the compared documents, whereas orthogonal document vectors indicate that few common terms are shared. The heuristic assumption is that term distribution within a document can be used as a substitute for the document's topic.

A query to a vector-based IR system is treated the same way as a document: its corresponding TF-IDF vector is built and compared to all document vectors in the corpus using the cosine similarity metric. Matching documents are retrieved in decreasing order of their similarity to the query.

2.1.2 Latent Semantic Indexing

Introduced by Furnas et al. [50], Latent Semantic Indexing (LSI) organizes a corpus containing d documents and a total of t terms into a $t \times d$ matrix of term frequencies,

X. Each element x_{ij} of this matrix represents the number of times the term i appears in document j. A Singular Value Decomposition (SVD) operation is performed on the matrix X,

$$\underset{t\times d}{X} = \underset{t\times m}{T}\ \underset{m\times m}{S}\ \underset{m\times d}{D^{\mathrm{t}}}$$

resulting in matrices T and D, with orthonormal columns, containing the *left* and *right singular vectors*, and diagonal matrix S, containing the *singular values*. In order to retain only the significant structure of the data and to filter out noise and the high dimensionality, only the k (where $k < m$) largest singular values from the total of m present in matrix S (where $m = \min(t,d)$) are kept along with their corresponding columns in T and D:

$$\underset{t\times d}{X} \approx \underset{t\times k}{T'}\ \underset{k\times k}{S'}\ \underset{k\times d}{D'^{\mathrm{t}}}$$

The similarity between two documents (columns of matrix X) is given by their inner product. Theoretically, to obtain similarities for all possible pairs of documents, we could perform the matrix multiplication $X^{\mathrm{t}}X$, resulting in a $d \times d$ matrix in which any cell (i,j) would be the similarity between documents i and j. Using the component matrices obtained from the SVD, this operation would be equivalent to:

$$X^{\mathrm{t}}X = (TSD^{\mathrm{t}})^{\mathrm{t}}TSD^{\mathrm{t}} = DST^{\mathrm{t}}TSD^{\mathrm{t}} = DSSD^{\mathrm{t}} = (DS)(DS)^{\mathrm{t}}$$

In other words, the similarity between documents i and j can be computed by taking the inner product of rows i and j from the product matrix DS. Analogously, similarities between terms (in the context of the corpus) can be computed by taking the inner product of rows from the product matrix TS. These operations are made very efficient if the dimensionally reduced matrices (T', S', and D', where we assume $k \ll d$ and $k \ll t$) are used instead of the originals.

Queries posted to an LSI-based IR system are treated as pseudo-documents by being appended as new columns to matrix X. In order to use the efficient comparison features offered by LSI, a corresponding row in matrix D' needs to be computed. We know that

$$\underset{t\times 1}{X_q} = \underset{t\times k}{T'} \quad \underset{k\times k}{S'} \quad \underset{k\times 1}{D_q^t}$$

and we compute the pseudo-row of matrix D' that corresponds to our query (D_q):

$$D_q = X_q^t T' S'^{-1}$$

The similarity between an arbitrary document i from the corpus and our query is then computed by taking the inner product of row i from matrix $D'S'$ and the row vector given by $D_q S' = X_q^t T'$. Note that there is no need to compute the inverse of matrix S'.

A comparison between LSI and TF-IDF [89], performed as part of an experiment intended to locate experts within an organization based on their publications, shows the two methods have roughly the same performance, with LSI better suited to detect terms that are synonyms. Lately, LSI has been used with a fair amount of success in applications that almost (but not quite) require NLP, such as automated evaluations of essays and summaries [76]. While LSI's design makes it an excellent solution for traditional IR, it also prevents it from being easily maintained in an incremental fashion. This makes it a poor choice for *filtering* systems, where, as we will show later in this chapter, ease of incremental maintenance of learned information is crucial.

2.1.3 Probabilistic Model

This approach is based on the observation that some terms are better than others in their ability to discriminate between the documents in a collection. Salton [122] initially attempted to improve performance of the TF-IDF vector-model system by limiting the vocabulary to terms with document frequency (DF) between $1/10$ and $1/100$ of the collection size. The intuition is that terms with higher DF were too common, and terms with lower DF were too specialized to be useful discriminators.

Eventually, probability theory was used to pick the best terms with the highest accuracy. Two main approaches emerged: the 2-Poisson method, and a method based on Bayes' formula.

2.1.3.1 The 2-Poisson Model

Probabilistic indexing [14, 64, 115, 92, 116] is based on the observation that for any term in a corpus, the number of documents in which the term has a given term frequency follows a multiple Poisson distribution (a weighted sum of Poisson distributions, one for each subclass of documents that are similarly described by the term in question):

$$f(x) = \sum_{i=1}^{N} p_i \frac{e^{-\lambda_i} \lambda_i^x}{x!}$$

where λ_i is the average number of documents containing the term for the ith Poisson distribution in the sum; x is the term frequency, $f(x)$ is the percentage of documents in the corpus with term frequency x, and p_i are the different weights attached to each subclass of documents, and $\sum_{i=1}^{N} p_i = 1$.

Most frequently, as pointed out in [64], the distribution contains only two Poisson components: one for the documents that are relevant with respect to the query term, and the other for the remaining documents in the corpus:

$$f(x) = p \frac{e^{-\lambda_1} \lambda_1^x}{x!} + (1-p) \frac{e^{-\lambda_2} \lambda_2^x}{x!}$$

The two degenerate cases where either λ_1 or λ_2 are equal to 0 (leading to a single Poisson distribution) occur for terms that only have one corresponding document "subclass" that spans the entire corpus. These terms are either poor indicators of document content, or good indicators of content for *all* documents in the corpus – of little use for indexing purposes in either situation. Examples for the first case include stopwords, such as "the", which occur in all documents but have no bearing on any document topic. For the second case, an example could be the term "computer" in a corpus in which all documents are about computers and computer science. While such terms *do* have a strong relationship to the document topic, they contribute no information to help us discriminate between *sub*-topics of the corpus, which is our ultimate goal.

In order to build an index, a histogram of document count by term frequency is built for each word. Then, the parameters of the underlying 2-Poisson distribution (λ_1, λ_2, and p) are inferred from the histogram using either the method of moments [111], or the more modern method of maximum likelihood (of which a good example is the Levenberg-Marquardt method implemented in [107], pp.683-688). Each term is assigned a weight based on the Poisson parameters, and other factors such as document length [116]. If the parameters indicate only a simple Poisson distribution, the term is not used for indexing (i.e., it is assigned a weight equal to 0). The relevance of a document with respect to a query is computed by summing the weights of the query terms contained in the document.

2.1.3.2 The Bayesian Model

A simpler approach to probabilistic indexing uses Bayes' Rule as proposed by Robertson et al. [114, 113] and Croft et al. [37]. However, these systems are not fully automated as they depend on manually supplied training examples of relevant and non-relevant documents for each topic. Each term t is assigned a weight proportional to its contribution to the relevance of the documents in which it appears:

$$w(t) = \log \frac{P(t|\text{relevant})}{P(t|\text{nonrel})}$$

We assume the training set consists of D documents, R of which are relevant and D_t of which contain the term t. R_t is the number of documents in the training set that are both relevant and contain term t. We have

$$P(t|\text{relevant}) = R_t/R$$

$$P(t|\text{nonrel}) = (D_t - R_t)/(D - R)$$

which results in

$$w(t) = \log \frac{R_t/(R - R_t)}{(D_t - R_t)/((D - R) - (D_t - R_t))}$$

Again, the relevance of a document is computed by summing the weights of all terms that also occur in the query.

In contrast with the vector space model, where documents were ranked based on similarity to a template of what is relevant, the Bayesian model ranks documents based on their probability of relevance. However, this probability is computed based on individual terms' probabilities of appearing in a relevant or non-relevant training example, which ends up being a roundabout way of computing similarity between a new document and a model of the positive or negative document classes.

A better formal explanation of this process is provided in Mitchell's description of the Naive Bayes Classifier (NBC) [95](pp.177–184). Given a document d, and a number of possible classes c_i into which this document could be classified, we wish to assign the document to the most likely class $c_{\max}$, i.e., the one for which $P(c_{\max}|d)$ is maximized. As applied to IR, there are typically two classes: the *relevant* class, containing documents that are relevant to a chosen topic, and the *non-relevant* class, containing all other (non-relevant) training samples. In other words, we seek

$$c_{\max} = \arg\max_{c_i} P(c_i|d)$$

Given Bayes' theorem,

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}$$

we then have

$$c_{\max} = \arg\max_{c_i} \frac{P(d|c_i) \cdot P(c_i)}{P(d)}$$

and since $P(d)$ is constant with respect to the class,

$$c_{\max} = \arg\max_{c_i} [P(d|c_i) \cdot P(c_i)]$$

The NBC is called *naive* because it assumes conditional independence of the document terms given the target class c_i. In other words, the document is treated as a bag-of-words,

just like in the vector space model. This approach also ignores phrases or separated co-occurrences of terms. If a document consists of a string of n terms

$$d = t_1, t_2, ..., t_n$$

then the NBC assumption is that for any class c_i,

$$P(d|c_i) = P(t_1, t_2, ..., t_n|c_i) = \prod_j P(t_j|c_i)$$

Returning to the search for our most probable class, we have

$$c_{\max} = \arg\max_{c_i} P(c_i) \cdot \prod_j P(t_j|c_i)$$

In our particular situation, we have $c_1 = relevant$ and $c_2 = nonrel$. Given the D training documents as mentioned earlier in this section, we have

$$P(relevant) = R/D$$

$$P(nonrel) = (D-R)/D$$

For each term t occurring in the training set, we compute $P(t|relevant)$ and $P(t|nonrel)$ as described earlier.

If we rank each candidate document d_{new} by how much more likely it is to be relevant than non-relevant

$$\text{rank}(d_{\text{new}}) = \log P(d_{\text{new}}|relevant) - \log P(d_{\text{new}}|nonrel)$$

we observe that this is equivalent to ranking by

$$\text{rank}(d_{\text{new}}) = \sum_{t \in d_{\text{new}}} w(t)$$

with $w(t)$ as proposed by [114]. Note that terms that didn't occur in the training set are ignored when they appear in new documents.

Such a system requires a training set that includes both negative and positive examples, which is why this model is not so widely used in traditional IR, but is rather popular for text filtering (discussed later in this chapter).

2.2 Relevance Feedback

Rocchio [118] observed that users' queries to an information retrieval system range from the very well informed, used to verify the presence of a document in the collection, to exploratory, where users need information on an unfamiliar topic, and hence do not know what they are looking for. The latter case may also include users lacking the skills to express their information need by submitting a good query. Relevance feedback is introduced as a solution for the latter type of user. The process is inspired from feedback loops in control theory and consists of iteratively refining and expanding queries based on the user-provided relevance of the documents retrieved during the previous iteration.

First, Rocchio proved that for any subset **R** of a document collection, there exists an optimal query Q_{opt} that would separate **R** from its complementary subset of the collection, **N**:

$$Q_{opt} = \arg\max_{Q} f(Q) = \frac{\sum_{R \in \mathbf{R}} sim(Q,R)}{|\mathbf{R}|} - \frac{\sum_{N \in \mathbf{N}} sim(Q,N)}{|\mathbf{N}|}$$

Given that $sim(A,B) = \cos(\angle(A,B)) = A \cdot B$ (assuming all TF-IDF vectors have been normalized), we can rewrite $f(Q)$ as

$$f(Q) = Q \cdot \left[\frac{\sum_{R \in \mathbf{R}} R}{|\mathbf{R}|} - \frac{\sum_{N \in \mathbf{N}} N}{|\mathbf{N}|} \right]$$

Since $f(Q)$ is a dot-product of two vectors, it is maximized when the two vectors (Q and the expression in square braces) are collinear. Without loss of generality, it can be said that

$$Q_{opt} = \frac{\sum_{R \in \mathbf{R}} R}{|\mathbf{R}|} - \frac{\sum_{R \in \mathbf{N}} R}{|\mathbf{N}|}$$

In other words, the optimal query is the vector difference between the centroid of the relevant subset **R** and of its complementary subset (of non-relevant documents) **N**.

Obviously, there is no realistic expectation that users would submit optimal queries, since having the ability to do so implies knowledge of the relevant set **R**, which precludes the need to search for information in the first place. Therefore, a method is

suggested to allow users to start with an arbitrary query Q_0 and iteratively refine it using the following formula:

$$Q_{i+1} = f(Q_i, \mathbf{RF}_i, \mathbf{NF}_i) = Q_i + \frac{\sum_{R \in \mathbf{RF}_i} R}{|\mathbf{RF}_i|} - \frac{\sum_{N \in \mathbf{NF}_i} N}{|\mathbf{NF}_i|}$$

where $\mathbf{RF}_i$ and $\mathbf{NF}_i$ are the sets of relevant and non-relevant documents identified by the user during iteration i of the process. As i increases, Q_i converges toward Q_{opt}. In Rocchio's original experiment, significant improvements in retrieval performance were observed even after one single iteration of relevance feedback.

Salton [120] compares relevance feedback performance on vector vs. probabilistic IR systems and concludes that the vector-model outperforms probabilistic models (such as used by [114]) by a significant margin. Harman [62] offers another comparison of the vector and probabilistic models; the article underlines the importance of using relevance feedback to also expand queries by adding new terms, in addition to simply re-weighting existing ones. When multiple iterations of relevance feedback are performed, feedback should continue within an iteration until all relevant documents in that iteration have been seen.

Buckley et al.[18] suggest a method for automatic relevance feedback, by which performance is improved by simply using all retrieved documents from the first iteration to modify the query submitted in the second iteration. The downside to this is that, when casual users type short and sub-optimal queries, the many non-relevant documents retrieved in the first iteration tend to cause *topic drift* by being included in the automated relevance feedback process. Mitra [96] proposed a solution based on manually or automatically constructed Boolean filters that exclude from automated relevance feedback documents that do not contain *enough* query terms and thus are likely to cause topic drift. Harman [61] introduced *interactive* relevance feedback, where potential terms that would be used in query expansion were first shown to the users. Users picked which terms they would like to use. This method significantly increased performance,

but often at significant cost of slow response time and extra effort requirements from users. A later study [91] measured performance improvements given the experience level of the users and found that inexperienced users not only didn't benefit, but actually experienced degraded retrieval performance.

More recent research on relevance feedback is usually performed within the context of text filtering (discussed in the following section). Relevance feedback is one of the several main mechanisms used by filtering systems to learn models of the relevant documents they are looking for.

2.3 Text Filtering

Text filtering has evolved from traditional IR, as an adaptation to changes in the environment in which document retrieval systems are expected to operate. Filtering moves away from the model of a centralized, static document repository, on which one-time, stateless queries induce a ranking in decreasing order of relevance. Instead, filtering systems operate on a dynamic document stream (such as Usenet news, email, or Web documents encountered during browsing or crawling). Information needs persist over a longer period of time, and binary decisions must be made as soon as each document is encountered.

Differences from traditional IR make text filtering suitable for use on the client-side instead of as a centralized server. For the same reasons, filtering has great potential for privacy and personalization, which makes it an excellent basis for building Web information agents (discussed in the next chapter).

According to [9, 80, 1, 10], the main distinguishing factors between traditional IR and text filtering are the source of documents, the level of involvement of the user into the process, and the time-frame of document retrieval.

Information source. Traditional IR systems are designed to operate on a theo-

retically *static* document collection or database. In practice, documents are added or removed from the collection all the time, but at a frequency that is negligible when compared to how often the system accepts user queries. Filtering systems, on the other hand, are designed to operate on a *stream* of documents, which is *dynamic*. Space for storing individual documents (or data structures representing them) for future reference is at a premium, and storage is generally discouraged. In essence, the difference is that traditional IR systems have *random access* to the documents on which they operate, whereas filtering systems access their documents in a *sequential* manner.

Dissemination decision. An immediate consequence of the nature of their respective information sources is that, by having random access to documents, traditional IR systems have tended to *rank* all documents they return in response to a query by their similarity to said query. Due to the lack of random access to documents, filtering systems must make immediate binary decisions on whether or not to disseminate each new document they encounter. This decision must usually be made without the ability to compare the document to any other previously encountered one.

User involvement. Querying a traditional IR system requires *active* user participation. The user normally types in a number of query terms, submits the query, and receives a list of documents in response. The user's information need may or may not be fulfilled, and the user has the option of continuing to use the system by submitting another query, modifying and re-submitting the original one (with or without the assistance of a relevance feedback feature of the system). In other words, using a traditional IR system is necessarily a user-*supervised*, interactive operation. With filtering systems, the user is not necessarily required to perform any activities beyond setting up an original query. The system monitors the incoming document stream on its own, and occasionally sends the user a document it decided to disseminate. Filtering systems may, therefore, be more or less *unsupervised* and non-interactive, where the user is assigned

a *passive* role. From the user's point of view, traditional IR systems use a *pull* model, whereas filtering systems operate according to a *push* model.

Retrieval time-frame. Because of the difference between levels of interactivity, traditional IR systems are used on a much shorter time-frame than filtering systems. When a user submits a query to a traditional IR system, they expect to either fulfill their information need right away, or give up (a document that would fulfill their need for information cannot be found in the collection). On the other hand, a filtering system is used on a *long-term* basis, with only optional training and/or user supervision.

Query complexity. Again, because of differences between levels of interactivity, queries submitted to traditional IR systems tend to be short and simple. It is rare that relevance feedback is used for more than one iteration beyond the original query. Filtering systems, on the other hand, tend to use *multiple* iterations of relevance feedback, causing queries to grow ever more complex over time, as the system receives feedback on its dissemination decisions. Filtering queries are sometimes referred to as *user profiles*, models of the user's information needs that are much more accurate than the usual IR query.

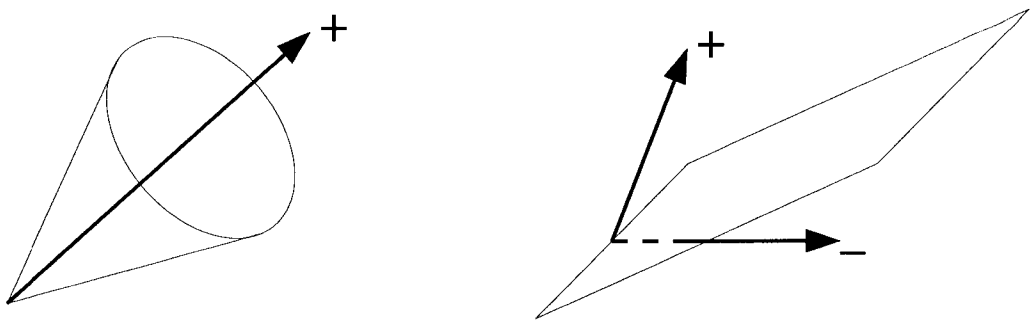
Personalization. Since traditional IR requires a large document repository, its obvious implementations use large, centralized databases and offer access to simple, passive clients that are required to trust the server with whatever information they provide, including the submitted queries. On the other hand, filtering systems can be implemented as smart, active clients, which are perfectly capable of doing their job while remaining under the total control of the users. This property makes filtering systems very well suited for application where the privacy of users and personally-tailored service are essential.

2.3.1 User Feedback

To improve the filtering system's performance, users may provide training (feedback) by rating disseminated documents as relevant or non-relevant, and, possibly, by providing examples of relevant documents to be incorporated into the filtering query or user profile. This is an extension of the Relevance Feedback mechanisms used with traditional IR systems. Some filtering systems require that both positive and negative examples be used in the construction of the user profile. These systems are usually probabilistic (Bayesian) [81, 101], but, theoretically, vector-based models such as TF-IDF could also be implemented this way. When a new document is considered for dissemination, it is *classified* into either the positive or negative category, by comparing its distances to the center of the positive and negative classes, respectively. For Bayesian systems, the *probabilities* of the document being positive or negative are compared. For a vector-based system, the cosine similarity between the document and the centroid vectors of the positive and negative classes could be compared. The document is disseminated if it is more likely to be positive (or closer to the centroid of the positive class than to that of the negative class).

Needless to say, systems that require negative feedback in addition to positive feedback force the user to expend extra effort in training. For this reason, some filtering systems (mostly vector-based ones [21, 148, 4, 45]) are designed to require only positive examples to be incorporated into the user profile. In order to facilitate binary dissemination decisions, a *dissemination threshold* is used along with the representation of the positive class in the user profile. This dissemination threshold indicates the minimum value of similarity between any new document and the centroid of the positive class for the new document to be disseminated.

Essentially, the dissemination threshold forms a cone around the vector representing the centroid of the positive class. Any document that falls inside this cone is likely



a) b)

Figure 2.1: Single positive centroid with dissemination threshold (a), and positive and negative centroids with separation plane (b)

relevant and will therefore be disseminated. Documents outside the cone are likely non-relevant, and will be ignored by the system. In the case where both a positive and a negative class are learned, we can imagine a plane that separates the two centroid vectors; documents on the same side of this plane as the positive centroid will be disseminated, whereas documents on the negative centroid's side will not. Figure 2.1 graphically illustrates these scenarios.

2.3.2 Profile Construction and Filtering

Most filtering systems are implemented using either the vector-space model (TF-IDF), or the probabilistic model (Bayesian classifier). We review some of the main works in each of these categories, as well as some novel, experimental techniques that do not follow either of the established models.

2.3.2.1 TF-IDF Filtering

TF-IDF based filters build and maintain one or more TF-IDF vectors representing topics of interest. Document vectors are compared against these topic vectors using the cosine similarity metric.

Allan [1] presents a method for incremental feedback for filtering, based on the idea

that documents can't be stored for future reference. The method consists of selecting 100 terms from each set of feedback documents (at each intermediary point) and adding them to the *original* user query. Precision and recall are consistent with most TF-IDF results seen so far (their harmonic mean appears to be approximately 0.4). A small number of past judgments are kept across iterations for context, which addresses changes in user interests (query drift) by measuring relevance not only against the original query, but also against recently learned positive examples.

Callan [21] proposes a filtering system based on TF-IDF vectors, with no negative feedback, and introduces the concept of an adaptively learned dissemination threshold. The threshold starts out low, and increases as non-relevant judgments indicate lack of precision. Recall was between 0.2 and 0.4, and precision between 0.2 and 0.3. These low values are a consequence of the overly restricted approach allowing the dissemination threshold to change in only one direction.

Zhai et al.[148, 147] propose an improved method of learning an adaptive threshold that can both increase and decrease as dictated by feedback. Their technique was based on storing documents between instances of feedback and adjusting the threshold to a value that would have maximized performance given the latest set of cached feedback documents.

Arampatzis [4] and Zhang [149] set thresholds based on the assumption that scores for relevant documents follow a Gaussian distribution and scores for non-relevant documents follow an exponentially decreasing distribution. Parameters of such distributions are learned from training documents and used to set dissemination thresholds that significantly boost filtering performance. These methods determine the optimal dissemination threshold for a given training set, but require storing all training data before the threshold is calculated.

2.3.2.2 Bayesian Filtering

Bayesian filtering is a straightforward application of the Naive Bayes Classifier (NBC) to text filtering. Documents are classified into one of two classes, *positive* and *negative*. A new document is disseminated if its probability of being positive is higher than that of it being negative.

Lewis and Gale [81] propose using uncertainty sampling to improve a classifier, by iterating through the following steps:

1. Build initial classifier;
2. While receiving feedback:
   (a) classify unlabeled examples;
   (b) pick subset of least certain documents, and ask user to label them;
   (c) train new classifier.

This approach requires considerable user interaction for the training phase and also requires an algorithm-driven interaction: the algorithm determines the documents for which additional feedback is required. In a later study, Lewis et al. [82] compare Rocchio against two other algorithms (Widrow-Hoff, and Kivinen-Warmuth EG) on a *supervised* (i.e., interactive) filtering task. Both algorithms outperform Rocchio's relevance feedback algorithm in this scenario.

Nigam [101] proposes the use of expectation maximization (EM) to iteratively improve the performance of NBC by repeatedly classifying unlabeled samples with the previous iteration's classifier, and using them to build a new classifier. Liu [86] and Li [83] propose a method of selecting the most likely negative documents, in order to limit requirements for labeling documents to positives only. So-called "spies" (a small random sample of positive documents) are used between iterations to measure if a classifier

adequately separates positive documents from negative ones. The most certain negatives and labeled positives are then used to build a final classifier, which is used to filter new incoming documents with a high degree of accuracy (precision, recall, and their harmonic mean all score above 0.6).

2.3.2.3 Other Filtering Methods

Several other methods have been used in filtering experiments: Latent Semantic Indexing (LSI , an alternative vector-space method), evolutionary algorithms used to train populations of simple filtering "agents", and approaches based on information theory. Foltz [47, 48] proposed a filtering system based on LSI and k-nearest neighbors. Based on singular value decomposition (SVD) and a list of relevant examples, the system can identify which new items are close enough to a sufficient number of relevant examples to be also considered relevant. Hull [70] compares filters based on LSI and those based on the vector model using the Cranfield [35] document collection. His findings are that while LSI can yield better performance, it does so at a significant computational and storage cost.

Sheth and Maes [127] use a genetically evolved population of agents to filter Usenet newsgroup feeds; their system (NewT) starts out with an internal representation containing several fields, among which the most important are the name of the newsgroup and a set of keywords. During the operation of the GA, agents trade keywords to improve their version of the user's profile. The agents are evaluated based on how much positive feedback was generated on articles they helped disseminate, and achieve performance consistent with other filtering systems: precision and recall are in the neighborhood of 0.5, and each drops off severely when the other is optimized. Clack et al. [33] uses a more complex internal representation for their agent population, based on parse trees. This approach yields high accuracy (authors report better than 95%), but only on a small test collection and with questionable potential for scalability to larger collections.

Amati [3] proposes a framework for a filtering system (ProFile) based on information theory. Weights are computed for terms based on the expected relevance of documents containing them. This approach is conceptually similar to using a Bayesian classifier, and reported results are also consistent with other Bayesian filtering systems.

2.4 Evaluation of IR Systems

The performance of an IR system depends on a large number of factors, many of them subjective. However, given that an IR system's purpose is to maximize relevant results and minimize non-relevant ones, a formal evaluation methodology has evolved based on collections of documents with known relevance to one or more topics. This approach is also suitable for filtering systems, and may also be used for evaluation of filtering components of Web information agents discussed in subsequent chapters. We review the basic performance metrics of an IR system: precision and recall. We also look at alternative metrics that have been proposed in the literature and conclude with a review of a few standardized document collections frequently used by IR researchers.

2.4.1 Recall and Precision

Typically, IR systems are evaluated in conjunction with a known document repository (or corpus) by submitting queries for which all documents in the corpus have been rated as relevant or non-relevant by expert judges. The relationship between the system's behavior and the expert judgments can be represented by a contingency table [80], such as the one in Figure 2.2. Expert judgments are marked either R for relevant or N for non-relevant. System predictions are either F for found, or M for missed. Thus, RF is the number of relevant documents found by the system. RM is the number of false negatives, i.e., relevant documents missed by the system. NF is the number of false positives, i.e., non-relevant documents mistakenly retrieved by the system.

		Expert judged	
		relevant	non-rel.
System predicted	relevant	*RF*	*NF*
	non-rel.	*RM*	*NM*

Figure 2.2: Contingency table of system predictions vs. expert judgments

The most frequently used performance metrics for IR systems are recall:

$$recall = \frac{RF}{RF + RM}$$

and precision:

$$precision = \frac{RF}{RF + NF}$$

Recall measures how good the system is at identifying relevant documents, while precision indicates how well the system can avoid retrieving non-relevant documents.

Other metrics mentioned in the literature, but less frequently used in practice are fallout [112, 80]:

$$fallout = \frac{NF}{NF + NM}$$

which estimates a pseudo non-relevant recall, and error rate [80]:

$$error\ rate = \frac{RM + NF}{RF + RM + NF + NM}$$

Well-performing IR systems would have both high recall (find most of the relevant documents in a collection) and high precision (find as few non-relevant documents as possible); ideally, both recall and precision should have value 1.0. Metrics such as fallout and error rate should have value 0.0. Typically, however, recall and precision are optimized at each other's expense. Systems tuned for high recall may include many false positives, and thus have low precision. When optimized for precision, systems may leave out

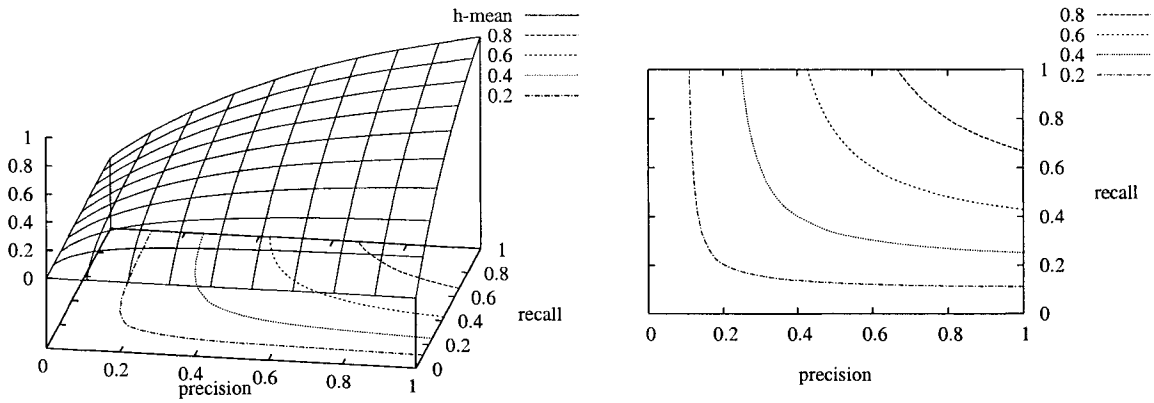


Figure 2.3: Harmonic mean of precision and recall

many relevant documents, leading to poor recall. In order to capture both precision and recall in a single metric, an *effectiveness* or E-measure has been proposed by Rijsbergen [139] and Shaw [125]:

$$E = 1 - \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

The parameter β allows one to specify the relative importance assigned to precision versus recall: sub-unit values of β indicate that recall is more important. When $\beta > 1$, the user values precision more. Sometimes, the E-measure is used in its complement form [81, 126]:

$$F = 1 - E$$

For the special case where $\beta = 1$, both precision and recall are valued equally, and we end up with their harmonic mean:

$$HM = F_{\beta=1} = \frac{2PR}{P+R} = \frac{2 \cdot RF}{2 \cdot RF + RM + NF}$$

With both precision and recall varying in the interval [0,1], the harmonic mean is a surface like the one depicted in Figure 2.3.

As IR systems don't usually *select* documents from their corpus to be retrieved in response to a query, but rather *rank* documents in decreasing order of their similarity to

the query vector, it is possible to imagine (at least in theory) that *all* documents in the corpus are ranked. If an imaginary cut-off is set after the top n ranked documents, recall and precision can be computed as if the system had selected these top n documents. As we increase n, recall increases as more relevant documents are included above the cut-off, and precision decreases as more non-relevant documents are also included above the cut-off. A method of representing IR system performance proposed by Baeza and Ribeiro [7] consists of plotting precision values corresponding to *standard* recall points of $0.0, 0.1, 0.2, ..., 1.0$.

IR systems are usually evaluated based on *multiple* different queries. Precision values computed at standard recall points are averaged across all queries, resulting in *average precision-recall* graphs used extensively in literature on traditional IR systems.

2.4.2 Benchmarks and Alternative Metrics

TREC (Text REtrieval Conference) [63, 141] offers a venue to support meaningful comparison of the various systems that are being developed. The benchmarks used at TREC consist of a large (and growing) collection of documents (currently distributed on six CD-ROMs), and a set of several hundred evaluation queries (also referred to as *topics*). For each query, a subset of the existing documents has been rated as relevant or non-relevant by a human judge. We are particularly interested in the TREC filtering track, in which filtering systems are compared by being supplied with a chronological stream of documents from the benchmark collections. Recently, filtering systems at TREC have been evaluated using *utility functions* [71]:

$$utility = u1 \cdot RF - u2 \cdot NF$$

which reward finding relevant documents (with a $u1$ reward each) and penalize retrieval of non-relevant documents (with a $u2$ penalty each). Some of the reward/penalty values used at TREC conferences were $(u1 = 3, u2 = 2)$ and $(u1 = 4, u2 = 1)$.

Recall, precision, and their derivatives have been criticized [109, 136] for requiring full knowledge of all candidate documents and their relevance to all queries used, which is frequently not available in practice. Another point is that these metrics have been designed for traditional, batch-mode IR systems, and are not well suited for evaluation of modern techniques (such as text filtering systems).

Simpler metrics designed to work with partial knowledge of the test corpus are described in [7]. Assuming only a subset of the existing relevant documents in a collection is known to the user, the values from which recall is computed (*RF* and *RM*) are further divided into *known* and *unknown*. One metric used is *coverage*:

$$coverage = \frac{knownRF}{knownRF + knownRM}$$

which is the equivalent of recall on the known subset of relevant documents. Another metric is *novelty*:

$$novelty = \frac{unknownRF}{knownRF + unknownRF}$$

which measures the ratio of relevant documents retrieved that were previously unknown.

Coverage, novelty, and utility are metrics specifically geared toward filtering systems, designed as workarounds to the lack of control over the test data (i.e., lack of comprehensive knowledge of relevance values for all documents in the data set). The downside of these metrics is that they are much less *portable* than recall and precision: scores given to filtering systems by these metrics are only relevant in the context of comparing systems to *each other* given one particular data set. For example, utility completely disregards the number of relevant documents *missed* by a filtering system. If full knowledge of relevance values with respect to the topics used is available, precision and recall remain the more meaningful performance metrics.

Chapter 3

IR on the Web: Search Engines and Web Agents

While text filtering provides a way to move IR systems away from centralized systems and toward personalization, search engines provide a move away from IR systems based on proprietary, local information collections and toward applying IR to the widely available, public information repository which is the World Wide Web. The first part of this chapter provides a brief overview of the evolution and shortcomings of search engines, as well as a series of techniques that help optimize their behavior. The second half is an overview of related work in the field of Web information agents. We draw the same parallel between search engines and Web agents as between traditional information retrieval and text filtering in the previous chapter.

3.1 Search Engines and Related Optimization Techniques

In the early 1990s, the Web was at the beginning of its extremely rapid growth, and locating information of interest was becoming a problem. Several "directory" sites were built, but with no real chance at scalability. Thus, search engines were born. Sites such as AltaVista[2] and Lycos[90] originally started out by using straightforward IR methods (as discussed in Section 2.1) on a corpus collected by programs known as

crawlers. Crawlers were originally simple depth- or breadth-first traversals of the Web (possibly several instances seeded at various different locations), collecting pages into a corpus that would then be searched in response to user queries.

In the late 1990s, the growth of the Web became even more explosive, and simple straightforward search engines such as the ones described above could no longer keep up [78]. According to Lawrence et al.[79], it was taking too long (months) before a new site would be crawled and added to the corpus, and none of the crawl/IR search engines were indexing more than cca. 16% of the available documents on the Web. Users were submitting poorly formulated (usually, one- or two-word long) queries, and last, but not least, site designers and search engine operators started manipulating their implementations to skew results. For instance, many sites contained huge amounts of keywords invisible to the viewer (either fonts using the background color, or keywords hidden in meta tags) in order to skew the rating received from the search engine. Also, many search engines were accepting payment for higher ranking, without necessarily disclosing this bias to their users.

In order to deal with the limitations of early search engines, several directions of research were pursued. Searches were distributed across multiple search engines to improve recall (meta-search); search results were post-processed and clustered by topic, to help users quickly find the topic they were really looking for; smaller, topic-specific search engines were built using focused crawlers that would only index documents related to the given topic. Also, search engines began exploiting the additional information that came hidden in the documents' link topology [16, 103], leading to the highly successful Google search engine [55].

Currently the most popular search engine, Google's status was achieved through a clean, clutter-free interface, and use of link popularity derived from Web topology [16, 103] in addition to traditional IR methods. Google also offers various personal-

ized services, such as WebAlerts [56], which sends users email when changes to their queries' top-10 results are detected. However, there are still open questions [142] regarding Google's privacy policies, tracking of users through permanent cookies, and the inherent fairness of topology-driven rankings. A recent study by Cho[31], performed over a period of seven months, indicates that the top 20% most linked Web pages will receive over 70% of new links, regardless of content.

Shifting control toward users by minimizing the information disclosed to third parties is a major step toward privacy and personalization. An example is Copernic[36] – a personal, client-side agent offering services such as meta-search (farming out searches to multiple search engines and integrating returned results), notification of new search results, and change monitoring. Being a proprietary closed-source application, the algorithms and inner workings of Copernic are difficult to evaluate or audit.

3.1.1 Meta-Search

The concept of meta-search on the Web has been inspired by GlOSS[58, 57], a system for simultaneously querying multiple text databases. Meta-search engines like MetaCrawler[123, 124], SavvySearch[43, 44, 69], NECI[77], and Vivisimo[140] are placed one step above standard search engines in the Web information food chain. They do not search the Web themselves, but rather send off queries to other search engines, and then integrate those results and present them to the user in a consistent fashion. The added value of meta-search engines consists in potentially more complete coverage of the Web, leading to improved recall – while individual search engines only covered small portions of the Web[79], querying several of them simultaneously offered access to the union of these portions. Meta-search engines can also learn to associate specific types of queries to search engines that are the most likely to come up with good results for them, a technique called *query routing* [135]. Thus, meta-search can be improved by only querying search engines expected to be able to find relevant information.

A meta-search system generally consists of the following components:

- a user interface that allows the entering of queries and displaying of results,
- multiple interfaces (or *wrappers*) to search engines, used for conversions between the formats required by the search engines and the internal format of the meta-search system,
- and, in systems which implement query routing, a learning module which uses past experiences to identify which search engines are most likely to find relevant documents with respect to a new user query.

The meta-search concept has been applied to other Web-related activities, most notably comparison shopping [41]. For a certain set of products (e.g., books, CDs, videos, etc.), the service looks up prices on a variety of vendor websites and lists the results in decreasing order of their extended price. The main problem confronting meta-search services is wrapper maintenance. Vendors change their user interfaces more frequently than search engines, which requires the comparison-shopping site operator to rewrite the wrappers. Proposed approaches to this problem are presented in [34, 99].

3.1.2 Clustering of Search Results

Based on the assumption that relevant documents are more similar to each other than to non-relevant ones, studies [66, 146] have suggested that search results should be clustered into categories instead of being ranked according to how well they match the user query. This is one of the possible ways to deal with the problem of vague user queries. An additional user-specified number of keywords can be extracted from the categories resulting from the clustering operation and used to help users decide which documents to examine first. The preferred distance metric used by the clustering algorithm is the TF-IDF based cosine metric. Users in both studies have claimed improved satisfaction

and ease of finding relevant information when search engine results were presented in a clustered form. Examples of search engines which cluster search results are Northern Light[85] and Vivisimo[140], which automatically cluster the list of hits when it becomes available, and Yahoo[144] which is based on human-generated categories.

3.1.3 Directed Crawling

Large, centralized search engines attempt to perform exhaustive crawls of the entire Web, which can cause large delays before new information is made available to the users. Smaller-scale search engine implementations can be designed to crawl the Web on an as-needed basis, given instructions as to what information to look for. As an example, Shark-Search[67] will start with a user query and an initial "seed" document and will perform a width and depth bounded heuristic search for information with high relevance with respect to the query. The expectation of finding relevant information is increased by an incoming link from a relevant document and decreased by the distance (both in width and depth) from the origin of the search. Shark-Search brings about a two-fold improvement in performance over Fish-Search[39], the original client-side Web crawling algorithm. Performance is measured using "sum of information", obtained by adding together the similarity between each encountered document and the query on which the system is currently focused. A similar system has been developed by Chakrabarti et al.[25, 24], consisting of three components: a *classifier* which assesses the relevance of encountered pages, a *distiller* which determines visit priorities, and the *crawler* itself, which performs a personal search for the user based on information from the other components. The focused crawler maintained an average relevance between 0.4 and 0.6, while the baseline, unfocused crawler's relevance quickly dropped to 0. Another example is given by Mukherjea[100], where a small-scale, topic-specific search engine is proposed based on a topic-focused crawler and a user interface to allow searching the collected topic-specific documents.

These approaches exhibit a tendency to *personalize* the services offered to the particular needs of individual (groups of) users. The systems are typically running locally, serving individuals or small groups rather than large masses of off-site users.

3.1.4 Refinement Based on Link Topology

In addition to traditional Information Retrieval techniques, search engines have access to a wealth of information hidden in the hypertext link structure of the web pages they collect [15]. Such information has a tremendous potential to improve the quality of returned results.

An early example of a system which leverages topology information is ParaSite[134], which classifies links on a Web page into one of the following types:

Upward: leading to a page with more general information, or to a superset of the current page;

Downward: leading to more specialized information, or a subset;

Crosswise: links to equally specialized but different members of the same superset;

Outward: links to off-site pages on the same topic.

The type of a link can be inferred using the URLs: if the host is the same, then a shorter, longer, or equally long path would predict upward, downward, or crosswise links, respectively. Otherwise, we are dealing with an outward link.

Another approach, with more solid mathematical underpinnings, is used in the HITS algorithm [75, 53, 23]. HITS is based on the assumption that a link to a Web document represents an implicit acknowledgment of its relevance to the document containing the link, by the authors of the latter. Thus, the algorithm is designed to find two kinds of Web documents: *authorities* which contain highly relevant information with respect to a certain query, and *hubs* which contain links to many strong authorities. According to

Gibson et al.[53], hubs and authorities exhibit a "mutually reinforcing relationship": the quality of a hub is given by the number and quality of the authorities it points to, and the quality of an authority is indicated by how many good hubs are pointing to it. In order to isolate the best hubs and authorities on a given topic, the authors describe the following algorithm:

1. Submit a query to a search engine; the set of retrieved documents will form the *root set.*

2. Gather documents pointed to by members of the root set, and documents that point to members of the root set. The latter can be accomplished by querying a search engine with a URL: the retrieved documents will be the ones containing a link to this URL. The union of these new documents and the root set will form the *base set.*

3. Each document d in the base set will be assigned a hub weight $h(d)$ and an authority weight $a(d)$, all initialized to 1. The weights are then updated iteratively as follows:

$$a(d) = \sum_{p \to d} h(p)$$

$$h(d) = \sum_{d \to p} a(p)$$

where $p \to q$ means that document p has a link to document q. Both the authority weight vector and the hub weight vector are normalized after each iteration.

4. After the weights converge, the highest ranking hubs and authorities are presented to the user.

While highly effective, this algorithm has several weaknesses, as identified by Bharat et al.[12]:

Mutual reinforcement: if a set of documents on a site all point to a single document (e.g., a table of contents), they drive up its authority score, which in turn drives up their own hub score, thus giving disproportionately high importance to the opinion of a single information source. This can also happen with automatically generated pages, where a standard set of links is always inserted in every page (e.g., a link to the front page of the site).

Topic drift: Sometimes the neighborhood of the root set contains non-relevant documents that are very well connected, and the highest ranking hubs and authorities will be about a different topic than originally requested. The example given by Bharat et al.[12] is the query "jaguar *and* car", where the algorithm settled into the more general topic "car" and returned the homepages of various car manufacturers as the top authorities.

The authors suggested further weighting of authority and hub scores during the iterative computation for those pages that are prone to the mutual reinforcement problem: an extra multiplicative weight of $1/k$ should be introduced, where k is the number of documents on the same site linking to or being linked to by another single document. For the topic drift problem, the suggested fix is to prune the base set of those nodes that do not pass a threshold of similarity with the original user query. The authors' experiment suggests that these fixes improve precision over the original HITS algorithm by 45%.

PageRank[103] (used by Google[16, 55]) is to date the most successful published link-topology based ranking scheme for web pages. Web pages are ranked according to the following formula:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

where B_u is the set of pages linking to document u, and N_v is the number of pages pointed to by document v, and c is a damping factor designed to deal with pages that

have no outgoing links. The R PageRank values are computed iteratively, and converge in a matter of hours for sets of millions of Web documents.

3.1.5 Continuous Queries

Continuous queries are a push technology originally proposed in the database research community. The term was coined by Terry et al.[138], authors of the Tapestry system. Tapestry allows users to register queries with a database server, which then compares every new document added to the database to all existing queries upon insertion. All owners of matching queries are immediately notified of the availability of a new potentially relevant document. Tapestry supports append-only databases exclusively, which makes the system functionally identical to a filtering system implemented server-side. The similarities between filtering (discussed in Section 2.3) are also noted by Babu and Widom[6]. Barbara[8] extends the theoretical foundation for continuous queries to databases that allow deletion and modification to already existing documents.

Several web-related applications of continuous queries have been proposed. Chawathe et al.[27] present a system for monitoring and reporting changes in semi-structured HTML documents, such as restaurant listings. Changes are not checked for relevance, but the system offers a mechanism to display and query the detected changes. OpenCQ[87] uses wrappers to access information available on various Web sites and prompts users when certain conditions specified in their continuous queries are met. The main application consists of monitoring meteorological data such as temperatures, water levels, etc., and notifying users when certain values are reached. NiagaraCQ[28] optimizes the time and computational expense of executing continuous queries, by grouping them together based on similarity.

Continuous queries could be viewed as a server-side step towards personalization. However, comparing each newly crawled document against a potentially large list of registered user queries does not scale well, which is why services such as Google's

WebAlerts[56] typically run the user's query through the search engine and email out new links that appear in the top 10. Rather than a *true* implementation of a continuous query service (e.g., using triggers), these methods are based on polling. When many resources with varying change frequencies must be monitored, an adaptive technique may be used. For instance, CAM (Continuous Adaptive Monitor) [104] adaptively schedules polling for different resources based on their update frequency learned during a tracking phase. CAM's resource allocation policy leads to performance an order of magnitude better than that of a baseline uniform policy.

3.2 Web Information Gathering Agents

Search engines mainly operate on the principles of traditional IR: gather a huge repository of documents, and assume random access to them when responding to simple user queries. Returned documents are ranked by decreasing order of expected relevance. Similar to the way text filtering allows for personalization in the field of IR, Web information gathering agents personalize Web document search and retrieval. In fact, the type of Web information agent that is the scope of this dissertation is *based* on text filtering. Machine learning [97] and IR/filtering techniques [108] are used to build and maintain complex filtering queries or user profiles that help improve future performance. Methods to gather candidate documents vary widely, from crawling the Web just like a search engine [94], to generating and sending queries to search engines [19], to monitoring online newspaper front pages for interesting new articles [29], etc.

The remainder of this section reviews several well-known Web information agents and the techniques on which they are based: methods for generating incoming document streams for the filtering component, maintaining user profiles via clustering, and detecting redundancy in addition to document relevance. Finally, we explore techniques used in evaluating the performance of Web information agents.

Agent	Purpose	Profile Representation	Learning	Doc. Gathering
Letizia	Rate Links	TF-IDF	Rocchio	Crawl/Rank
WebWatcher	Rate Links	Winnow Wordstat TF-IDF	Supervised	Rank Links
Amalthaea	Collect docs	TF-IDF	GA	Gen. Query
Syskill & Webert	Rate Links	Decision Tree Nearest Neighb. Neural Net TF-IDF	Supervised	Crawl
WebAce	Collect docs	TF-IDF	Clustering	Gen. Query
WebMate	Collect docs	TF-IDF	Clustering	Crawl
Watson	Collect docs	Weigh. Term Vec.	Clustering	Gen. Query
InfoSpiders	Rate Links	Neural Net	GA	Crawl
NewsAgent	Collect docs	TF	Clustering	Crawl

Table 3.1: Summary of example Web information agents

3.2.1 Web Information Agent Examples

Because of our main objectives (personalization and privacy), our focus is on agents that are capable of running as standalone systems on the users' local machines. The ability to take advantage of serving multiple users for increased performance is a bonus, but should by no means be a requirement. The following agent prototypes (listed in chronological order) allow us to review some examples of how user profiles are represented internally, what learning mechanisms are used, and which methods are used to collect potentially relevant documents. This information is also summarized in Table 3.1.

Letizia [84] learns a profile (based on TF-IDF) from heuristically assessing user interest in browsed Web documents (based on indications such as bookmarking a document, time spent viewing, etc). Breadth first search is used to crawl documents linked from the page currently being viewed. These documents are then compared to the learned profile, and ranked according to their relevance. Upon request, Letizia will then make recommendations as to which link is the most worth following. The advantage

of profiling based on indirect factors is that it minimizes the burden on the user; unfortunately, it also severely reduces accuracy since these factors may also be significantly influenced by causes other than user interest.

WebWatcher [5, 73] compares several methods of learning a user profile for a Web browsing assistant, which is supposed to recommend the most interesting link to follow given a document the user is currently viewing. The methods compared were Winnow (a linear threshold function over a set of Boolean features), Wordstat (a probabilistic approach based on computing the conditional probability of a link being followed), and TF-IDF. Their performance is similar, and clearly better than the random baseline. One large drawback to WebWatcher is the necessity of conducting off-line training sessions on the user's information request (goal) before testing the performance of the agent. In our experience, users are unlikely to put up with such inconvenience in a real-life scenario.

Amalthaea [98] evolves a population of information agents using a genetic algorithm. The purpose of the system is to have a subset of the agents gather documents from search engines by generating queries, and another subset filter these documents and submit the relevant ones to the user for review. The agents contain TF-IDF vectors representing portions of a user profile, and are evaluated based on their filtering ability or the quality of results from the search queries they generate. The population is bootstrapped based on an initial profile built from positive examples supplied by the user. The system was tested using synthetic data: manually built profiles were fed into the system, and returned documents were scored based on their similarity to the profile. As expected, the agent population converged towards high-performing individuals retrieving documents similar to the profile.

Syskill&Webert [106, 105] is a link-recommender system used as a test-bed for comparing several different machine learning techniques (decision trees, nearest neigh-

bors, neural nets, and naive Bayes) to a TF-IDF relevance feedback technique for learning user profiles. The authors' main conclusions are that a Bayes classifier and TF-IDF Rocchio relevance feedback compete for the top spot for best learning performance, and that good predictions could be made based on only the initial portions of Web pages. Also, pre-loading the profile with user-supplied relevant documents was found to significantly increase its usefulness.

WebAce [60] learns a set of TF-IDF vectors by clustering positive examples of documents supplied by the user. It then generates queries automatically to retrieve new documents on behalf of the user. Two novel clustering mechanisms (AHRP and PCDP) are specifically targeted for efficient operation in high-dimensional term vector spaces. Association Rule Hypergraph Partitioning (ARHP) clusters documents by maximizing the number of terms shared by documents within each cluster. Principal Component Divisive Partitioning (PCDP) is similar to Hierarchical Agglomerative Clustering (HAC) in that it starts with one cluster encompassing the whole set and splits leaf nodes with the largest scatter factor according to the direction of maximum variance into two new sub-nodes. WebAce caches documents and uses a non-incremental clustering algorithm on the accumulated cache at regular intervals. The clustering algorithms were evaluated using entropy as a measure of performance. Lower values for entropy indicate better performance, as clusters contain fewer different classes of documents. PCDP scores best with entropy 0.69, compared to ARHP (entropy 0.79) and the baseline given by a Bayesian classifier with entropy 2.05.

WebMate [29] learns a set of clusters (represented via TF-IDF vectors) from positive feedback provided by the user. The clustering algorithm uses an incremental-greedy approach, merging the two closest clusters when running out of space. The agent keeps a list of user-supplied Web pages as starting points for searching for new documents, and uses its profile to filter and disseminate relevant documents encountered along the way.

Disseminated documents are collected into a "personal newspaper" which is shown to the user periodically. By using an incremental clustering algorithm, the agent can learn without requiring storage to cache copies of relevant training examples.

Watson [19] uses a set of heuristics such as term placement within the document, count, and typeface to compute a set of term weights for documents currently being read or worked on by users in their word processor. User profiles are built using a greedy incremental clustering algorithm similar to WebMate. Top weighted terms from the profile are used to generate queries to a search engine, and search results are then organized and displayed in a window next to the one containing the document currently being worked on.

InfoSpiders [94] uses a genetic algorithm to represent and learn a user profile based on neural networks. Each agent in the population has a genotype consisting of a set of keywords. A feed-forward neural net is used to predict a document's relevance based on the term frequency of terms contained both by the document and by the agent's genotype. The weights of the neural net are learned during the lifespan of the agent, whenever feedback is made available by the user. During reproduction, two-point crossover is used on the keyword list, and random noise is added to each term's corresponding weights in the neural network. The individual members of the population act like miniature crawlers and are evaluated based on the relevance of the documents they chose to follow. The entire population of agents is used to recommend links worth following by the user. The system is evaluated on the online version Encyclopedia Britannica (EB), taking advantage of the known relevance of EB documents with respect to a given set of topics.

PersonalSearcher/NewsAgent [54] uses TF vectors to store profiles. Terms appearing in titles or links are given extra weight. Training samples are stored explicitly, and k-nearest-neighbors clustering is used to identify the user's topics. The application

is building a "personalized newspaper" by filtering items linked from a set of online newspapers. The system's performance, evaluated via F-measure (or harmonic mean of precision and recall) was in the range of 0.4 to 0.6. Data was collected by asking users to monitor online newspapers such as the New York Times and offer feedback on the documents collected by NewsAgent on their behalf.

3.2.2 Generating an Incoming Document Stream

Although not explicitly addressed as a separate issue by current research, a Web agent can be viewed as two separate components: a text filter similar to those studied by IR research, and a method of document gathering which supplies the filter with an incoming document stream.

One document gathering technique, used by Letizia[84] and WebWatcher[5, 73], is to extract links available within the current document and use some variant of best-first search to follow these links in a manner similar to a *directed crawler* (see 3.1.3). Documents encountered during such crawls are supplied as an incoming stream for the agent's filtering algorithm. Another technique, used by WebMate[29], is to regularly extract links from a user-supplied list of hub sites (such as online newspaper front pages) and treat the collected set of documents as a source for the filtering algorithm. Last, but not least, agents such as WebAce[60] and Watson[19] build a query based on the user profile and submit it to one or more search engines. The documents returned by the search engine are then forwarded to the filtering algorithm for further examination.

3.2.2.1 Query Generation

Generating search engine queries is a promising and inexpensive way to increase the potential relevance of the documents submitted to the filtering component of a Web information agent (i.e., the quality of the incoming stream of documents). Well-formulated queries have the potential to off-load significant effort to search engines, which should

provide the agent with a stream of documents with high likelihood of relevance. Using a search engine should save effort both in terms of filtering performed by the agent, and in terms of the cost of the Web crawl that would be required to gather the documents in the first place.

Query generation has evolved from the more general technique of relevance feedback [118], which modified the users' original queries with the purpose of improving the quality of returned results. Filtering queries (a.k.a. the agent's user profile) are not directly suitable for submission to a general-purpose Web search engine, in part due to the large number of terms and complex weighting schemes. Even without weighting, too many terms would overly restrict the range of potential results the search engine might return. Query generation refers to the technique of extracting relevant terms from the user profile, and using them to augment (or construct from scratch) a query suitable for a search engine.

In order to generate a new search engine query from scratch, only a few highly representative terms are typically extracted from the user profile. For WebAce[13], the authors propose that queries be generated from a cluster of documents based on the average term count (or term frequency – TF) across all documents within the cluster and on the number of documents in the cluster that contain a term (i.e., the document frequency of the term – DF). One set of documents with the highest TF and another one of the same size based on highest DF are selected. The intersection of these sets is used as the final query that is sent to the search engine (Yahoo[144], in this particular study). Only anecdotal performance information is provided for this query generation method (the authors generated and submitted a few queries and were satisfied with the results), since the study mainly focuses on building user profiles using various clustering algorithms.

CorpusBuilder[52] uses automatic query generation to collect relevant Web docu-

ments. Query generation is used to avoid a brute-force crawl and expensive filtering to isolate documents of interest. Queries are generated from the corpus collected so far, hoping to significantly increase the chances that search results would be relevant, thus speeding up the collection process. The authors use several different query generation methods:

- uniform – select n terms from the relevant documents with equal probability;
- term-frequency – select n most frequent terms from the relevant documents;
- probabilistic term-frequency – select n terms from the relevant documents with probability proportional to their term frequency;
- odds-ratio – select n terms with highest odds-ratio scores, as given by the formula:

$$OR_t = \log_2 \frac{P(t|rel) \cdot (1 - P(t|nonrel))}{P(t|nonrel) \cdot (1 - P(t|rel))}$$

  where t is the term, and $P(t|rel)$ and $P(t|nonrel)$ are the conditional probabilities of t appearing in a relevant and non-relevant document, respectively;
- probabilistic odds-ratio – select n terms with probability proportional to their odds-ratio score;

Best results were obtained with queries of size $n = 4$ and the simple odds-ratio method. Since the relevance criterion in this study was that documents must be in a particular language (Slovenian), the applicability of these results to other systems (where relevance has a semantic component) is not guaranteed. Filtering for topical relevance is necessarily more difficult and subtle than determining whether a document is written in any particular language.

3.2.3 Filtering Techniques for Web Agents

Several issues related to filtering have an important influence on the user-friendliness of a Web information agent: whether negative feedback is required in addition to positive, whether explicit topic information is required with each training instance, and whether the system can detect (and avoid disseminating) relevant but redundant documents.

For traditional IR, we have discussed the tradeoff between either requiring both positive and negative training examples, or using a dissemination threshold with positive-only feedback earlier (see Section 2.3). Thresholding was largely ignored by earlier research on Web agents, even when positive-only feedback was specified (e.g., Letizia[84], WebMate[29], etc.). We explore learning mechanisms for dissemination thresholds in more detail in Chapter 4.

In this section, we focus on simplifying the user's involvement with the process of training a text filter by automatically classifying training samples. Only relevance, but no topic label is required from the user. This is accomplished by using incremental clustering algorithms that automatically maintain a set of clusters representing the user's topics of interest, but without the need to explicitly store previous training examples.

We also examine techniques for detecting redundancy within newly encountered documents. Some of these documents may be relevant to one of the user's topics of interest, but have no *additional* relevant information compared to previously disseminated documents. As such, it is the filtering system's job to avoid disseminating these documents to the user.

3.2.3.1 Incremental Document Clustering

Franz et al.[49] emphasize the contrast between learning users' topics of interest (topic detection), and filtering relevant documents once the topics are known (topic tracking). Topic detection involves:

- Unsupervised learning: relevant training samples will not have associated topic labels.

- Hard decisions: each training sample must be assigned to one cluster representing a new or existing topic (n-ary classification).

For topic tracking, we have:

- Supervised learning: new documents are compared to a set of existing topics (supplied by either a topic detection algorithm, or explicitly by the user)

- Soft decisions: new documents may be assigned to more than just one topic, or to neither. There are n separate binary classifiers, one for each existing topic.

In practice, we observe that these modes of operation frequently coexist. While unknown new documents are encountered by the agent, their relevance is predicted using topic tracking. If the user points out a relevant training sample (but without naming an applicable topic), topic detection comes into play, and the agent updates its list of learned topics automatically.

Web agents such as WebMate[29] and Watson[19] are capable of automatically learning to identify topics of interest by clustering positive training samples supplied without an associated topic label. The overall profile is learned in a supervised manner (since users are required to provide training examples). However, learning the list of topics of interest is unsupervised, since no topical information is expected or required from the users. This is an extra step towards making such agents user friendly, by minimizing the amount of effort required for training.

Typically, an upper limit is set on the number of allowable clusters k. New documents must be partitioned into up to k clusters so as to minimize the maximum cluster size. Most frequently, IR systems use hierarchical agglomerative clustering (HAC) [110]. This method assumes that all n documents are available a priori and starts out

by assigning each document to its own cluster. Pairs of clusters are merged iteratively, based on the distance between their centroids, until only k clusters are left. These clusters can then be treated as automatically learned models of the user's "topics" of interest.

Unfortunately, a clustering algorithm that requires the presence of all documents on which it is supposed to operate is not practical for client-side, personalized filtering systems or Web agents. First, there are storage considerations: a filtering system would need to store all training samples it has ever received explicitly, so it can re-cluster them periodically when updating its information on the user's current topics of interest. Secondly, re-clustering a growing number of training samples from scratch can become quite expensive as more and more samples are collected.

The solution is to use *incremental* clustering algorithms. Only information about the current cluster centroids is stored, and explicit document information is discarded after it has been used to update the cluster centroids. An incremental clustering algorithm maintains up to k clusters: every time a new training sample is received, it is either added to one of the existing clusters, or it becomes the seed of a new cluster. If the maximum number of k clusters is exceeded, two or more existing clusters are merged together, such that the resulting number of clusters never exceeds the limit k.

The ARAM clustering algorithm [137] incrementally modifies the appropriate cluster if a match criterion is met, or else adds a new cluster. There is no upper limit on the number of allowed clusters, and documents are stored explicitly to allow *splitting* of clusters when necessary. Clusters are labeled with the top few terms by TF-IDF weight and presented to the user as a means of automatically categorizing relevant documents. Although no quantitative performance information is provided, ARAM offers an interesting graphical interface allowing users to visualize the main keywords and weight of each learned cluster.

Charikar et al.[26] present a set of incremental clustering algorithms suited to such a hybrid approach, along with mathematical performance analysis based on a metric called *performance ratio*. This metric is computed as the ratio between the maximum cluster diameter obtained by the incremental algorithm in the worst case (i.e., assuming the most inconvenient update sequence), and the maximum cluster diameter size obtained by an optimal algorithm with random access to all points. Lower performance ratios indicate better algorithms, and 1.0 is the theoretical optimum. Three incremental algorithms are compared: *greedy*, *doubling*, and *clique partition*.

Greedy Algorithm. A fixed number of k clusters are maintained. The first k documents are assigned each to their own new cluster. Once k clusters have been formed, each new document is treated as a $(k+1)$st cluster, and the closest two existing clusters are merged in order to bring the total back to k. Thus, a new document may either be assigned to an existing cluster, or it may form a new cluster and force two existing clusters to be merged. Greedy incremental clustering is the simplest of the three methods and captures existing practice in personal Web agents (e.g., WebMate and Watson). According to Charikar et al.[26], the performance ratio of the greedy algorithm is at least $2k-1$, which is rather poor when compared with the other two algorithms.

Doubling Algorithm. As with greedy clustering, initially, each feedback document up to the kth is placed into a separate cluster. However, merging allows for multiple clusters to be joined, such that the total number of clusters may well be lower than k (while still never being allowed to exceed it). The clusters to be merged are found based on a distance threshold, d. When the number of clusters first exceeds k, d is initialized to twice the distance between the closest two clusters. A *merging stage* is entered, in which groups of clusters close to each other are joined by picking an arbitrary cluster and merging it and every cluster that is closer to it than d. This process is repeated until all groups of close-enough clusters have been merged. New documents are added during

the *update stage*, in which the algorithm normally operates while the cluster count is below k. A document is added to the closest cluster if the distance between them is less than $2d$; otherwise, a new cluster is created. Whenever the limit of k clusters is exceeded, the algorithm *doubles* the distance threshold d (hence the name of the algorithm), and enters another merging stage. The doubling algorithm has a constant performance ratio of 8, independent of the number of clusters k.

Clique Partition Algorithm. This algorithm differs from the doubling algorithm only by its merging stage. Instead of picking clusters at random, and merging them with every other cluster within distance d, the set of existing clusters becomes the set of vertices of a graph, with edges connecting any pair of clusters that are closer than d. A minimum clique partition is performed on this graph, and members of each clique are merged. A detailed description and solution of the clique partition problem (and its complement, the graph coloring problem) is presented by Mehrotra and Trick[93]. Although the minimum clique partition problem is NP-hard, solving it may not be too inefficient for reasonable values of a user's maximum number of topics k (expected to be in the tens or single digits). The performance ratio of the clique partition clustering algorithm was proven to be 6, slightly better than that of the doubling algorithm.

3.2.3.2 Redundancy Detection

The field of Topic Detection and Tracking (TDT) has been concerned with detecting *novelty* [145, 51] – i.e., finding the first occurrence of a document on a new topic. Researchers of Web information agents (and filtering systems) are becoming interested in a related problem: detection of *redundancy*. When a large number of relevant documents is disseminated from an incoming stream of Web documents, it is increasingly likely that some of them will be duplicates of each other due to aliasing and mirroring on the Web. Kelly et al.[74] report that duplicate Web traffic accounts for cca. 36% of data transferred over the Web. A somewhat harder problem is detecting whether different

documents make each other redundant. Broder et al.[17] propose a measure of *containment* between pairs of documents based on the intersection between sets of n-grams sampled from each document. This measure can be used to a certain extent to verify whether a document is a super or subset of another document. Crouch et al.[38] point out that, in order to be redunant with respect to an earlier document, a new document must necessarily be similar to it. Their approach for detecting redundancy is based on Natural Language Processing (NLP), by mapping document text to a knowledge base of concepts specific to the application at hand (copier machine repair notes, in the authors' particular application). This approach is hence unlikely to scale up to a general-purpose redundancy detection system for Web agents.

Carbonell et al.[22] propose "maximal marginal relevance" (MMR) as a measure for picking out new relevant documents from a set *New*, which are the least redundant with respect to already selected documents (*Sel*) with respect to a profile or query q:

$$d = \arg\max_{d_i \in New} \left[\lambda \left(sim(d_i, q) - (1 - \lambda) \max_{d_j \in Sel} sim(d_j, d_i) \right) \right]$$

where parameter λ allows the user to control focus between maximizing relevance with respect to the existing profile ($\lambda = 1$) and maximizing exploration ($\lambda = 0$). Values of $\lambda \in [0.3, 0.7]$ have been found particularly effective in detecting relevant and non-redundant documents.

Zhang et al.[150] propose several different approaches to redundancy detection. One is based on TF-IDF similarity: a regular similarity threshold is used to determine whether two documents are on similar topics, and another (higher) threshold to predict whether the earlier one makes the later one redundant. The heuristic assumption is that documents which are *too similar* are in fact highly likely to be redundant, and that non-redundant documents on the same topic have pairwise similarities within a range given by the two thresholds. Another approach is to compute the size of the set difference between terms appearing in the new document and those from the earlier one, which

measures the *novelty* of the new document. The approach using a second dissemination threshold has an 18% error rate, while the one based on set difference exhibits a slightly higher, 28% error rate.

3.2.4 Evaluation of Web Information Agents

Individual building blocks of a Web agent can usually be evaluated using automated experiments. For example, the filtering subsystem described in Section 2.4 can certainly be evaluated using automated experiments as demonstrated at the TREC conferences [63, 141, 71]. Haveliwala et al.[65], demonstrate the use of the Open Directory Project (ODP) [102] in the evaluation of various similarity metrics for Web documents. The ODP is a collaborative effort whereby Web pages on a great many topics are manually classified into a hierarchy of categories by human volunteers. Documents from within the same category, or from related categories *should* be rated more similar by a metric than documents from different, unrelated categories. Similarity metrics that give the highest score to pairs of similar documents (selected based on their ODP category) should work best in practice. The authors use the derived similarity metrics to implement "Related Pages" functionality – i.e., to find Web pages that are similar to the current document being viewed by the user.

However, the overall functionality and usefulness of a Web agent can only be validated by its users. Some of the agents reviewed earlier in this chapter did go past an evaluation of individual subsystems, and showed results collected from a user study. For example, WebMate[29] was evaluated by asking users to rate documents collected in the personal newspaper. Generally, users found that between 50% and 60% of the collected documents were relevant to their interests. Similarly, Watson[19] asked 10 users to submit one of their papers, and then rate the relevance of documents collected by the agent based on these papers. 80% of the users indicated that at least one of the collected documents was useful.

None of these studies attempted to compute IR-style metrics such as precision and recall, demonstrating the disconnect that still existed at the time between the Web agent and IR communities. While it is true that recall cannot be computed in practice (due to the impossibility of having full a priori knowledge of all relevant documents that *should* have been retrieved), precision values could have been easily computed from the data available to these studies, and would have nicely complemented the supplied results.

A thorough evaluation of a personalized Web agent must deal with several issues. First, an appropriate corpus must be made available to the agent. If the agent is being tested on the Web directly, we must deal with the potential lack of adequate knowledge of document relevance. Finally, the frequency and detail of training solicited from users must be taken into account. We address these issues throughout the subsequent chapters of this dissertation, but mainly in Chapter 7, where we present an overall user study of our QueryTracker Web agent.

Chapter 4

Lightweight Filtering for Web Agents

Filtering is one of the principal building blocks of our personalized Web information agent (Figure 1.2). As the agent encounters various documents on the Web, its filtering component must decide when a document is relevant to the user's interests. This chapter describes our efforts to build and optimize a filtering algorithm for use with personal Web information agents. To be consistent with our design goals outlined in Chapter 1, filtering must be fast, lightweight, and unobtrusive.

Our desired filter must only maintain an aggregate model of the ideal relevant document, and avoid storing past training samples. In other words, we are looking for a filtering algorithm capable of updating its filtering queries *incrementally*. While physical storage and disk access time have greatly improved in recent years, managing individually stored documents would add significant complexity to the design, and should therefore be avoided. Also, the filtering algorithm should preferably require only positive training samples. Requiring negative training poses a potentially significant burden on the users, requiring them to provide documents that are *non*-relevant with respect to their topic(s) of interest, and thus violating our goal of making the system user friendly. We study the theoretical implications of constructing a filtering query (or user profile), and the implications of relying on negative (in addition to positive) training in Section 2.3.

We begin by constructing a new filtering algorithm that meets all our requirements

outlined above in Section 4.1. This algorithm uses TF-IDF vectors as representation for documents, queries, and user profiles. Only positive feedback is expected, and, therefore, a dissemination threshold is associated with the vectors modeling the user profile. We perform a factorial experiment in which several parameters of the algorithm are tuned to optimize filtering performance. In Section 4.2, we implement the incremental clustering algorithms reviewed in Section 3.2.3, to allow automatic learning of users' topics of interest. This further reduces the obtrusiveness of the system by eliminating the requirement that users explicitly associate a topic with each relevant training sample they provide. This is the first empirical study on how much filtering performance is improved by using more advanced clustering algorithms over the simple greedy technique currently employed by other Web agents. We compare our TF-IDF filtering system with a Naive Bayes Classifier (NBC) in Section 4.3. NBCs require negative training samples in addition to positive ones. We examine various techniques that work around the requirement to provide explicit negative training to a NBC, as well as various scenarios in which a NBC could fit into the design of a Web information agent. In addition to the NBC, we explore a hybrid TF-IDF based algorithm which uses two vectors to represent both positive and negative document classes. Finally, we compare these algorithms and assess their relative merits and suitability for use with a Web information agent in Section 4.4.

4.1 Implementing and Tuning a TF-IDF Filter

Of all filtering algorithm classes studied in Section 2.3, TF-IDF best fits our design goals. Vectors representing filtering queries can be maintained incrementally by adding together all training document vectors supplied by the user during operation. By maintaining a dissemination threshold associated with the filtering query, we can avoid asking users for non-relevant training examples.

This section presents our implementation of such a TF-IDF based filtering system, along with an experiment used to fine-tune various parameters, such as query and threshold learning rates. Our experiments also measures how gracefully filtering performance degrades when the available training is limited.

4.1.1 Filtering Algorithm

As documents arrive from the incoming stream, they are first transformed into TF vectors by counting the number of occurrences for each distinct term. TF-IDF vectors are then obtained by weighing each term by its current IDF value (see Section 2.1).

New documents are compared against accumulated query vectors using the cosine similarity metric. The similarity between two vectors ranges between 1, when the vectors are perfectly collinear (and thus heuristically considered to be on the same topic), and 0, when the vectors are orthogonal, and have no common terms.

Filtering query vectors (maintained as part of the user profile) generalize topics of interest to a user. Our algorithm only requires *relevant* examples to construct a TF-IDF vector representing each topic (i.e., positive feedback only). In addition to this vector, each topic entry in the profile also contains a dissemination threshold, the minimum similarity between the vector of an incoming document and the topic's query vector required for disseminating the document. Dissemination thresholds range between $[0, 1]$. When a new document arrives, its vector is compared to each topic query vector; if the similarity exceeds the dissemination threshold for at least one topic, the document is shown to the user.

Filtering algorithms start with profile queries created by converting a description of each topic into a TF-IDF vector; the vectors are updated based only on feedback from documents that have been disseminated [71]. Our algorithm starts with an empty profile. Topics are maintained explicitly by the user; new topic vectors are created from the first document submitted by the user under a new topic name. The document's TF-

IDF vector forms the initial query vector for the topic. When different initial values are chosen for the dissemination threshold, differences in measurements quickly disappear after only a few (less than ten) training samples. We therefore conclude that the initial value is not important, and set it to 0.5, in the middle of its allowable range.

For existing topics, the query vector is learned using a mechanism inspired by relevance feedback [118]. The query vector is initialized to the first submitted training sample. Both the query vector and the current document vector are always normalized to eliminate the influence of document length. Before being added to the query vector, the document vector is also weighted by a query learning rate w which controls how much influence the original query vector and the new document vector will each have on the updated query.

Like the query vector, the dissemination threshold also needs to be adapted. Callan[21] proposes a dissemination threshold that starts low and grows in time to improve precision. The system presented by Zhai et al.[147, 148] features dissemination thresholds that can both increase and decrease, but the algorithm heavily relies on numerous cached documents to periodically optimize the dissemination threshold.

To keep our filtering system lightweight, we prohibit the algorithm from caching past document vectors. Keeping around old training samples introduces a storage burden[1] which might be manageable on a modern workstation, but also causes additional algorithmic complexities, such as deciding how long to keep such samples around, and whether to assign less weight to older samples than to newer ones, what weighting scheme to use, etc. Instead, our threshold tracks the similarity between the original query vector and the latest feedback document only. We can imagine the original query vector to be at the center of a "bubble", with a radius given by the dissemination threshold. If the new feedback document falls inside this bubble, we can raise the threshold

[1]QueryTracker, one of our testbed systems, encountered approx. 200 documents per topic per day.

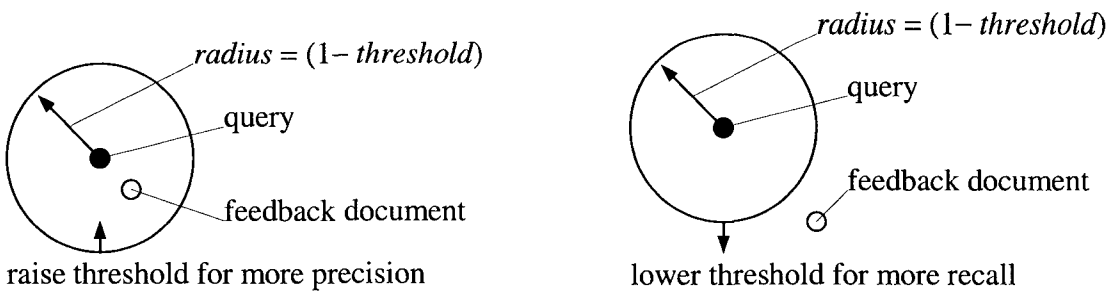


Figure 4.1: Learning the dissemination threshold

```
AddDocToTopic(doc,topic)
    s = similarity(doc.vector,topic.vector);
    normalize(doc.vector);
    normalize(topic.vector);
    topic.vector += w · doc.vector;
    normalize(topic.vector);
    topic.threshold += α · (s − topic.threshold);
```

Figure 4.2: Topic learning mechanism

(shrink the bubble), to improve precision. For feedback documents falling outside the bubble, we must lower the threshold to improve recall. This process is illustrated graphically in Figure 4.1. Unlike Callan's method, ours permits non-monotonic changes to the threshold vector. The learning algorithm, for both the query vector and the dissemination threshold, is presented in pseudocode in Figure 4.2. The pseudocode includes two critical parameters: w (query learning rate) and α (threshold learning rate).

4.1.2 Performance Evaluation

The purpose of this study is to determine the optimal combination of query and threshold learning rates, as well as the sensitivity of the filter to the amount of available training. The experiment proceeds as follows: For each new incoming document, the system predicts relevance based on the current user profile. The new document is compared to each internal topic vector, and a record is made when any of the dissemination thresholds

have been passed. Then, the simulated "user" provides feedback for relevant documents, which is simulated with the Boolean relevance judgments included with the datasets. The new document is added to all topics for which it is known to be relevant. Our independent variables are:

- w – query learning rate that controls how much influence the new document vector exerts on the updated query. Values below one favor the original query vector. Our experiments use: 0.1, 0.15, 0.2, 0.25, 0.5, 1.0, 2.0, and 4.0.
- α – threshold learning rate, with values of 0.1, 0.3, 0.5, 0.7, and 0.9.
- U_{max} – the maximum number of training samples for each topic, with values of 5, 10, 20, 30, 40, 50, 60, 70, 100, 200, and ∞ (as many as available).

We use two datasets: the Foreign Broadcasting Information Service (FBIS) collection, and the Los Angeles Times (LATIMES) collection from TREC disk #5. The FBIS and the LATIMES collections consist of 48527 and 36330 news articles respectively, classified into 194 topics which span both datasets. Because the first relevant document under a topic is used to create an entry for the topic in the user profile, we only use topics with more than one relevant document, which reduces the total number of topics to 186.

For each combination of independent variables, we computed recall, precision, their harmonic mean (HM), and the TREC-8 linear utility functions [71]

$$LF1 = 3RF - 2NF$$

$$LF2 = 3RF - NF$$

as described in Section 2.4. We compute utility functions in this experiment to ascertain that our algorithm has performance comparable to other established research in the field.

	FBIS			LATIMES		
Metric	α	w	Best Value	α	w	Best Value
Recall	0.9	0.5	0.511	0.9	0.5	0.493
Precision	0.1	0.5	0.516	0.1	1.0	0.510
HM	0.1	0.2	0.453	0.3	0.5	0.386
$LF1$	0.1	0.5	4533	0.1	1.0	1012
$LF2$	0.1	0.25	8441	0.1	0.25	2097

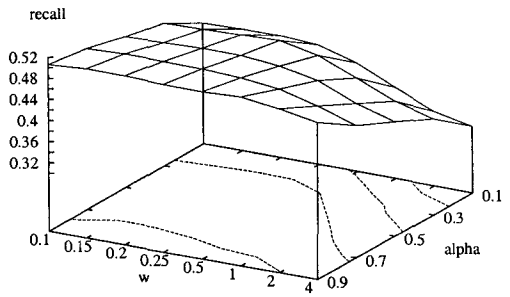
Table 4.1: Optimal parameter values for each metric

First, we test parameter influence under ideal conditions (all available feedback used, $U_{max} = \infty$). We also evaluate how a filtering algorithm optimized for the "long haul" behaves when training is limited to smaller values of U_{max}. Since individual users building profiles cannot be expected to provide thousands of training samples per topic, it is important that performance degrades gracefully with limited feedback.

4.1.2.1 Impact of Parameters α and w when $U_{max} = \infty$

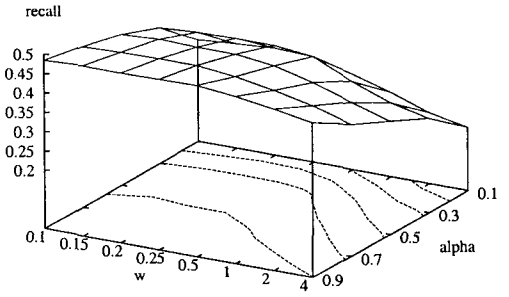
A two-way Analysis of Variance indicates that both α and w, individually and jointly, strongly influence recall ($p < 0.00001$). Recall is consistently good for values of $\alpha \geq 0.7$ and $w \leq 0.5$, but strongly decreases outside this region. A surface plot as well as an ANOVA table for each dataset are shown in Figure 4.3. Performance depends primarily on α and is close to optimal as long as w is within a range of values. The best parameter values are in Table 4.1. These values are comparable to the performance of other filtering systems evaluated at TREC-8 [71], with the caveat that different datasets were used there, and that the best coefficients for use with TREC linear utility functions are still up for debate within the filtering research community.

Precision depends strongly on α ($p < 0.00001$). Precision is highest for the smallest tested value of $\alpha = 0.1$, the most conservative update. The query learning rate w also has a significant effect on precision, albeit much smaller ($p < 0.01$). The best values for precision seem to occur for $w \in [0.25, 1.0]$. The interaction effects are negligible.



ANOVA	F	p
α	403.36	0.00001
w	142.44	0.00001
α:w	27.59	0.00001

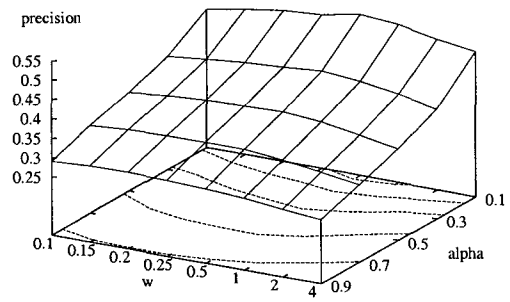
a) Dataset: FBIS



ANOVA	F	p
α	435.12	0.00001
w	96.07	0.00001
α:w	8.43	0.00626

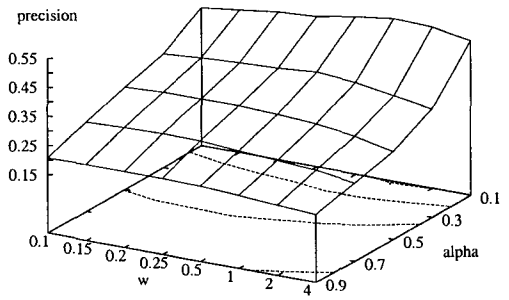
b) Dataset: LATIMES

Figure 4.3: Effects of threshold (α) and query (w) learning rates on recall



ANOVA	F	p
α	613.63	0.00001
w	9.52	0.00389
α:w	0.99	0.32598

a) Dataset: FBIS

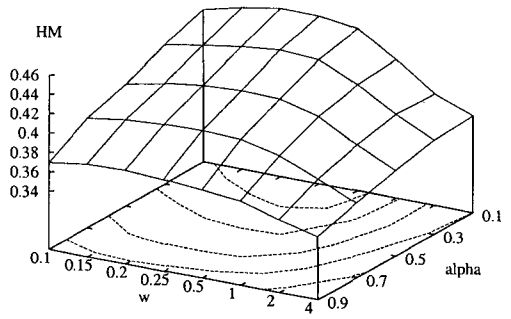


ANOVA	F	p
α	427.54	0.00001
w	3.53	0.06824
α:w	2.41	0.12896

b) Dataset: LATIMES

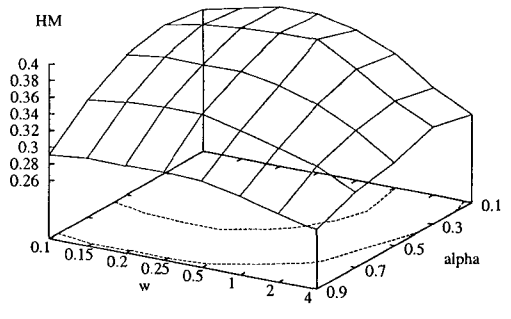
Figure 4.4: Effects of threshold (α) and query (w) learning rates on precision

The previous analysis considers recall and precision in isolation, which is clearly unrealistic for our application. The measures that encompass both recall and precision (HM, $LF1$, $LF2$) show strong effects of w and α (see Figures 4.5,4.6,4.7). HM and $LF1$ show interaction effects. HM shows best performance within the interval, $\alpha \leq 0.3$ and $w \leq 0.5$.



ANOVA	F	p
α	348.49	0.00001
w	209.67	0.00001
α:w	17.58	0.00017

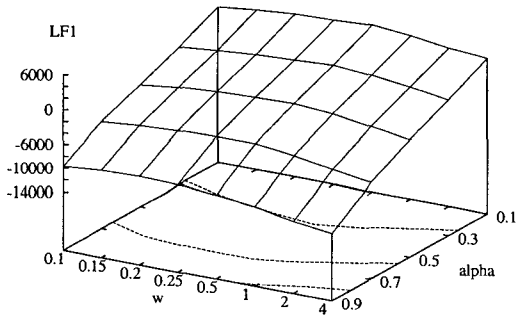
a) Dataset: FBIS



ANOVA	F	p
α	127.47	0.00001
w	99.41	0.00001
α:w	6.42	0.01574

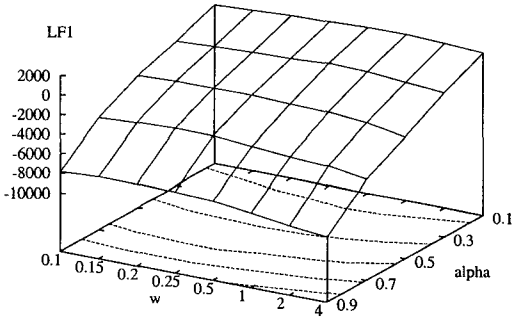
b) Dataset: LATIMES

Figure 4.5: Effects of threshold (α) and query (w) learning rates on *HM*



ANOVA	F	p
α	1736.64	0.00001
w	36.18	0.00001
α:w	6.23	0.01726

a) Dataset: FBIS

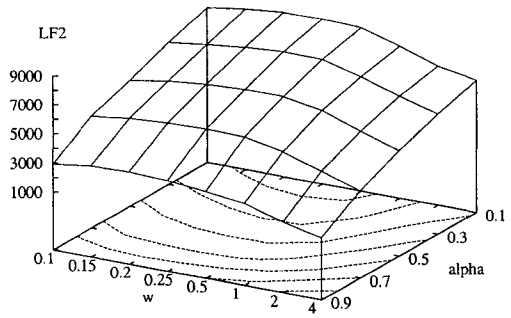


ANOVA	F	p
α	1070.65	0.00001
w	10.36	0.00272
α:w	5.06	0.03063

b) Dataset: LATIMES

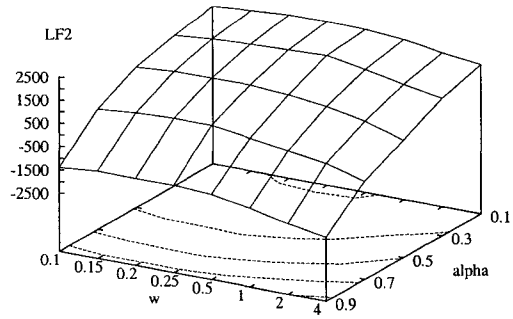
Figure 4.6: Effects of threshold (α) and query (w) learning rates on TREC8 *LF*1

In conclusion, with unlimited training, the best learning parameters for our algorithm are $w \in [0.2, 1.0]$, and $\alpha \leq 0.3$ for everything except recall, where values of $\alpha \geq 0.9$ give the best results. Fortunately for our application, there was no statistical difference between the two datasets; thus, we would expect our best parameter settings to generalize to other data sets as well.



ANOVA	F	p
α	643.15	0.00001
w	109.24	0.00001
α:w	0.11	0.73996

a) Dataset: FBIS



ANOVA	F	p
α	426.38	0.00001
w	29.97	0.00001
α:w	0.99	0.32714

b) Dataset: LATIMES

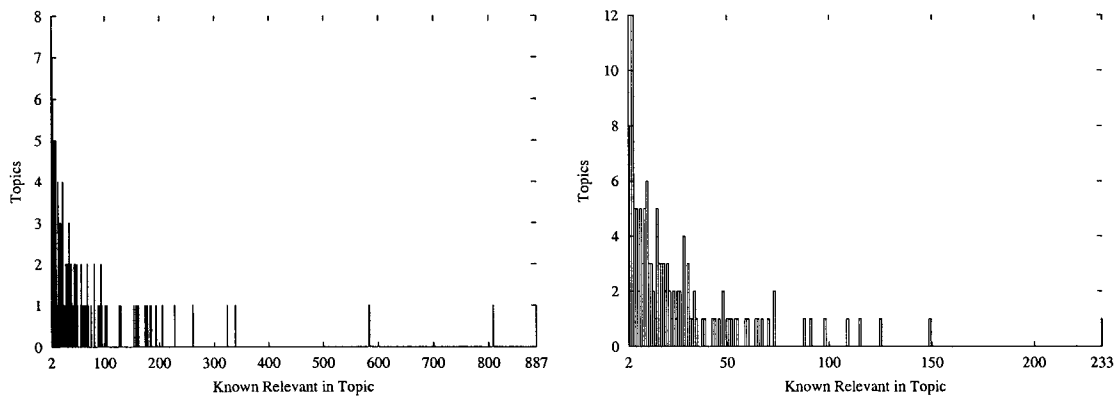
Figure 4.7: Effects of threshold (α) and query (w) learning rates on TREC8 $LF2$

Having established that the performance of our lightweight algorithm is comparable to other existing research in the field, we continue our subsequent studies by focusing on recall, precision, and HM, as they are more established and their meaning is better understood.

4.1.2.2 Robustness with Less Feedback (U_{max})

The two datasets vary wildly in the number of relevant documents per topic. Histograms of the distribution of topics according to the number of known relevant documents they contain are given in Figure 4.8. To determine the effects of limiting training, we had to focus on topics for which many documents were available. We used the topics in the fourth quartile of the distributions in Figure 4.8: this includes 36 topics from the FBIS dataset, with 75 or more known relevant documents, and 36 topics from the LATIMES dataset, with 31 or more known relevant documents. These topics include the majority of known relevant documents in our datasets: 7,678 for FBIS, and 2,368 for LATIMES.

We plotted recall, precision, and HM against U_{max} for each combination of α and



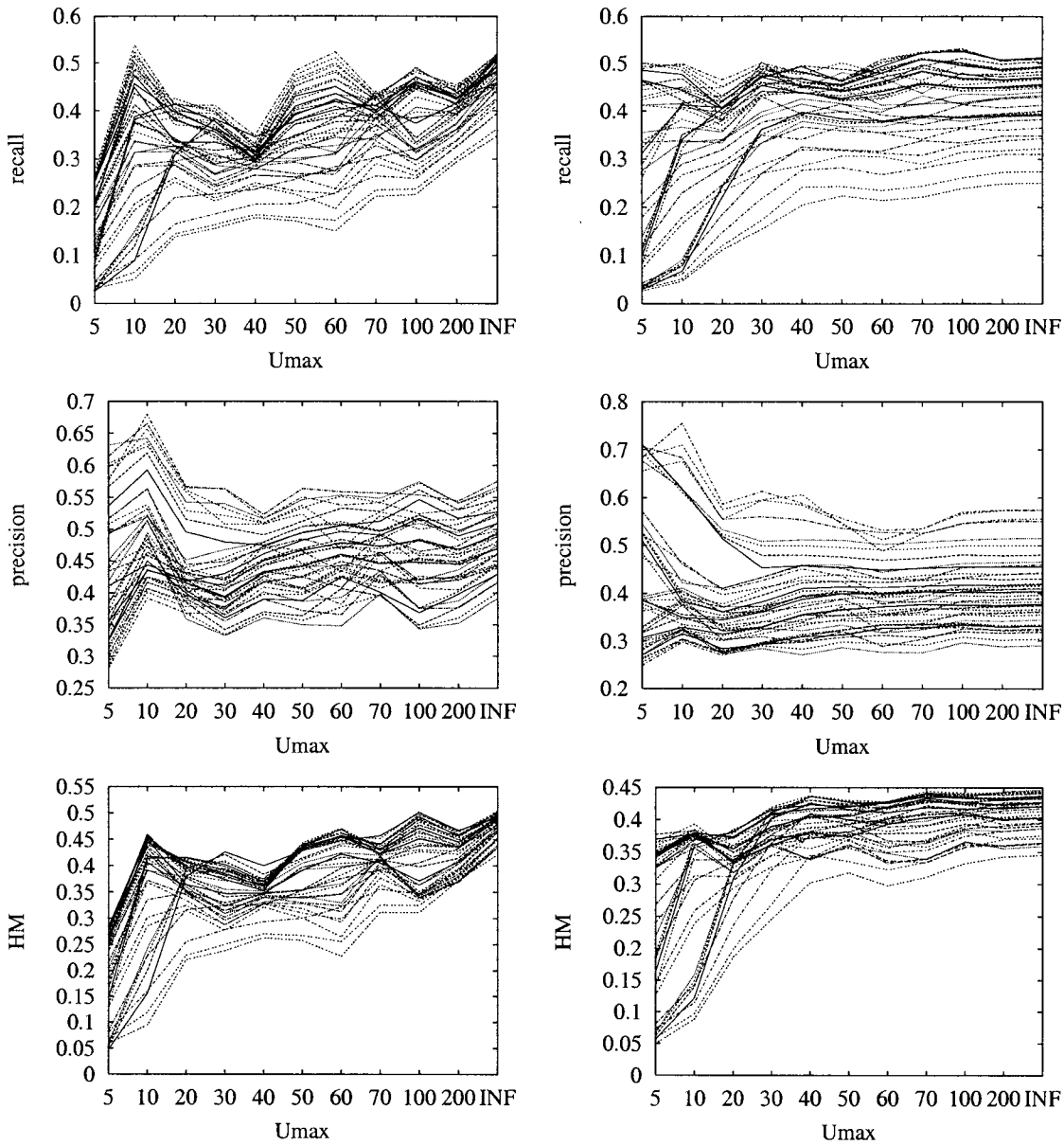
a) Dataset: FBIS b) Dataset: LATIMES

Figure 4.8: Distribution of known relevant documents across topics

w in Figure 4.9. These plots show that the interaction effects between U_{max} and the two parameters are small for both datasets. Additionally, the effect of U_{max} is non-monotonic, and, while no parameter combination dominates all values of U_{max}, some are consistently among the top few.

ANOVA tests indicate that U_{max} strongly influences recall: $F = 128.34, p < 0.00001$ for FBIS, and $F = 36.30, p < 0.00001$ for LATIMES. In comparison, the influence on precision is significant only on the FBIS dataset ($F = 47.36, p < 0.00001$ for FBIS, $F = 1.02, p < 0.31$ for LATIMES). The U_{max} on HM is significant for both datasets: $F = 100.59, p < 0.00001$ for FBIS, and $F = 31.07, p < 0.00001$ for LATIMES. In contrast to the unlimited updates, limited updates favor high values of α, while w remains in the same range. This result follows from the need to make the most out of the available data.

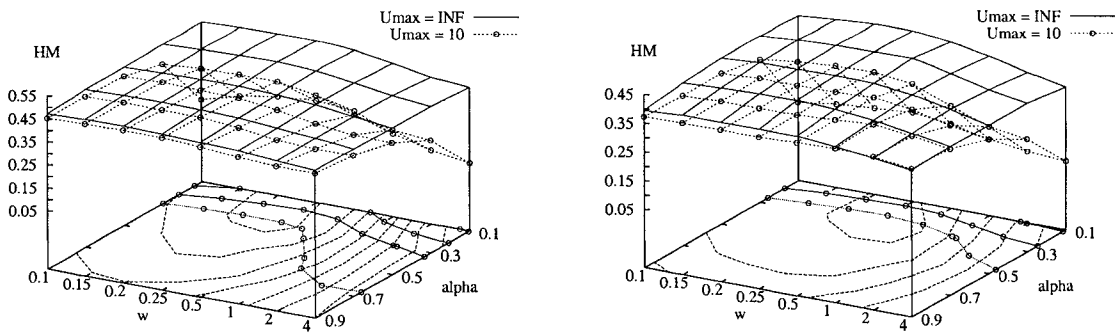
Interestingly, our plots show that very good results can be obtained after as few as 10 updates per topic. After 10 updates per topic for FBIS, we reached 0.4489 recall and 0.4501 precision, resulting in 0.4495 HM at $\alpha = 0.9$ and $w = 0.5$ (99% of the value at $U_{max} = \infty$); LATIMES showed 0.4055 recall, 0.3818 precision, and HM of 0.3933 (102% of the value at $U_{max} = \infty$ in Table 4.1, which suggests a slight overfitting when



a) Dataset: FBIS b) Dataset: LATIMES

Figure 4.9: Influence of U_{max} on recall, precision, and HM for combinations of α and w. These plots are intended to convey general, common trends across the studied topics.

more training is supplied), at $\alpha = 0.7$ and $w = 1.0$. As an example, Figure 4.10 displays the surface plots of HM at $U_{max} = 10$ and $U_{max} = \infty$. Both data sets exhibited similar performance, and therefore we expect these results to generalize for other data as well.



a) Dataset: FBIS b) Dataset: LATIMES

Figure 4.10: Influence of U_{max} on HM for the FBIS dataset

Performance seems robust within a relatively wide interval of parameter values. Importantly for our target application, it turns out that the algorithm is robust against limited training information; we observe a slow degradation in performance as the amount of feedback was reduced down to 10. When used in a Web information agent, we expect our algorithm to be trained with up to 10 or 20 positive examples per topic. In consequence, the algorithm must start out with a large threshold learning rate.

4.2 Automatically Learning Implicit Topics

The filtering algorithm introduced in the previous section does not deal with the management of user's topics of interest. It defaults to the most straightforward option, which is to expect users to completely control the creation and maintenance of their topics. Each time the user provides a training document, it must be accompanied by the identifier of a topic for which it is relevant. This topic will be updated if it exists already, or otherwise newly created.

Unfortunately, the maintenance of a user profile containing explicit topics can become burdensome, and therefore violates our design requirement for unobtrusiveness. Therefore, in a departure from the methods employed for text filtering, authors of per-

sonal information agents usually choose to automate the maintenance of the user profile. Under this paradigm, there is no longer a strict mapping between the components of the user profile and a user's designated topics of interest. For example, WebMate[29] only requires the user to indicate when a document is interesting, without the need to provide a specific topic. An incremental clustering algorithm is used to automatically select the component of the user profile (pseudo-topic) that will receive the update.

In this section, we implement two incremental clustering algorithms proposed by Charikar et al.[26] (described in Section 3.2.3): Greedy and Doubling. We use these algorithms with our filtering system to automatically learn and maintain users' topics implicitly, and compare the performance tradeoffs between the original and modified filtering systems.

4.2.1 Incremental Clustering Methods

Incremental clustering algorithms typically set an upper limit k on the number of allowed clusters. Periodically, clusters must be merged to maintain the established limit. The merging operation acknowledges that two clusters may have moved close enough together to be better captured as a single generalization. Established clustering algorithms usually assume that clusters are represented by their center of mass and a diameter. When clusters are merged, the new center of mass is computed from all points now contained in the resulting cluster, and the diameter is set to the distance between the two points in the cluster that are the farthest apart.

In order to adapt this principle to TF-IDF vectors, when two clusters are merged in our implementation, the query vector becomes the sum of the two original query vectors, weighted by their mass (the number of points contained in each original cluster). The same is true for the dissemination threshold, which is set to the weighted sum of the dissemination thresholds of the original clusters. This operation is described in Figure 4.11. Note that if the second cluster involved in the merging operation consists

```
MergeClusters(cl1, cl2)
    if(cl2_npoints) is 1 then
      AddDocToCluster(cl2, cl1);
      return(cl1);
    else
      normalize(cl1_vector);
      normalize(cl2_vector);
      newcl_vector = cl1_npoints · cl1_vector+
                     cl2_npoints · cl2_vector;
      normalize(newcl_vector);
      newcl_threshold = (cl1_npoints · cl1_threshold+
                         cl2_npoints · cl2_threshold)/
                        (cl1_npoints + cl2_npoints);
      newcl_α = (cl1_npoints · cl1_α+
                 cl2_npoints · cl2_α)/
                (cl1_npoints + cl2_npoints);
      newcl_npoints = cl1_npoints + cl2_npoints;
      return(newcl);
```

Figure 4.11: Merging two clusters

of a single point, we consider this to be a document addition to the first cluster. The merging thus defaults to the AddDocToCluster procedure from Figure 4.2, described in the previous section.

The following paragraphs describe the three filtering systems in this study: Explicit Topic Feedback (XPL), using the Greedy Algorithm (GRD – standard practice with other existing Web agents), and the Doubling Algorithm (DBL – theoretically shown to outperform DBL).

Explicit Topic Feedback (XPL). Our baseline is given by the unmodified filtering algorithm described in the previous section. Explicit topic IDs accompany each (positive only) training sample. One cluster is maintained for each distinct topic ID and is initialized to the TF-IDF vector of the first training sample provided for that topic. Subsequent documents are added to the topic vector with a query learning rate w, and the dissemination threshold (initialized to 0.5) is updated with a learning rate α. This method serves

```
FeedbackGRD(document)
    cluster[k+1] = newcluster(document);
    (i, j) = closest two clusters, with i < j;
    newcl = MergeClusters(cluster[i], cluster[j]);
    delete(cluster[i]);
    delete(cluster[j]);
    insert(newcl);
```

Figure 4.12: Agent feedback using greedy incremental clustering

as an example for the best performance that could be achieved, even though we do not view it as ultimately viable for Web information agents, due to its excessive requirement for user interaction. Pseudocode for this algorithm was given in Figure 4.2.

Greedy Algorithm (GRD). This algorithm initially assigns each of the first k documents offered as feedback to its own cluster. Once k clusters have been formed, each new document is treated as the $(k+1)$st cluster, and the algorithm then proceeds to merge the closest two clusters in order to bring the number of clusters back to k. Thus a new document may be placed in an existing cluster or may form its own cluster forcing existing clusters to merge. The pseudocode for this algorithm is in Figure 4.12. This is the simplest of the two incremental clustering algorithms in our study, and captures the current practice in personal WWW agents (e.g., WebMate [29]).

Doubling Algorithm (DBL). As with greedy clustering, initially, each feedback document, up to the kth, is placed in a separate cluster. Once the limit of k clusters is reached, the algorithm enters a merging stage, during which multiple clusters may be combined together. Merging stages alternate with update stages, during which new training samples are either added to existing clusters, or lead to the creation of new ones, eventually forcing a new merging stage.

Merging is based on a distance threshold d, which is periodically increased according to two parameters, a and b. When the number of clusters first exceeds k, d is initial-

```
FeedbackDBL(doc)
    i = closest cluster to doc_vector;
    if dist(cluster[i]_vector, doc_vector) ≤ a · d then
      AddDocToCluster(doc, cluster[i]);
    else
      insert newcluster(doc);
    while number of clusters > k do merging stage :
      d = b · d;
      build graph G: for all pairs (i, j) do
        if dist(cluster[i]_vector, cluster[j]_vector) < d
        then connect i and j;
      repeat
        pick arbitrary node n in G;
        foreach neighbor m of n do :
            MergeClusters(n, m);
            disconnect node m from G;
        disconnect node n from G;
      until empty(G);
```

Figure 4.13: Agent feedback using doubling incremental clustering

ized to b times the distance between the closest two clusters. The subsequent merging stage combines groups of clusters that are separated by a distance less than the current value of d.

During an update stage, a training document is added to an existing cluster if the distance between their TF-IDF vectors is less than $a \cdot d$, or else it forms a new cluster. After each feedback document is inserted, the algorithm checks whether the number of clusters exceeds k, and, if so, executes another merging stage. Before each merging stage, d is multiplied by b. Typically, a and b are set to 2, hence the name *doubling* algorithm. The pseudocode for this procedure is presented in Figure 4.13. Because the threshold is b times the known minimum, several pairs of clusters may be merged during the process.

The doubling algorithm was originally developed for points in a metric space [26], and therefore it assumes that the distance between two points can take values anywhere

in the interval $[0,\infty)$. However, for IR applications such as this, we compute similarity between two TF-IDF vectors using the cosine measure, which ranges from 0 for dissimilar to 1 for a perfect match. Therefore, for our application, we convert similarity into distance using the following formula:

$$\text{dist}(vector1, vector2) = \frac{1}{\text{similarity}(vector1, vector2)} - 1$$

which maps a similarity of 1 to a distance of 0, and a similarity of 0 to a distance of ∞.

This algorithm has been theoretically proven [26] to perform better than the greedy algorithm (by building tighter clusters). The metric used was the *performance ratio* of the incremental algorithm, which is the ratio of its maximum cluster diameter to that of the optimal clustering for a given set of points. The doubling algorithm is proven to have a performance ratio of 8, whereas the greedy algorithm has a performance ratio of $(2k-1)$, which degrades with the number of clusters. Thus, we include the doubling algorithm in the set because it would appear to offer a superior alternative to current practice.

4.2.2 Empirical Evaluation of Incremental Clustering

Our study is intended to answer the following questions:

- Does filtering performance suffer when the topic information associated with training documents is unavailable ?
- Which of the two proposed incremental algorithms (GRD or DBL) performs better in practice, and how can it be optimized?
- How does performance degrade when the amount of feedback is limited?

We present a factorial experiment for each of the three algorithms described above (XPL, GRD, and DBL), computing recall, precision, and *HM*. The following independent variables are used:

- Datasets: FBIS and LATIMES

- Query learning rate w: the weighting factor by which document vectors are scaled before being added to a cluster's query vector. Sub-unit values of w are used to assign more weight to the original cluster's query vector, whereas values greater than one are used to favor the new document vector. This parameter controls the relevance feedback process.

- Threshold learning rate α and decay factor δ: XPL, GRD, and DBL all feature dissemination thresholds that track the cluster's similarity to the latest feedback document with a decaying learning rate. Both α and δ are allowed values in the interval $[0, 1]$. Using information from earlier pilot experiments we decided to use $\alpha = 1.0$ when varying δ, and $\delta = 1.0$ when varying α (i.e., it only makes sense to start with a high α when using decay).

- Maximum number of allowed clusters k: applicable to the incremental clustering cases (GRD and DBL) only.

- Doubling algorithm settings: a, b: Parameters only applicable to the doubling algorithm (see Figure 4.13).

In addition to the variables used previously for our filtering experiment (see Section 4.1), our threshold learning rate α is now allowed to decay with a factor δ. We also added k, the maximum number of allowed clusters for GRD and DBL, as well as doubling parameters a and b for DBL. Value sets for these parameters change as we move from one experiment to the next, focusing on value ranges which are found to be more promising with respect to performance.

4.2.2.1 Explicit Topics vs. Incremental Clustering

Our first question is whether automatically learning topics using incremental clustering is detrimental to filtering performance as compared to XPL, where clusters are provided explicitly by the user. While convenient, incremental clustering has the potential to negatively impact the agent's filtering performance. To find out just how much performance is lost, we compare our explicit-topic algorithm (XPL) with GRD and DBL. Our independent variables (introduced above) are given the following values:

- Datasets: FBIS and LATIMES.
- $w \in \{0.1, 0.15, 0.2, 0.25, 0.5, 1, 2, 4\}$.
- $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ with $\delta = 1.0$, and $\alpha = 1.0$ with $\delta \in \{0.1, 0.5, 0.9\}$.
- For GRD and DBL, $k \in \{10, 20, 30, 40, 50\}$. Since topics are provided explicitly by the user, k is not applicable to XPL.
- For DBL only, $a, b \in \{1.5, 2, 3\}$.

When comparing XPL against GRD and DBL, which both use incremental clustering, we compute XPL's metrics the same way as if it had also been an incremental clustering algorithm: A document is added to *RF* if it was judged relevant to *any* topic and XPL found it relevant under *any* topic, even if the topics don't match. If a document judged relevant under any topic is not found relevant by XPL under at least one topic, *RM* is incremented instead. *NF* is incremented when a document with all non-relevant judgments is found relevant by XPL under any topic.

When measuring *HM* for each of the three algorithms, across all combinations of the other parameters, we obtain the results given in Figure 4.14. We use two-sample t-tests to compare the algorithms against each other. As expected, XPL is better than both GRD ($t = 3.9425$, $p \leq 0.0001$) and DBL ($t = 7.0205$, $p \leq 0.0001$). Surprisingly,

Algorithm	Best HM	Mean HM	Std. Dev.
XPL	0.4586	0.2932	0.0817
GRD	0.3726	0.2652	0.0714
DBL	0.4257	0.2062	0.1397

Figure 4.14: Best and mean HM, and standard deviation for XPL, GRD, and DBL

GRD seems to have better average performance than DBL ($t = 10.546$, $p \leq 0.0001$). However, DBL has a best value that comes very close to the best value of XPL, and a much higher variance than both of the other algorithms. We suspect this variance to be caused by our choice of values for the doubling parameters a and b. A two-way ANOVA with a and b as the independent variables, and HM as the dependent, indicates that both a and b have strong influence on HM: $F = 44.629$ for a, $F = 4698.292$ for b, and $F = 41.789$ for their interaction effect, all three with $p \leq 0.0001$. Looking at the box plot of HM versus a and b in Figure 4.15, we observe that runs with $b = 1.5$ cause a steep degradation in performance.

Once all runs with $b = 1.5$ are discarded, the HM results for DBL have a mean of $HM = 0.2940$ and $\sigma = 0.0714$. The average is slightly better and the variance tighter than even those of XPL, while the samples are statistically indistinguishable ($t = 0.1178$, $p = 0.9063$). Without being hampered by a bad parameter setting ($b = 1.5$), DBL clearly beats GRD ($t = 9.0205$, $p \leq 0.0001$). Also, re-running the two-way ANOVA shows that neither a nor b have a significant effect on HM.

We now have empirical verification of the theory presented by Charikar et al. [26], that DBL clearly outperforms GRD in terms of clustering, leading to better filtering performance. In some situations, the performance obtained by DBL can even equal that of XPL.

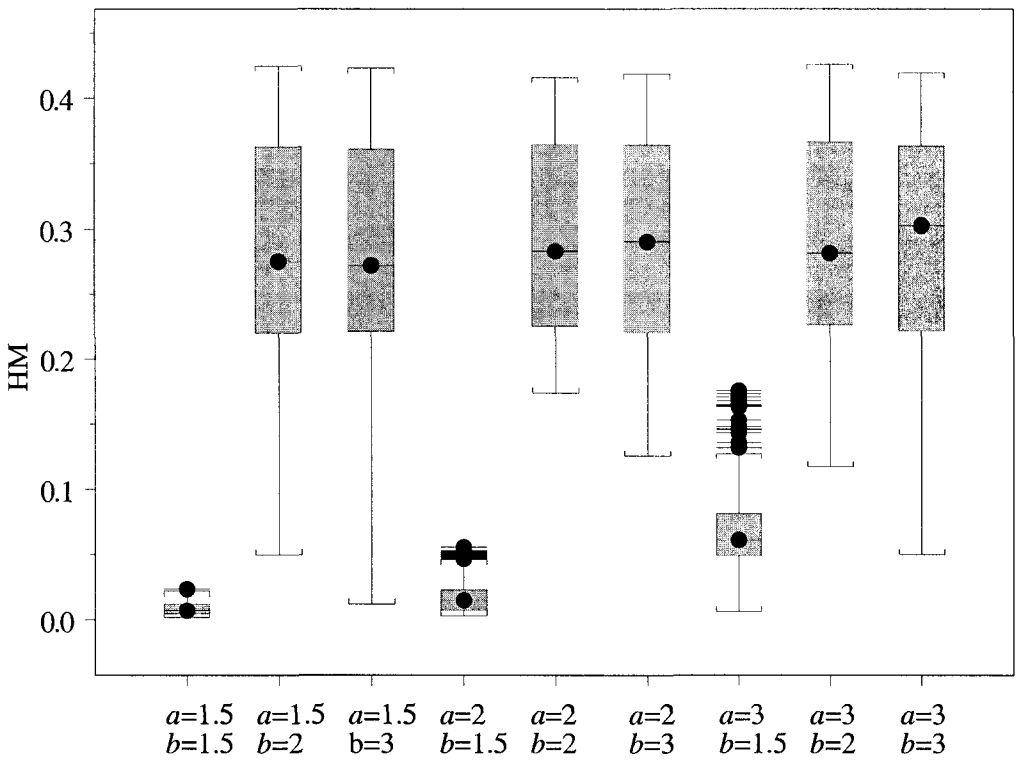


Figure 4.15: Influence of the Doubling algorithm (DBL) parameters a and b on HM

4.2.2.2 Optimizing DBL

So far, we have shown that DBL's performance approaches that of XPL, *without* asking users to provide explicit topic labels for feedback documents. We now proceed to find the optimal settings DBL's parameters (w, α, k, a, and b).

First, we split our dataset on DBL into two components: one for decaying α (i.e., α is always initialized to 1.0 and we vary the decay δ), and another for non-decaying α (i.e., we vary α and keep δ constant at 1.0). A two-sample t-test on the two subsets ($t = 3.023$, $p \leq 0.01$) shows that runs with non-decaying α have a slightly better average $HM = 0.2968$ and less variance $\sigma = 0.0723$ than runs with decaying α (average $HM = 0.2892$, $\sigma = 0.0794$). However, the best run is obtained using decaying α ($HM = 0.4257$); the best run using fixed α has $HM = 0.4189$. A box plot of HM for each of the two subsets is given in Figure 4.16.

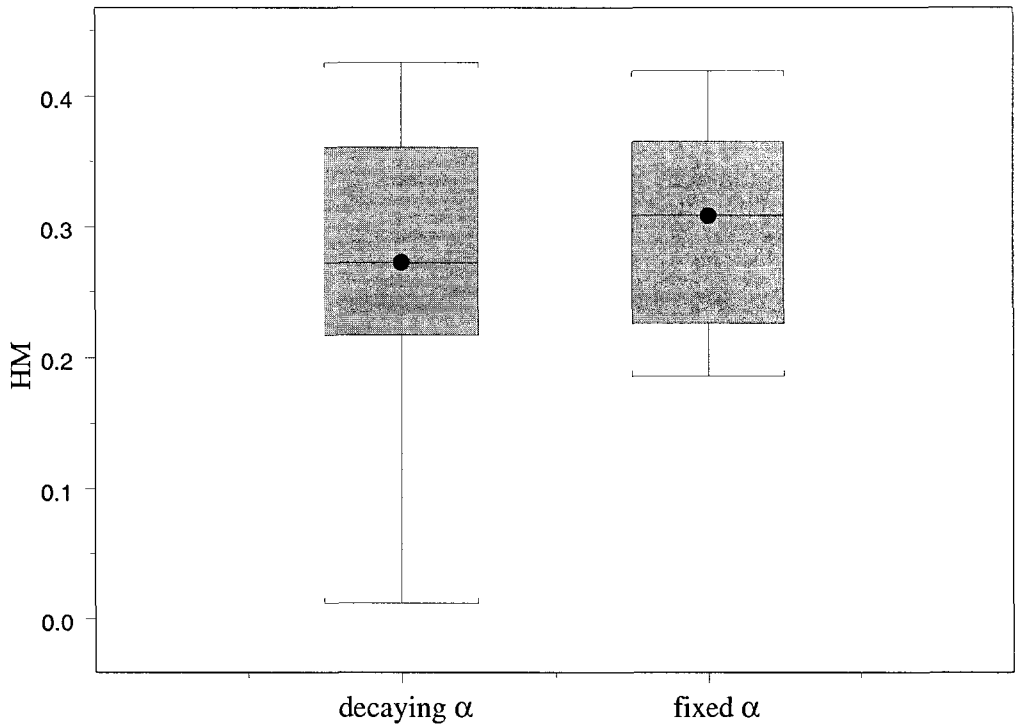


Figure 4.16: Box plots of DBL's *HM*, comparing performance of decaying vs. fixed α

Using ANOVA studies, we find that k has a strong individual effect (but no interaction effects) on *HM*, both for decaying α ($F = 21.4027$, $p \leq 0.0001$), and for fixed α ($F = 23.3866$, $p \leq 0.0001$). A graphical representation of these effects is given in Figure 4.17. From both graphs, it appears that an upper limit of $k = 30$ clusters seems to be enough to optimize DBL's performance. This emphasizes an added benefit of DBL over XPL: in addition to not requiring explicit topics from the user, DBL reduces memory use, by tracking up to 30 clusters at a time, rather than 194 explicit topics.

When using decaying α, the decay δ has a very strong individual effect on *HM* ($F = 93.64454$, $p \leq 0.0001$), and also an interaction effect together with the query learning rate w ($F = 7.53013$, $p < 0.01$). The individual effect of w on *HM* is almost negligible ($F = 3.62280$, $p = 0.0572$). We plot the influence of δ and w on *HM* in Figure 4.18. Evidently, slow decay ($\delta = 0.9$) is better, and w should be between 0.2 and 0.5.

When fixed α is used, both α and w have significant individual effects on *HM*:

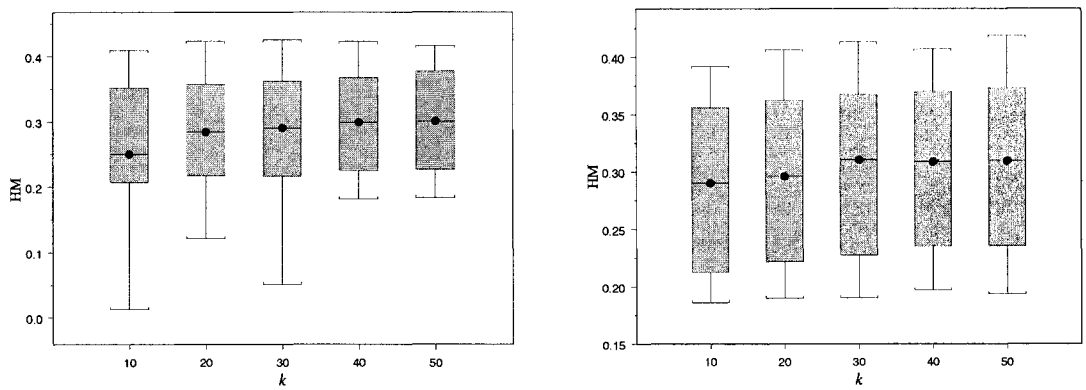


Figure 4.17: Effects of k on DBL's HM with decaying α (a), and fixed α (b)

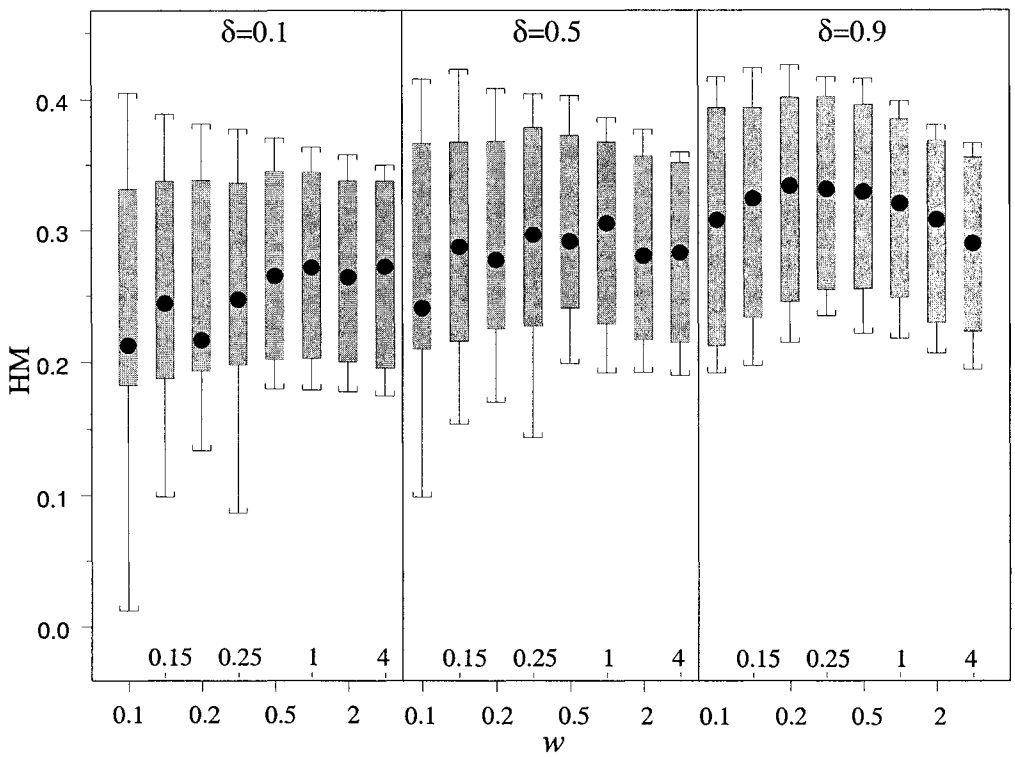
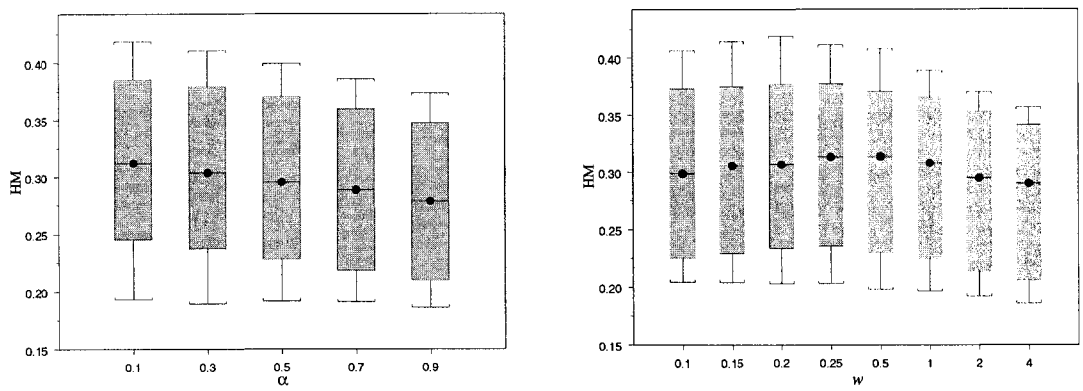


Figure 4.18: Influence of decay δ and query learning rate w on DBL's *HM*

$F = 75.3278$, $p \leq 0.0001$ for α, and $F = 48.9528$, $p \leq 0.0001$ for w. No significant interaction effects could be found. Plots of this behavior are available in Figure 4.19. Small α and w between 0.2 and 0.5 lead to the best performance.

In conclusion, when using DBL on filtering problems with a relatively large number



a) b)

Figure 4.19: Effects of α (a) and w (b) on DBL's HM

of topics (we analyze DBL's behavior for a smaller set of topics in our next experiment), a small fixed α or a slowly decaying one are preferable. The query learning rate w should have values between 0.2 and 0.5. Our best result with DBL was $HM = 0.4257$, obtained with initial $\alpha = 1.0$, $\delta = 0.9$, $w = 0.2$, $a = 3$, $b = 2$, at $k = 30$ on the FBIS dataset. Our best non-decay result was $HM = 0.4189$, obtained with $\alpha = 0.1$, $\delta = 1.0$, $w = 0.2$, $a = 3$, $b = 3$, at $k = 50$, also on FBIS.

4.2.2.3 Influence of Limited Feedback on DBL

In our previous experiment we optimized the performance of DBL filtering assuming feedback after every disseminated relevant document. While significantly less obtrusive than XPL, we would like the filtering component of our Web agent to perform well after a limited number of relevant training samples. We first test the behavior of DBL under limited feedback in the presence of the entire dataset, and then proceed with a small subset of topics in an attempt to simulate the behavior of a single user.

Limiting the Total Amount of Feedback on the Full Dataset. First, we examine the effects of limiting the total number of feedback instances on performance for the entire dataset. To save time, we restrict the range of parameters to known good values, as determined in the previous subsection. An extra parameter U_{max} controls the amount of

feedback updates received by the algorithm. Since relevant documents on all monitored topics are scattered uniformly throughout the document collections, we simply limit the total number of documents which can be supplied as training examples to the algorithm, and expect that subsequent documents be identified correctly without further training. The parameter values used in this experiment are:

- Datasets: FBIS and LATIMES.
- $w \in \{0.2, 0.25, 0.5\}$.
- $\alpha = 0.1$ with $\delta = 1.0$, and $\alpha = 1.0$ with $\delta = 0.9$.
- $k \in \{20, 30, 40, 50\}$.
- $a \in \{1.5, 2, 3\}, b \in \{2, 3\}$.
- $U_{max} \in \{500, 1000, 1500, 2000, 2500, 3000, 4000, 5000, 6000, 7000, 8000, 9000\}$; Since LATIMES has 3297 relevant documents, any value of U_{max} of 4000 or more amounts to unlimited feedback. Same goes for FBIS and its 8544 relevant documents for $U_{max} = 9000$.

When only the best combinations are used for decaying and fixed α respectively, the decaying version scores slightly better ($t = 4.8044$ at $p \leq 0.0001$) at mean $HM = 0.3050$, maximum $HM = 0.4257$, and $\sigma = 0.0706$ than the fixed α version, at mean $HM = 0.2917$, maximum $HM = 0.4198$, and $\sigma = 0.0740$. In addition, an ANOVA study with HM as the dependent shows strong influences from w ($F = 18.5182$, $p \leq 0.0001$), and k ($F = 12.1850$, $p \leq 0.001$). The best values for these parameters (which optimize HM) are $w = 2$ and $k = 30$.

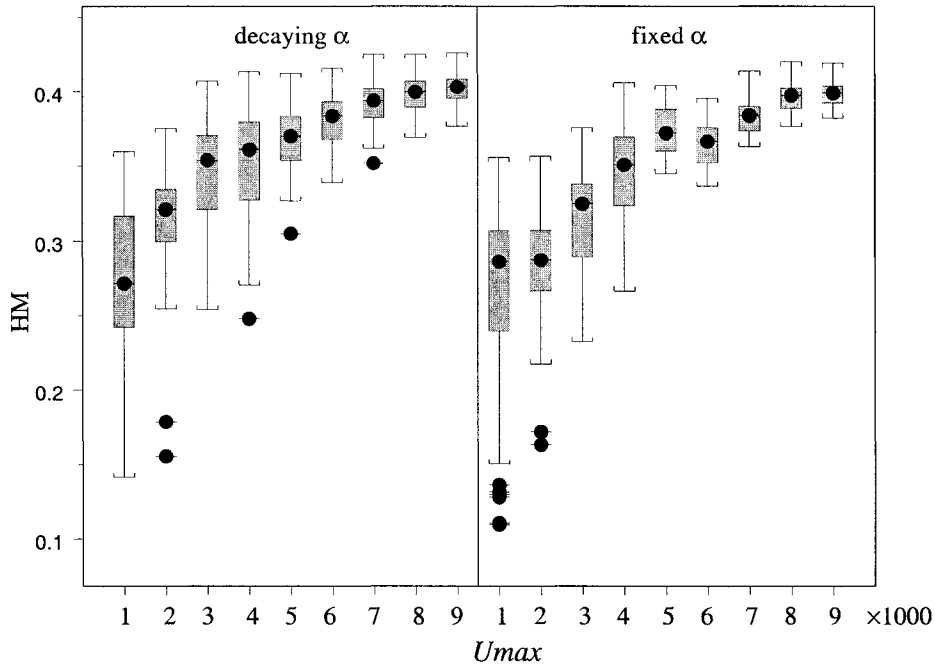
When restricting our focus to the FBIS dataset (it has more than twice the number of known relevant documents present in LATIMES), U_{max} is found to have both a direct influence on HM ($F = 2766.415$, $p \leq 0.0001$), as well as a strong interaction effect

with the threshold learning mechanism ($F = 68.493$, $p \leq 0.0001$). The boxplots of HM for each value of U_{max} and each of the two mechanisms for learning the dissemination threshold are presented in Figure 4.20. It can be observed that for decaying α, performance decreases gracefully when the amount of feedback given by U_{max} is decreased. For LATIMES, the situation is similar: $F = 1405.667$, $p \leq 0.0001$ for the direct effect of U_{max} on HM, and $F = 42.478$, $p \leq 0.0001$ for the interaction effect of U_{max} and the threshold learning mechanism.

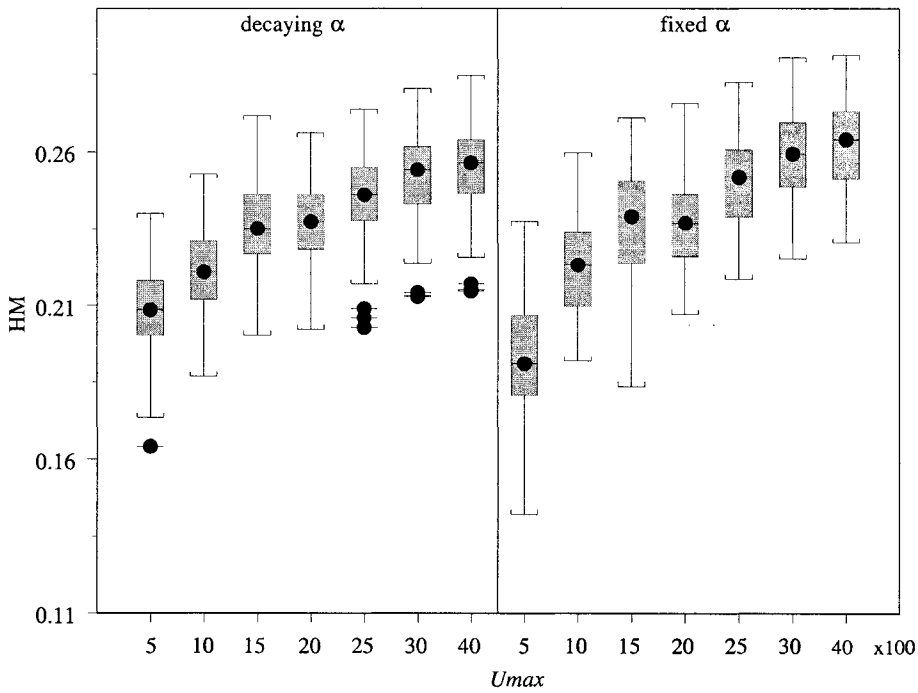
From Figure 4.20 we observe that HM increases approximately logarithmically with the amount of feedback. For instance, with decaying α on FBIS, the average HM obtained at $U_{max} = 3000$ is cca. 90% of the average HM at $U_{max} = \infty$. More importantly, best HM values degrade even more slowly when U_{max} is limited.

Per-Topic Limits in Simulated Single User Study. For our second study, we simulate the behavior of a single user, who is expected to have a smaller number of topics of interest. We picked nine topics randomly from the FBIS dataset (which contains a total of 194 topics), with the only requirement that the number of relevant documents be between 90 and 110 for each topic. We simulate a user who would be interested in these topics, willing to provide a limited amount of feedback for each topic, and then expect the agent to find the remaining relevant documents. Our "user" is aware of the nine topics and is only willing to give feedback to the agent on the first U_{max} documents encountered on each individual topic. Internally, DBL is still free to maintain clusters the way it sees fit. The parameter values used in this study are:

- Datasets: subset of 9 topics selected from FBIS (with 90 to 110 known relevant documents each; known non-relevant documents were used in the computation of the results, but not in the selection of the dataset).
- $w \in \{0.2, 0.25, 0.5\}$.
- $\alpha = 0.1$ with $\delta = 1.0$, and $\alpha = 1.0$ with $\delta = 0.9$.



FBIS



LATIMES

Figure 4.20: DBL's HM by α learning mechanism and by amount of feedback U_{max}

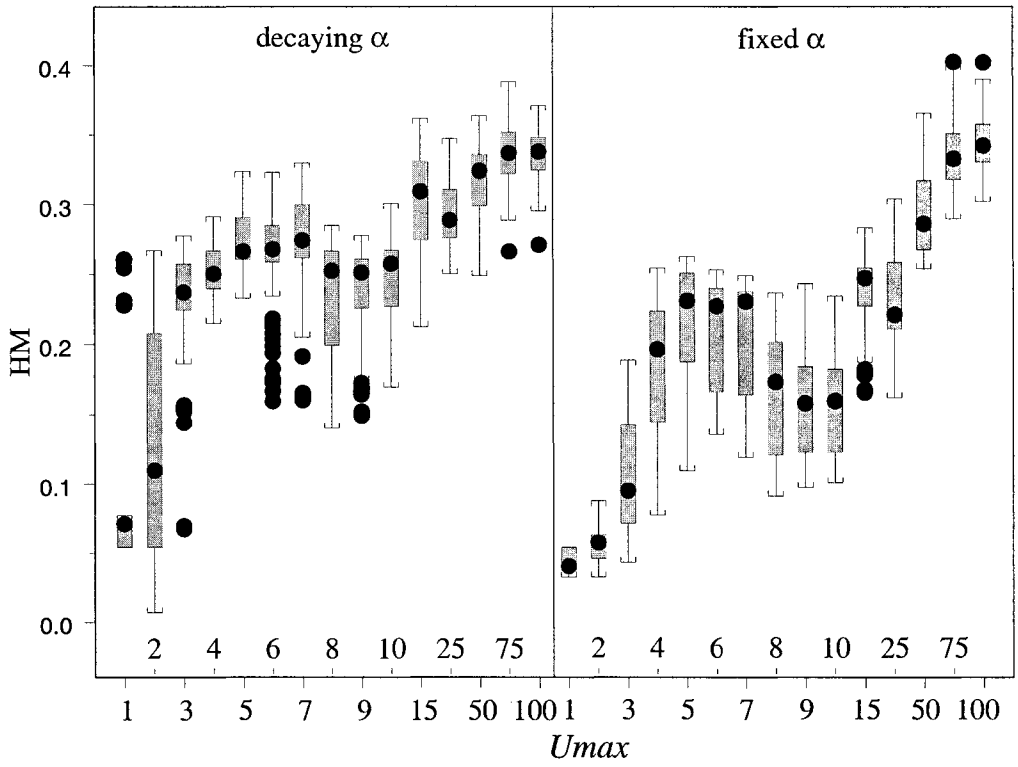


Figure 4.21: Effects of the learning mechanism and U_{max} on DBL's HM (single user study)

- $k \in \{4,5,6,7,8,10,15\}$.
- $a \in \{1.5,2,3\}, b \in \{2,3\}$.
- $U_{max} \in \{1,2,3,4,5,6,7,8,9,10,15,25,50,75,100\}$.

We observe that decaying α favors moderate amounts of feedback, whereas extensive feedback works best with fixed α. Also, fewer clusters work best with moderate feedback, whereas more clusters help optimize performance with extensive feedback.

Using ANOVAs with *HM* as the dependent variable, we observe that the parameters with strongest effects are the learning mechanism ($F = 832.327$), query learning rate w ($F = 72.658$), cluster limit k ($F = 30.586$), and the amount of feedback U_{max} ($F = 3201.379$), all at $p \leq 0.0001$. There are also interaction effects between U_{max} and the learning mechanism, and between U_{max} and the cluster limit k. As with our previous

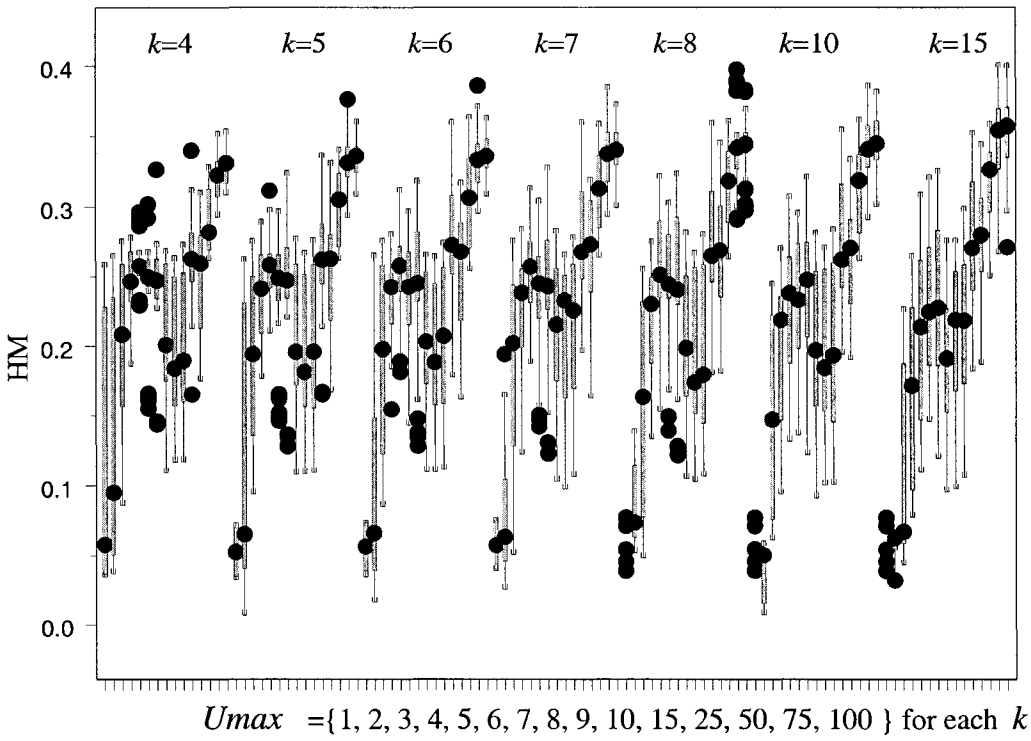


Figure 4.22: Effects of the cluster limit k and U_{max} on DBL's HM (single user study)

experiments, the optimal query learning rate is $w = 0.2$. We present box plots of the effects on HM of the combination between U_{max} and the learning mechanism in Figure 4.21 and of the combination between U_{max} and k in Figure 4.22. The discontinuities present in the graphs are most likely caused by topic drift and/or overtraining, followed by an adaptation of the user profile if more training is made available.

We conclude that an incremental clustering algorithm of good quality (such as DBL – the doubling algorithm) can learn topics automatically such as to yield comparable performance to an explicit algorithm where users provide the topics. Thus, the convenience of using a Web information agent may be significantly improved. Also, after analyzing the effect of various parameters on the performance of DBL, we find several parameter combinations that optimize filtering performance as measured by HM, the harmonic mean of *precision* and *recall*. Last, but not least, HM performance degrades gracefully when reducing the amount of training samples supplied to DBL.

4.3 Advantages of Negative Feedback

We started out by using TF-IDF vectors to represent user profiles, because, in combination with a dissemination threshold, they allow the user to avoid having to provide negative feedback (see Figure 2.1a). How much could performance be improved if restrictions against negative feedback were lifted? We compare our TF-IDF filter presented earlier [129] against the Naive Bayes Classifier (NBC) with Expectation Maximization (EM) [101], which is another popular method used in both IR/filtering and machine learning, and which, by design, requires both positive and negative training samples (see Figure 2.1b).

We also present a new method that is a variant on NBC. Inspired by the method for automatic selection of pseudo-negative training samples used by Liu et al.[86] (see Section 2.3.2.2), our version does not require explicit negative feedback. With this version, we can assess whether negative feedback is worth the additional user effort. While all methods are capable of achieving good performance, we find that each incurs a different penalty with respect to our requirements. TF-IDF requires more training than NBC, which in turn requires negative training samples. The version of NBC modified to eliminate explicit negative feedback requires storage of previous training documents. We show how each method behaves when the amount of training is limited: while precision remains relatively constant (between 0.4 and 0.6) for all methods, their overall *HM* performance is dictated by recall. While methods using Bayes start out with recall above 0.6 after as little as 50 positive training samples, it takes the more performant TF-IDF based methods at least three times as much training to achieve comparable recall.

Finally, we briefly present the results of a user study on QueryTracker showing that if users are willing to provide negative feedback, filtering performance significantly improves. The study also shows that combining filtering decisions by disseminating based on either NBC or TF-IDF leads to improved recall.

Topic No.	Relevant Documents	Non-rel. Documents	Position of median rel.	Dataset
189	584	695	62595	FBIS
301	339	433	50695	FBIS
354	175	715	64424	FBIS
374	109	315	58943	LATIMES
422	98	840	70875	LATIMES
426	145	626	67988	LATIMES

Table 4.2: Topics used in TF-IDF vs. NBC comparison

4.3.1 Comparing TF-IDF and the Naive Bayes Classifier

For each dataset, we picked the topics with the best balance between relevant and non-relevant documents. The topics include low, medium, and high numbers of relevant documents from each of the two collections. Relevant documents are distributed uniformly throughout the collections. The topics are shown in Table 4.2

For each topic and method, a profile is built from the first half of the available relevant documents (up to and including the median relevant document). Once the position of the median relevant document is reached in the document stream, the remaining documents are used to test the filtering performance of the profile. Similarly to the previous sections in this chapter, we use the harmonic mean of precision and recall (HM) to evaluate filtering performance. The slight differences in evaluation methodology are required to allow for a fair comparison between TF-IDF and Bayes filtering systems.

4.3.2 TF-IDF Results

We use the TF-IDF filtering system and parameter settings derived in Section 4.1. The only additional variation introduced in this experiment is a more detailed exploration of the dissemination threshold learning mechanism. In addition to the *adaptive* learning rate α used earlier, we explore setting the threshold at a fixed fraction between the similarities of the documents which were most, and least, similar to the user profile,

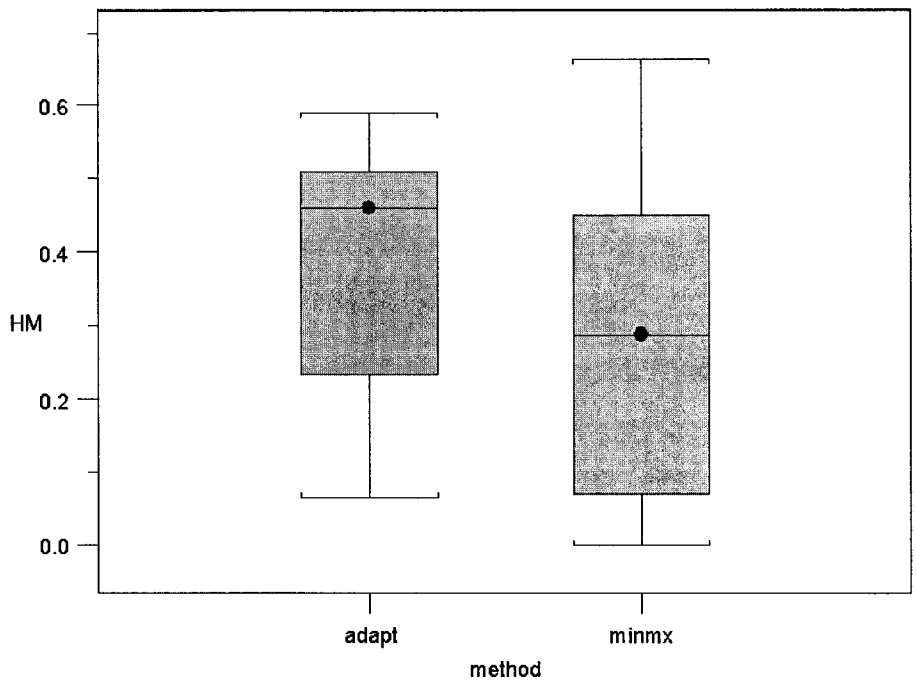


Figure 4.23: Discrepancy between average and best results for adaptive and min-max on *HM* across all parameter settings

respectively. The latter method will be referred to as *min-max*, and the parameter controlling where the threshold is set will be named *fraction*. The motivation behind trying to set the dissemination threshold to a fixed value is to verify how well *adaptive* threshold learning approaches the values (and filtering performance) of a system that benefits from the "hindsight" of knowing the distance range of all its relevant training samples.

4.3.2.1 Comparing Thresholding across All Parameters

In this experiment, we assess the impact on each thresholding method described above (*adaptive* and *min-max*) by its thresholding parameter (learning rate and min-max fraction, respectively). This first experiment is exploratory in nature, as we search over a wide range of values for the regions where each parameter optimizes performance.

Both adaptive learning rate and min-max distance fraction are given values in the interval [0.05..0.7], in 0.05 increments. A two-sample t-test of the results yields $t =$

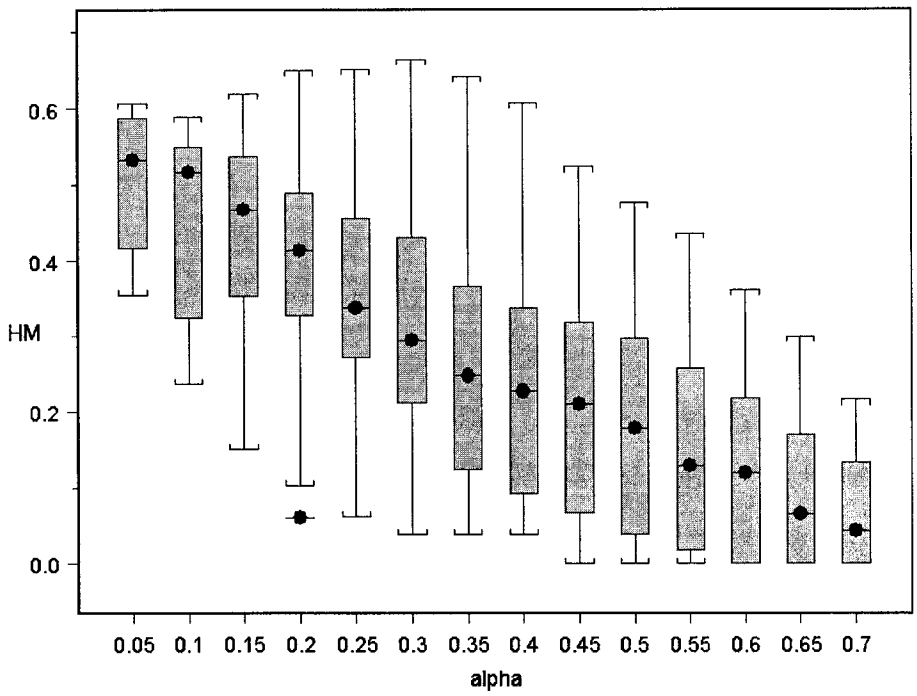


Figure 4.24: Effects of min-max fraction on *HM* performance across all other parameter settings.

5.642, $p \leq 0.0001$, showing that adaptive results are better on average than min-max results (adaptive mean *HM* was 0.369, min-max mean *HM* was 0.275). However, the best value for min-max exceeds the best value for adaptive (as shown in Figure 4.23). This suggests that the range of the min-max fraction parameter that leads to optimal min-max *HM* performance is different, possibly narrower, than the range for the adaptive learning rate that optimizes adaptive *HM*.

A one-way ANOVA on min-max *HM* results by min-max fraction shows significant effects ($F = 122.921$, $p \leq 0.0001$). A box-plot of *HM* by min-max fraction is shown in Figure 4.24. The variability is caused by difference in performance across the multiple topics. Over this range of the min-max fraction parameter, the performance decreases almost linearly, with a slight reduction at the lowest value. This suggests that min-max may be optimized by a value even smaller than 0.05.

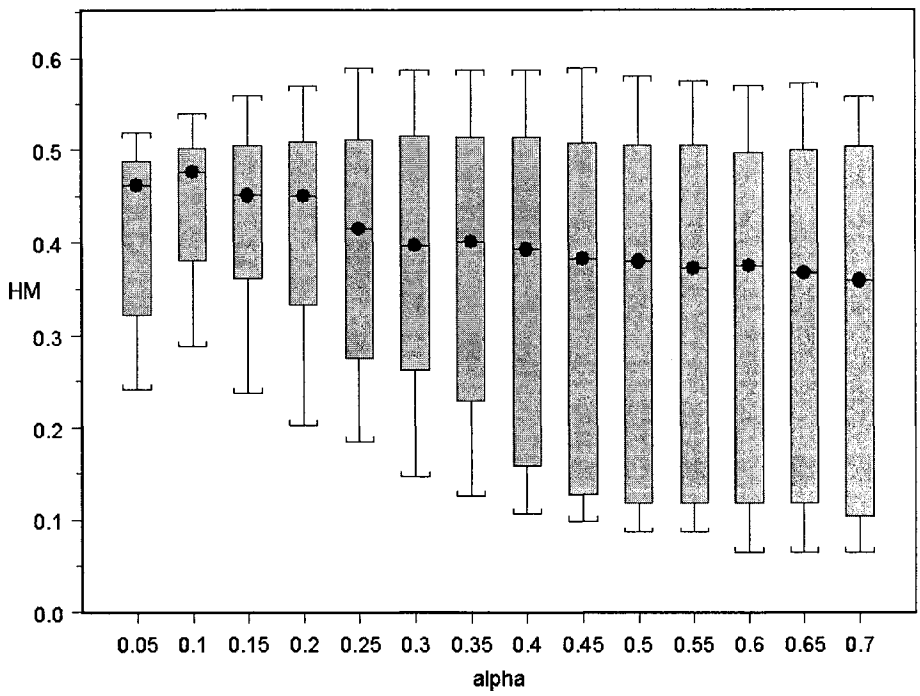


Figure 4.25: Effects of learning rate on adaptive *HM* performance across all other parameter settings

A one-way ANOVA on adaptive *HM* results by learning rate also shows significant effects ($F = 9.980$, $p \leq 0.0001$). A box-plot of *HM* by learning rate is shown in Figure 4.25. The performance variability is much smaller over the range of learning rates, which explains why adaptive results were better on average than min-max ones. The best adaptive results occur for a learning rate of 0.1, again, a relatively low, conservative value, consistent with the results from Section 4.1.

When we pick the best empirical min-max fraction (0.05) and the best adaptive learning rate (0.10) and compare results only on these restricted data sets, min-max obtains better performance (mean $HM = 0.501$) than adaptive threshold learning (mean $HM = 0.438$). However, the difference is not significant; a two-sample t-test yields $t = 1.625$, $p \leq 0.118$.

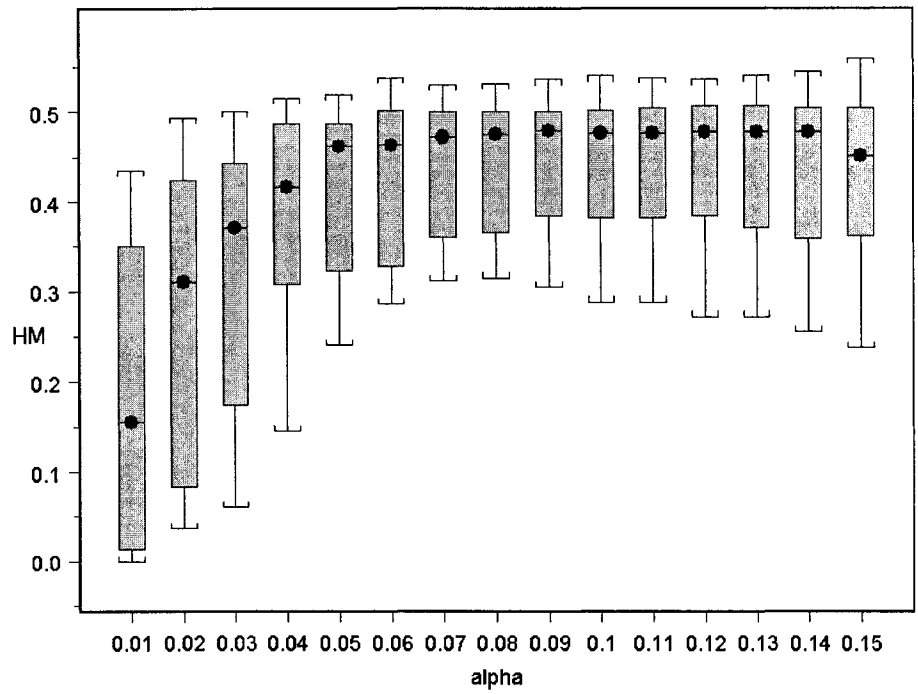


Figure 4.26: Effects of learning rate on adaptive *HM* performance (narrower parameter range)

4.3.2.2 Comparing Thresholding on Focused Parameter Settings

The previous experiment suggests that min-max *may* outperform adaptive learning given the right parameter setting. We repeat the experiment, this time using a narrower range focused around values of min-max fraction and adaptive learning rate which respectively optimized performance in the previous experiment. Both parameters will take values in the interval [0.01..0.15], with 0.01 increments. This experiment should be viewed as a "zoom-in" operation on the behavior of our algorithms over the parameter value ranges that did well the first time around.

For the adaptive method, the learning rate over the interval of lower values also has a significant effect on *HM* ($F = 36.091$, $p \leq 0.0001$), as shown in Figure 4.26. The learning rate leading to the best mean *HM* performance (0.446) seems to be 0.09.

For min-max, the distance fraction over the interval of lower values also has a sig-

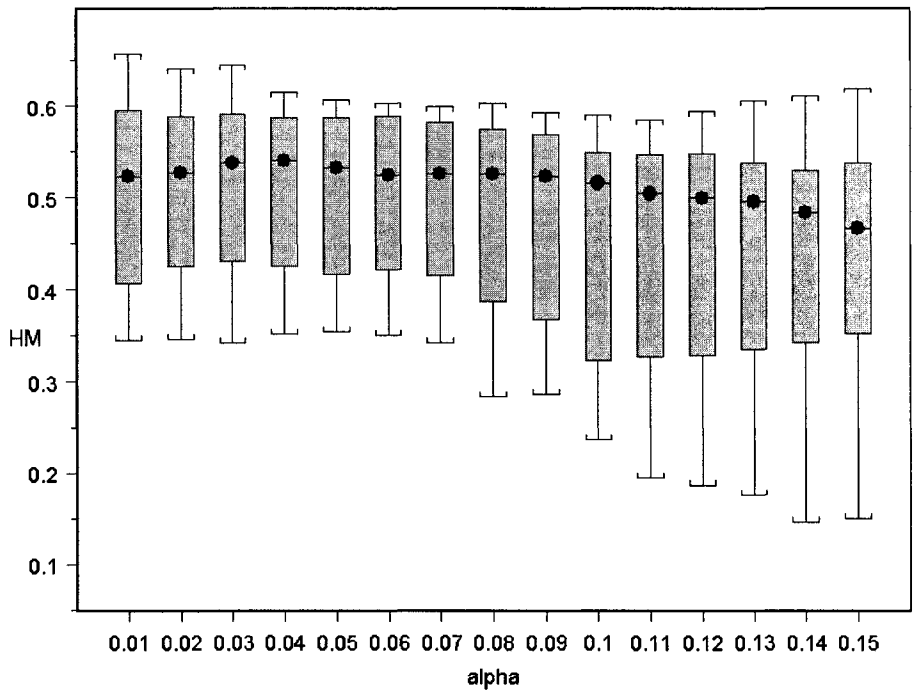


Figure 4.27: Effects of min-max fraction on HM performance (narrower parameter range)

nificant effect on performance ($F = 10.911$, $p \leq 0.001$), as shown in Figure 4.27. Best results ($HM = 0.509$) are obtained when the distance fraction between least and most similar feedback documents is set to 0.04.

A comparison of the two methods shows that min-max clearly outperforms the adaptive method (both mean and max value is higher for min-max, as shown in Figure 4.28). A two-sample t-test on the HM performance of min-max vs. adaptive yields $t = 6.065$, $p \leq 0.0001$. Clearly, storing documents and using the benefit of hindsight to set the dissemination threshold to a fixed value allows min-max to outperform adaptive threshold learning.

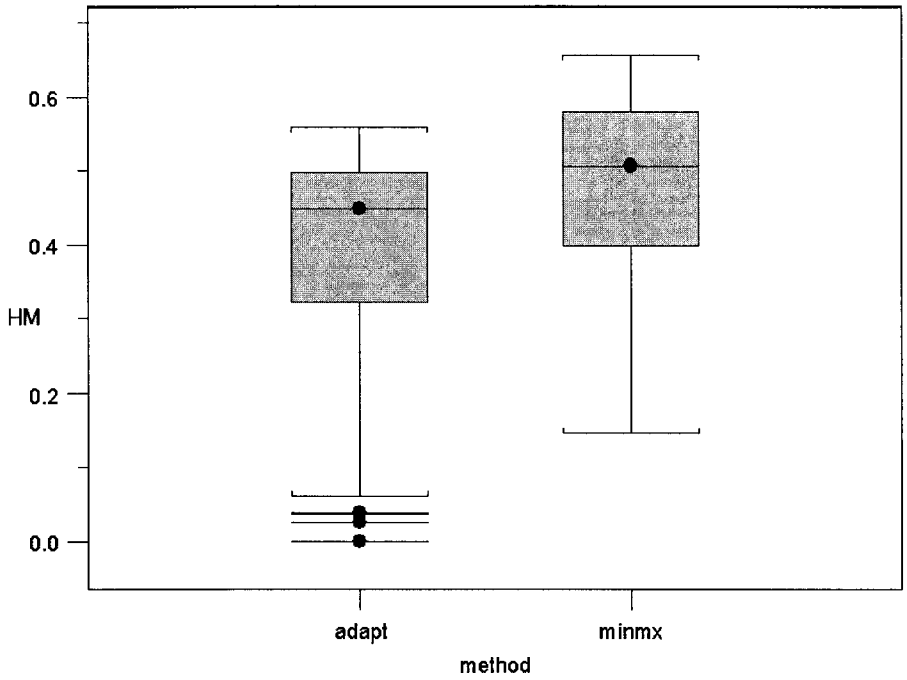


Figure 4.28: Comparison between adaptive and min-max methods over a more relevant range of learning rates and min-max distance fractions

4.3.2.3 Analysis of Results

Although the best min-max settings outperform the best adaptive thresholding, the min-max method requires the availability of all of the feedback documents, stored as TF-IDF vectors. Should that not be the case (e.g., due to storage constraints), we need to fall back on adaptive threshold learning. To figure out just how much of a performance hit we should expect in such a situation, we compared the best adaptive run (at learning rate 0.09) to the best min-max run (at distance fraction 0.04). A two-sample t-test across the queries yields $t = 1.813$, $p = 0.084$, which is perhaps marginally significant. The results are presented graphically in Figure 4.29 (two leftmost bars); the means differ, but the variance is about the same for both, suggesting that performance of both vary somewhat with queries. The low value of the min-max fraction (0.04) suggests that recall is much more sensitive to this parameter than precision.

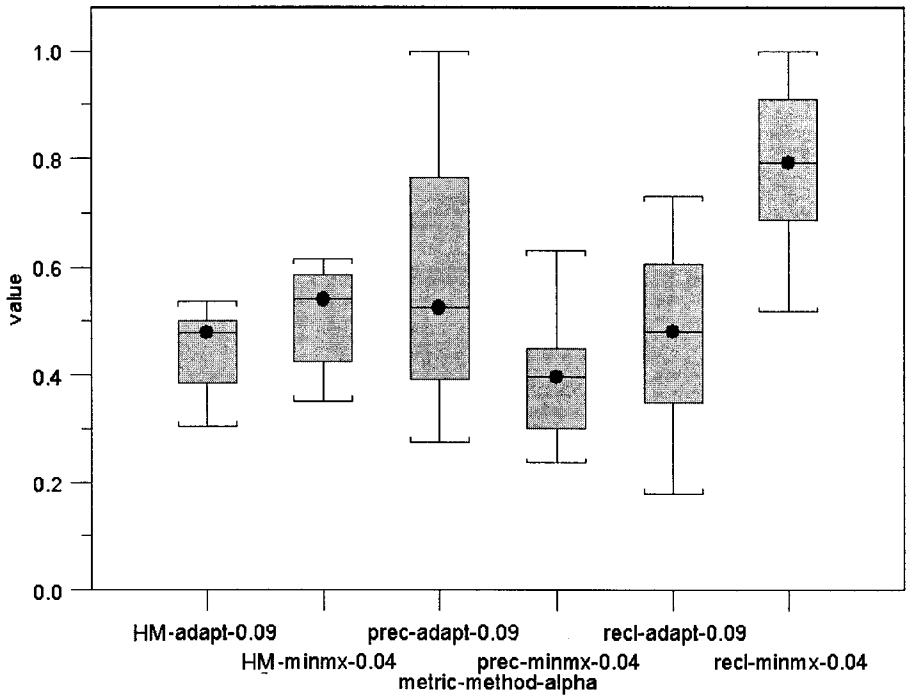
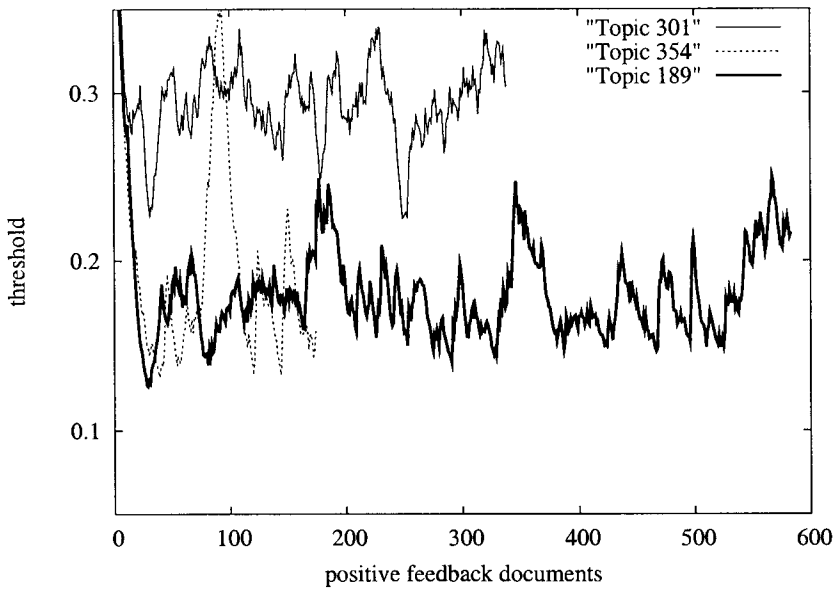


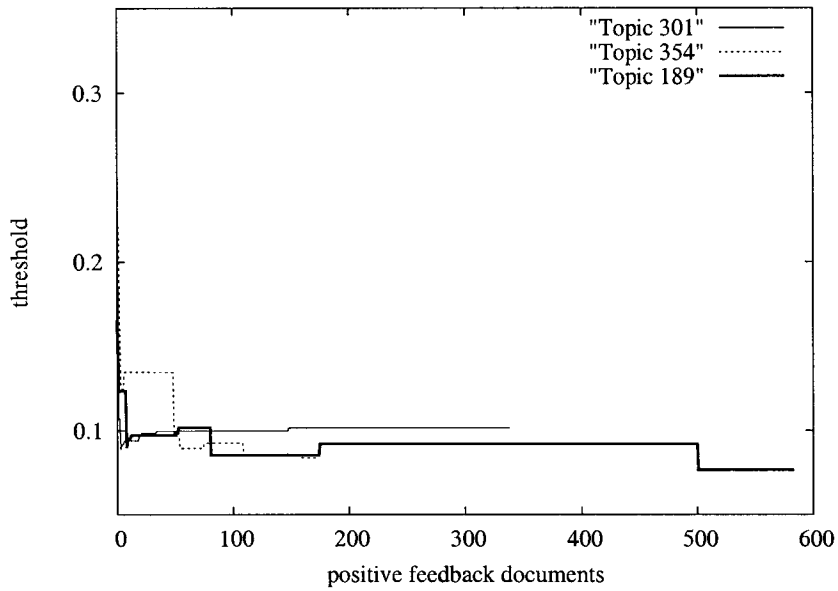
Figure 4.29: *HM*, precision, and recall for best runs of adaptive and min-max. X-labels consist of metric, filtering method (min-max or adaptive), and the value of either α (for adaptive) or distance fraction (for min-max).

To assess the amount of training required, we plot the evolution of the threshold after each new known-relevant document was added to the profile for a subset of the studied topics. Rather than stopping after exhausting the training samples, we continue adjusting the threshold and plotting its value until the end of the corpus was reached. The results are shown in Figure 4.30. Note that graphs for each topic have different lengths on the x axis, reflecting different numbers of positive training samples.

Adaptive thresholds fluctuate within ranges specific to each topic. Min-max thresholds quickly stabilize after approximately 25 positive training examples with only minor changes after that. Interestingly, for all topics, min-max thresholds end up in a narrow range in the proximity of 0.1. This stability suggests that we may not need to learn min-max thresholds at all, and instead simply use TF-IDF with a hard-coded dissemination



a) Adaptive $\alpha = 0.09$



b) Min-Max ratio = 0.04

Figure 4.30: Dissemination threshold for topics 189, 301, and 354: a) best adaptive α = 0.09, and b) best min-max ratio = 0.04

threshold of 0.1. This situation is common across all six of the topics we used. If hard-coding the TF-IDF dissemination threshold proves viable outside of our tested topics, it would eliminate the need to store the document vectors.

As expected, due to higher thresholds, the adaptive results have better precision than min-max, but min-max has better recall (see the last four bars in Figure 4.29.) Precision for min-max degrades less due to its lower threshold than recall degrades for adaptive thresholds due to its higher threshold. When combined, the *HM* values for min-max are better than those for adaptive thresholds, mainly due to min-max maintaining and leveraging more information in the form of explicitly stored training samples used to set the dissemination threshold.

4.3.3 NBC Results

We implement the Naive Bayes Classifier in order to test how much of a performance penalty we incur with TF-IDF by not requiring users to supply negative training. Since an NBC profile mainly consists of conditional probability values associated with terms in the vocabulary, we first test whether we should limit the vocabulary to terms occurring in *labeled* training samples only, or whether it would be advantageous to use all terms encountered during training, just like using such terms helps build better corpus-wide statistics for TF-IDF (e.g., the IDF component). Then, we consider a new filtering algorithm based on the NBC, which does not require explicit negative feedback.

4.3.3.1 Limiting Vocabulary to Labeled Training Samples

Some documents in the TREC test collection have no relevance judgments for any of the existing topics. With TF-IDF, including these documents into the IDF corpus-wide calculations helps gain a better overall view of the collection, and improves performance. We are interested in determining whether this still holds true for NBC: in other words, does including terms occurring only in unlabeled documents into the vocabulary help or hurt the filtering performance of a Bayesian classifier?

A paired t-test between profiles using all terms from the training set on one hand and profiles using only terms from the labeled training samples on the other, resulted in $t =$

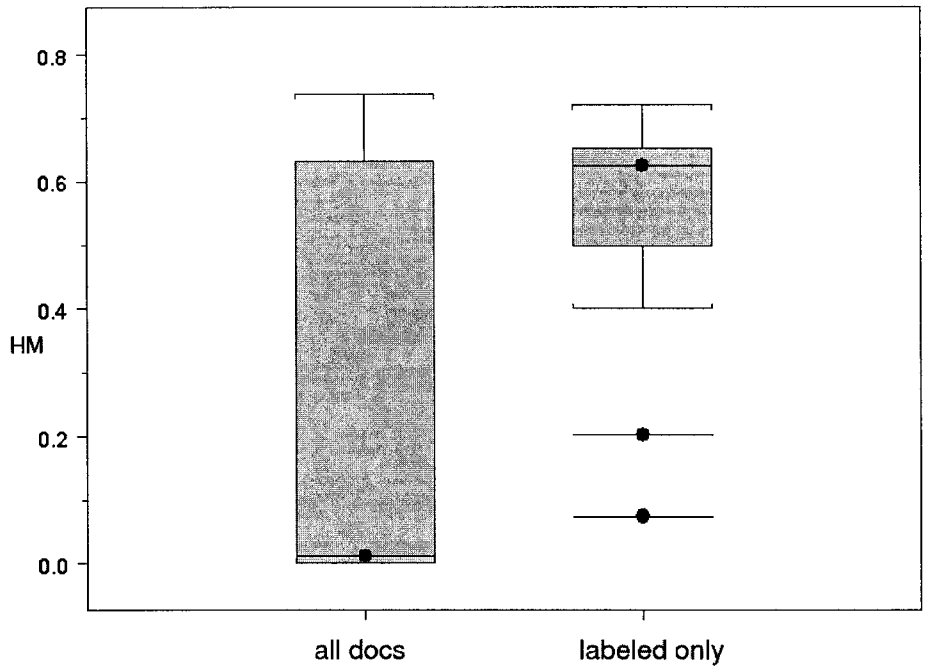


Figure 4.31: Impact of restricting vocabulary to labeled training samples on NBC's filtering performance

5.596, $p < 0.0001$. The mean and standard deviation were 0.27 and 0.32 for all terms, and 0.56 and 0.16 for terms from labeled training samples only. Clearly, including terms from the unlabeled training documents decreases performance, as shown in Figure 4.31.

The worst effects are observed on topic 354 from the FBIS corpus and all three topics from LATIMES. The differences in performance coincide with the fact that the affected topics have several times more labeled negative training samples than labeled positives (715 and 175, respectively). Considering the computation underlying NBC, we recall that the conditional probability of a term t given class C (where C is either the positive or the negative class) is:

$$p(t|C) = \frac{occurrences(t) + 1}{npos(C) + nterms}$$

where $occurrences(t)$ is the number of times t shows up in class C, $npos(C)$ is the number of word positions in class C, and $nterms$ is the total number of terms in the vocabu-

lary. Terms that do not show up in the labeled examples will have $occurrences(t) = 0$, and their conditional probabilities will all be equal to:

$$p(t|C) = \frac{1}{npos(C) + nterms}$$

When the negative class contains many more documents than the positive class, we have $npos(-) > npos(+)$, and, as a consequence, $p(t|+) > p(t|-)$ for all terms t that do not appear in labeled training samples. Since we know that:

$$\sum_t p(t|C) = 1$$

it follows that all those terms that never appear in labeled training samples will diminish the conditional probabilities of those terms that do, and, moreover, diminish the conditional probabilities given the positive class more than those given the negative class. This process is illustrated in Figure 4.32. The net effect of this phenomenon is a significant decrease in recall for the affected topic: the probability of a document belonging to the positive class, as perceived by the algorithm, is diminished.

4.3.3.2 Need for Negative Feedback

One of the desirable characteristics for a filtering algorithm used with a Web information agent is that negative feedback should not be required. We implemented a simplified version of S-EM [86] (described in Section 2.3.2.2) which requires only positive samples to be labeled within the training set, as follows:

1. Assume all unlabeled training documents are negative, and build an initial NBC based on this assumption.

2. Classify all unlabeled training documents using the classifier in Step 1, and sort them according to the difference in their conditional probabilities:

$$\Delta = [p(doc|-) - p(doc|+)]$$

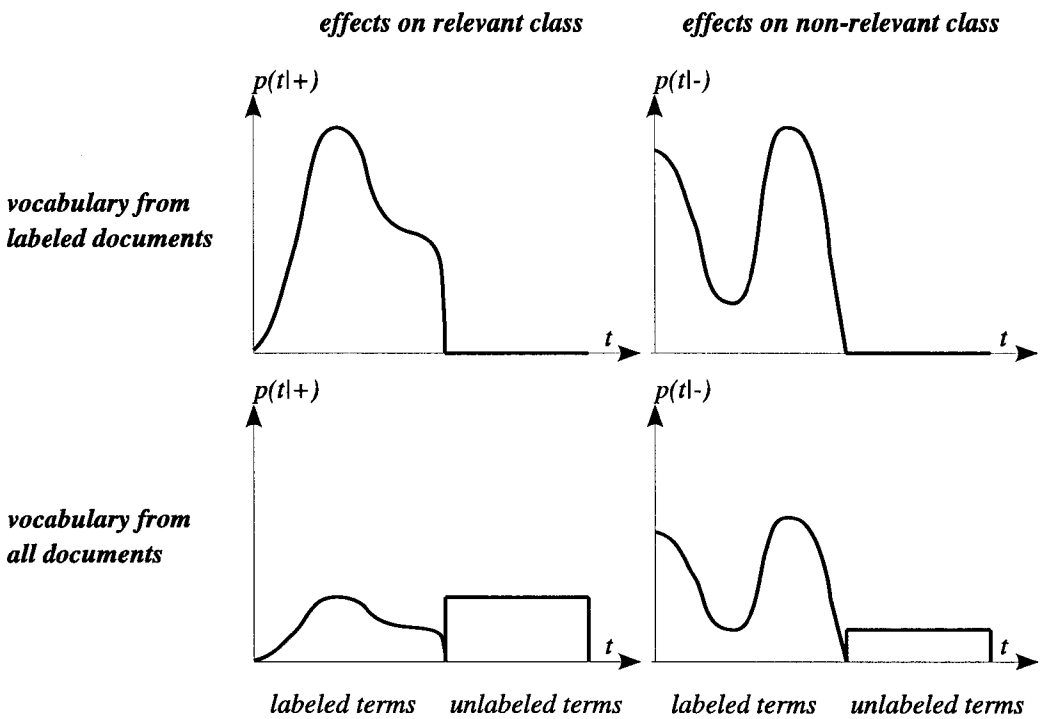


Figure 4.32: Effects of adding terms from unlabeled training samples on conditional probabilities of existing terms

3. Pick the *npdocs* unlabeled documents with the largest Δ, where *npdocs* is the number of *labeled positive* documents. Use these documents as *pseudo-labeled negatives.*

4. Using the labeled positive and pseudo-labeled negative training samples, build a new NBC.

5. Optionally, use EM[101]: classify the rest of the unlabeled training samples, and rebuild the NBC once again.

6. Test the final classifier on the test documents.

This algorithm is based on the assumption that the unlabeled documents are mostly negative. If we stopped after Step 1 of the algorithm, we would find similar problems as encountered when labeled negative training samples vastly outnumbered labeled posi-

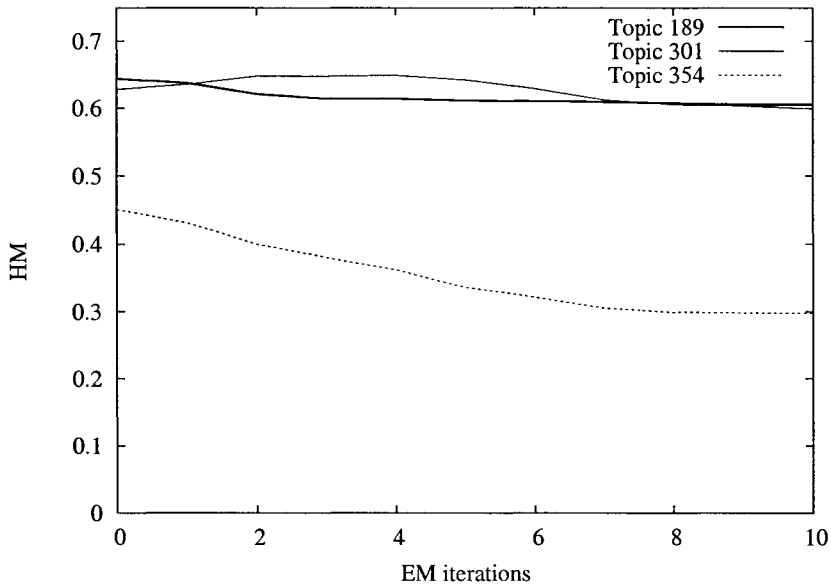


Figure 4.33: Decaying *HM* due to overfitting caused by EM with modified NBC

tives: the conditional probability given the positive class for terms contained by positive documents would become unfairly low, leading to extremely poor recall. This is avoided by picking a number of negative documents equal to the number of known positives. The original S-EM mixes known-relevant documents into the unlabeled set (so-called "spies") in order to compute a threshold intended to help avoid selection of unlabeled relevant documents as pseudo-negatives.

The only variable parameter of this experiment was the number of EM iterations performed in Step 5 of the algorithm. Unlike with the "regular" NBC, using EM does not help with the modified algorithm. *HM* either stays constant or decreases with the number of additional EM iterations. The explanation is that, since we are already "guessing" the negative training samples, adding EM steps leads to overfitting the training data and, ultimately, to poor performance on the test data. We illustrate this phenomenon in Figure 4.33.

4.3.4 TF-IDF and Negative Feedback: A Hybrid Algorithm

NBC-based filters are not the only way to learn a linear separation over the document space. It is also possible to represent positive and negative document classes within a topic using two TF-IDF vectors instead of sets of conditional probabilities. This hybrid approach will later be compared to both the NBC and single-vector thresholded TF-IDF, and will help clarify the source of their different learning speeds and overall performance.

Two vectors are learned for each topic of interest, consisting of the sum of document vectors representing positive and, respectively, negative training samples encountered by the algorithm. Decisions on whether to classify a document as relevant or nonrelevant are made based on which of the two profile vectors are closer to the document vector, as measured by the cosine metric. In other words, we implemented a classifier that uses TF-IDF as its underlying representation rather than Bayes conditional probabilities.

We wish to determine whether performing an EM step on the training data, or restricting the vocabulary to rated documents has any effect on filtering performance, similarly to the way these optimizations helped improve the performance of our Bayesian filter. For this hybrid algorithm, EM is implemented as follows:

1. An initial pair of positive and negative TF-IDF vectors is built from *labeled* training samples.

2. *Unlabeled* training samples are classified using cosine similarity to the positive and negative vectors from Step 1.

3. A new pair of positive and negative TF-IDF vectors is built using both originally labeled training samples *and* the newly classified ones from Step 2.

4. Test documents are classified using cosine similarity and the class vectors computed in Step 3.

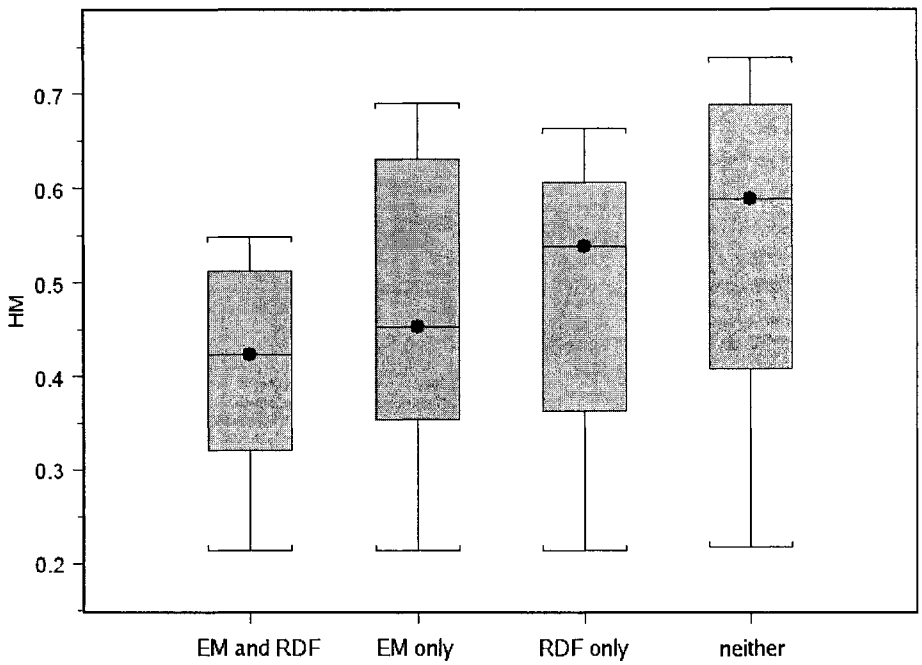


Figure 4.34: Effects of using EM and restricting the vocabulary (RDF) on NBC's filtering performance

A paired t-test indicates that, unlike with the NBC, two-vector TF-IDF does not benefit from EM ($t = 2.496$, $p = 0.03$; mean HMs are 0.512 when EM is not used, and 0.436 when using one EM iteration). Another paired t-test indicates that restricting the vocabulary to terms in labeled training samples is also not helpful ($t = 3.563$, $p = 0.005$; mean HMs are 0.501 for all terms included, and 0.447 when vocabulary is limited to terms in labeled documents). The interaction effect between EM usage and vocabulary restriction is negligible, as shown by an ANOVA test ($F = 0.003$, $p = 0.956$). The effects of EM usage and vocabulary restriction on performance as measured by HM are illustrated in Figure 4.34.

set	topic	Adapt. TF-IDF	Min-Max TF-IDF	NBC	NBC w. auto-neg.	Pos-Neg TF-IDF
FBIS	189	0.481	0.607	0.632	0.644	**0.668**
	301	0.495	0.579	0.692	0.628	**0.738**
	354	0.321	0.352	**0.479**	0.450	0.407
LA	374	0.450	**0.614**	0.608	0.462	0.509
	422	0.405	**0.406**	0.366	0.210	0.218
	426	0.477	0.451	0.641	0.351	**0.688**

Table 4.3: Comparison of *HM* performance, topics × optimized algorithms

4.4 Trading Between Performance and Convenience

We identify three desired characteristics of filtering algorithms for Web information agents. First, the method should use only positive feedback, as that is more easily obtained from users. Second, the method should work well given limited feedback (training examples). Third, the filtering algorithm should be incremental, so that it best uses available storage. In other words, given an existing classifier C_n, learned from documents $d_1..d_n$, it should be possible to derive $C_{n+1} = f(C_n, d_{n+1})$ without the need to store explicit information on documents $d_1..d_n$.

After having optimized a series of filtering methods, we now assess their relative merits with respect to performance and ease of use. We first compare our filtering algorithms for performance. Next, we discuss tradeoffs in learning speed, and show a typical example on topic 189. The desired features (incremental learning, no need for negative training) are also compared. We briefly discuss the applicability of each method to Web information agents, and show a pilot study, which will be the basis of a larger user study presented in Chapter 7.

We compare *HM* values obtained by each method's optimized settings on each of our tested topics in Table 4.3. One-way ANOVAs with the method as the independent variable, and recall, precision, and, respectively, *HM* as the dependents indicate that

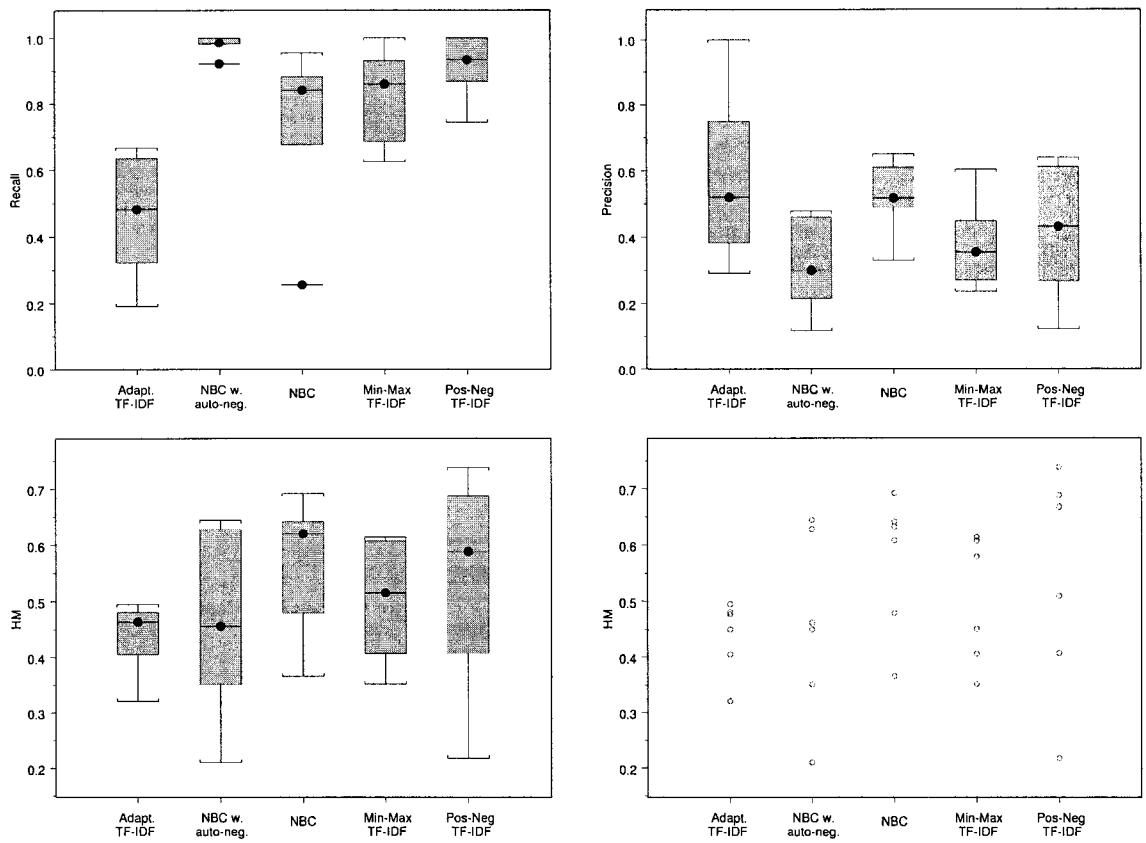


Figure 4.35: Comparison of optimized filtering methods: recall (top-left), precision (top-right), and *HM* (boxplot bottom-left, scatterplot bottom-right)

the method used does not significantly impact *HM* ($F = 0.901$, $p = 0.479$) or precision ($F = 2.098$, $p = 0.111$), but does impact recall ($F = 9.156$, $p < 0.001$). From the boxplots in Figure 4.35, we observe that the impact on recall originates from the relatively poor performance of adaptive TF-IDF, and the unusually high performance of the modified NBC auto-negative method. The relatively worse recall exhibited by adaptive TF-IDF is caused by the dissemination threshold oscillating within higher ranges than, e.g., min-max TF-IDF (see Figure 4.30). The high recall obtained by the modified Bayesian classifier is due to the way negative documents are picked to be farthest away from the positive ones, thus making new documents easier to classify as relevant. From the *HM* boxplots in Figure 4.35, the best performer on average seems to be the unmodi-

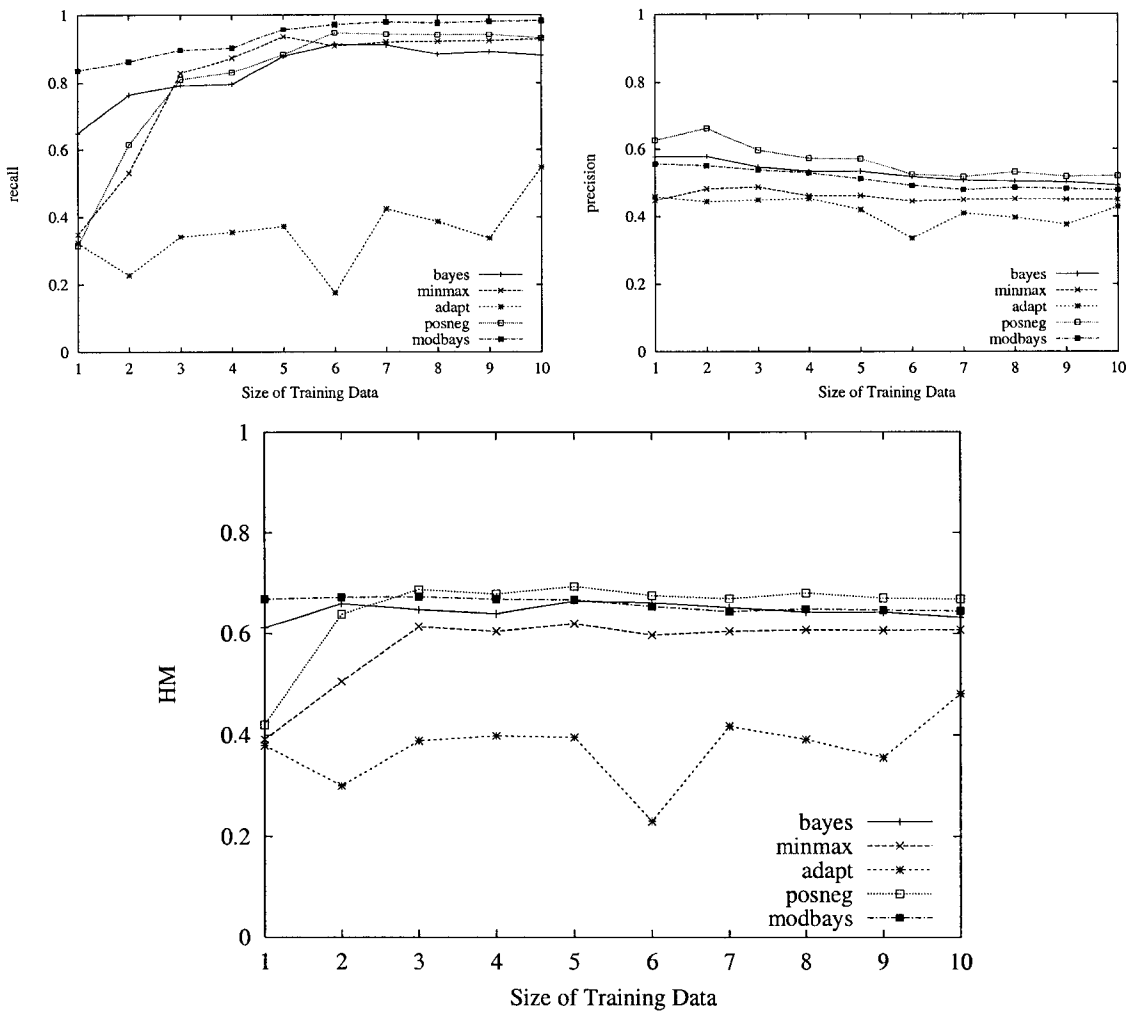


Figure 4.36: Learning speed of optimized filtering methods on Topic #189: recall (top-left), precision (top-right), and *HM* (bottom)

fied NBC, closely followed by min-max TF-IDF. Hybrid (two-vector, pos-neg) TF-IDF yields average performance values close to those of the NBC (and comes out on top for more of the topics than NBC), but suffers from higher variability which originates from the precision component of *HM*.

To assess how quickly each method learns to separate relevant from non-relevant documents, we plot the evolution of precision, recall, and *HM* for topic #189 as we varied the amount of training data in increments of one tenth (from one tenth of the avail-

Method	Incremental Learning	Positive-only Training	Comments
Bayes	yes	no	—
Bayes (auto-neg.)	no	yes	—
2-vector TF-IDF	yes	no	—
2-vector TF-IDF (auto-neg.)	no	yes	not tested
min-max TF-IDF	*	yes	possibly incremental
adaptive TF-IDF	yes	yes	slow learning recall

Table 4.4: Summary of features for each algorithm

able training samples all the way to the complete training set). Results on topic #189 are fairly representative of the average case. It can be observed how adaptive TF-IDF is slow at improving its recall, which hurts its overall *HM* performance. All methods are comparable as far as precision is concerned. Figure 4.36 presents a comparison of learning speeds for the various methods using topic #189 as an example. The graphs also illustrate how, after cca. one third of the training samples are processed, *HM* performance remains relatively flat for most methods. Methods which use negative feedback reach their best performance faster, after only two tenths of the training data has been learned.

A summary of the user profiling/filtering methods considered in this study is presented in Table 4.4. The two methods that use both a positive and a negative class (NBC and hybrid TF-IDF) can have the profile updated incrementally, without requiring the storage of past training samples. The downside of these algorithms is that they require negative feedback in addition of relevant examples. When modified to automatically infer which unlabeled training documents are most likely to be negative, the resulting algorithms require access to all past training examples, and can no longer be updated incrementally (i.e, the classifier must be recomputed *after* a new set of likely-to-be-negative documents has been extracted, every time the set of positive examples is modified).

Thresholded TF-IDF requires no negative feedback. However, when adaptive thresh-

old learning is used to make the process incremental, performance (recall) is slow to improve with training. Setting the threshold using min-max precludes incrementality, as it requires access to previous training examples. The promising news is that min-max thresholds have consistently ended up in the $0.05 - 0.1$ range. With more data sets and topics, it may turn out that setting the dissemination threshold to a constant will give similar performance while also preserving the incremental nature of the algorithm. Another modification that could make min-max TF-IDF incremental would be to maintain most current values for minimum and maximum similarity of a relevant training document to the topic vector. While these values would not always apply to the *current* topic vector (due to the latter being updated by relevant training samples that are neither minimally or maximally similar), this may turn out to be a good enough approximation to preserve the min-max algorithm's performance while achieving incremental updates without explicit storage of training samples.

To summarize, all methods studied in this chapter achieve filtering performance consistent with current practice, but each of them incurs a different penalty. TF-IDF methods require more training. NBC requires negative feedback, and our version of NBC without explicit negative feedback requires storage of past documents. Given that users apparently can be cajoled into providing negative feedback (see Chapter 7), we advocate NBC as the best performing method, on both a pre-judged corpus and in a user study, for applications, such as Web information agents which provide restricted feedback and storage.

Chapter 5

Detecting Relevant Changes and Redundancy

Sometimes, Web agents repeatedly encounter the same document. For instance, QueryTracker (Figure 1.2 in Chapter 1) monitors users' queries over time by repeatedly submitting them to a search engine. Returned lists of documents have a high degree of overlap between subsequent iterations. The simplest approach (used by Google's WebAlerts[56]) is to identify *new*, previously unseen documents (by URL), within the top ten returned hits. This, however, does not account for possible modifications made over time to old, previously encountered documents.

A related problem [74] is that, frequently, content is either *mirrored* (i.e., exact copies of the same document appear at multiple different addresses), or *aliased* (e.g., when multiple online newspapers copy the same article or press release, but with their own surrounding "boilerplate" formatting). These documents may be identified by employing *redundancy detection* techniques [22, 150].

In this chapter, we present the technique used by QueryTracker to determine if relevant changes were made to a previously encountered document. We also compare two methods of redundancy detection, assessing their relative merits and ease of use.

5.1 Change Relevance Detection

When is a new version of a previously encountered document worth examining anew? Changes between versions occur quite frequently (often on a daily basis), but, most of the time, they are not relevant to any user interest or information need. We wish to ignore changes in advertisements, banner ads, and counters, but detect when the useful content is modified in a meaningful way. We disseminate a new version of an old document only when the changes are themselves relevant to the users' profile.

Utilities such as UNIX `diff` (which is based on the detection of longest common subsequences [68, 72]), or its HTML-ized variants [42, 30] are focused heavily on pointing out the *location* where changes occur within a document. Since our techniques are based on IR and text filtering, we end up judging the relevance of changes via bag-of-words (TF-based) methods such as TF-IDF or NBC. Therefore, we are only interested in converting the text of the changes into a TF vector, and would be discarding the location information. As such, a more direct and efficient approach is to simply perform a vector difference between the TF vectors representing the new and original documents, respectively. This method loosely resembles the *set difference* technique used by Zhang et al.[150] for redundancy detection (see Section 3.2.3). However, the specific details differ as the intended applications are different (redundancy detection vs. prediction of change relevance).

The difference vector is an aggregate representation of the modifications made to the original:

$$V_{\text{diff}} = V_{\text{new}} - V_{\text{orig}}$$

Terms with positive weights in V_{diff} represent content added to the original, and may be represented separately as V_{add}. Terms with negative weights in V_{diff} represent removed content, and may be represented separately as a positive-only vector V_{del}, such that

$$V_{\text{add}} - V_{\text{del}} = V_{\text{diff}}$$

We may now compute the relevance of additions and deletions to the original document independently of each other. This allows for more fine-grained dissemination decisions based not only on the relevance of the changes, but also on their direction (addition vs. deletion). Based on our observations, users are typically more interested in cases when relevant content is added.

This section presents a user study which is used to evaluate the adequacy of using V_{add} and V_{del} to predict the relevance of changes to Web documents.

5.1.1 Experimental Setup

The purpose of our study is to measure how well QueryTracker is able to predict the relevance of changes made to previously encountered documents. QueryTracker users were asked to submit queries matching their ongoing interests, and to provide feedback on the relevance of disseminated documents. We received a total of six queries from four users:

flushots *flu+shot+pregnancy* – effects and interactions of flu shots during pregnancy;

harry *harry+potter+order+phoenix+book* – news on the (at that time) yet to be released 5th Harry Potter book;

graves *graves+thyroid+disease* – news and information on Graves' hyperactive thyroid disease;

romania *romania+integration+european+union+nato* – articles and news on Romania's progress toward integration into the E.U. and NATO;

scheduling *oversubscribed+scheduling* – news and articles on oversubscribed scheduling research;

vorticity *potential+vorticity+coordinate* – news and articles on a subdomain of atmospheric science research.

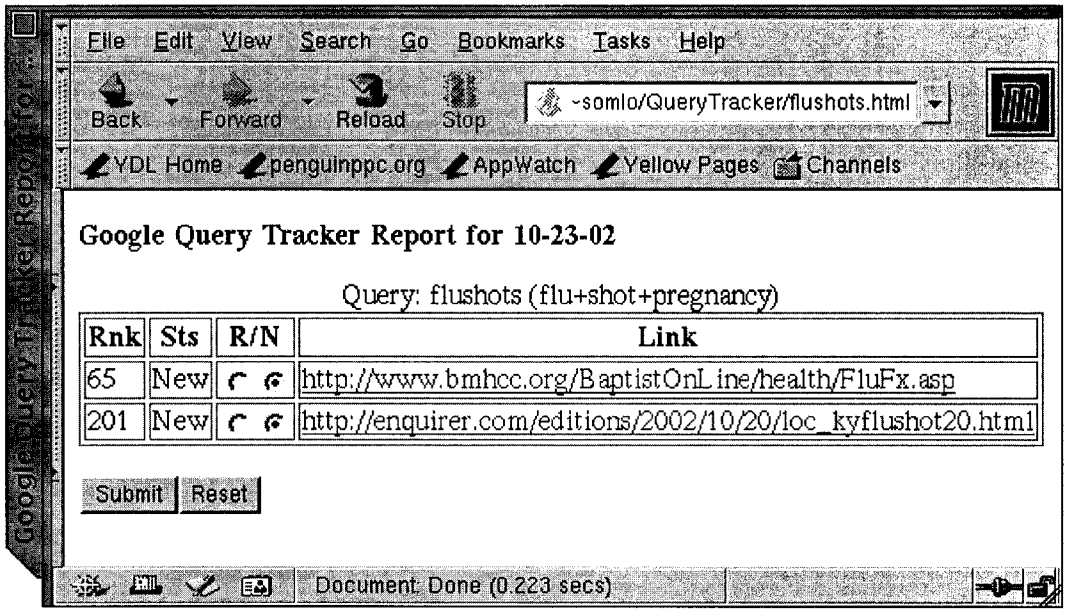


Figure 5.1: Daily survey on relevance of newly found documents

Each day over the period of one month, users were presented with a list of *new* documents (disseminated daily from the top 100 results returned by sending the original query to Google) and asked to identify the ones they found relevant. An example of this daily survey is shown in Figure 5.1.

During the trial, QueryTracker saved the modified versions of documents previously marked as relevant by users. A document was considered to have changed if its TF-IDF vector changed.

At the end of the trial, users were asked to fill out a final survey: documents they had identified as relevant and for which changed versions were later detected were offered to them side by side with the changed version. Users were asked to indicate whether significant additions or deletions were made between the original and changed versions of the document. An example of the final survey form is shown in Figure 5.2.

For each pair of documents listed in the final survey (relevant original and its subsequent changed version), we extracted V_{add} and V_{del}, as described earlier, and used them to predict change relevance using one of three methods:

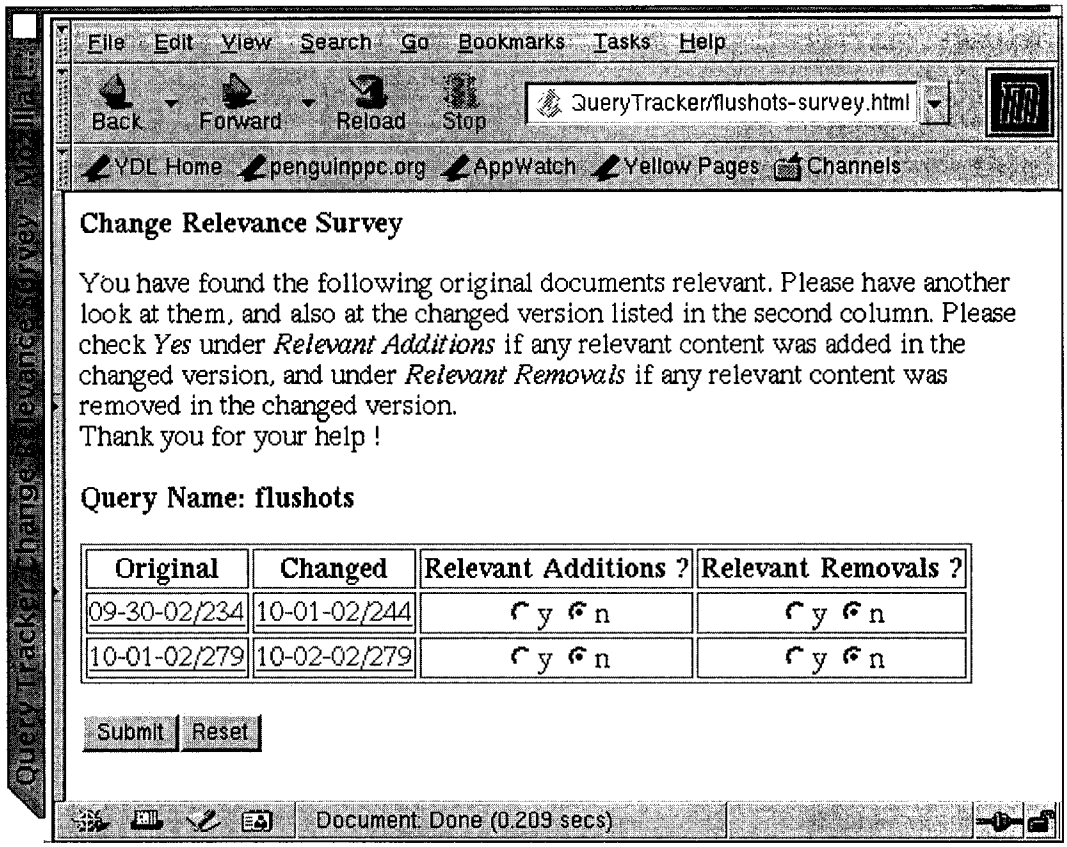


Figure 5.2: Final survey on relevance of changes between document versions

profile Based on all positive user-provided training examples. For each query, a topic profile vector was built by adding together the TF-IDF vectors of all documents identified as relevant to that query during the trial. V_{add} and V_{del} were compared to this profile vector to predict change relevance.

orig Based on the original document. V_{add} and V_{del} were compared to the TF-IDF vector of the original.

query The user query was converted into a TF-IDF vector and compared to V_{add} and V_{del} to predict change relevance.

The first method (**profile**) is the most expensive and inconvenient, as it requires users to provide constant feedback to QueryTracker during its normal operation. On the other

Query	New	NR	NRC	RelAdd	RelDel
graves	469	164	20	3	2
harry	542	109	29	7	5
romania	515	63	18	5	7
scheduling	384	49	4	1	0
flushots	508	9	2	0	0
vorticity	510	33	0	0	0

Table 5.1: Summary of new documents retrieved

hand, we expect this method to be the most accurate in predicting change relevance. The other two methods are designed to allow the unsupervised operation of QueryTracker by eliminating the need for user-supplied feedback and training. The second method (**orig**) is based on the fact that the original document itself is an example of relevant text on the user's topic of interest and assumes that relevant changes would occur along a similar direction in TF-IDF space. The third method (**query**) compares changes against the continuous query itself and depends on the query being descriptive enough, i.e., having sufficient terms to be convertible into a halfway decent TF-IDF vector. We expect the last method to be the least accurate.

5.1.2 Results

During the course of our experiment, we collected a total of 8071 distinct documents, which were either new hits or contained changes from their previous versions. For each query, we collected the information shown in Table 5.1. The significance of each column is as follows:

Query a keyword identifier for the query;

New number of new documents disseminated for the query, over the period of the trial;

NR subset of new documents which were rated relevant by the user;

NRC subset of NR for which changed versions were retrieved during the trial;

RelAdd, RelDel subset of NRC for which the changed version contained additions (and deletions) considered relevant by the user.

Based on this information, it became apparent that we did not have sufficient data on three of our six queries:

vorticity No new relevant document detected over the trial period had a subsequently changed version.

flushots Only two of the new relevant documents had subsequently changed versions, but none of these changes were rated relevant by the user.

scheduling Only four new relevant documents had changed versions, and only one of these had relevant additions.

We observe that documents representing general interest queries, such as **graves**, **harry**, and **romania**, will receive both a higher hit rate from search engines (e.g., Google), and a higher rate of change than documents representing more narrow research interests, such as **flushots**, **scheduling**, and **vorticity**.

We begin by establishing a baseline score for change relevance between pairs of unrelated documents. Ten random pairs of relevant (but unrelated) documents from each topic were selected. In order to convert one member of such a pair into the other, one would have to make relevant deletions (practically, delete the entire original), and relevant additions (add the text of the latter document). We used each of our three methods (**profile**, **query**, and **orig**) to compute addition and deletion relevance scores for these pairs, and averaged them over the ten trials within each topic (see the column named "base" in Table 5.2). Since in practice we wish to detect the relevance of changes between *related* documents (i.e., subsequent versions of the *same* document), we expect this baseline change relevance to be an upper bound. Related documents are expected to be more similar to each other, and therefore the changes between them less relevant.

Query	method	base	t	$p \leq$	rel	$nrel$	threshold	fp	fn	recall	precision
graves	profile	.251	5.18	0.001	3	17	.058-.190	2-0	0-1	.600-1	1-.667
	query	.143	5.28	0.001	3	17	.038-.161	1-0	0-1	.750-1	1-.667
	orig	.056	0.84	0.41	3	17	.043-.106	2-1	1-2	.500-.500	.667-.333
harry	profile	.208	5.85	0.001	7	22	.092-.135	2-0	0-3	.778-1	1-.571
	query	.084	3.07	0.005	7	22	.042-.143	2-0	0-3	.778-1	1-.571
	orig	.037	3.11	0.005	7	22	.081-.212	5-0	1-5	.545-1	.857-.286
romania	profile	.263	2.90	0.01	5	13	.040-.162	5-0	0-3	.500-1	1-.400
	query	.115	1.21	0.24	5	13	.032-.059	1-1	2-2	.750	.600
	orig	.030	0.35	0.73	5	13	.003-.160	9-2	0-4	.357-.333	1-.200

additions

Query	method	base	t	$p \leq$	rel	$nrel$	threshold	fp	fn	recall	precision
graves	profile	.248	5.38	0.001	2	18	.053-.077	2-0	0-1	.500-1	1-.500
	query	.142	5.01	0.001	2	18	.019-.067	2-0	0-1	.500-1	1-.500
	orig	.909	7.13	0.001	2	18	.407-.591	0-0	0-0	1	1
harry	profile	.223	7.88	0.001	5	24	.061-.067	0-0	0-0	1	1
	query	.112	7.04	0.001	5	24	.047-.048	0-0	0-0	1	1
	orig	.945	1.57	0.13	5	24	.158-.688	11-0	0-4	.313-1	1-.200
romania	profile	.252	3.84	0.002	7	11	.052-.058	2-0	0-1	.778-1	1-.857
	query	.091	2.53	0.022	7	11	.015-.015	0-0	0-0	1	1
	orig	.948	6.23	0.001	7	11	.259-.437	1-0	0-2	.875-1	1-.714

deletions

Table 5.2: Study of threshold ranges for change relevance prediction; fp, fn, recall, and precision ranges are shown for meaningful ranges of the dissemination threshold.

One interesting anomaly can be observed for the **orig** method, where relevant additions receive a disproportionately low score when compared to deletions. This method is heavily biased in favor of the original document. When the original is converted into a different document, a large portion of its content needs to be removed (hence the large relevance of deletions, essentially a comparison of the original document to itself). Also, a significant portion of the new document needs to be added, which explains the low score of the addition vector (the result of comparing the original document to its mostly unrelated "new version"). Because of this bias, we expect this method might still prove useful for predicting deletion relevance.

To establish how well our methods distinguish between pairs with relevant and non-relevant changes, we performed a two-sample t-test for each method, query, and direction of change, with the null hypothesis that QueryTracker's computed change relevance was similarly distributed regardless of the actual, user-supplied judgment. Results are shown in the t and p columns of Table 5.2. In general, all methods made a clear distinction between relevant and non-relevant changes, with the exception of **orig**, which had most trouble with predicting addition relevance, and the query **romania** on additions, which posed the most difficulties for all three methods (although $p < 0.01$ for **profile**, this is an order of magnitude less significant than results obtained for other queries).

We must now set dissemination thresholds for the change relevance predictions made by QueryTracker. Our data show that such thresholds need to be specific to each query and direction of change. First, we sort the document pairs by their predicted change relevance. Then, we set the lower limit of the threshold range to the score of the first pair with known relevant changes, and the upper limit to the score of the last pair with changes known to be non-relevant. At the lower limit of the threshold range ("threshold" column in Table 5.2), we will have no false negatives (all relevant changes correctly identified), but we may have false positives (non-relevant changes may be mistakenly classified as relevant). Setting the threshold here would optimize recall. At the upper limit of the range, there will be no false positives, but false negatives may exist, and precision will be optimized. Values outside this range are not meaningful, as recall and precision are only influenced by variations of the threshold *within* the interval. Given the number of relevant changes rel, and the number of false positives fp and false negatives fn, recall and precision are computed thus:

$$recall = \frac{rel - fn}{rel - fn + fp}$$

$$precision = \frac{rel - fn}{rel}$$

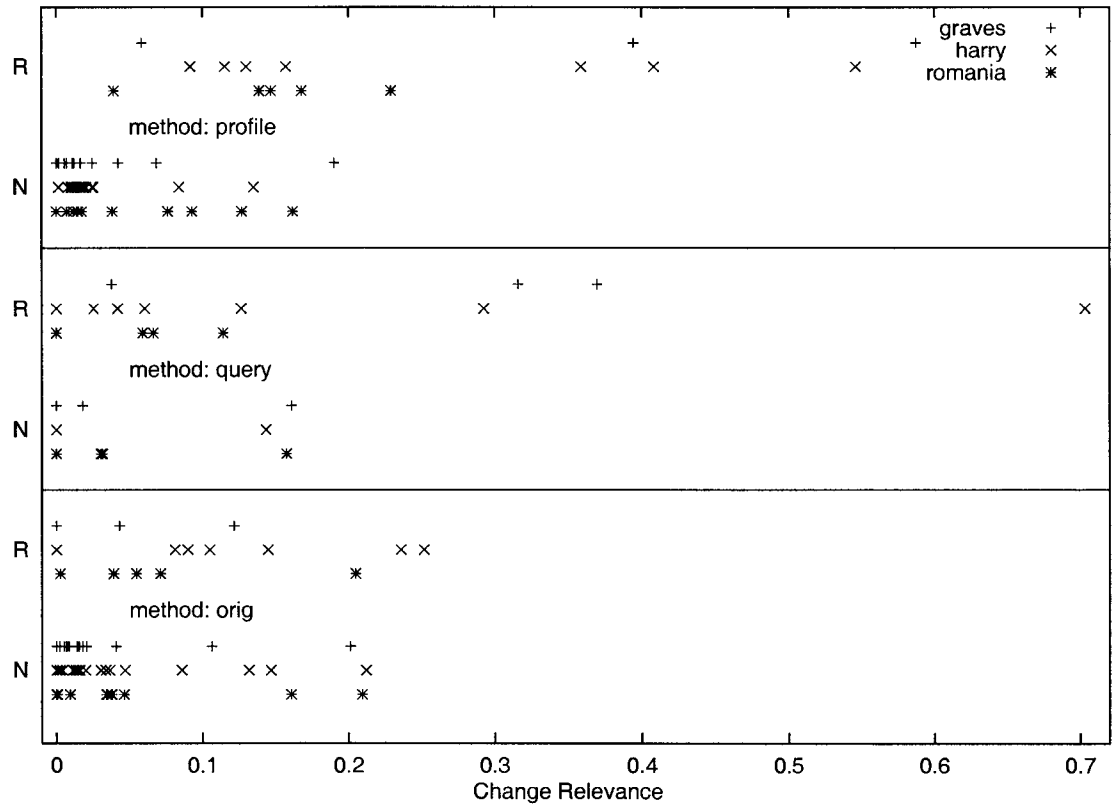


Figure 5.3: QueryTracker predictions for relevance of additions: similarity between V_{add} and profile, query, and orig, respectively

The number of pairs with relevant changes *rel* and non-relevant changes *nrel* for each query are shown in the respective columns of Table 5.2. We also present the threshold ranges and subsequent ranges for the number of false positives fp, false negatives fn, recall and precision.

Change relevance predictions generated by QueryTracker were plotted against user's Boolean relevance judgments: additions in Figure 5.3, and deletions in Figure 5.4. From these results, it appears that relevant and non-relevant changes are relatively easy to separate, allowing QueryTracker to do a good job of predicting relevant changes in HTML documents. Both the **profile** and **query** methods work well on predicting the relevance of additions to the original document, while they yield mixed results on predicting the relevance of deletions. On the other hand, the **orig** method works poorly on additions,

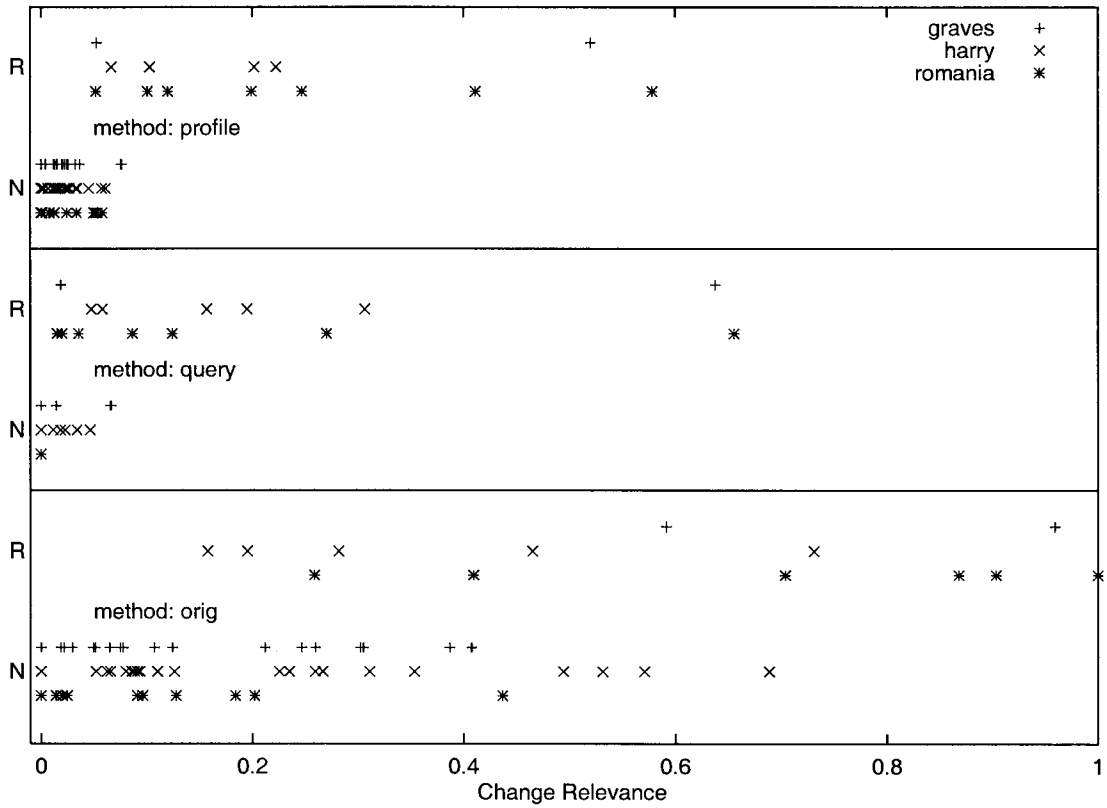


Figure 5.4: QueryTracker predictions for relevance of deletions: similarity between V_{del} and profile, query, and orig, respectively

but yields better results on deletions. It has been our observation that users most frequently care about the relevance of additions, where **profile** (or, in the absence of a user profile, **query**) are the best choice. However, if both addition and deletion relevance detection are important, some weighted combination of **profile** or **query** with **orig** seems to be needed to fully address the problem of predicting the relevance of both directions of change in HTML documents. Addition and deletion relevance detection are complementary operations, as they base their predictions on different data (V_{add} and V_{del}, respectively).

5.2 Redundancy Detection

Documents on the Web are often replicated at several different sites (mirroring), or the same document is reproduced with slight variation in versions by multiple sites (aliasing). Search engines do not typically check for the occurrence of mirroring and aliasing, and, if a query matches, return all of them as different results. We are interested in eliminating this type of redundancy before disseminating such documents to the users.

While also interested in true redundancy detection (in a semantic sense, where information contained in a more recent relevant document has already been covered in earlier documents), our main goal is to eliminated mirrored and aliased Web pages. We compare a redundancy detection algorithm proposed by Zhang et al.[150] (described in Section 3.2.3) to our own lightweight method. Zhang's method essentially consists of learning *two* thresholds: one for dissemination of relevant documents, and a second, higher one, for pruning documents which are *too similar* to other previously disseminated documents (and thus expected to be redundant).

Our proposed alternative method is based on the change detection technique outlined in the previous section. When a new document is encountered (and determined to be relevant), we compute V_{add} between it and a set of previously disseminated documents. If V_{add} itself passes the relevance test, we assume the new document is novel; otherwise, the new document is considered redundant. This method has the main advantage of transparently using the underlying filtering algorithm, without requiring a similarity metric to compare two arbitrary documents to each other. Another advantage is that we do not require a secondary threshold, with the added complexity of its learning mechanism.

5.2.1 Experimental Setup

Our experiment is designed to measure how well QueryTracker is able to predict whether a newly encountered document is made redundant by a previously seen one. In addition to our original six queries from the previous section (**flushots**, **harry**, **graves**, **romania**, **scheduling**, and **vorticity**), we collected redundancy feedback on seven more queries (from four more users) covering a wide range of user selected topics:

agents *information+agent+filtering+tracking+text*, 53 of 147 documents relevant,

multimedia *text+image+information+retrieval+multimedia*, 50 of 145 documents relevant,

kidvac *kid+friendly+resort+vacation+beach*, 59 of 169 documents relevant,

mp3edu *music+piracy+riaa+college+campus+user+education*, 284 of 340 documents relevant,

scosuit *linux+ibm+sco+lawsuit*, 213 of 235 documents relevant,

diabetes *gestational+diabetes*, 124 of 230 documents relevant,

genre_ir *genre+classification+information+retrieval*, 13 of 84 documents relevant.

From all documents judged as relevant by our users, we constructed all possible pairs of documents (d_1, d_2), both on the same topic, where d_1 was disseminated *before* d_2. After computing the similarity between all pairs, we sampled 90 documents from each original query (only 78 pairs were available for **flushots**), and only 30 from each of the additional queries (which received less feedback from their users). One third of the sampled pairs were taken from among the most, median, and least similar pairs, respectively. We then presented these pairs to our users in random order and asked them to specify whether the second document in a pair is made redundant by the first one. The

number of selected pairs was picked such as to make this task manageable for the users. Two of our original queries (**flushots** and **vorticity**) yielded no examples of redundant document pairs.

We measure the performance of redundancy detection using metrics introduced by Zhang et al.[150], based on the concepts of precision and recall. Redundancy recall (*r_recall*) is the fraction of redundant document pairs correctly identified by the system. Redundancy precision (*r_prec*) is the fraction of the pairs predicted by the system to be redundant that were actually correctly labeled as redundant. Redundancy mistake (*r_mistake*) is the fraction of the total documents that were mis-classified. Knowing the user-supplied number of redundant pairs *red*, and the number of false positives fp and false negatives fn, we compute redundancy recall and precision thus:

$$r\_recall = \frac{red - fn}{red}$$

$$r\_precision = \frac{red - fn}{red - fn + fp}$$

$$r\_mistake = \frac{fp + fn}{total\_documents}$$

Just as with regular recall and precision, we use *r_HM* (the harmonic mean of *r_recall* and *r_prec*) as our overall performance metric to be optimized.

5.2.2 Results

User feedback regarding redundancy is plotted against similarity measured between document pairs in Figure 5.5. Only topics containing at least one redundant pair are plotted. This figure shows how diverse the similarity rankings are across the queries. Some of the queries (e.g., **kidvac** and **mp3edu**) produced an almost bimodal distribution of similarity values, while others (e.g., **genre_ir** and **romania**) covered most of the range for non-redundant documents. This shows that similarities may vary widely across topics in general, documents that are redundant still stay in a narrow range with high similarity as measured by our metrics.

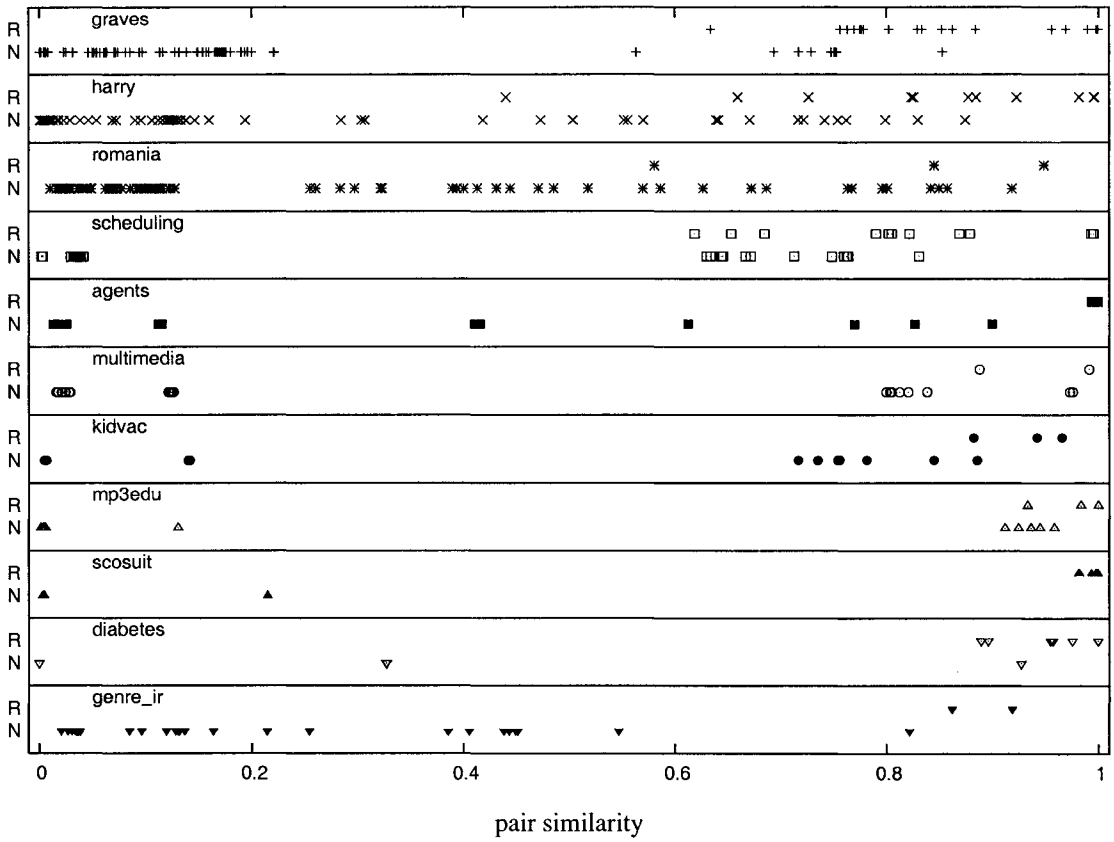


Figure 5.5: Redundancy vs. similarity (R = redundant, N = not redundant) on pairs for which users provided feedback

In Table 5.3, we present the number of redundant *red* and non-redundant *nred* pairs for each remaining topic, and the applicable redundancy threshold ranges and subsequent ranges for false positives fp, false negatives fn, recall *r_recl* and precision *r_prec* for Zhang's method. The redundancy threshold ranges were computed similarly to the dissemination thresholds in the previous experiment on change relevance. The data show that redundancy detection is clearly an issue in a Web monitoring application; 3.33% to 33.3% (with a mean of 15%) of the documents were marked as redundant.

In Figure 5.6, we plot performance for both Zhang's (similarity-based) and our add-vector based method. Also shown on the plot is the filtering performance obtained when separating relevant from non-relevant documents in the overall collection. Relevant and

Query	*red*	*nred*	threshold	*fp*	*fn*	*r_recl*	*r_prec*	*r_HM*	*r_mistake*
graves	20	70	.599-.858	9-0	0-11	1-0.450	0.690-1	0.816-0.621	0.100-0.122
harry	12	78	.430-.876	17-0	0-6	1-0.500	0.414-1	0.585-0.666	0.189-0.066
romania	3	87	.575-.934	12-0	0-2	1-0.333	0.200-1	0.333-0.500	0.133-0.022
scheduling	12	61	.331-.850	12-0	0-8	1-0.333	0.500-1	0.666-0.500	0.164-0.109
agents	4	26	.900-.993	0-0	0-0	1-1.000	1.000-1	1.000-1.000	0.000-0.000
multimedia	2	28	.863-.984	2-0	0-1	1-0.500	0.500-1	0.666-0.666	0.066-0.033
kidvac	3	27	.864-.914	1-0	0-1	1-0.666	0.750-1	0.857-0.800	0.033-0.033
mp3edu	3	27	.929-.971	4-0	0-1	1-0.666	0.429-1	0.600-0.800	0.133-0.033
scosuit	10	20	.220-.980	0-0	0-0	1-1.000	1.000-1	1.000-1.000	0.000-0.000
diabetes	9	21	.608-.941	1-0	0-2	1-0.777	0.900-1	0.947-0.875	0.033-0.066
genre_ir	2	28	.822-.861	0-0	0-0	1-1.000	1.000-1	1.000-1.000	0.000-0.000

Table 5.3: Study of threshold ranges for detecting redundancy in selected document pairs; metric ranges are given for meaningful redundancy threshold ranges

non-relevant documents are best separated (w.r.t *HM*) when the dissemination threshold is between 0.2 and 0.35. Zhang's algorithm would perform optimally when it learned a redundancy threshold of 0.83 ($r\_HM = 0.73743$, $r\_recall = 0.81481$, $r\_precision = 0.67346$, $r\_mistake = 0.06225$). Our add-vector method (implemented using TF-IDF) is optimized when the dissemination threshold used for V_{add} is close to 0.1 ($r\_HM = 0.40650$, $r\_recall = 0.61728$, $r\_precision = 0.30303$, $r\_mistake = 0.19337$).

At its best, Zhang's method outperforms our add-vector algorithm. However, this is partially explained by a bias in our data collection. Originally intended for the evaluation of how Zhang's method performs when used with QueryTracker, we collected top, median, and bottom ranked document pairs *by similarity between documents* and saw correlation between this similarity and document redundancy. However, data collected this way has the potential of introducing a bias against the add-vector method, which might be more fairly evaluated using a *random* sample of document pairs.

It is important to note that, with redundancy detection systems, false positives (non-redundant, interesting document pairs mistakenly classified as redundant) cause the system to discard information, whereas false negatives (redundant pairs mistakenly consid-

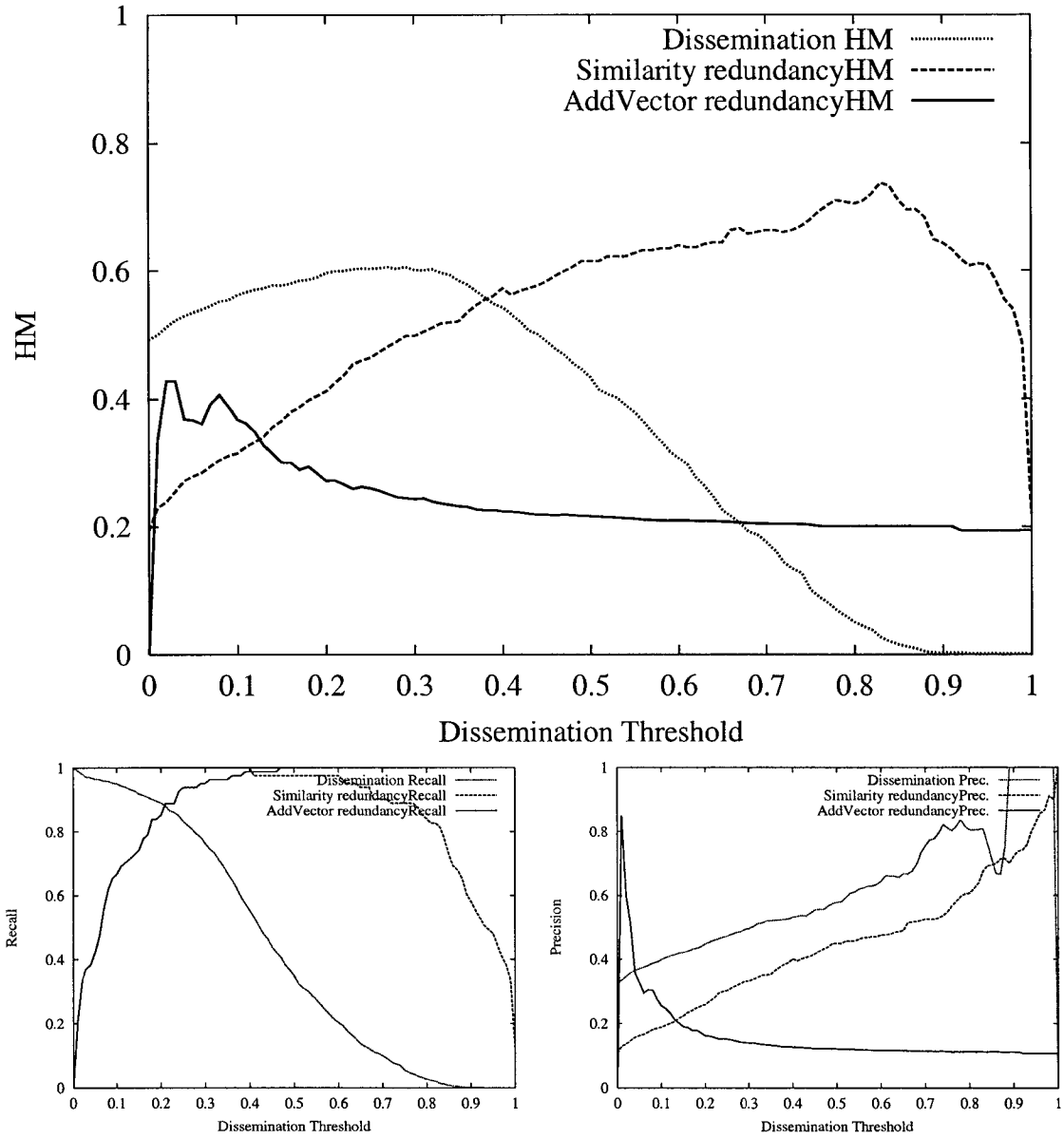


Figure 5.6: Effect of threshold on Harmonic Mean, recall, and precision

ered non-redundant and disseminated to the user) will cause noise and annoyance to the user. We examined the most similar non-redundant pairs and found the following:

- For **graves**, **scheduling**, and **kidvac**, the most similar non-redundant pairs consist of unrelated hub-type documents, each containing a large number of links; many of the links are identical, and technically, one of the documents could have been

considered redundant. The users noted that even though much of the content overlapped, they marked the pair as non-redundant because all the descriptions associated with each reference were worded differently.

- For **agents**, **multimedia**, and **genre_ir**, many of the hits were CiteSeer [32] pages reviewing various papers dealing with the respective topic. Many false positives were papers with common authors, with very little difference between them. These papers often were largely the same content published in different venues (e.g., journal versions of workshop papers).

- For all queries, clusters of non-redundant documents with high similarity (≥ 0.6) mostly consist of hub documents, containing several links to various articles, plus a large amount of identical boilerplate website design material. While the links are different (which is why the users chose to mark the pairs as non-redundant), the high similarity is caused by the boilerplate content which is identical between pairs.

Our data show that the redundancy threshold can be set between 0.8 and 0.9, without risk of missing out on too many relevant, non-redundant documents; targeting for high redundancy precision, we can obtain good redundancy recall, from 0.33 to 0.5, which is consistent with the results in [150]. These results show that the method can be easily adapted to Web documents (although with room for improvement). Additionally, the method can deal with documents generated from individually biased streams; our documents were derived from search engine queries specified and judged by users based on their own information need.

5.3 Summary

From the three change detection methods we tested, **profile** and **query** are helpful in detecting relevant additions to a document, and **orig** helps detect relevant deletions. Verbal feedback indicates that users assign a higher importance to detecting new information that is added to a relevant Web document. We also tested two second stage filtering algorithms for detecting redundancy [150], which can successfully eliminate a significant portion of duplicate documents posted to the Web under different addresses and also identify differently worded, but extremely similar documents.

Our results show that these methods can carry over to the Web. However, future work will address the Web specific discrepancies in the definitions of a "relevant" document (e.g., whether to include hub pages with links to relevant articles, or only the relevant articles themselves) and a "redundant" one (e.g., if the latter document is totally different from the former, but provides the same information, should it be considered redundant?). In order to further reduce false positives in redundancy detection, future work should investigate methods of eliminating boilerplate HTML design elements and focusing on structure based objects as in Liu et al.[88] before parsing the document text. The false positives observed in academic articles, for example, might be further reduced by automatically inferring query specific thresholds.

Chapter 6

Generating Search Engine Queries

Up to this point, we have explored filtering techniques, including change and redundancy detection. Another important component of a personalized Web information agent is a document gathering mechanism, which is tasked with finding and supplying candidate documents to the filtering component, which then decides whether to disseminate them to the user.

Web document gathering techniques currently in use include focused crawling (see Section 3.1.3), monitoring well-known hub sites for links to interesting content, and filtering through documents returned by a search engine query (see Section 3.2.1). Of these, we find the latter to be most interesting, as they offload a significant portion of the effort to a large, centralized entity, while still remaining able to preserve user privacy and personalization. Documents are harvested from a larger pool than what could be crawled by a client-side agent, and, if queries are well chosen, the returned document set would still be richer in relevant items than a random crawl.

In this chapter, we explore techniques that generate a search engine query based on a learned user profile (typically represented by a complex TF-IDF vector or set of Bayes probabilities). The search engine query is expected to optimize the relevance of search engine results with respect to the topic represented by the user profile. We evaluate the quality of query generation mechanisms indirectly by evaluating the relevance of search

engine results obtained using the generated queries. We are interested in answering the following main questions:

1. *How do different query generation methods perform relative to each other?* For our case, query generation involves constructing queries of specified length based on a user profile; these are then submitted to a search engine for the purpose of harvesting documents.

2. *Is positive-only relevance feedback adequate to support the task?* To minimize user overhead, we prefer to solicit only *positive* relevance feedback. Obviously, this provides relatively weak knowledge, requiring the profiling mechanism to self-organize the categories of interest and to trade-off precision.

3. *Can a user profile be learned independently of the service?* In other words, the profile may be built from documents collected via *other* means than by giving feedback on the performance of the service in question (e.g, training documents supplied a priori, or feedback on the performance of some other service). If so, then user overhead can be minimized and multiple services provided as part of the same agent.

Out of several query generation algorithms compared, we find three that perform well, both by extracting sufficient numbers of documents from the search engine, and in terms of the relevance of returned links. Positive-only relevance feedback does indeed incur a performance penalty, which, however, may be worthwhile due to the increased convenience for users.

We implement several different methods suggested by Ghani et al.[52] (described in detail in Section 3.2.2). In addition, we use deterministic and probabilistic extraction of highest weight terms from TF-IDF profile vectors. The complete list of methods used is given below:

Uniform (Unif) baseline case, select n terms with uniform probability;

Boley select n terms from the intersection of the k top ranking terms according to term frequency, and k top ranking terms according to document frequency;

TF select n top ranking terms according to term frequency;

Probabilistic TF (P-TF) select n terms with probability proportional to their term frequency;

OR select the top ranking n terms according to their odds-ratio score;

Probabilistic OR (P-OR) select n terms with probability proportional to their odds-ratio score;

TF-IDF select n terms with the highest TF-IDF weight from the profile vector;

Probabilistic TF-IDF (P-TF-IDF) select n terms with probability proportional to their TF-IDF weights;

The probabilistic versions are included because injection of small amounts of randomness was found to be helpful in related research by Ghani et al.[52]. Specifically, they found that probabilistic TF outperformed straightforward TF in their early studies, which motivated them to try a probabilistic version of their Odds-Ratio (OR) method.

For each topic, we collected and/or computed the following data, which was subsequently used to rank terms according to each query generation method:

- average term frequencies for terms in relevant documents;
- document frequencies for terms in relevant documents;
- TF-IDF vector built from relevant documents;

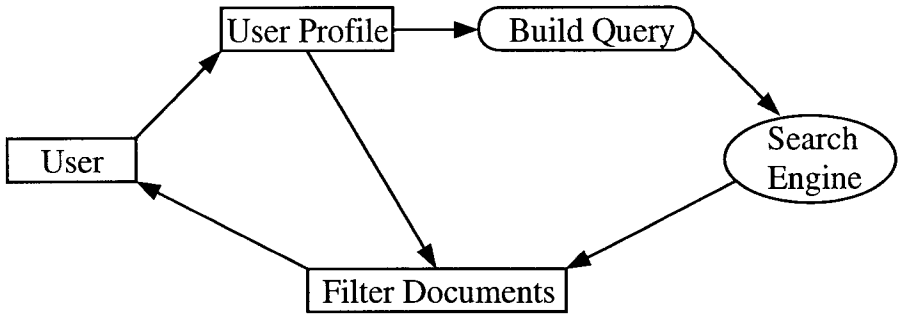


Figure 6.1: Evaluation protocol for query generation techniques

- odds-ratio scores OR_t for terms t in relevant documents; odds-ratio scores are based on both relevant and non-relevant documents related to the topic; the following formula shows how the odds-ratio score is computed:

$$OR_t = \log_2 \frac{P(t|rel) \cdot (1 - P(t|nonrel))}{P(t|nonrel) \cdot (1 - P(t|rel))}$$

6.1 Comparing Query Generation Methods

We wish to find both the method and query length that would optimize the relevance of the documents returned by the search engine with respect to the user profile. Methods where the query length is computed on the fly (along with the query itself) may also work well, although, according to our observations, increasing the query length too much (past eight terms) might be detrimental to recall. We conduct a factorial experiment with query generation method and query length as the independent variables, and response consisting of the two evaluation metrics: return count and relevance. The basic protocol for our study (illustrated in Figure 6.1) consists of the following steps:

1. construct user profiles,
2. generate queries from those profiles using each of the query generation methods,
3. submit queries of different lengths to Google, and evaluate results.

As in some of our other studies, we expedite experimental control and data collection by constructing user profiles based on data from TREC disk #5. We hand-picked topics with reasonable number of known relevant documents, consistent with the amount of training expected to be provided by a user building a profile. Since the TREC #5 disk contains two main document collections (FBIS and LATIMES), we selected one topic from each collection. Our first topic is on airport security measures and was constructed based on 46 relevant documents which appeared in the Los Angeles Times over the course of two years (1989-1990); this topic will be referred to as LATIMES-412. The second topic is on genetic research and was constructed based on 55 relevant documents from the Foreign Broadcasting Information Service appeared in 1996; this topic will be referred to as FBIS-82.

Having established the impact of negative feedback in Chapter 4, TREC data also allows us to test the scenario in which negative feedback is available for query generation, and assess how much performance might be sacrificed by restricting feedback to only positive examples. The number of positive documents used in the construction of each topic (46 and 55, respectively) is realistic compared to what users would have been capable of providing while building their profiles in real life. The method which requires negative feedback (OR) is based on probabilities computed as part of the Bayes profile (see Chapter 4).

From this information, we generated queries of seven lengths (two to eight terms) for each of the eight methods. For the four probabilistic methods, we generated 10 queries of each length, which means their reported results will be averages over those 10 queries.

For Boley's method, we iterated through increasing values of k, and intersected the sets consisting of the top k ranking terms by TF and DF respectively. The resulting intersection was used as the search engine query. We retained all distinct queries of

length between two and eight. For FBIS, no value of k generated a query of length $n = 6$. Similarly, for LATIMES, no value of k generated a query of length $n = 7$.

We generated a total of 488 queries (244 for each topic[1]). We submitted these queries to Google and collected the top 10 returned hits (first pageful of results), which were subsequently retrieved and stored locally.

We discarded (and decremented our hit count accordingly) all dead links and all hits that were in a format other than ASCII (including all variants and versions of HTML, XML, etc.) or PDF: a total of 312 out of 2917 hits were discarded. The PDF documents were converted into ASCII using the `pdftotext` utility. All resulting documents were then converted into TF-IDF vectors.

6.2 Results

To avoid a manual relevance evaluation of each returned document, we use TF-IDF cosine similarity between the document vector and the corresponding topic vector as a heuristic measure of the document's relevance. This decision introduces a bias in favor of TF-IDF-based query generation (this bias will be removed in two subsequent studies).

For each combination of query generation method and query length, we recorded the number of hits received, the relevance of the best hit, and the average relevance over all hits received. For the probabilistic methods, these measurements represent average values obtained over the ten repetitions for each combination of parameters. The results are summarized in Table 6.1 for the FBIS topic, and Table 6.2 for the LATIMES topic. The three rows corresponding to each method indicate average relevance (top), maximum relevance, and number of returned hits (bottom).

All methods return at least seven documents with query lengths of 2, but most taper

[1]Each probabilistic method had 7×10 queries/topic; non-probabilistic methods had 7 queries/topic each, except for Boley, which had 6.

method		query length 2	3	4	5	6	7	8
Unif	avg:	.022	.046	.059	.018	—	—	.011
	max:	.051	.077	.101	.019	—	—	.011
	cnt:	7.7	4.3	3.2	0.4	0	0	0.1
P-TF	avg:	.026	.054	.044	.053	.069	.091	.082
	max:	.059	.096	.079	.102	.120	.192	.138
	cnt:	8.9	7.7	7.9	5.2	6.5	7.2	6.3
P-OR	avg:	.039	.047	—	.019	—	—	—
	max:	.099	.090	—	.019	—	—	—
	cnt:	9.0	3.2	0	0.1	0	0	0
P-TF-IDF	avg:	.045	.058	.088	.069	.035	.034	.030
	max:	.100	.110	.178	.097	.054	.035	.055
	cnt:	9.1	6.1	8.4	2.4	2.7	0.6	1.4
Boley	avg:	.053	.077	.090	.081	—	.111	.088
	max:	.120	.112	.136	.168	—	.239	.239
	cnt:	9	9	10	7	0	8	9
TF	avg:	.036	.031	.048	.082	.081	.087	.083
	max:	.065	.059	.129	.134	.103	.130	.135
	cnt:	10	9	10	9	9	10	9
OR	avg:	.123	.186	.102	—	—	—	—
	max:	.155	.361	.190	—	—	—	—
	cnt:	9	8	2	0	0	0	0
TF-IDF	avg:	.100	.144	.160	.176	.214	.278	.242
	max:	.377	.377	.377	.279	.399	.404	.242
	cnt:	9	10	10	7	10	4	1

Table 6.1: Average relevance, Maximum relevance, and count of returned hits for the FBIS topic on genetic technology

off in the number returned with longer query lengths. For the deterministic methods, the relevance increases as the query length increases (until 7 or 8), but the relevance for the probabilistic methods tends to plateau early.

All methods consistently outperform the baseline uniform term selection. Probabilistic methods are outperformed by the non-probabilistic ones, which is consistent with the observations in Ghani et al.[52]. The best results for the FBIS topic were obtained using TF-IDF at query length 7: a total of 4 hits are returned, with an average

method		query length 2	3	4	5	6	7	8
Unif	avg:	.012	.012	.012	.013	.004	—	—
	max:	.024	.024	.028	.019	.006	—	—
	cnt:	8.0	5.3	3.9	1.0	0.6	0	0
P-TF	avg:	.017	.026	.025	.028	.032	.024	.010
	max:	.042	.073	.062	.061	.046	.042	.011
	cnt:	9.1	9.5	6.0	6.5	2.0	4.0	0.7
P-OR	avg:	.017	.018	.016	.011	—	.007	—
	max:	.052	.039	.029	.013	—	.007	—
	cnt:	8.2	8.3	4.0	0.9	0	0.1	0
P-TF-IDF	avg:	.026	.036	.064	.063	.059	.020	.010
	max:	.058	.103	.125	.156	.106	.036	.014
	cnt:	9.2	8.1	8.1	5.7	5.3	1.3	0.2
Boley	avg:	.040	.098	.135	.193	.199	—	.167
	max:	.086	.199	.299	.343	.359	—	.299
	cnt:	8	9	8	8	8	0	7
TF	avg:	.107	.058	.030	.048	.041	.069	—
	max:	.222	.093	.051	.075	.069	.069	—
	cnt:	7	10	10	7	6	1	0
OR	avg:	.048	.036	.348	—	—	—	—
	max:	.122	.096	.402	—	—	—	—
	cnt:	9	9	2	0	0	0	0
TF-IDF	avg:	.115	.144	.155	.171	.144	.153	.143
	max:	.331	.331	.357	.299	.276	.349	.349
	cnt:	9	7	8	8	9	9	9

Table 6.2: Average relevance, Maximum relevance, and count of returned hits for the LATIMES topic on airport security

relevance of .278, and a maximum relevance of .404. The best results for the LATIMES topic are obtained using OR at query length 4: two hits are returned, with average relevance of .348 and maximum relevance of .402. The difference in results is likely caused by a more narrower focus of the LATIMES topic, which also explains the better performance of simple TF-based query generation on this topic.

Query lengths 2 and 3 are the only ones where all methods lead to non-empty returns for both topics. To test whether the differences between methods are significant, we

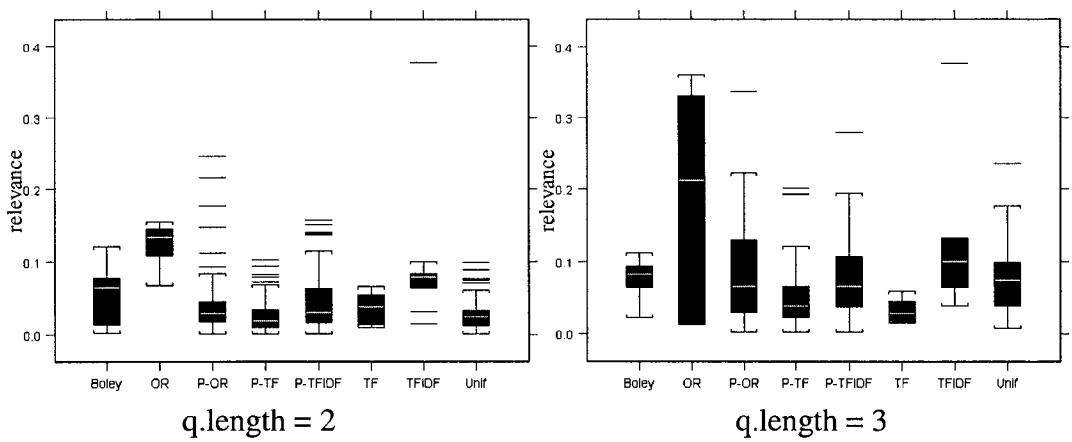


Figure 6.2: Box plot of relevance by method for FBIS topic at query length 2 and 3

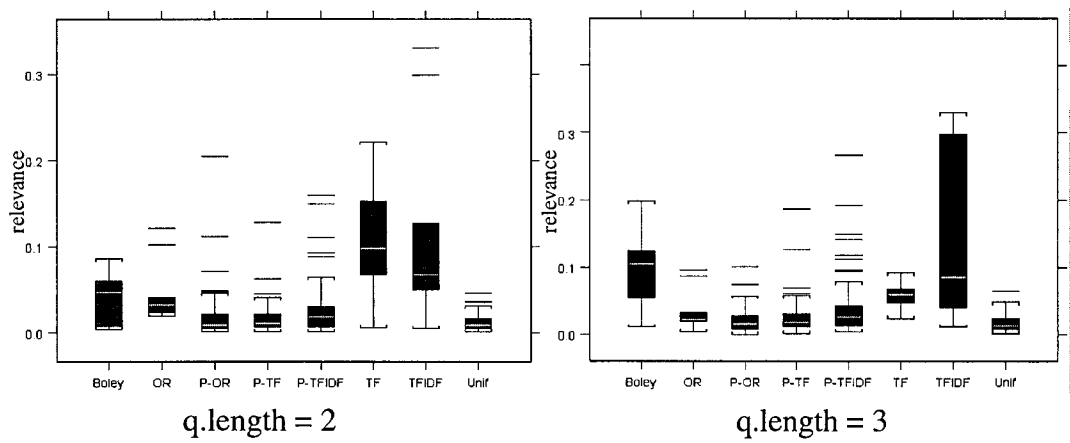


Figure 6.3: Box plot of relevance by method for LATIMES topic at query length 2

perform an ANOVA on each topic at query lengths 2 and 3, with the query generation method as the independent variable, and relevance as the dependent. The effects are significant in each case: for FBIS, we obtain $F = 14.007$, $p < 0.001$ at query length 2, and $F = 8.692$, $p < 0.001$ at query length 3; for LATIMES, we obtain $F = 24.027$, $p < 0.001$ at query length 2, and $F = 20.277$, $p < 0.001$ at query length 3, meaning that the methods produce distinctly different results.

Box plots of relevance by method for query lengths 2 and 3 are given in Figure 6.2 for FBIS, and Figure 6.3 for LATIMES. Note that medians rather than means are marked on the graph. These plots illustrate several points obscured by the previous table. First,

while TF-IDF returns the best match and second best median in all cases, overall better results are obtained using OR for FBIS, and TF and Boley for LATIMES. Second, each method can show high variance in the results, although TF-IDF tends generally to higher variance. Additionally, the results for query length 3 have higher variance than those for query length 2. Finally, the distributions are often skewed. For example, Boley consistently has a higher median than mean.

Because relevance for all returned documents is measured against the TF-IDF vector of the corresponding topic, the experiments are biased in favor of the TF-IDF query generation method. Our experiments cannot prove that TF-IDF query generation is sufficient, but its good performance coupled with the not always good performance of OR suggests that we do not incur a significant loss by leaving out negative feedback.

Both Boley and TF-IDF consistently result in many links being returned, even for long query lengths. Many hits are desirable when the agent will be pruning out previously visited sites and filtering the returned links according to a filtering threshold.

When the length of the generated query increases, so does the "focus" of the returned documents. If the correct terms are picked, the relevance of the results is increased. When the selected terms are incorrect, result relevance is sharply decreased. The main caveat of these conclusions is that they are based on the behavior observed for only two queries. Chapter 7 presents a more detailed exploration of query generation behavior.

6.3 Removing TF-IDF Bias

To gain an understanding of the performance of TF-IDF query generation without the bias present in our experiments with synthetically generated topics, we also conducted a study with one topic built by a real user (a fellow CS graduate student), containing 34 documents on stress relief, obtained from the user's bookmarks. The user profile was limited to using a TF-IDF vector (to limit solicited feedback to positive documents

method		query length 2	3	4	5	6	7	8
TF-IDF	#rel:	8	3	4	4	4	—	—
	cnt:	10	10	10	10	9	0	0
P-TF-IDF	avg. #rel:	1.0	1.8	0.7	0.4	0.0	0.5	—
	avg. cnt:	9.5	8.1	7.7	4.1	1.3	1.3	0.0

Table 6.3: Number of relevant documents and count of returned hits for the user-generated topic on stress relief

only). Query generation was in turn limited to the TF-IDF and P-TF-IDF methods. We followed the same protocol as in the previous section: query lengths were between 2 and 8, and results were averaged over ten trials for the probabilistic version P-TF-IDF. A total of 369 documents were returned by Google, 343 of which were distinct (a 5.5% overlap). We shuffled these documents and asked the user to examine each one of them, marking the ones found to be relevant to the original topic. 56 documents out of the total of 343 were found relevant. Table 6.3 presents the number of relevant documents and the number of hits returned for each parameter combination.

This study supports the hypothesis that TF-IDF based queries will generate an adequate incoming stream: queries of length up to six returned at least nine hits from Google. A more comprehensive user study, which removes the bias introduced by the evaluation metric used in this chapter, is presented in Chapter 7.

6.4 Summary

We studied several methods for generating queries from a user profile to answer three questions related to the design of Web information agents. First, how do different query generation algorithms perform relative to each other? In fact, we observed significantly different performance among the eight methods tested. Overall, Boley, TF-IDF and to a lesser extent TF provided a good number of hits and relatively high relevance.

Second, is positive relevance feedback adequate to support the task? We found that leaving out negative training examples does not incur a significant performance loss in terms of expected relevance of returned documents. Odds-Ratio was found to excel on one topic, but its competitive advantage does not appear to be worth the additional overhead expected from the user. TF-IDF and Boley, requiring only positive relevance feedback, generated queries that resulted in relevant hits.

Third, can user profiles be learned independently of the service? The results from TF-IDF and the pilot experiment are suggestive. However, the pilot study also suggests that either user relevance judgments may be a bit lower (harsher) than the automated method or that the profile may not adequately reflect the users' interests. In fact, the good performance of Boley and TF indicates that in some cases it might be worthwhile to collect more than TF-IDF information from the user-supplied positive training examples. This last question will be examined in more detail in the future.

Chapter 7

Filtering and Query Generation: a User Study

Our prior experiments mainly used simulated user interaction, with newspaper articles from benchmark collections in lieu of real Web documents. We must consider whether those results generalize to our target application/context. This chapter presents a user study where we obtain true relevance judgments, thus also eliminating the bias introduced in our earlier query generation experiment, where similarity of new documents to the user profile was used in lieu of their actual (unknown) relevance. We seek answers to the following questions:

1. Which query generation methods works best, and how does it compare against the original, user-supplied query? We expect Odds-Ratio to slightly outperform TF-IDF, based on our earlier study from Chapter 6.

2. Which filtering method works best? Given our results in Chapter 4, we expect the Naive Bayes Classifier (NBC) to outperform thresholded TF-IDF.

3. Is there an interaction between query generation and filtering, and, if yes, which combination performs best?

4. Would performance be helped if users were allowed to supply their own training samples (in addition to giving feedback on documents disseminated by Query-

Tracker)? We expect additional user-supplied training to give an extra "jump-start" to the user profile, leading to increased performance.

7.1 QueryTracker Test Setup

QueryTracker is configured to monitor multiple user queries over time. When users submit training data (feedback to documents disseminated by the agent, or examples of relevant documents found independently by the user), a profile is created. This profile serves a dual purpose: query generation, and document filtering.

In addition to the original user-supplied query, the profile is used to build up to three additional queries. We use Odds-Ratio and TF-IDF (described in Chapter 6), as well a Bayesian method which selects top-ranking terms according to their prior probability ratio ($P(t|+)/P(t|-)$). In Chapter 6, Odds-Ratio came out slightly ahead of TF-IDF, in spite of the evaluation being biased in favor of the latter. Since this time relevance judgments are provided directly by the users, this study will remove the bias introduced in our earlier experiment.

For text filtering, we use TF-IDF and the NBC when a corresponding profile already exists, and SearchRank (a method based on the ranking of documents returned by the search engine) in the absence of a user profile.

QueryTracker submits user queries to a search engine (Google, in this case) every 24 hours. For each query, the first 100 returned hits are filtered for relevance, and the ones that pass through the filter are disseminated to the users. In turn, users may provide feedback which is used to train a user profile. Before a profile is created (i.e., before the first training sample is submitted by the user), filtering is performed using a document's search engine rank, based on the expectation that the search engine lists documents in what it perceives to be their decreasing order of relevance to the query. In order to allow rank-based filtering to use a threshold learning mechanism similar to the one employed

with TF-IDF, the rank of each document is converted into a score ranging between 0 and 1, according to the following formula:

$$rank\_score = 1 - \frac{rank - 1}{max\_rank - 1}$$

where $max\_rank = 100$ is the rank of the last returned search engine hit we accept for consideration. The top returned document will have $rank\_score = 1$, and the last (100^{th}) one will have $rank\_score = 0$. We set an initial dissemination threshold for this method (henceforth referred to as SearchRank) to 0.9, guaranteeing the dissemination of the top 10 returned documents. This threshold is adaptively adjusted based on the rank of the supplied training documents, just like the TF-IDF dissemination threshold is adapted based on the similarity between training documents and the current TF-IDF profile vector.

Once positive feedback becomes available for a query, QueryTracker creates a TF-IDF profile vector and starts using TF-IDF filtering in addition to SearchRank. TF-IDF based generation of an alternative query is also started at this point. A separate dissemination threshold set using the *min-max* method with $\alpha = 0.05$, as recommended by the results in Section 4.3.

Both SearchRank and TF-IDF filtering use probabilistic dissemination: a document is disseminated with certainty if its score exceeds the threshold *thr*, and with a decreasing probability otherwise. The dissemination probability is computed using the following formula:

$$P(dissemination) = \begin{cases} 1, & \text{if } score > thr \\ 1 - \frac{\sqrt{thr^2 - score^2}}{thr}, & \text{otherwise} \end{cases}$$

where *score* is either *rank_score* for SearchRank, or similarity to the profile vector for TF-IDF. Probabilistically disseminating documents that would otherwise be pruned by the dissemination threshold allows for a certain amount of exploration and novelty among the disseminated documents.

When *both* positive and negative feedback are available, an NBC (see Section 4.3) is built in addition to the TF-IDF profile, and is also used for filtering. Bayesian and Odds-Ratio query generation is also started at this time. The NBC does not require a threshold, but instead disseminates a document if its likelihood of being relevant is greater than that of it being non-relevant. Our three filtering techniques are used in a logic-OR fashion: a document is disseminated if *either* method supports its dissemination.

7.2 Filtering and Query Generation User Study

To answer the questions posed at the beginning of this chapter, we conduct a study in which seven distinct users submitted a total of 19 different queries (summarized in Table 7.1) to QueryTracker. Four of these users also participated in our previous experiments. Queries were monitored over a period of approximately one month each. Experiments were conducted in two installments, due to the availability of our test subjects. First, we collected results on a set of ten queries (from four different users), using only TF-IDF query generation in addition to the original user query. The second installment contains nine queries (collected from five distinct users), and alternative queries were generated using all three methods: TF-IDF, Odds-Ratio, and Bayesian. In both cases, filtering was performed using SearchRank, TF-IDF, and NBC.

Users could access the QueryTracker Web service to monitor their queries at their convenience. They were not required to judge their queries daily. When the users accessed their results, they were asked to provide feedback on the disseminated documents by marking them either relevant or non-relevant. When documents were not marked one way or the other, it was assumed that users had not read them yet, and they continued to be treated as *new* rather than *previously seen* the next time they were returned by the search engine (usually the following day when the query was re-submitted). An example of the form used to solicit user feedback is shown in Figure 7.1.

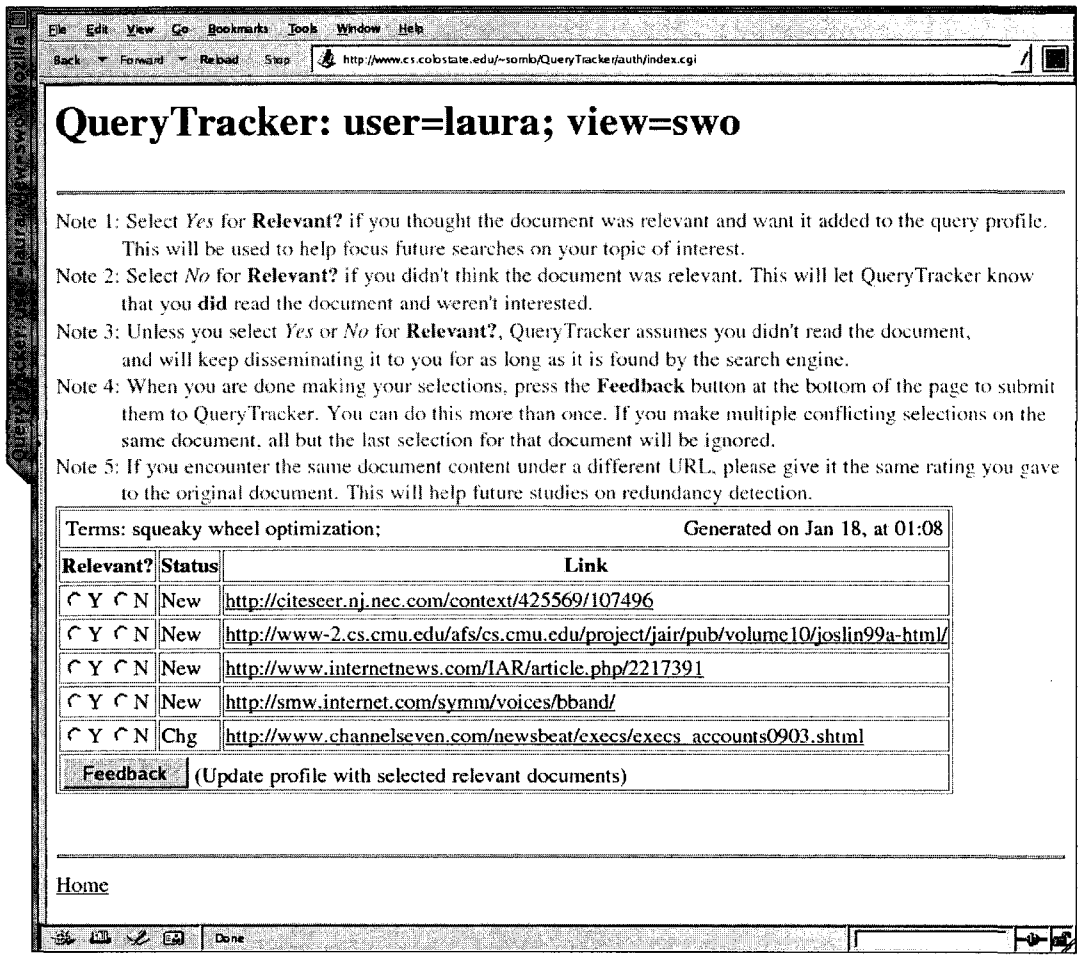


Figure 7.1: View of disseminated documents and feedback input form

For each disseminated document, we recorded the type of query that led to its retrieval (Original, TF-IDF, Odds-Ratio, or Bayesian), and all filtering methods that voted to disseminate the document (SearchRank, TF-IDF, or NBC). A document was disseminated if at least one of the filtering methods' vote was favorable.

For each query type and method described above, we computed the following values:

RF number of relevant documents disseminated via the query type or dissemination method in question,

NF number of non-relevant documents disseminated (false positives),

pseudo RM number of relevant documents missed by the query type or method in question, that were found by some other method (false negatives).

Based on these numbers, we computed precision and (pseudo-)recall as:

$$\text{precision} = \frac{RF}{RF + NF}$$

$$\text{pseudo\_recall} = \frac{RF}{RF + pseudo\_RM}$$

Sound *RM* and recall cannot be computed without complete knowledge of (and random access to) the entirety of the documents indexed by the search engine. The values we compute using available data should, however, be sufficient to compare our query generation and dissemination methods *to each other*. Values for *pseudo_recall* and *precision* were used to compute a *pseudo_HM* value, similar to what was used in our previous experiments. For simplicity of notation, we will henceforth use *recall* and *HM* in our notation, with the understanding that they are only meaningful for comparing methods relative to each other, within the framework of this experiment.

7.3 Results

A list of the queries used, along with summary data such as number of days running, total relevant/non-relevant documents collected, and overlaps (documents disseminated by more than one filtering method) is presented in Table 7.1. We observe that only a small fraction of documents were disseminated by more than one filtering method.

7.3.1 Should the original query be modified?

We wish to assess whether the generated query helps performance. As shown by the last two columns in Table 7.1, the number of overlapping documents is a small percentage of the total returned. Generated queries clearly produce documents not returned by the original query, thus potentially contributing to increased overall recall. We first examine

batch	query #	topic/area of interest	# days running	total rel	total non	overlap rel	overlap non
	1	educ. softw.	10	39	98	1	1
	2	agents	9	41	54	3	0
	3	multimedia	10	39	62	0	0
	4	scheduling	10	25	65	1	0
1	5	vacations	15	72	149	3	0
	6	genetic alg.	14	198	86	18	3
	7	optimization	16	76	161	0	0
	8	audio	5	51	67	0	0
	9	linux pda	21	39	286	0	0
	10	copyright	23	343	72	1	0
	1	multimedia	11	28	44	2	5
	2	scheduling	13	37	64	1	2
	3	veterinary	12	41	177	1	7
	4	veterinary	10	35	241	3	14
2	5	veterinary	14	59	222	2	5
	6	agents	9	77	34	13	7
	7	multimedia	11	32	52	2	5
	8	scheduling	13	30	63	1	1
	9	agents	9	79	57	8	11

Table 7.1: Summary data on user queries

the latter batch of nine queries, to compare all our different query generation methods. Boxplots comparing performance values obtained using each query generation technique are shown in Figure 7.2. To our (not unpleasant) surprise, TF-IDF query generation is the only method with reasonable performance, not only removing our earlier evaluation bias, but positioning TF-IDF as the clear winner in terms of recommended query generation technique. Mean *HM* for TF-IDF query generation is 0.258, with standard deviation $\sigma = 0.114$. The original query results in mean $HM = 0.539$ with $\sigma = 0.193$. For completeness, we confirm the strong effect of the choice of query generation method on *HM* performance via a one-way ANOVA, resulting in $F = 32.345$ at $p < 0.0001$. Performance is dominated by recall (ANOVA: $F = 115.633$, $p < 0.0001$), while precision does not differ much across query generation methods (ANOVA: $F = 3.292$, $p = 0.033$).

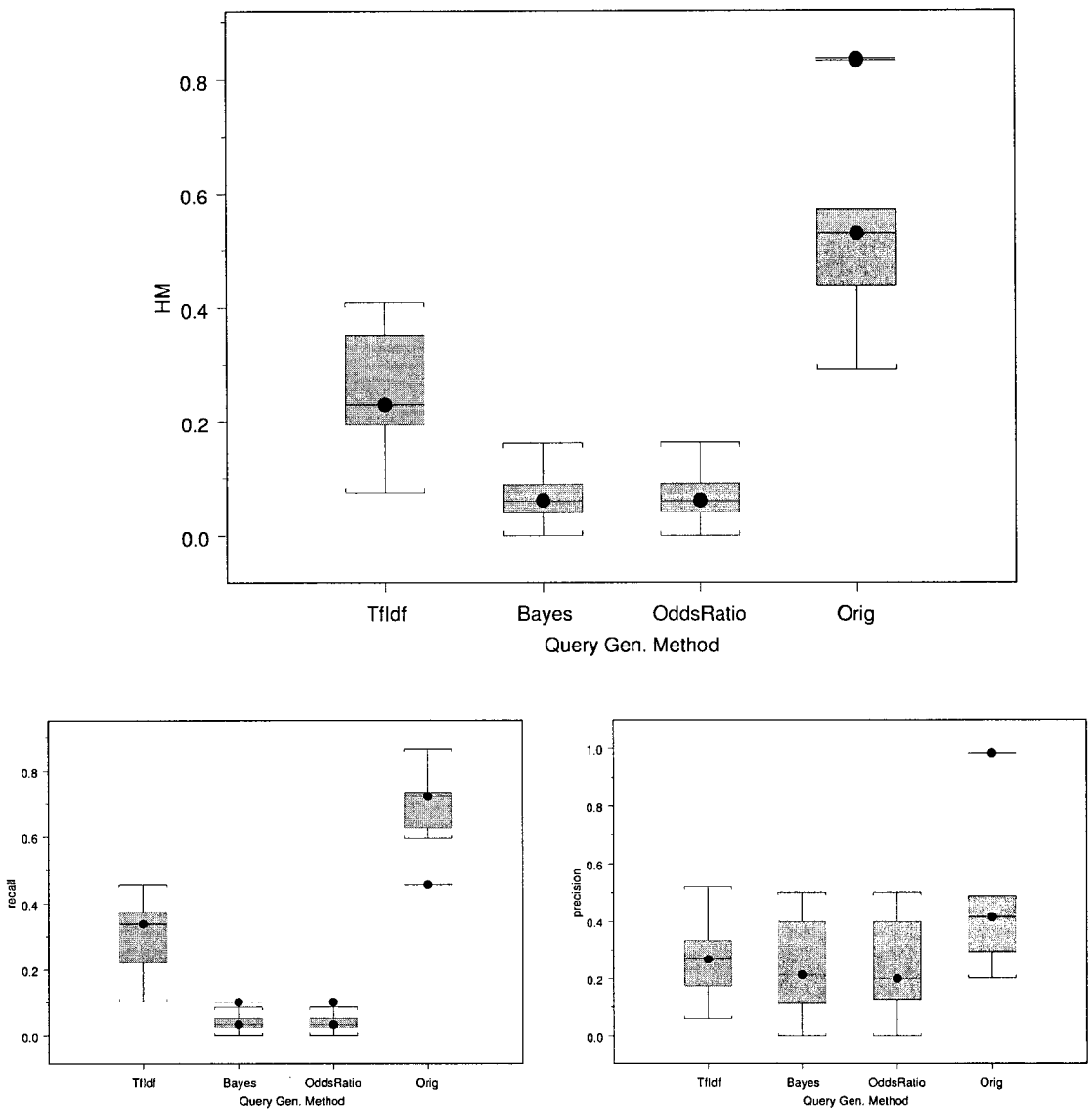


Figure 7.2: Performance comparison between query generation methods on first batch of 10 queries

Having established TF-IDF as our most viable query generation method, we proceed to examine the cumulative set of all 19 queries, since the use of Original and TF-IDF query generation is common across all of them. A new boxplot is shown in Figure 7.3. The relationship between Original and TF-IDF performance remains unchanged. Differences remain significant (a two-sample t-test yields $t = 4.788$ at $p < 0.0001$). Mean HM and σ are 0.279 ($\sigma = 0.197$) for TF-IDF, and 0.562 ($\sigma = 0.165$) for Original.

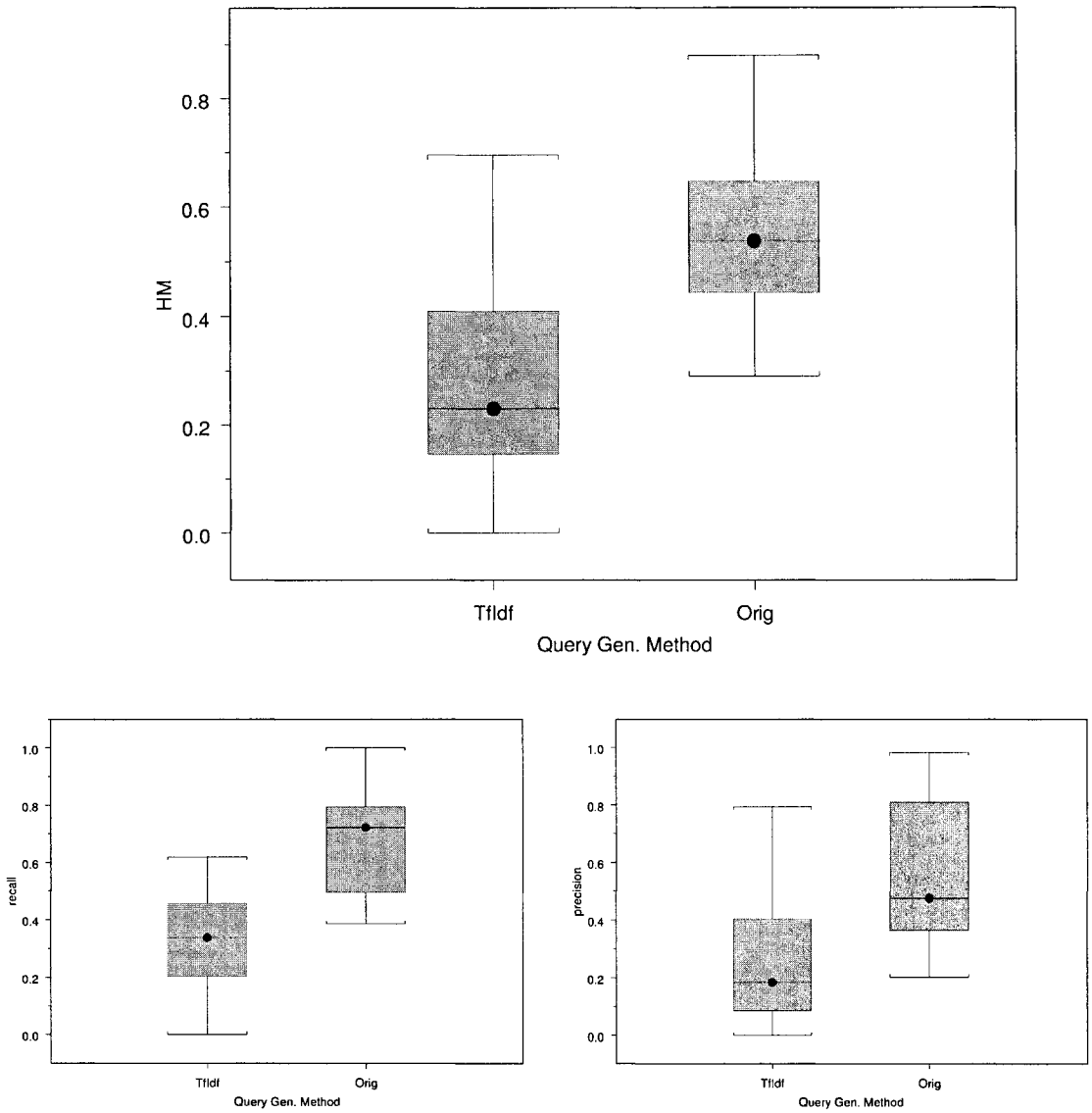


Figure 7.3: Performance comparison between Original and TF-IDF generated queries on entire data set

While our analysis suggests that generated queries perform worse than original ones, in four of the queries, the performance obtained using a TF-IDF generated query matched and even exceeded the performance from the original query, mainly because of improved recall. These queries also had more relevant feedback (i.e., "yes" judgments), leading to the hypothesis that, over time, performance of the generated query improves with training. An example using query #10 from our first batch (this query was supplied

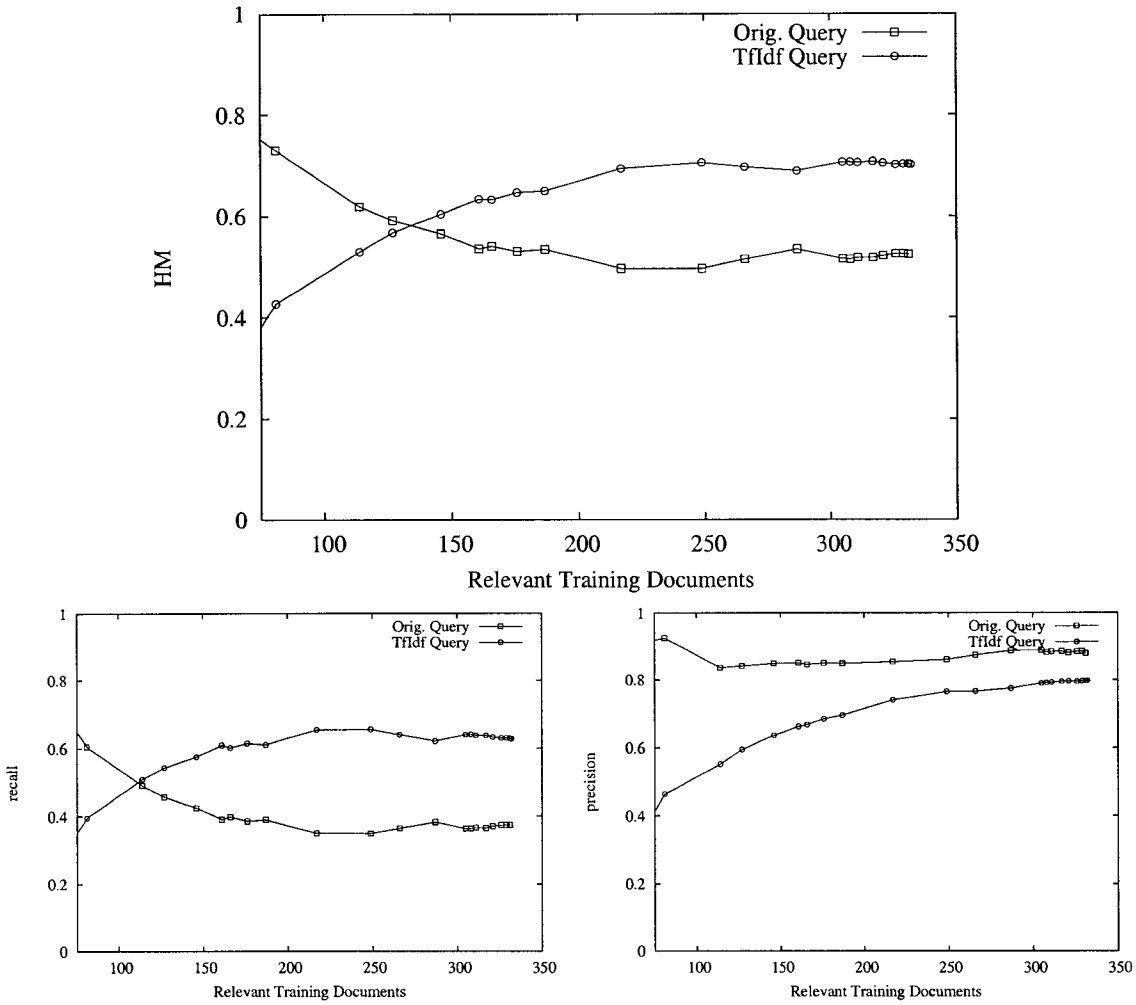
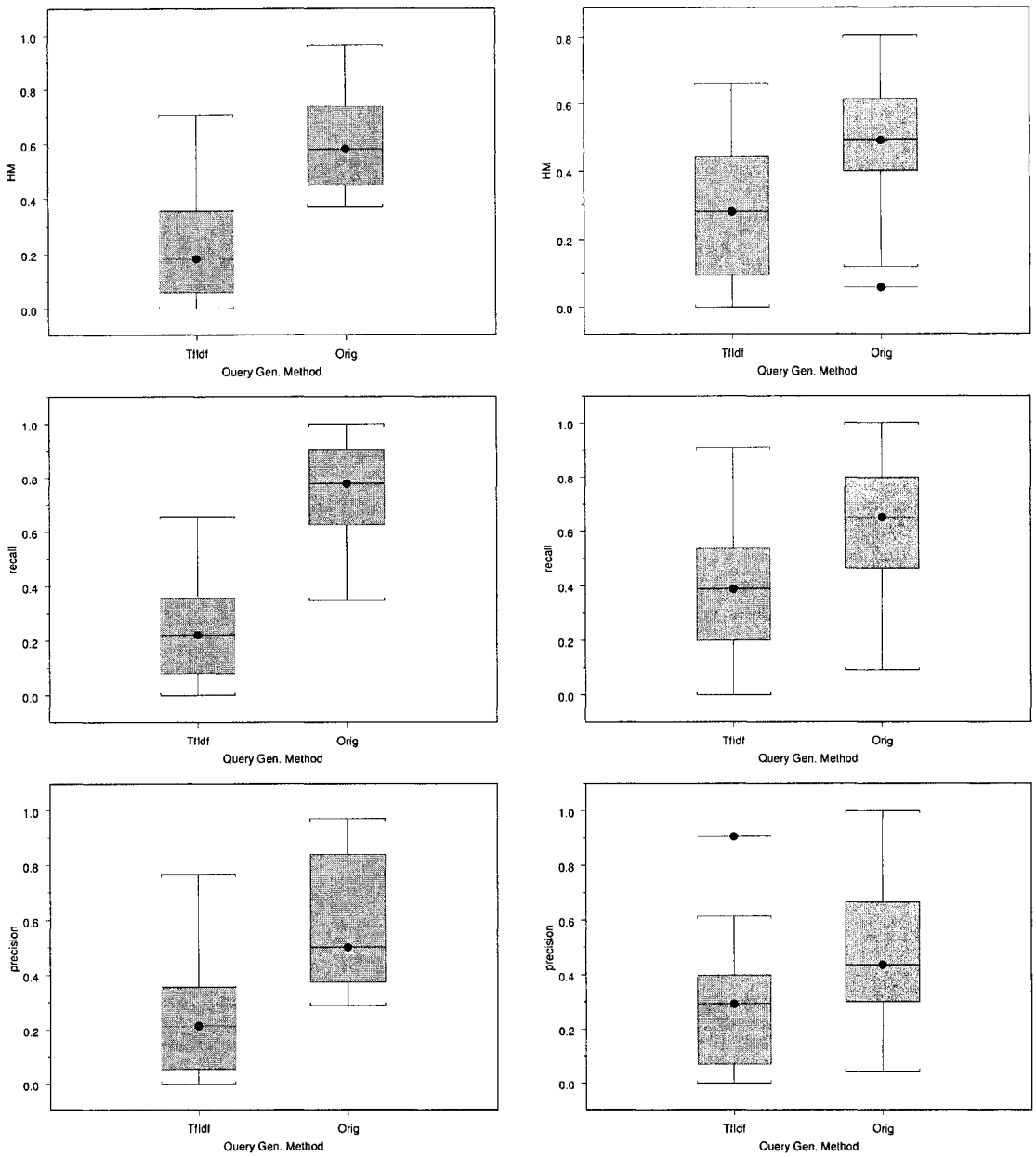


Figure 7.4: Evolution of original and TF-IDF-generated query performance (query #10)

with the most intensive training) is shown in Figure 7.4, where *HM* is plotted for both Original and TF-IDF after each day. Indeed, after day 5, the performance of TF-IDF query generation exceeds that of the user-supplied original.

In other cases, the performance of the generated query also improves with feedback, but never manages to outperform the original. In looking at individual queries, it appears that queries on narrowly focused topics are more difficult to improve, although, even then, the generated query leads to additional relevant documents to be found. To determine how much query generation performance changed over time, we plotted *HM*,



First Half Second Half

Figure 7.5: Performance comparison between Original and TF-IDF generated queries during first and second half of experiment duration

recall and precision split by whether data was gathered during the first, or, respectively, the second half of the query's lifespan. These plots are shown in Figure 7.5. Paired t-tests show that while the performance of the generated query increases significantly

in the second half of a query's lifespan, ($t = 3.94$, $p < 0.001$), the performance of the original remains the same ($t = 1.27$, $p = 0.219$).

In conclusion, once a reliable TF-IDF user profile is learned, the automatically generated query built from its top-weighted terms can become quite useful. Also, there is very little observed overlap between relevant documents found via the original query and those resulting from the generated alternative. A hypothesis explaining this observation is that the generated query explores different areas of the search engine's document space than the original user query: at first, because the wrong terms are selected while the user profile is still being learned; later, when a good profile is learned, the generated queries are focused on the truly relevant portions of the document space, which is different than what the search engine focuses on when seeing the original query. Generating alternative queries is therefore useful because it extends the pool of candidates from which the filtering algorithm can disseminate potential relevant documents.

When studying the evolution of generated queries over time, and comparing them to their respective originals, a few interesting observations become apparent. Queries generated using Odds-Ratio are frequently identical to those generated using Bayesian prior probabilities, which suggests that the two methods are more or less equivalent. For all query generation techniques, the query terms are completely unrelated to the original query during the first two to four days. During the following five to ten days, terms change frequently from one day to the other, and begin to include words which are recognizably related to the topic of the query. After that, change is much reduced, to possibly one term being replaced roughly every seven to ten days. This reflects the stage in which a user profile has mostly stabilized, and new feedback documents mostly solidify the agent's learned model of the topic of interest.

Since TF-IDF outperforms other query generation techniques due to better recall, we conclude that its term weighting scheme is better at picking out the really important

keywords from a user profile, thus leading to better results from the search engine. It is also worth noting that, for query generation, recall is more important than precision – after all the purpose of query generation is to gather documents to be submitted to the filtering mechanism.

7.3.2 Which filtering method is best?

Using the full set of 19 queries, we plot *HM* performance for all three filtering methods in Figure 7.6. As expected (based on earlier results from Chapter 4), NBC filtering performs best (mean $HM = 0.512$, $\sigma = 0.159$). SearchRank also provides adequate performance (mean $HM = 0.388$, $\sigma = 0.109$), and may be quite useful when filtering decisions must be made before users provide the necessary training to create a profile. TF-IDF performs rather poorly (mean $HM = 0.119$, $\sigma = 0.092$). A one-way ANOVA on *HM* by filtering method produces $F = 50.333$ at $p < 0.0001$. As expected, filtering performance is dominated by precision (ANOVA: $F = 103.938$, $p < 0.0001$), whereas recall is fairly uniform across methods (ANOVA: $F = 0.291$, $p = 0.748$).

We illustrate the evolution of filtering performance over time in Figure 7.7. It can be observed how Bayes learning is quite fast, SearchRank performance is relatively flat, and TF-IDF improves, albeit very slowly. Compared to the other filtering methods, TF-IDF suffers from low precision (too many false positives being disseminated).

7.3.3 How Do Query Generation and Filtering Interact?

To complete the picture, we search for an interaction effect between filtering method and query generation. Running a two-way ANOVA with *HM* as dependent variable, we obtain $F = 7.299$ at $p < 0.01$ for the interaction effect between query generation and filtering methods. Based on the boxplot in Figure 7.8, the overall winner is NBC filtering applied to results from the original user query. SearchRank filtering does a decent job in the absence of a user profile (which would then allow NBC to supersede it). TF-

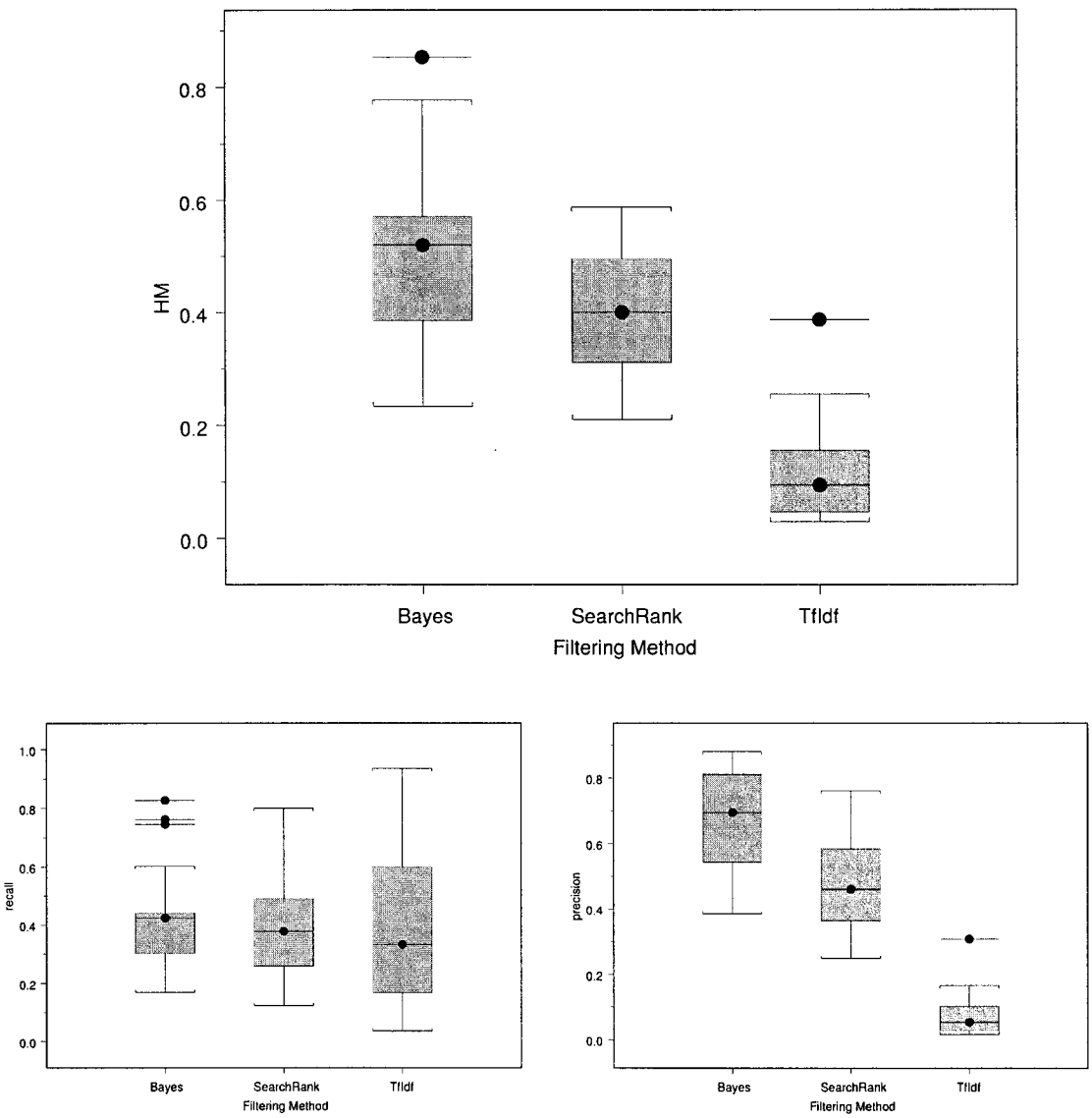


Figure 7.6: Performance comparison between filtering methods

IDF query generation provides a decent additional way of boosting recall, given the fact that it generates documents that have little overlap with those generated by the original query. We observe that, once again, precision is dominant (and mainly influenced by the choice of filtering method), with a smaller influence from recall (mainly influenced by the query generation method).

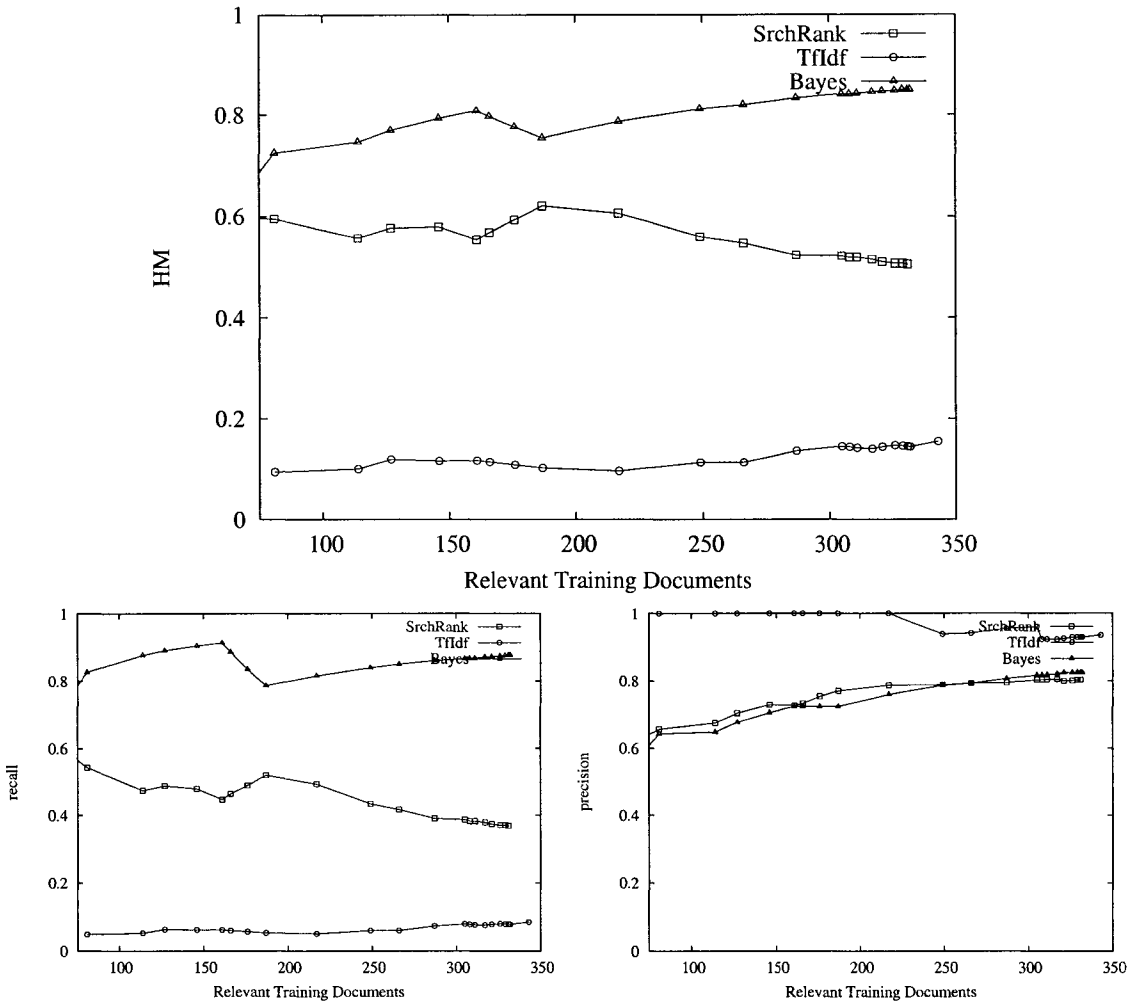


Figure 7.7: Evolution of filtering method performance (query #10)

7.3.4 Does Extra Training Help?

Only two of our users (on three different queries) used the feature allowing them to supply extra positive training for their QueryTracker profiles. Therefore, statistically significant results are currently unavailable for this question. While most of the time performance with extra feedback evolves similarly to the situation when extra feedback was not used, there are instances where performance during the first few days in the lifetime of a query is significantly boosted. As an example, *HM* performance for an example query (NBC filtering on results from original user query) is shown in Figure 7.9.

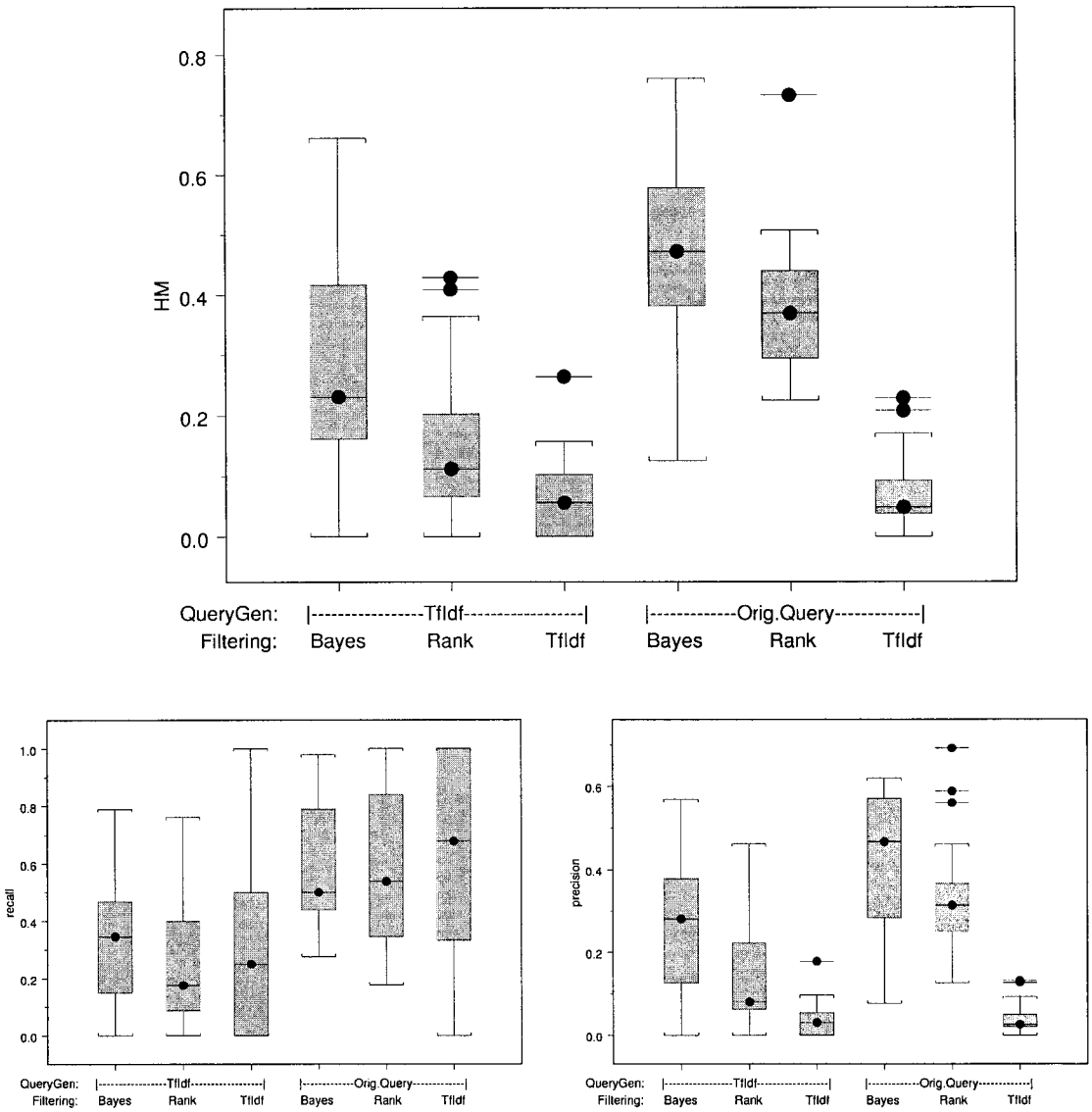


Figure 7.8: Performance comparison between combinations of filtering and query generation methods

The version of the query which received the extra feedback performed significantly better for the first 5 days, while the version without extra feedback needed the extra time to learn its profile from scratch. In general, TF-IDF generated queries differed on average by two terms in the beginning (between versions with extra training and those without). Towards the end of the trial, the average difference decreased to a single term out of a total of four per query.

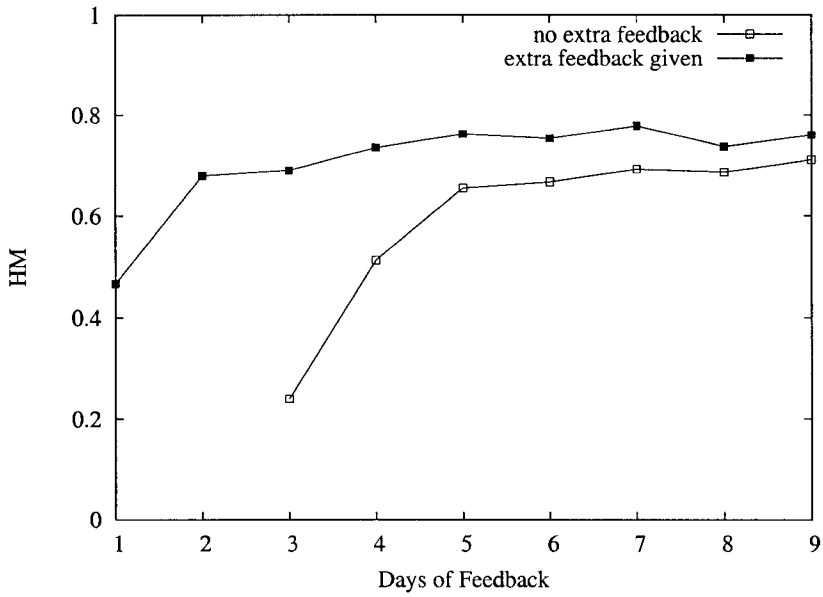


Figure 7.9: HM performance comparison for preload

7.4 Summary

Query generation based on a TF-IDF profile has good potential; the technique led to relevant results that had not been obtained by the original query. However, it appears to require significant amounts of feedback before becoming effective. Even so, we are pleasantly surprised that, after having removed the evaluation bias favoring TF-IDF in our earlier experiments, it not only didn't lose ground as compared to Odds-Ratio, but really turned out to outperform it significantly.

On filtering, we find that TF-IDF is rather slow-learning when compared to NBC and is often outperformed by the search engine's ranking. This is mainly due to TF-IDF being slow to improve its precision as compared to the other filtering methods.

Jump-starting the profile by allowing users to submit relevant examples of documents independently appears to help boost performance early on in the lifetime of the query, but more experiments would be required in order to achieve statistical significance of this claim.

Another interesting conclusion of this study is that negative training may be easier to obtain from users than we originally anticipated. In QueryTracker's usage scenario, in particular, it makes it easier for users to mark read but non-relevant documents in order to distinguish them from those remaining unread. Users expressed the requirement that unread documents keep showing up as new during subsequent iterations of retrieval from the search engine, until such a time when they are marked as either relevant or non-relevant.

Why does TF-IDF perform well for query generation and poorly for filtering, while the reverse is true for NBC? We think that, due to using negative feedback, NBC is quickly learning a linear separation function over the document space, which seems an adequate model for distinguishing between relevant and non-relevant documents. TF-IDF, on the other hand, quickly learns the most relevant position in the document space (which helps query generation), but needs a longer time to learn a boundary between relevant and non-relevant documents (which is shaped as a hyper-cone, due to the use of a dissemination threshold – see Figure 2.1).

To offer additional verification as to the adequacy of using linear separation to discriminate between relevant and non-relevant documents, we compared the NBC with an implementation of the k-nearest neighbors (kNN) classifier (with $k \in \{1,2,\ldots,9\}$), on the six TREC benchmark topics we used throughout our other filtering experiments. We chose kNN for its ability to learn non-linear boundaries between classes of documents, and thus offer insight into how much performance (if any) is lost due to our other methods being limited to learning linear boundaries. NBC leads to slightly worse precision, significantly better recall, and overall better performance than kNN. The value of k did not significantly influence performance across the observed range. kNN assumes random-access to all training samples, and thus severely violates our requirements for an incremental filtering system. However, the purpose of this comparison was to verify

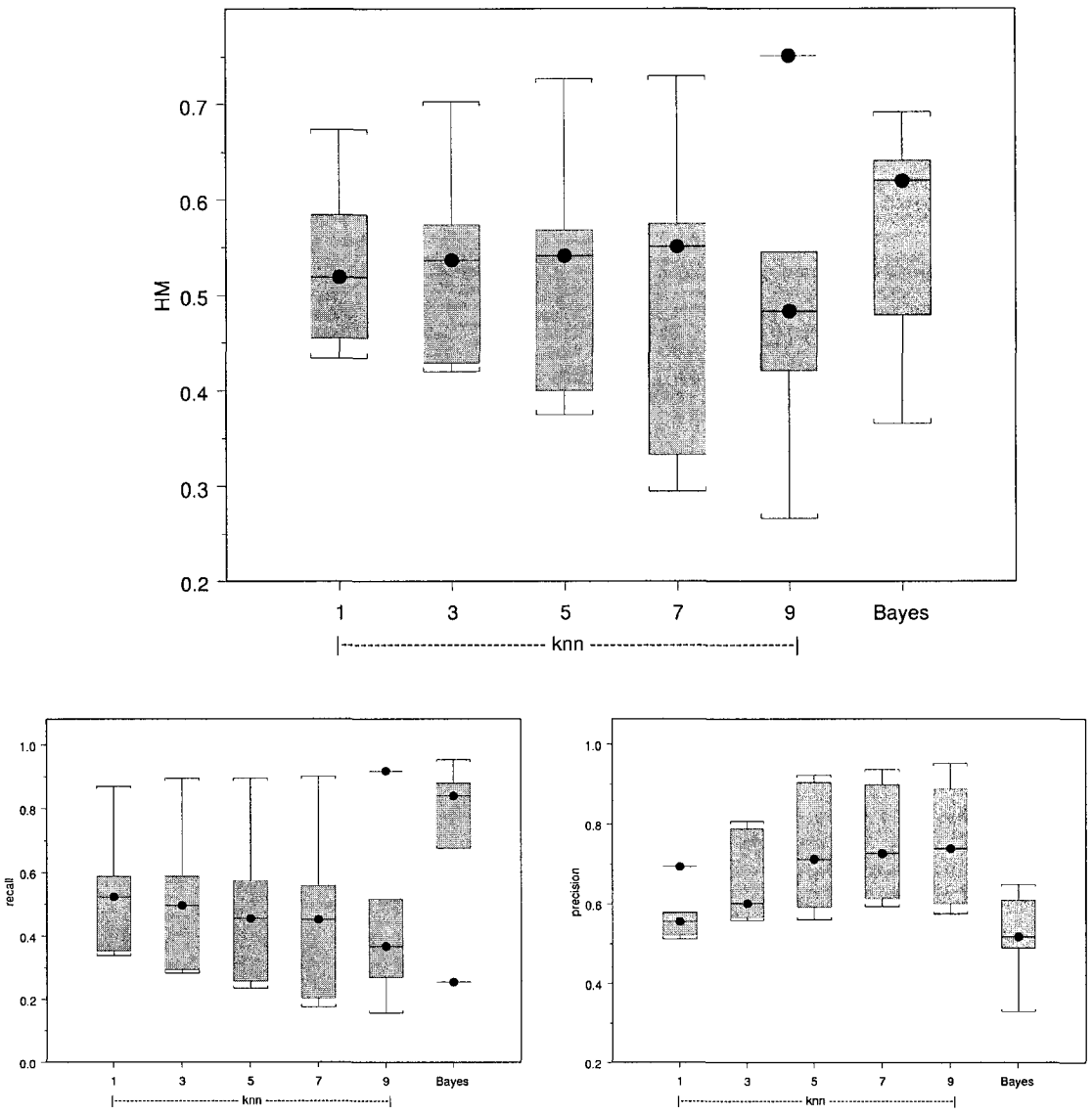


Figure 7.10: Comparing k nearest neighbors and the NBC on TREC data

whether a method capable of learning a non-linear separation function offers additional filtering performance, which did not happen in our case. The performance comparison is presented graphically in Figure 7.10.

Chapter 8

Summary, Future Work, and Conclusions

This dissertation describes a detailed approach to building a personalized Web information agent. We build on the strong parallels between IR text filtering and the machine learning techniques used with agents to offer personalized retrieval of Web documents.

The key contributions of this work include the development and evaluation of several lightweight filtering techniques for personalized information retrieval, change and redundancy detection in Web documents, and search engine query generation from a user profile. These techniques are evaluated individually and as an integrated system.

Our design goals are centered around maximizing personalization, privacy, and user friendliness (by reducing the amount of interaction originated by the agent) and minimizing the burden on local computing resources. The main conclusions we draw are that it is possible to build high-performance, lightweight text filters, especially when the operational environment facilitates the solicitation of both positive and negative training samples; that lightweight and efficient methods can detect when a document undergoes relevant content changes, or is made redundant by previously encountered documents; and that relevant search engine queries can easily be extracted from a TF-IDF user profile and used to supplement the incoming document stream for significantly improved recall.

8.1 User Profiles and Filtering

TF-IDF. Based on one of our main design goals (agent should require an absolute minimum amount of attention from the user), we started out by selecting TF-IDF as our main method for filtering and representation of the user profile. Only relevant training examples are required, and relevance of new documents is predicted based on similarity to the profile and a dissemination threshold. We devised an adaptive mechanism for learning the threshold which, unlike other work in the area, does not require access to (and storage of) past training samples. We conducted a factorial experiment to deduce the optimal values of parameters such as threshold learning rate, weighting of documents added to the profile TF-IDF vector, and to assess the evolution of filtering performance as a function of the amount of training provided.

Implicit Topics. Another step toward improving the quality of the user experience was to eliminate the requirement that users provide a topic for each relevant training sample given as feedback to the agent. We implemented and empirically evaluated several incremental clustering algorithms which were used as mechanisms to automatically *learn* the appropriate user topics. We demonstrated that the Doubling algorithm outperforms Greedy incremental clustering (used with existing Web agents), experimentally verifying existing theoretical results [26].

Bayesian Filtering. We compared TF-IDF filtering to a system based on the Naive Bayes Classifier, to assess how much performance is lost by ignoring negative feedback. We also presented and evaluated a modification allowing a Bayes classifier to automatically select its negative class from unlabeled training samples (a simplified adaptation based on [83]). We found that each algorithm has different strengths and weaknesses. While Bayes learns much faster than TF-IDF, it does so by soliciting negative training samples. When modified to automatically select pseudo-negatives from among unlabeled samples, it requires access to (and, therefore, storage of) past training documents.

8.2 Change Relevance and Redundancy

Web information agents often encounter the same document repeatedly, at different points in time. Existing systems typically maintain a history list of previously encountered documents and ignore documents that are already on the list. The unfortunate side effect of this is that potentially relevant changes to such documents are also ignored. We devised a technique which efficiently evaluates the relevance of such changes, and allows for previous documents to be re-disseminated if the changes are found relevant.

A related problem is the frequent mirroring and aliasing of Web documents. While existing research on redundancy detection is mainly concerned with *semantic* redundancy, we used one such technique to address the more immediate (and tractable) problem of duplicate content. We also proposed our own lightweight and efficient method for detecting duplicate content, which uses the underlying filtering mechanism without requiring an extra threshold and learning mechanism.

8.3 Query Generation

Frequently, Web agents will formulate and submit queries to search engines as an economical way to generate a document stream from which to filter relevant documents based on their user profile. We provided a systematic study of several existing methods for extracting search engine queries from user profiles. Our experiment used pre-judged benchmark data to build a profile, and TF-IDF similarity as a measure of the quality of documents returned by the search engine. Our results showed that using both positive and negative examples (Odds-Ratio) has a slight advantage over positive-only profiles (TF-IDF). However, our quality measure introduced a bias in favor of TF-IDF, which may have skewed our experimental results.

8.4 Interaction of Filtering and Query Generation

We studied the interaction of filtering and query generation in a real-world, live deployment of the QueryTracker system. Users' Web queries were monitored over a period of several weeks, and new results were filtered based on profiles built from user feedback on previous results. We devised a method allowing us to compare the performance of various filtering and query generation methods based on precision, recall, and their harmonic mean, even in the absence of full relevance knowledge of documents on the Web. While true recall values cannot be computed without such knowledge, we used *relative* recall computed with respect to relevant documents found by *competing* methods, which was sufficient to compare all competitors to each other.

Our experiments not only removed the bias in comparing TF-IDF and Odds-Ratio query generation, but, to our surprise, showed that TF-IDF generated queries are the clear winners, trailing only the original user queries with respect to retrieval performance. Query generation performance is dominated by recall, and generated queries retrieve fewer relevant documents as compared to the original. While generated queries do not typically outperform user's originals, we showed a strong increase in performance with the amount of training provided by users. Also, since there is very little overlap between documents retrieved by the original query and the generated one, using both will guarantee an improvement in combined recall values.

We also reconfirmed TF-IDF's slow learning curve as compared to NBC filtering. Our TF-IDF implementation was successfully used for filtering scenarios where hundreds/thousands of documents were used for training. In a personalized Web agent, users cannot be expected to provide this much feedback, which leaves the NBC as a strong contender. Another observation made during this experiment is that negative training may be easier to elicit from users than we originally thought. When presented with a large number (tens) of disseminated documents daily, users will typically only

manage to read (and rate) a few of them (cca. 10). If only positive feedback is required (accepted) by the agent, unread documents become indistinguishable from non-relevant ones. Allowing the users to mark a document as read (but non-relevant) allows unread documents to keep showing up as newly disseminated in subsequent iterations, while at the same time avoiding re-dissemination of the same non-relevant document over and over again.

8.5 Limitations and Future Work

The filtering work described in Chapter 4 is heavily tilted toward TF-IDF, based on our original assumption that users would be inconvenienced by having to provide negative feedback. In retrospect, it turns out that users *are* willing to provide negative feedback under the right circumstances. However, they are not willing to provide *large amounts* of feedback. This illustrates another point of contrast between centralized systems which process many (tens of thousands) of documents in one run on one hand, and personalized filtering systems which deal with tens of documents at once (several orders of magnitude fewer documents). Similarly, our work on incremental clustering was performed within the same TF-IDF context.

Our Bayesian filtering was performed by building one classifier for each distinct topic, with a positive (relevant) and negative (non-relevant) class. In a future experiment, we plan to build a filter that uses a single Bayesian classifier for all topics. Each topic will be assigned a class, which is where relevant documents for that topic will be placed. Non-relevant documents will be assigned to a separate, additional class. Incremental clustering may then be used to create, manage, and merge automatically learned topics/classes. This technique will hopefully combine the learning speed of Bayesian classifiers with the convenience of automatically inferred topics of interest contributed by incremental clustering.

We used a "bag-of-words" approach to feature extraction from documents. A term frequency vector was built from each document, by simply counting how often each distinct term occurred within the document. Other approaches use more complex weighting schemes which take into account the position and graphical characteristics (e.g., font size, emphasis, etc.) of a term within a document. A generally applicable method which takes cues from document layout in addition to simply counting terms may further improve our systems' performance.

It would also be interesting to gain more theoretical insight into the differences between TF-IDF and Bayesian profiles. So far, it appears that Bayesian filters outperform thresholded TF-IDF by quickly learning a boundary between relevant and non-relevant regions of the document space, but without having (or requiring) a high-quality representation of the ideal relevant document. Thresholded TF-IDF learns such a representation well, but has trouble separating relevant documents from non-relevant ones. Future experiments will use dual-vector TF-IDF profiles which, according to our results in Chapter 4, perform as well as Bayesian classifiers on filtering tasks, yet have the same good representation of the relevant document class as thresholded TF-IDF.

Practical uses of this work might include organizations monitoring their own Web presence for changes in relevant content or ratings, various types of intelligence gathering (counter-terrorism, business, research, etc.).

8.6 Conclusion

This dissertation presents a framework for the design and implementation of a Web information gathering agent. Although many different Web agents already exist, our research goes one step further and provides a thorough analysis and evaluation of the architecture, individual components and building blocks, and their overall interaction. Drawing on the strong relationship between IR and text filtering on one hand, and ma-

chine learning and agents on the other hand, we implement and evaluate a set of building blocks required to build a personalized Web information agent.

REFERENCES

[1] J. Allan. Incremental relevance feedback for information filtering. In *Proceedings of the 19th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

[2] AltaVista. The altavista search engine. http://www.altavista.com.

[3] G. Amati. Probabilistic learning for selective dissemination of information. *Information Processing & Management*, 35(5):633–654, 1999.

[4] A. Arampatzis. Unbiased s-d threshold optimization, initial query degradation, decay, and inceremental ity, for adaptive document filtering. In *Proceedings of the Tenth Text Retrieval Conference (TREC-10)*, Gaithersburg, MD, USA, 2001.

[5] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Resources*, Stanford, CA, USA, 1995.

[6] S. Babu and J. Widom. Continuous queries over data streams. *ACM SIGMOD Record*, 30(3):109–120, 2001.

[7] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[8] D. Barbara. The characterization of continuous queries. *International Journal of Cooperative Information Systems*, 8(4):295–323, 1999.

[9] 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, 1992.

[10] T.A.H. Bell and A. Moffat. The design of a high performance information filtering system. In *Proceedings of the 19th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

[11] T. Berners-Lee. A short history of web development. http://www.w3.org/People/Berners-Lee/ShortHistory.html.

[12] K. Bharat and M.R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[13] D. Boley, M. Gini, R. Gross, E. Han, K. Hastings, G. Karypis, V. Kumar, M. Bamshad, and J. Moore. Document categorization and query generation on the world wide web using webace. *AI Review*, 13(5-6):365–391, 1999.

[14] A. Bookstein and D.R. Swanson. Probabilistic models for automatic indexing. *Journal of the American Society for Information Science*, 25(5):312–318, 1974.

[15] J. Boyan, D. Freitag, and T. Joachims. A machine learning architecture for optimizing web search engines. In *Working Notes of AAAI-96 Workshop on Internet-Based Information Systems*, Portland, OR, USA, 1996.

[16] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[17] A.Z. Broder, S.C. Glassman, M.S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara, CA, USA, 1997.

[18] C. Buckley, G. Salton, J. Allan, and A. Singhal. Automatic query expansion using smart: Trec 3. In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, Gaithersburg, MD, USA, 1994.

[19] J. Budzik and K. Hammond. Watson: Anticipating and contextualizing information needs. In *Proceedings of the 62nd Annual Meeting of the American Society for Information Science*, Washington, DC, USA, 1999.

[20] J. Callan. Document filtering with inference networks. In *Proceedings of the 19th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

[21] J. Callan. Learning while filtering documents. In *Proceedings of the 21st International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[22] J. Carbonell. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[23] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J.M. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, 1998.

[24] S. Chakrabarti, P. Kunal, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, Honolulu, HI, USA, 2002.

[25] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of the Eighth International World Wide Web Conference*, Toronto, Canada, 1999.

[26] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, El Paso, TX, USA, 1997.

[27] S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, USA, 1997.

[28] J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, TX, USA, 2000.

[29] L. Chen and K. Sycara. Webmate: A personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN, USA, 1998.

[30] Y. Chen, F. Douglis, H. Huang, and K. Vo. Topblend: An efficient implementation of htmldiff in java. In *Proceedings of the World Conference on the WWW and Internet (WebNet 2000)*, San Antonio, TX, USA, 2000.

[31] J. Cho and S. Roy. Impact of search engines on page popularity. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW'04)*, New York, NY, USA, 2004.

[32] CiteSeer. The citeseer scientific literature digital library. http://www.citeseer.com.

[33] C. Clack, J. Farringdon, P. Lidwell, and T. Yu. Autonomous document classification for business. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, USA, 1997.

[34] W.W. Cohen and W. Fan. Learning page-independent heuristics for extracting data from web pages. In *Proceedings of the Eighth International World Wide Web Conference*, Toronto, Canada, 1999.

[35] Cranfield Collection. 1398 abstracts on aerodynamics. ftp://ftp.cs.cornell.edu/pub/smart/cran/.

[36] Inc. Copernic Technologies. Copernic agent professional. http://www.copernic.com/en/products/agent/professional.html.

[37] W.B. Croft and D.J. Harper. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35(4):285–295, 1979.

[38] D. Crouch, C. Condoravdi, R. Stolle, T. King, V. de Paiva, J. Everett, and D. Bobrow. Scalability of redundancy detection in focused document collections. In *Proceedings of Workshop on Scalability in Natural Language Understanding (ScaNaLU)*, Heidelberg, Germany, 2002.

[39] P. DeBra, G.J. Houben, Y. Kornatzky, and R. Post. Information retrieval in distributed hypertexts. In *Proceedings of the 1995 conference on Intelligent Multimedia, Information Retrieval System and Management*, New York, NY, USA, 1994.

[40] Dogpile. The dogpile meta-search engine. http://www.dogpile.com.

[41] R.B. Doorenbos, O. Etzioni, and D.S. Weld. A scalable comparison-shopping agent for the world wide web. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, USA, 1997.

[42] F. Douglis, T. Ball, Y. Chen, and E. Koutsofios. The at&t internet difference engine: Tracking and viewing changes on the web. *World Wide Web*, 1(1):27–44, 1998.

[43] D. Dreilinger and A.E. Howe. An information gathering agent for querying web search engines. Technical Report TR-CS-96-111, Colorado State University, Computer Science Department, 1996.

[44] D. Dreilinger and A.E. Howe. Experiences with selecting search engines using meta-search. *ACM Transactions on Information Systems*, 15(3):195–222, 1997.

[45] D. Eichmann and P. Srinivasan. Adaptive filtering of newswire stories using two-level clustering. *Information Retrieval*, 5:209–237, 2002.

[46] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616: Hypertext transfer protocol, http/1.1, 1999. http://www.faqs.org/rfcs/rfc2616.html.

[47] P.W. Foltz. Using latent semantic indexing for information filtering. In *Proceedings of the Conference on Office Information Systems*, Cambridge, MA, USA, 1990.

[48] P.W. Foltz. Latent semantic analysis for text-based research. *Behavior Research Methods, Instruments and Computers*, 28(2):197–202, 1996.

[49] M. Franz, T. Ward, J.S. McCarley, and W.J. Zhu. Unsupervised and supervised clustering for topic tracking. In *Proceedings of the 24th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, LA, USA, 2001.

[50] G.W. Furnas, S. Deerwester, S.T. Dumais, T.K. Landauer, R.A. Harshman, L.A. Streeter, and K.E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of the 11th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Grenoble, France, 1988.

[51] E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie: Providing personalized newsfeeds via analysis of information novelty. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW'04)*, New York, NY, USA, 2004.

[52] R. Ghani, R. Jones, and D. Mladenić. On-line learning for query generation: finding documents matching a minority concept on the web. In *Proceedings of the First Asia-Pacific Conference on Web Intelligence*, Maebashi City, Japan, 2001.

[53] D. Gibson, J.M. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*, Pittsburgh, PA, USA, 1998.

[54] D. Godoy, S. Schiaffino, and A. Amandi. Interface agents personalizing web-based tasks. *Cognitive Systems Research*, (5):207–222, 2004.

[55] Google. The google search engine. http://www.google.com.

[56] Google. The google webalerts service. http://www.google.com/webalerts.

[57] L. Gravano and H. García-Molina. Generalizing gloss to vector-space databases and broker hierarchies. In *Proceedings of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, 1995.

[58] L. Gravano, H. García-Molina, and A. Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the 1994 International ACM-SIGMOD Conference on Management of Data*, Minneapolis, MN, USA, 1994.

[59] V.N. Gudivada, V.V. Raghavan, W.I. Grosky, and R. Kasanagottu. Information retrieval on the world wide web. *IEEE Internet Computing*, 12:58–68, 1997.

[60] E.H. Han, D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Webace: A web agent for document categorization and exploration. In *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN, USA, 1998.

[61] D.K. Harman. Towards interactive query expansion. In *Proceedings of the 11th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Grenoble, France, 1988.

[62] D.K. Harman. Relevance feedback revisited. In *Proceedings of the 15th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Copenhagen, Denmark, 1992.

[63] D.K. Harman. Overview of the third text retrieval conference. In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, Gaithersburg, MD, USA, 1994.

[64] S.P. Harter. A probabilistic approach to automatic keyword indexing. *Journal of the American Society for Information Science*, 26(4):197–206,280–289, 1975.

[65] T.H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the web. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, Honolulu, HI, USA, 2002.

[66] M. Hearst and J.O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of the 19th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

[67] M. Hersovici, M. Jacovi, Y.S. Maarek, D. Pelleg, M. Shatalhaim, and S. Ur. The shark-search algorithm - an application: Tailored web site mapping. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, 1998.

[68] D.S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977.

[69] A.E. Howe and D. Dreilinger. Savvysearch: A meta-search engine that learns which search engines to query. *AI Magazine*, 18(2):19–25, 1997.

[70] D. Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *Proceedings of the 17th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, 1994.

[71] D.A. Hull and S. Robertson. The trec-8 filtering track final report. In *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, Gaithersburg, MD, USA, 1999.

[72] G. Jacobson and K. Vo. Heaviest increasing/common subsequence problems. In *Proceedings of the Third Annual Symposium of Combinatorial Pattern Matching*, Tucson, AZ, USA, 1992.

[73] T. Joachims, D. Freitag, and T. Mitchell. Webwatcher: A tour guide for the world wide web. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, 1997.

[74] T. Kelly and J. Mogul. Aliasing on the world wide web: Prevalence and performance implications. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, Honolulu, HI, USA, 2002.

[75] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, USA, 1998.

[76] T.K. Landauer, P.W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[77] S. Lawrence and C.L. Giles. Context and page analysis for improved web search. *IEEE Internet Computing*, 2(4):38–46, 1998.

[78] S. Lawrence and C.L. Giles. Searching the world wide web. *Science*, 280:98–100, 1998.

[79] S. Lawrence and C.L. Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[80] D.D. Lewis. Evaluating and optimizing autonomous text classification systems. In *Proceedings of the 18th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA, USA, 1995.

[81] D.D. Lewis and W.A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, 1994.

[82] D.D. Lewis, R.E. Schapire, J. Callan, and R. Papka. Training algorithms for linear text classifiers. In *Proceedings of the 19th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

[83] X. Li and B. Liu. Learning to classify text using positive and unlabeled data. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

[84] H. Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

[85] Northern Light. The northern light search engine. http://www.northernlight.com.

[86] B. Liu, W.S. Lee, P.S. Yu, and X. Li. Partially supervised classification of text documents. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-02)*, Sydney, Australia, 2002.

[87] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[88] L. Liu, W. Tang, D. Buttler, and C. Pu. Information monitoring on the web: A scalable solution. *World Wide Web Journal*, 5(4), 2002.

[89] K.E. Lochbaum and L.A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing & Management*, 25(6):665–676, 1989.

[90] Lycos. The lycos search engine. http://www.lycos.com.

[91] M. Margennis and C.J. van Rijsbergen. The potential and actual effectiveness of interactive query expansion. In *Proceedings of the 20th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, PA, USA, 1997.

[92] E.L. Margulis. Modeling documents with multiple poisson distributions. *Information Processing & Management*, 29(2):215–227, 1993.

[93] A. Mehrotra and M.A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[94] F. Menczer and R.K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the web. *Machine Learning*, 39(2-3):203–242, 2000.

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

[96] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *Proceedings of the 21st International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[97] D. Mladenić. Text-learning and related intelligent agents. *IEEE Intelligent Systems*, 14(4):44–54, 1999.

[98] A. Moukas. Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, 1996.

[99] I. Muşlea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, USA, 1999.

[100] S. Mukherjea. Wtms: A system for collecting and analyzing topic-specific web information. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, The Netherlands, 2000.

[101] K. Nigam, A.K. McCallum, S. Thrun, and T.M. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 39(2-3):103–134, 2000.

[102] ODP. The open directory project. http://www.dmoz.com.

[103] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[104] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *Proceedings of the Twelfth International World Wide Web Conference (WWW'03)*, Budapest, Hungary, 2003.

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

[106] M.J. Pazzani, J. Muramatsu, and D. Billsus. Syskill & webert: Identifying interesting web sites. In *Proceedings of the 1996 National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, USA, 1996.

[107] W.H. Press, S.A. Teukolsky, W.T. Vellerling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition.* Cambridge University Press, 1992.

[108] A. Pretschner and S. Gauch. Personalization on the web. Technical Report ITTC-FY2000-TR-13591-01, University of Kansas, Department of Electrical Engineering and Computer Science, 1999.

[109] V.V. Raghavan, G.S. Jung, and P. Bollmann. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Office and Information Systems*, 7(3):205–229, 1989.

[110] E. Rasmussen. Clustering algorithms. In W. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[111] P.R. Rider. Estimating the parameters of mixed poisson, binomial, and weibull distributions by the method of moments. *Bulletin of the Institute for International Statistics*, 39:225–232, 1961.

[112] S.E. Robertson. The probability ranking principle in ir. *Journal of Documentation*, 33(4):294–304, 1977.

[113] S.E. Robertson. On term selection for query expansion. *Journal of Documentation*, 46(4):359–364, 1990.

[114] S.E. Robertson and K. Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[115] S.E. Robertson, C.J. van Rijsbergen, and M.F. Porter. Probabilistic models for indexing and searching. In Oddy et al., editor, *Information Retrieval Research*, pages 35–56. Butterworths, 1981.

[116] S.E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, 1994.

[117] S.E. Robertson and S Walker. Threshold setting in adaptive filtering. *Journal of Documentation*, 56(3):312–331, 2000.

[118] J.J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[119] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1988.

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

[121] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[122] G. Salton, C.S. Yang, and C.T. Yu. A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, 26(1):33–44, 1975.

[123] E. Selberg and O. Etzioni. Multi-service search and comparison using the metacrawler. In *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, USA, 1995.

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

[125] W.M. Shaw. On the foundation of evaluation. *Journal of the American Society for Information Science*, 37(5):346–348, 1986.

[126] W.M. Shaw, R. Burgin, and P. Howell. Performance standards and evaluations in ir test collections. *Information Processing & Management*, 33(1):1–36, 1997.

[127] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, Orlando, FL, USA, 1993.

[128] A.F. Smeaton. Using nlp or nlp resources for information retrieval tasks. In T. Strzalkowski, editor, *Natural language information retrieval*, pages 99–111. Kluwer Academic Publishers, 1999.

[129] G.L. Somlo and A.E. Howe. Adaptive lightweight text filtering. In *Proceedings of the Fourth International Symposium on Intelligent Data Analysis (IDA 2001)*, Lisbon, Portugal, 2001.

[130] G.L. Somlo and A.E. Howe. Incremental clustering for profile maintenance in information gathering web agents. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, 2001.

[131] G.L. Somlo and A.E. Howe. Using web helper agent profiles in query generation. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, Melbourne, Australia, 2003.

[132] G.L. Somlo and A.E. Howe. Filtering for personal web information agents (poster presentation). In *Proceedings of the 27th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Sheffield, UK, 2004.

[133] G.L. Somlo and A.E. Howe. Querytracker: An agent for tracking persistent information needs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, New York, NY, USA, 2004.

[134] E. Spertus. Parasite: Mining structural information on the web. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara, CA, USA, 1997.

[135] A. Sugiura and O. Etzioni. Query routing for web search engines: Architecture and experiments. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, The Netherlands, 2000.

[136] J. Tague-Sutcliffe. Measuring the informativeness of a retrieval process. In *Proceedings of the 15th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Copenhagen, Denmark, 1992.

[137] A.H. Tan, H.L. Ong, H. Pan, J. Ng, and Q.X. Li. Foci: A personalized web intelligence system. In *Proceedings of the IJCAI-01 Workshop on Intelligent Techniques for Web Personalization*, Seattle, WA, USA, 2001.

[138] D.B. Terry, D. Goldberg, D. Nichols, and B.M. Oki. Continuous queries over append-only databases. In *Proceedings of the 1999 ACM-SIGMOD International Conference on Management of Data*, San Diego, CA, USA, 1992.

[139] C.J. van Rijsbergen. Foundation of evaluation. *Journal of Documentation*, 30(4):365–373, 1974.

[140] Vivisimo. The vivisimo meta-search engine. http://www.vivisimo.com.

[141] E.M. Voorhees and D.K. Harman. Overview of the sixth text retrieval conference (trec-6). In *Proceedings of the Sixth Text Retrieval Conference (TREC-6)*, Gaithersburg, MD, USA, 1997.

[142] Google Watch. A look at google's monopoly, algorithms, and privacy policies. http://www.google-watch.org.

[143] Search Engine Watch. Search engine ratings and reviews. http://www.searchenginewatch.com.

[144] Yahoo. The yahoo portal and search engine. http://www.yahoo.com.

[145] Y. Yang, T. Ault, T. Pierce, and C.W. Lattimer. Improving test categorization methods for event tracking. In *Proceedings of the 23th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Athens, Greece, 2000.

[146] O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to web search results. In *Proceedings of the Eighth International World Wide Web Conference*, Toronto, Canada, 1999.

[147] C. Zhai, P. Jansen, N. Roma, E. Stoica, and D.A. Evans. Optimization in clarit trec-8 adaptive filtering. In *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, Gaithersburg, MD, USA, 1999.

[148] C. Zhai, P. Jansen, E. Stoica, N. Grot, and D.A. Evans. Threshold calibration in clarit adaptive filtering. In *Proceedings of the Seventh Text Retrieval Conference (TREC-7)*, Gaithersburg, MD, USA, 1998.

[149] Y. Zhang and J. Callan. Maximum likelihood estimation for filtering thresholds. In *Proceedings of the 24th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, LA, USA, 2001.

[150] Y. Zhang, J. Callan, and T. Minka. Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Tampere, Finland, 2002.