Chapter 15

Multiplicative Adaptive User Preference Retrieval and Its Applications to Web Search

Zhixiang Chen

Department of Computer Science, University of Texas-Pan American
1201 West University Drive, Edinburg, Texas 78539, USA
E-mail: chen@cs.panam.edu

Abstract

Existing popular algorithms for user preference retrieval, such as Rocchio's similarity-based relevance feedback algorithm and its variants [Rocchio (1971); Ide, (1971a)], the Perceptron algorithm [Rosenblatt, (1958)] and the Gradient Descent Procedure [Wong *et al.*, (1988)], are based on linear additions of documents judged by the user. In contrast to the adoption of linear additive query updating techniques in those algorithms, in this chapter two new algorithms, which use multiplicative query expansion strategies to adaptively improve the query vector, are designed. It is shown that one algorithm has a substantially better mistake bound than the Rocchio and the Perceptron algorithms in learning a user preference relation determined by a linear classifier with a small number of non-zero coefficients over the real-valued vector space $[0,1]^n$. It is also shown that the other algorithm boosts the usefulness of an index term *exponentially*, while the gradient descent procedure does so *linearly*. Applications of those two algorithms to Web search are also presented.

15.1 Introduction

Consider a collection of documents $\mathcal{D}$. For any given user, her preference about documents in $\mathcal{D}$ is a *relation* defined over documents in $\mathcal{D}$ with re-

spect to her information needs or search queries. For any two documents in the collection, she may prefer one to the other or considers them as being equivalent. In other words, she may rank one higher than the other or gives them the same rank (or does not care the actual ranks for the two). Unfortunately, her preference may have various representations and, to make the things even worse, is *unknown* to an information retrieval system. However, one thing a system can do is to *"learn"* the unknown user preference through a series of adaptive improvements on some hypothetical guess. Among a variety of such *"learning"* processes, the most popular one is relevance feedback.

Research on relevance feedback in information retrieval has a long history [Baeza-Yates and Ribeiro-Neto, (1999); Frakes and Baeza-Yates, (1992); Ide, (1971a,b); Raghavan and Wong (1986); Rocchio, (1971); Salton, (1989); Spink and Losee, (1996)]. It is regarded as the most popular query reformation strategy [Baeza-Yates and Ribeiro-Neto, (1999)]. The central idea of relevance feedback consists of selecting important terms, or expressions, attached to the documents that have been judged as relevant or irrelevant by the user, and of enhancing the importance of those terms in a new query formation. Usually, relevance feedback works in a step by step fashion. In practice, at each step, a small set of top ranked documents (say, 20) are presented to the user for judgment of relevance. The expected effect is that the new query will be moved towards the relevant documents and away from the irrelevant ones.

In the vector space model [Salton, (1989); Salton *et al.,* (1975)], both documents and queries are represented as vectors in a vector space. A vector space may be binary in the simplest case, or real-valued when more realistic term weighting and indexing methods such as *tf-idf* are used. In the vector space model, relevance feedback is essentially an adaptive supervised learning algorithm: A query vector $\mathbf{q}$ and a similarity measure m are used as a hypothetical guess for the user preference relation to classify documents as relevant or irrelevant and to rank the documents as well; the user's judgments of the relevance or irrelevance of some of the classified documents are used as examples for updating the query vector and hopefully for improving the hypothetical guess towards the unknown user preference relation. The effectiveness of a concrete relevance feedback process depends on, among other factors, the query updating strategy used by the process. There exist a wide collection of relevance feedback algorithms [Salton, (1989); Baeza-Yates and Ribeiro-Neto, (1999)] with designs based on two basic techniques: query expansion with additions of documents judged by the user and term reweighting for modifying document term weights based on the user relevance judgment. We will focus ourselves on the query expansion technique.

The query expansion technique used in the existing relevance feedback algorithm is essentially a *linear additive* query updating or modifying method: Adding a linear combination

$$\alpha_1 \mathbf{d}_1 + \cdots + \alpha_s \mathbf{d}_s$$

to the current query vector, where $\mathbf{d}_1, \ldots, \mathbf{d}_s$ are the vectors of the documents judged by the user at the current iteration of relevance feedback, and α_i are real-valued updating factors for $1 \leq i \leq s$. Certainly, a concrete algorithm will choose its own favorite updating factors in the above linear combination. Rocchio's algorithm and its variants [Rocchio, (1971); Ide, (1971a,b); Salton, (1989); Baeza-Yates and Ribeiro-Neto, (1999)] are the most popular relevance feedback algorithms with linear additive query updating strategies. Especially, those algorithms are very similar to the Perceptron algorithm [Rosenblatt, (1958); Duda and Hart, (1973)], a well-known and extensively studied algorithm in fields such as Artificial Intelligence, Machine Learning, and Neural Networks. The similarities between those algorithms were mentioned and discussed in [Salton and Buckley, (1990); van Rijsbergen, (1979); Chen and Zhu, (2002)]. Another type of the algorithm, the gradient descent procedure, was designed in [Wong *et al.,* (1988)] for finding a linear structured user preference relation (or acceptable ranking strategy). This procedure also resembles the Perceptron algorithm in its adoption of linear additive query updating technique for minimizing its current ranking errors.

The main advantage of the linear additive query updating techniques used in those existing algorithms is their simplicity and good results. The simplicity is due to the fact that the modified term weights (query vector components) are computed directly from the set of retrieved documents. The good results are observed experimentally and are due to the fact that the modified query vector does reflect a portion of the intended query semantics. The main disadvantage is that no optimality criterion is adopted in practice without knowing the user preference ahead of the time, though such criterion exist in theory [Rocchio, (1971); Salton, (1989); Baeza-Yates and Ribeiro-Neto, (1999)]. In other words, those algorithms may have a *slow rate* to converge to the target user preference relation. In fact, it has been proved that, when used to learn a monotone disjunction of m relevant variables over the n dimensional binary vector space, both Rocchio's algorithm and the Perceptron algorithm have an $\Omega(n)$ lower bound on their classification errors [Kivinen *et al.,* (1997); Chen and Zhu, (2002)]. The gradient descent procedure can, after the first iteration from a zero initial query vector, boost the usefulness of an index term *linearly* [Wong *et al.,* (1988)]. The slow converging rate and the linear boosting achieved by those algorithms may not be liked by users in the real-world problem of Web

search, because Web search users have no patience to try, say, more than 10 iterations of relevance feedback to gain some significant search precision increase.

The main contributions of this chapter are as follows: In contrast to the adoption of linear additive query updating techniques in the existing algorithms, two types of algorithms, the multiplicative adaptive query expansion algorithm MA and the multiplicative adaptive gradient search algorithm MG, are designed. Those two algorithms use multiplicative query updating techniques to adaptively improve the query vector. It is proved that algorithm MA has an $O(m \log n)$ upper bound on classification errors in learning a user preference relation determined by a linear classifier $a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_m} > 0$ over the real-valued vector space $[0, 1]^n$. This means that algorithm MA has substantially better performance than the Rocchio's and the Perceptron algorithms for learning the same type of user preference relations. It is shown that after the first iteration from a zero initial query vector algorithm MG boosts the usefulness of an index term *exponentially*, while the gradient descent procedure does so linearly. This means that a document with a *good* index term will be ranked *exponentially higher* than the one without the *good* index term, thus more ideal ranking effect will be generated by algorithm MG for users. The work is this chapter is enlightened by algorithm Winnow [Littlestone, (1988)], a well-known algorithm equipped with a multiplicative weight updating technique. However, algorithm MG is a gradient descent search algorithm, which is different from algorithm Winnow. Furthermore, algorithm MA generalizes algorithm Winnow in the following aspects: Various updating functions may be used; multiplicative updating for a weight is dependent on values of the corresponding indexing terms, which is more realistic and applicable to real-valued vector space; and finally, a number of documents which may or may not be counterexamples to the algorithm's current classification are allowed as relevance feedback to the algorithm. Two Applications of algorithms MA and MG have been discussed. The first application is the project *MARS* [Meng and Chen, (2002)] (*M*ultiplicative *A*daptive *R*efinement *S*earch), which is built upon algorithm MA. The second is the project *MAGrads* [Meng *et al.,* (2003)] (*M*ultiplicative *A*daptive *Gra*dient *D*escent *S*earch), which is built upon algorithm MG. Other related work on intelligent Web search tools [Chen *et al.*, (2002)] includes *WebSail* [Chen *et al.,* (2002)], *Features* [Chen *et al.,* (2001)], *Yarrow* [Chen *al.,* (2000)], and *PAWS* [Meng and Chen, (2003)].

The rest of this chapter is organized as follows. Section 15.2 gives a formal presentation of user preference in the vector space model. Section 15.3 includes the design of algorithm MA and its performance analysis in terms of classification errors. Section 15.4 includes the design of algorithm

MG and its performance analysis based on weighting of index terms. The project *MARS* is reported in Section 15.5. The project *MAGrads* is reported in Section 15.6. Section 15.7 lists several problems for future study.

15.2 Vector Space and User Preference

Let $\mathcal{R}$ be the set of all real values, and $\mathcal{R}^+$ be the set of all non-negative real values. Let n be a positive integer. In the vector space model in information retrieval a collection of n indexing terms $T_1, T_2, \ldots, T_n$ are used to represent documents and queries. Each document $\mathbf{d}$ is represented as a vector $\mathbf{d} = (d_1, \ldots, d_n)$ such that for any i, $1 \leq i \leq n$, the i-th component of d_i is used to determine the relevance (or weight) of the i-th term T_i in the document. Because a document vector can be normalized, without loss of generality we only consider the real-valued vector space $[0,1]^n$ in this chapter. Given any two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ in $[0,1]^n$ (or $\mathcal{R}^n$), we use $\mathbf{x} \cdot \mathbf{y}$ to denote their inner product $x_1y_1 + \cdots + x_ny_n$.

Let $\mathcal{D}$ be a collection of documents. As in the work [Bollmann *et al.*, (1987); Wong it et al., (1988)], given any two documents in $\mathcal{D}$, we assume that a user would *prefer* one to the other or regard both as being *equivalent* with respect to her information needs (or search queries). In other words, user preference of documents in $\mathcal{D}$ defines a preference relation $\prec$ over $\mathcal{D}$ as follows:

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \prec \mathbf{d}' \iff \textit{the user prefers } \mathbf{d}' \textit{ to } \mathbf{d}.$$

It has been shown in [Bollmann *et al.,* (1987); Fishburn, (1970); Roberts, (1976)] that if a user preference relation $\prec$ is a *week order* satisfying some additional conditions then it can be represented by a *linear classifier*. That is, there is a query vector $\mathbf{q} = (q_1, \ldots, q_n) \in \mathcal{R}^n$ such that

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \prec \mathbf{d}' \iff \mathbf{q} \cdot \mathbf{d} < \mathbf{q} \cdot \mathbf{d}'. \tag{15.1}$$

In general a linear classifier over the vector space $[0,1]^n$ is a pair of $(\mathbf{q}, \theta)$ which classifies any document $\mathbf{d}$ as relevant if $\mathbf{q} \cdot \mathbf{d} > \theta$, or irrelevant otherwise, where the query vector $\mathbf{q} \in \mathcal{R}^n$ and the classification threshold $\theta \in \mathcal{R}^+$. Recall that $\mathbf{q} \cdot \mathbf{d}$ is usually used as the relevance rank or (score) of the document d with respect to user preference.

A natural way to understand a user preference relation $\prec$ is *document ranking*: A user prefers a document $\mathbf{d}$ to a document $\mathbf{d}'$, if and only if she ranks $\mathbf{d}$ higher than $\mathbf{d}'$. When a user has no preference of $\mathbf{d}$ to $\mathbf{d}'$ nor $\mathbf{d}'$ to $\mathbf{d}$, then she is really not interested in how those two documents are actually ranked. Based on such understanding, the following linear

acceptable ranking strategy was proposed in [Wong *et al.*, (1988)]:

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \prec \mathbf{d}' \Longrightarrow \ \mathbf{q} \cdot \mathbf{d} < \mathbf{q} \cdot \mathbf{d}', \tag{15.2}$$

where $\mathbf{q} \in R^n$ is the query vector determined by the user.

Let $\mathcal{D}_r$ be the set of all relevant documents in $\mathcal{D}$ with respect to a user's information needs (or search queries), and $\mathcal{D}_{ir}$ the set of all irrelevant documents. If we assume that a user preference relation has a simple structure with only two levels, i.e., one level consisting of all relevant documents and the other consisting of all irrelevant documents. Within the same level, no preference is made between any two documents. Then, finding a user preference relation satisfying the expression (15.1) is equivalent to the problem of finding a linear classifier $(\mathbf{q}, \theta)$ over $[0, 1]^n$ with the property

$$\forall \mathbf{d} \in \mathcal{D}, \quad \mathbf{d} \in \mathcal{D}_r \Longleftrightarrow \mathbf{q} \cdot \mathbf{d} > \theta, \tag{15.3}$$

where $\mathbf{q} \in \mathcal{R}^n$ is the query (or weight) vector. Similarly, finding a linear acceptable ranking strategy satisfying expression (15.2) is equivalent to the problem of finding a query vector $\mathbf{q} \in R^n$ with the property

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \in \mathcal{D}_{ir} \textit{ and } \mathbf{d}' \in \mathcal{D}_r \Longrightarrow \mathbf{q} \cdot \mathbf{d} < \mathbf{q} \cdot \mathbf{d}'. \tag{15.4}$$

The goal of relevance feedback in information retrieval is to identify a user preference relation $\prec$ with respect to her information needs from the documents judged by that user. Since user preference relations vary from different users and may have various unknown representations, it is not easy for an information system to find such relations. The existing popular relevance relevance algorithms basically use linear additive query expansion methods to find a user preference relation as follows:

- *Start with an initial query vector* $\mathbf{q}_0$.
- *At any step* $k \geq 0$*, improve the* k*-th query vector* $\mathbf{q}_k$ *to*

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \alpha_1 \mathbf{d}_1 + \cdots + \alpha_s \mathbf{d}_s, \tag{15.5}$$

where $\mathbf{d}_1, \ldots, \mathbf{d}_s$ *are the documents judged by the user, and the updating factors* $\alpha_i \in \mathcal{R}$ *for* $i = 1, \ldots, s$.

One particular and well-known example of relevance feedback is Rocchio's similarity-based relevance feedback [Rocchio, 1971; Salton, 1989]. Depending on how updating factors are used in improving the k-th query vector as in expression (15.5), a variety of relevance feedback algorithms have been designed [Ide, (1971a,b); Salton, (1989); Baeza-Yates and Ribeiro-Neto, (1999)]. A similarity-based relevance feedback algorithm is essentially an adaptive supervised learning algorithm from examples [Salton and Buckley, (1990); van Rijsbergen, (1979); Chen and Zhu, (2002)]. The

goal of the algorithm is to learn some unknown classifier (such as the linear classifier in expression (15.3)) that is determined by a user's information needs to classify documents as relevant or irrelevant. The learning is performed by modifying or updating the query vector that serves as the hypothetical representation of the collection of all relevant documents. The technique for updating the query vector is linear additions of the vectors of documents judged by the user. This type of linear additive query updating technique is similar to what is used by the Perceptron algorithm [Rosenblatt, (1958)]. The linear additive query updating technique has some disadvantage: It's *converging rate* to the unknown target classifier is *slow*, because it has been proved that the Perceptron algorithm [Kivinen *et al.,* (1997)] and the Rocchio's relevance feedback algorithm [Chen and Zhu, (2002)] with any of the four typical similar measures [Salton, (1989)] have an $\Omega(n)$ lower bound on their performance of learning in the n dimensional binary vector space. In the real world of Web search, a huge number of terms (usually, keywords) are used to index Web documents. To make the things even worse, no users will have the patience to try, say, more than 10 iterations of relevance feedback in order to gain some significant search precision increase. This implies that the traditional linear additive query updating method may be too slow to be applicable to Web search, and this motivates us to design new and *faster* query updating methods for user preference retrieval in Section 15.3.

For a user preference with respect to her information needs, for any index term T, define

- $|\mathcal{D}|$ = the total number of documents in the collection $\mathcal{D}$
- $|\mathcal{D}_r|$ = the total number of relevant documents in $\mathcal{D}$
- η = the number of documents in $\mathcal{D}$ indexed by T
- γ = the number of relevant documents in $\mathcal{D}_r$ indexed by T

A gradient descent procedure has been designed in [Wong *et al.,* (1988)] to find an acceptable ranking strategy satisfying expression (15.2). The idea of the procedure is to minimize ranking errors through linear additions of $\mathbf{d}' - \mathbf{d}$ for all pairs of documents $\mathbf{d}'$ and $\mathbf{d}$ that, according to expression (15.3), are ranked incorrectly. When the gradient descent procedure is applied to find an acceptable ranking satisfying expression (15.4), it has been shown [Wong *et al.*, (1988)] that after the first iteration from a zero initial query vector the procedure weighs an index term T *linearly* in $\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|}$, an approximate measure of the usefulness of index term T for distinguishing relevant and irrelevant documents. It has also shown that under certain *good* index term probability distribution the above usefulness measure for the index term T reaches its expected maximum when $\eta = 0.5|\mathcal{D}|$, a justi-

fication for choosing mid-frequency terms in indexing [Salton, (1989)]. In contrast to the linear additive query updating strategy used in the gradient descent procedure [Wong *al.,* (1988)] for minimizing ranking errors, in Section 15.4 a new algorithm with the multiplicative query updating strategy will be designed. It is shown that after the first iteration from a zero initial query vector the new algorithm weights an index term T *exponentially* in $\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|}$. This means that *exponentially large gaps* will be generated for index terms with respect to measures of their usefulness. Hence, a document with a *good* index term will be ranked *exponentially higher* than one without the *good* index term, thus more ideal ranking effects will be generated for users.

15.3 Multiplicative Adaptive Query Expansion Algorithm

In this section, a multiplicative query updating technique is designed to identify a user preference relation satisfying expression (15.1). It is believed that linear additive query updating yields some *mild* improvement on the hypothetical query vector towards the target user preference. A query updating technique that can yield *dramatic* improvements is wanted so that the hypothetical query vector can be moved towards the target in a much faster pace. The idea is that when an index term is judged by the user, its corresponding value in the hypothetical query vector should be boosted by a multiplicative factor that is dependent on the value of the term itself.

Algorithm $MA(\mathbf{q}_0, f, \theta)$:
(i) Inputs:
$\mathbf{q}_0$, *the non-negative initial query vectory*
$f(x) : [0,1] \longrightarrow R^+$, *the updating function*
$\theta \geq 0$, *the classification threshold*
(ii) Set $k = 0$.
(iii) Classify and rank documents with the linear classifier $(\mathbf{q}_k, \theta)$.
(iv) While (the user judged the relevance of a document $\mathbf{d}$*) do*
{
for $i = 1, \ldots, n$, *do* {
/* $\mathbf{q}_k = (q_{1,k}, \ldots, q_{n,k})$, $\mathbf{d} = (d_1, \ldots, d_n)$ */
if $(d_i \neq 0)$ {
/* *adjustment* */
if $(q_{i,k} \neq 0)$ *set* $q_{i,k+1} = q_{i,k}$
else set $q_{i,k+1} = 1$
if ($\mathbf{d}$ *is relevant)* /* *promotion* */

set $q_{i,k+1} = (1 + f(d_i))q_{i,k+1}$
else /* *demotion* */
set $q_{i,k+1} = \frac{q_{i,k+1}}{1+f(d_i)}$
} *else*
set $q_{i,k+1} = q_{i,k}$
}
}
(v) If no documents were judged in the k-th step, then stop. Otherwise, let $k = k + 1$ *and go to step (iv).*
/* *The end of the algorithm MA* */

In this chapter, only non-decreasing updating functions $f(x) : [0, 1] \Longrightarrow \mathbf{R}^+$ is considered, because it is wanted that multiplicative updating for an index term is proportional to the value of the term. We are, in particular, interested in the following two examples of algorithm MA:

Algorithm **LMA:** *In this algorithm, we let the updating function in algorithm MA be* $f(x) = \alpha x$*, a linear function with a positive coefficient* $\alpha > 1$.

Algorithm **ENL:** *In this algorithm, we let the updating function in algorithm MA be* $f(x) = \alpha^x$*, an exponential function with* $\alpha > 1$.

The design of algorithm MA is enlightened by algorithm Winnow [Littlestone, (1988)], a well-known algorithm equipped with a multiplicative weight updating technique. However, algorithm MA generalizes algorithm Winnow in the following aspects: (1) Various updating functions may be used in algorithm MA, while only constant updating functions are used in algorithm Winnow; (2) multiplicative updating for a weight is dependent on the value of corresponding indexing terms, which is more realistic and applicable to real-valued vector space, while algorithm Winnow considers all the terms equally; and (3) finally, a number of documents which may or may not be counterexamples to the algorithm's current classification are allowed as relevance feedback; while algorithm Winnow is an adaptive learning algorithm from equivalence queries, requiring the user to provide a counterexample to its current hypothesis. The equivalence query model is hardly realistic, because a user in reality has no knowledge about the information system nor about the representation of her preference. What she may do, and is able to do, is that she can judge some documents as what she needs or not among those provided by the system. We can derive algorithm Winnow [Littlestone, (1988)] and algorithm TW2 [Chen *et al.*, (2002)] in the following.

Algorithm MA becomes algorithm Winnow [Littlestone, (1988)] when the following restrictions are imposed:

- the vector space is set to the binary vector space $\{0, 1\}^n$.

- the initial query vector is set to $q_0 = (1, \ldots, 1)$.
- the update function is chosen as $f(x) = \alpha$, a positive constant function.
- at step (iv), equivalence query is adopted. That is, the user is asked to judge at most one document that is a counterexample to the current classification of the algorithm.

Algorithm MA becomes algorithm TW2 [Chen *et al.,* (2002)] when the following restrictions are imposed:

- the vector space is set to the binary vector space $\{0, 1\}^n$.
- the initial query vector is set to $q_0 = (0, \ldots, 0)$.
- and the updating function is chosen as $f(x) = \alpha$, a positive constant function.

We now analyze the the performance of algorithm MA when it is used to identify a user preference satisfying expression (15.3), a linear classifier $(\mathbf{q}, 0)$. Here, we consider a zero threshold in expression (refeq3). We say that algorithm MA makes an classification error at step k is the user judged a document as a counterexample to the algorithm's current hypothesis. We estimate the total number of classification errors algorithm MA will make based on the worst-case analysis. We also let at most one counterexample may be provided to the algorithm at each step. From now on to the end of this section it is assumed that $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ is a non-negative query vector with m non-zero components and $\theta > 0$. Define

$$\beta = min\{q_i \mid q_i > 0, 1 \leq i \leq n\}.$$

Definition 3.1. *Documents in the collection $\mathcal{D}$ are said to be indexed with respect to a threshold δ, $0 < \delta \leq 1$, if for any document $\mathbf{d} = (d_1, \ldots, d_n) \in \mathcal{D}$ one has either $d_i = 0$ or $\delta \leq d_i$, $1 \leq i \leq n$.*

In other words, when a document is indexed with respect to a threshold δ, any index term with a value below the threshold δ is considered not significant, and hence set to zero. Recall that in the vector space model a document and its vector have the equivalent meaning, so we may not distinguish the two concepts.

Lemma 3.2. *Assume that documents are indexed with respect to a threshold δ. Let u denote the total number of promotions that algorithm MA needs to find the linear classifier $(\mathbf{q}, 0)$. Let m denote the number of non-zero components in $\mathbf{q}$. Then,*

$$u \leq \frac{m \log \frac{\theta}{\beta\delta}}{\log(1 + f(\delta))}.$$

Proof. Without loss of generality, we may further assume that the m non-zero components of $\mathbf{q}$ are $q_1, \ldots, q_m$. When a promotion occurs at step k, a relevant document $\mathbf{d}$ is given to the algorithm as a counterexample to its current classification. Because of document indexing with respect to the threshold δ, there is some i with $1 \leq i \leq m$ such that $d_i \geq \delta$. This means that the i-th component $q_{i,k}$ of the query vector $\mathbf{q}_k$ will be promoted to

$$q_{i,k+1} = (1 + f(d_i))q_{i,k} \geq (1 + f(\delta))q_{i,k}, \tag{15.6}$$

because f is non-decreasing. Since $q_{i,k}$ will never be demoted, it follows from expression (15.6) that $q_{i,k}$ can be promoted at most

$$\frac{\log \frac{\theta}{\beta\delta}}{\log(1 + f(\delta))} \tag{15.7}$$

times. Since each promotion yields a promotion for at least one $q_{i,k}$ for $1 \leq i \leq m$, the total number of promotions u is at most m times the value given in expression (15.7). □

Theorem 3.3. *Assume that documents in $\mathcal{D}$ are indexed with respect to a threshold δ, $0 < \delta \leq 1$. Let T denote the total number of classification errors algorithm MA makes in order to find the linear classifier $(\mathbf{q}, 0)$ over the real-valued vector space $[0,1]^n$, where all components in $\mathbf{q}$ are nonnegative. Let m denote the number of non-zero components in $\mathbf{q}$. Then,*

$$T \leq \frac{[(1 + f(1))(n - m) + \sigma](1 + f(\delta))(1 + \delta)}{f(\delta)\theta} + \left(\frac{(1 + f(1))(1 + f(\delta))(1 + \delta)}{f(\delta)\delta} + 1\right)\frac{m \log \frac{\theta}{\beta\delta}}{\log(1 + f(\delta))}$$

(Hence, if $\theta = \frac{n}{m}$ is chosen, $T = O(m \log n)$.)

Proof. Without loss of generality, we may assume again that the m non-zero components of $\mathbf{q}$ are $q_1, \ldots, q_m$. We estimate the sum of the weights $\sum_{i=1}^{n} q_{i,k}$. Let u and v be the number of promotion steps and the number of demotion steps occurred during the learning process, respectively. Let t_k denote the number of zero components in $\mathbf{q}_k$ at promotion step k. Note that once a component of $\mathbf{q}_k$ is promoted to a non-zero value, it will never become zero again. For a promotion at step k with respect to a relevant document $\mathbf{d}$ judged by the user, for $i = 1, \ldots, n$, we have

$$q_{i,k+1} = \begin{cases} q_{i,k}, & if\ d_i = 0, \\ (1 + f(d_i)), & if\ d_i \neq 0\ and\ q_{i,k} = 0, \\ (1 + f(d_i))q_{i,k}, & if\ d_i \neq 0\ and\ q_{i,k} \neq 0. \end{cases}$$

Since a promotion only occurs when

$$\mathbf{q}_k \cdot \mathbf{d} = \sum_{i=1}^{n} d_i q_{i,k} = \sum_{d_i \neq 0 \ and \ q_{i,k} \neq 0} q_{i,k} < \theta,$$

we have

$$\begin{aligned}
\sum_{i=1}^{n} q_{i,k+1} &= \sum_{d_i \neq 0 \ and \ q_{i,k}=0} q_{i,k+1} + \sum_{d_i \neq 0 \ and \ q_{i,k} \neq 0} q_{i,k+1} + \sum_{d_i=0} q_{i,k+1} \\
&= \sum_{d_i \neq 0 \ and \ q_{i,k}=0} (1 + f(d_i)) + \sum_{d_i \neq 0 \ and \ q_{i,k} \neq 0} (1 + f(d_i)) q_{i,k} + \sum_{d_i=0} q_{i,k} \\
&\leq (1 + f(1)) t_k + \frac{1 + f(1)}{\delta} \sum_{d_i \neq 0 \ and \ q_{i,k} \neq 0} \delta q_{i,k} + \sum_{i=1}^{n} q_{i,k}. \\
&\leq (1 + f(1)) t_k + \frac{1 + f(1)}{\delta} \sum_{d_i \neq 0 \ and \ q_{i,k} \neq 0} d_i q_{i,k} + \sum_{i=1}^{n} q_{i,k}. \\
&\leq (1 + f(1)) t_k + \frac{1 + f(1)}{\delta} \theta + \sum_{i=1}^{n} q_{i,k}. \qquad (15.8)
\end{aligned}$$

For a demotion at step k with respect to an irrelevant document $\mathbf{d}$ judged by the user, for $i = 1, \ldots, n$, we have

$$q_{i,k+1} = q_{i,k} - (1 - \frac{1}{1 + f(d_i)}) q_{i,k} \leq q_{i,k} - (1 - \frac{1}{1 + f(\delta)}) q_{i,k}.$$

Since a demotion occurs only when $\sum_{i=1}^{n} d_i q_{i,k} > \theta$, we have

$$\begin{aligned}
\sum_{i=1}^{n} q_{i,k+1} &\leq \sum_{i=1}^{n} q_{i,k} - (1 - \frac{1}{1 + f(\delta)}) \sum_{i=1}^{n} q_{i,k} \\
&= \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1 + f(\delta))} \sum_{i=1}^{n} \frac{d_i}{1 + \delta} q_{i,k} \\
&\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1 + f(\delta))(1 + \delta)} \sum_{i=1}^{n} d_i q_{i,k} \\
&\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1 + f(\delta))(1 + \delta)} \theta \qquad (15.9)
\end{aligned}$$

Let the sum of the initial weights be σ. Hence, by (15.8) and (15.9), after u promotions and v demotions,

$$\sum_{i=1}^{n} q_{i,k+1} \leq (1+f(1)) \sum_{i=1}^{u} t_i + \sum_{i=1}^{n} q_{i,0} + \frac{(1+f(1))\theta u}{\delta} - \frac{f(\delta)\theta v}{(1+f(\delta))(1+\delta)}$$
$$\leq (1+f(1))(n-m) + \sigma + \frac{(1+f(1))\theta u}{\delta} - \frac{f(\delta)\theta v}{(1+f(\delta))(1+\delta)}$$

Note that at any step the weights are never negative. It follows from the above relation that

$$v \leq \frac{[(1+f(1))(n-m)+\sigma](1+f(\delta))(1+\delta)}{f(\delta)\theta} + \frac{(1+f(1))(1+f(\delta))(1+\delta)u}{f(\delta)\delta}. \tag{15.10}$$

It follows from Lemma 3.2 and (15.10) that the total number of promotions and demotions, i.e., the total number of classification errors T, is bounded by

$$T \leq v+u \leq \frac{[(1+f(1))(n-m)+\sigma](1+f(\delta))(1+\delta)}{f(\delta)\theta} + \frac{(1+f(1))(1+f(\delta))(1+\delta)u}{f(\delta)\delta} + u$$
$$\leq \frac{[(1+f(1))(n-m)+\sigma](1+f(\delta))(1+\delta)}{f(\delta)\theta} + \left(\frac{(1+f(1))(1+f(\delta))(1+\delta)}{f(\delta)\delta} + 1\right)\frac{m \log \frac{\theta}{\beta\delta}}{\log(1+f(\delta))}$$

This completes our proof. □

15.4 Multiplicative Gradient Descent Search Algorithm

In this section, we design algorithm MG for finding a query vector $\mathbf{q}$ satisfying the acceptable ranking strategy condition (15.2). Algorithm MG uses a multiplicative query updating technique to minimize its ranking errors. It will be shown that algorithm MG boosts the usefulness of an index term *exponentially* in contrast to *linear* boosting achieved by the gradient descent procedure in [Wong *et al.*, (1988)].

Algorithm $MG(\mathbf{q}_0, f)$:
(i) Inputs:

$\mathbf{q}_0$, *the non-negative initial query vectory*
$f(x) : [0,1] \longrightarrow R^+$, *the updating function*
(ii) Set $k = 0$.
(iii) Let $\mathbf{q}_k$ *be the query vector at step* k. *Identify the set of mistakes*

$$\Gamma(\mathbf{q}_k) = \{< \mathbf{d}, \mathbf{d}' > | \mathbf{d} \prec \mathbf{d}', \mathbf{q}_k \cdot \mathbf{d} \geq \mathbf{q}_k \cdot \mathbf{d}'\}.$$

If $\Gamma(\mathbf{q}_k) = \emptyset$, *then stop.*
(iv) For each pair $< \mathbf{d}, \mathbf{d}' > \in \Gamma(\mathbf{q}_k)$, *do* {
for $i = 1, \ldots, n$, *do* {
if $(d_i' \neq 0)$ {
/* *adjustment* */
if $(q_{i,k} \neq 0)$ *set* $q_{i,k+1} = q_{i,k}$ *else set* $q_{i,k+1} = 1$
set $q_{i,k+1} = (1 + f(d_i'))q_{i,k+1}$ /* *promotion* */
}
if $(d_i \neq 0)$
set $q_{i,k+1} = \frac{q_{i,k}}{1+f(d_i)}$ /* *demotion* */
}
}
(v) Let $k = k + 1$ *and go to step (iii).*
/* *The end of the algorithm MG* */

Theorem 4.1. *Assume that algorithm MG is applied to find an acceptable ranking strategy satisfying condition (15.4). If one chooses the initial query vector* $\mathbf{q}_0 = \mathbf{0}$, *then after the first iteration, for any* i *with* $1 \leq i \leq n$, *the weight* $q_{i,1}$ *for the* i*-th index term in* $\mathbf{q}_1$ *is*

$$q_{i,1} = \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \frac{\tau(\mathbf{d}')}{\tau(\mathbf{d})},$$

where $\tau(\mathbf{d}') = 1 + f(d_i')$ *if* $d_i' \neq 0$ *or* 1 *otherwise, and* $\tau(\mathbf{d}) = 1 + f(d_i)$ *if* $d_i \neq 0$ *or* 1 *otherwise. In particular, when a linear updating function* $f(x) = \alpha$ *is chosen,*

$$q_{i,1} = (1 + \alpha)^{|\mathcal{D}| \cdot |\mathcal{D}_r| (\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|})},$$

where $|\mathcal{D}_r|$ *is the total number of relevant documents,* η *is the number of documents indexed by the* i*-th index term* T_i, *and* γ *is the number of relevant documents indexed by* T_i

Proof. Since the acceptable ranking strategy satisfying condition (15.4), it follows from $\mathbf{q}_0 = \mathbf{0}$ that at the first iteration one obtains

$$\Gamma(\mathbf{q}_0) = \{< \mathbf{d}, \mathbf{d}' > | \mathbf{d} \in \mathcal{D}_{ir}, \mathbf{d}' \in \mathcal{D}_r\}.$$

Here $\mathcal{D}_r$ and $\mathcal{D}_{ir}$ denote respectively the set of relevant documents and the set of irrelevant ones. Note that during the first iteration, for each pair $< \mathbf{d}, \mathbf{d}' > \in \Gamma(\mathbf{q}_0)$, for any i with $1 \leq i \leq n$, a promotion is performed for the i-th component of the query vector if $d_i' \neq 0$ and a demotion is performed for the i-th component if $d_i \neq 0$. This implies that for any i, $1 \leq i \leq n$, after the first iteration the value $q_{i,1}$ with respect to the i-th index term is

$$q_{i,1} = \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \frac{\tau(d')}{\tau(d)}.$$

When a constant updating function $f(x) = \alpha$ is used, it easily follows from the above expression that

$$\begin{aligned}
q_{i,1} &= \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \frac{\tau(\mathbf{d}')}{\tau(\mathbf{d})} \\
&= \frac{\prod_{\mathbf{d}' \in \mathcal{D}_r} \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \tau(\mathbf{d}')}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \tau(\mathbf{d})} = \frac{\prod_{\mathbf{d}' \in \mathcal{D}_r} (\tau(\mathbf{d}'))^{|\mathcal{D}_{ir}|}}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} (\tau(\mathbf{d}))^{|\mathcal{D}_r|}} \\
&= \frac{\prod_{\mathbf{d}' \in \mathcal{D}_r} (1 + f(d_i'))^{|\mathcal{D}_{ir}|}}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} (1 + f(d_i))^{|\mathcal{D}_r|}} = \frac{(1 + f(d_i'))^{|\mathcal{D}_{ir}|\gamma}}{(1 + f(d_i))^{|\mathcal{D}_r|(\eta - \gamma)}} \\
&= (1 + \alpha)^{|\mathcal{D}_{ir}|\gamma - |\mathcal{D}_r|(\eta - \gamma)} \\
&= (1 + \alpha)^{(|\mathcal{D}| - |\mathcal{D}_r|)\gamma - |\mathcal{D}_r|(\eta - \gamma)} \\
&= (1 + \alpha)^{|\mathcal{D}| \cdot |\mathcal{D}_r| (\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|})}
\end{aligned}$$

This completes our proof. □

In Fig. 15.1, we illustrate the boosting effects of algorithm MG (the exponential curve) and the gradient descent procedure (the linear curve). When the probability distribution of an index term has the pattern as shown in part (b) of Fig. 15.1, we can show in a way similar to [Wong *et al.,* (1988)] that the expected value of the usefulness $\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|}$ for an index term reached its maximum when $\eta = 0.5|\mathcal{D}|$, another justification for choosing mid frequent terms in indexing [Salton, (1989)].

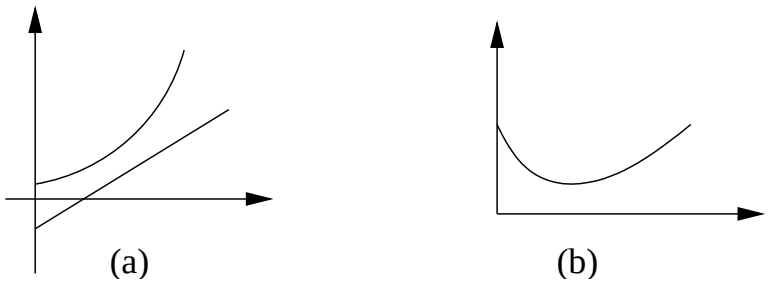

Fig. 15.1 (a)Boosting Rates. (b) Index Term Prob. Distribution

15.5 Meta-Search Engine MARS

In this section, we report the application of algorithm MA to the experimental meta-search engine MARS [Meng and Chen, (2002)] (*M*ultiplicative *A*daptive *R*efinement *S*earch). The architecture of MARS is shown in Fig. 15.2. User queries to MARS are accepted from a browser. Besides entering the query, a user can also specify a particular general-purpose search engine that she would like MARS to use and the maximum number of returned results (the larger the number is, the more time it takes to process). The `QueryConstructor` organizes the query into a format conforming to the specified search engine. One of the MetaSearchers sends the query to the general-purpose search engine. When the results are sent back from the general-purpose search engine, `DocumentParser`, `DocumentIndexer` and `Ranker` process the returned URLs and list them to the user as the initial search result. At this point, the rank is based on the original rank from the search engine. Constrained by the amount of space available on a typical screen, we list the top 10 URLs (highest ranked) and the bottom 10 URLs (lowest ranked). Once the results are displayed, the user can interactively work with MARS to refine the search results. Each time the user can mark a particular URL as *relevant* or *not relevant*. Upon receiving feedbacks from the user, MARS updates the weight assigned to each index term within the set of documents already returned from the specified search engine, according to the algorithm MA. The refined results are sorted based on their ranking scores and then displayed back to the user for further relevance feedback. This process continues until the satisfactory results are found or the user quits her search.

Some initial tests have conducted in [Meng and Chen, (2002)] to see how effectively and efficiently the algorithm MA can be applied to Web search using MARS meta-search engine. Two types of performance measures, the search precision improvement and the delay statistics of the MARS, were

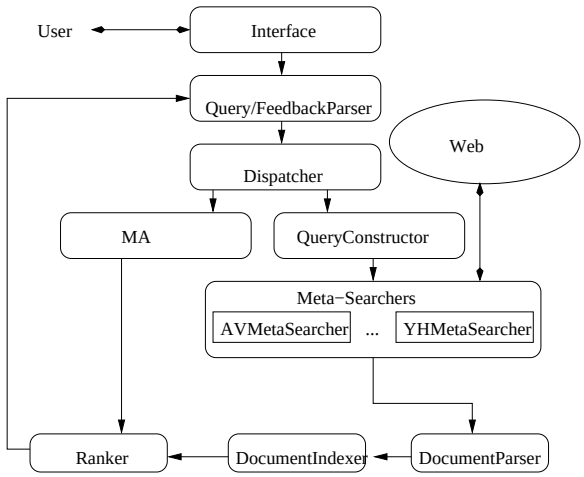

Fig. 15.2 The Architecture of MARS

used. A dozen of queries were sent to MARS to examine the response times. These response times are divided into two categories: The initial response time between the time issuing the query and the time receiving the response from an external search engine the user selected; and the time needed for the algorithm MA to refine the results. One calls these two time measurements *initial time* and *refine time*. The statistics in the Table 15.1 indicates the two measures. One should note that the *initial time* is needed to get any search results from the external search whether or not the algorithm MA is involved. As can be seen, the time spent in refining the search results is very small relative to the time to get the initial result.

Table 15.1: Response Time in Seconds

	Mean	Std Dev.	95% Conf. Interval	Maximum
Original	3.86	1.15	0.635	5.29
Refine	0.986	0.427	0.236	1.44

It has also been evaluated how algorithm MA improves the search precision. In the evaluation, a set of highly vague terms, such as *memory* and *language*, were selected. These words may mean completely differently in different areas. E.g., the term *memory* can mean human memory, or the memory chips used in computers; and the term *language* can refer to spoken language or computer programming language. We wanted to see if the search precision improves *dramatically* with very limited relevance

feedback. In all the tests, the search engine AltaVista [AV] was used. In the following, as an example, we lists the comparative results of *memory* and *language*.

Memory: The top-10 of the initial result sent back include two types of URLs, as expected. One is computer memory related, the other is related to memory in human beings. The following is the list (for space reason, only the top 10 is listed). For the underlying search purpose the ones that have an R in front are relevant; the ones with an X are irrelevant.

R http://www.memorytogo.com/
X http://memory.loc.gov/
X http://lcweb2.loc.gov/ammem/ammemhome.html
R http://www.datamem.com/
R http://www.samintl.com/mem/index.htm
X http://www.asacredmemory.com/
X http://www.exploratorium.edu/memory/lectures.html
X http://www.exploratorium.edu/memory/index.html
R http://www.satech.com/glosofmemter.html
R http://www.lostcircuits.com/memory/

With one round of refinement when a total of four URLs were marked (two marked in the top 10 list and two marked in the bottom 10 list), four of the original irrelevant URLs were eliminated and the revised top 10 URLs are as follows.

R http://www.streetprices.com/Electronics/...ware_PC
R http://www.memorytogo.com/
X http://www.crpuzzles.com/mem/index.html
R http://www.linux-mtd.infradead.org/
R http://fiberoptics.dimm-memory-infineon....owsides
R http://www.ramplus.com/cpumemory.html
X http://www.asacredmemory.com/
R http://www.computersupersale.com/shopdis..._A_cat_
R http://www.datamem.com/
R http://www.lostcircuits.com/memory/

Language: Similar to the term *memory*, the search results for *language* can be roughly divided into two classes, the ones related to human languages and the ones related to computer programming language. The following is the original list of top 10 URLs returned from AltaVista [AV] with R as relevant and X as irrelevant and we are looking for information about programming language.

X http://chinese.about.com/
R http://www.python.org/
X http://esl.about.com/
X http://esl.about.com/homework/esl/mbody.htm
X http://www.aliensonearth.com/catalog/pub/language/
X http://kidslangarts.about.com/
X http://kidslangarts.about.com/kids/kidslangarts/mb
X http://pw1.netcom.com/ rlederer/rllink.htm
X http://www.wordcentral.com/
X http://www.win-shareware.com/html/language.html

As can be seen, only one URL `www.python.org` is really relevant to what was looked for. With a refinement of three URLs marked, one marked irrelevant from the top 10 list, one marked relevant from the top 10 list, and one marked from the bottom 10 list (`www.suse.de/lang.html`), the refined list now contains six relevant URLs, compared to only one before refinement. Of these six URLs, one was originally in top 10 and was marked; one was originally in bottom 10 and was marked; the other four were not examined nor marked before at all. But they showed up in the top 10 list!

R http://www.suse.de/lang.html
R http://www.python.org/
X http://www.eason.ie/flat_index_with_area...L400_en
R http://www.w3.org/Style/XSL/
X http://www.hlc.unimelb.edu.au/
X http://www.transparent.com/languagepages/languages
R http://caml.inria.fr/
R http://home.nvg.org/ sk/lang/lang.html
R http://www.ihtml.com/
X http://www.aliensonearth.com/catalog/pub/language/

As can been seen from the above experimental results algorithm MA improve the search performance significantly.

15.6 Meta-Search Engine MAGrads

This section reports an experimental meta-search engine MAGrads [Meng *et al.,* (2003)] (*M*ultiplicative *A*daptive *Gra*dient *D*escent *S*earch), which is built upon algorithm MG. The architecture of MAGrads is similar to that of MARS as shown in Fig. 15.2. As in MARS, a user can query MAGrads

from a browser. The user can also specify a particular general-purpose search engine she would like MAGrads to use and the maximum number of returned results. The `QueryConstructor` organizes the query into a format conforming to the specified search engine. One of the MetaSearchers sends the query to the general-purpose search engine to receive the initial search result. `DocumentParser`, `DocumentIndexer` and `Ranker` work in the similar ways as they do in MARS. Once the results are displayed, the user can interactively work with MAGrads to refine the search results. Each time the user can mark a particular URL as *relevant* or *not relevant*. Upon receiving feedbacks from the user, MAGrads updates the weight assigned to each index term within the set of documents already returned from the specified search engine, according to the algorithm MG. The refined results are sorted based on their ranking scores and then displayed back to the user for further relevance feedback. This process continues until the satisfactory results are found or the user quits her search.

In order to provide some measures of system performance we conducted a set of experiments and compared the search results between AltaVista and MAGrads in regard to their search accuracy. The evaluation is based on the following approach that is similar to the standard *recall* and *precision* used in information retrieval.

For a search query q, let A denote the set of documents returned by the search engine (either AltaVista or MAGrads), and let R denote the set of documents in A that are relevant to q. For any integer m with $1 \leq m \leq |A|$, define R_m to be the set of documents in R that are among the top m ranked documents according to the search engine. One defines the *relative recall* R_{recall} and the *relative precision* $R_{precision}$ as follows.

$$R_{recall} = \frac{|R_m|}{|R|}$$

$$R_{precision} = \frac{|R_m|}{m}$$

The relative recall R_{recall} measures the percentage of relevant documents in R are listed in the top-m list m positions. The more relevant documents are listed within the top-m list, the better the performance is of the search engine. The relative precision $R_{precision}$ is a measure of how many documents are relevant among the top-m list.

In the evaluation, a list of 20 queries were selected. The queries were tested against AltaVista and MAGrads, respectively. The performance results were collected and recorded. When a query was issued to AltaVista, One manually went through the returned results (the set A), marking the ones that were relevant (the set R). The number of relevant documents within the top-10 and top-20 were also marked (the set R_m where $m =$

Table 15.2: Relative Precision and Recall Comparison

	$\mathbf{R_{precision}}$		$\mathbf{R_{recall}}$	
	AltaVista	MAGrads	AltaVista	MAGrads
(50,10)	0.55	0.82	0.23	0.40
(50,20)	0.52	0.74	0.40	0.69
(100,10)	0.55	0.85	0.14	0.27
(100,20)	0.52	0.77	0.24	0.46
(150,10)	0.55	0.86	0.11	0.23
(150,20)	0.52	0.79	0.19	0.40
(200,10)	0.55	0.88	0.09	0.21
(200,20)	0.52	0.80	0.17	0.35

10, 20). We then again manually went through the documents list to check the number of documents appeared at the top-10 and top-20 list. For each of the 20 queries, one selected the total number of documents to be returned by AltaVista and used by MAGrads to be 50, 100, 150, and 200, respectively. The relative recall and relative precision were computed for each of the query. The average was taken for comparison between AltaVista and MAGrads. The results of the comparison are listed in Table 15.2.

In Table 15.2, each row is labeled by a pair of integers $(|A|, m)$, where $|A|$ is the total number of documents retrieved by the search engine for a given query and m is the limit of the size of the top-listed documents. Some observations and analysis are obtained from the data listed in the table:

(1) AltaVista holds relative precisions, $R_{precision} = \frac{|R_m|}{m}$, constant across the document sets of size 200, 150, 100, and 50 for the given value of m (the number of top documents listed to the user). The reason is that the document set returned by AltaVista and its order in response to the given query is fixed. Thus the set R_m is a constant with regard to a given value of m. In the underlying experiments, m was assigned the value of 10 and 20.

(2) On the other hand for MAGrads the set R_m changes as the user enters its feedback and MAGrads re-adjusts the document relevance according to the user feedback. On average more relevant documents sift up towards the top-listed document set based on the user relevance feedback. Thus it has a better average relative precision than that of AltaVista.

(3) As for relative recall, $R_{recall} = \frac{|R_m|}{|R|}$, the case is different. For AltaVista, although the set R_m, the set of relevant documents that are actually listed in the top-m documents is fixed, the base size $|R|$ is changing from 50 to 200. Thus the average relative recall is in fact decreasing (from 0.23 to 0.09 for $m = 10$ and from 0.40 to 0.17 for $m = 20$) as the base

document size increases.

(4) The relative recall of MAGrads is decreasing as the base size increases for the same reason for a given value of m. However because of the relevance feedback MAGrads has a much higher average relative recall rate than that of AltaVista.

As can be seen from the table, MAGrads has a much better average performance than that of AltaVista, as far as relative precisions and relative recalls are concerned. The primary reason is the use of multiplicative gradient descent algorithm MG in MAGrads. Since the query list is chosen randomly and the comparison is done for the same set of documents between AltaVista and MAGrads, the results here are convincing that the MAGrads algorithm performs very well.

15.7 Concluding Remarks

The motivations of the work in this chapter come from the reality of Web search: Web search users usually have no patience to try, say, more than 5 iterations of relevance feedback for some intelligent search system in order to gain certain significant search precision increase. Existing popular algorithms for user preference retrieval have their own beauty and advantages. Because of the adoption of *linear additive* query updating techniques, when used to identify user preference determined by a linear classifier those algorithms, such as Rocchio's algorithm and its variants [Rocchio, (1971); Ide, (1971a,b)], the Perceptron algorithm [Rosenblatt, (1958); Duda, (1973)] and gradient descent procedure, [Wong *et al.,* (1988)] have either a *slow* converging rate or *small* boosting on the usefulness of an index term.

In contrast to the adoption of linear additive query updating techniques in the those existing algorithms, two new algorithms MA and MG have been designed in this chapter. Those two algorithms use multiplicative query updating techniques to adaptively learn the user's preference from relevance feedback. It is proved that algorithm MA has substantially better performance than the Rocchio and the Perceptron algorithms in the case of identifying a user preference relation that is determined by a linear classifier with a small number of non-zero coefficients. It is also shown that algorithm MG boosts the usefulness of an index term *exponentially* to identify a linear structured user preference after the first iteration, while the gradient descent procedure does so *linearly.* Two applications MARS and MAGrads of algorithms MA and MG are also presented, and the experimental results show that both algorithms can achieve significant search precision increase.

The current theoretical results have been obtained for algorithms MA and MG in the worst-case performance analysis. It is interesting to analyze

the average-case performance of those two algorithms. We feel that this task is very challenging. It is also interesting to conduct emperimental studies to understand the behaviors of the algorithms MA and GA with real world data sets. Finally, we would like to investigate more applications of the two algorithms to Web search.

Acknowledgment The author thanks Professor Xiannong Meng for many valuable discussions on the topic. The work in this chapter is supported in part by the Computing and Information Technology Center of the University of Texas-Pan American.

Bibliography

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*, Addison Wesley.

Bollmann, P. and Wong, S.K.M. (1987). Adaptive linear information retrieval models, *Proc. the Tenth ACM SIGIR Conf. on Research and Development in Information Retrieval*, ACM Press, pp. 157-163.

Chen, Z. (2001). Multiplicative adaptive algorithms for user preference retrieval, *Proc. the Seventh Annual Intl. Conf. on Combinatorics and Computing*, Lecture Notes in Computer Science 2108, Springer-Verlag, pp. 540-549.

Chen, Z. and Meng, X. (2002). MARS: Multiplicative adaptive user preference retrieval in Web search, *Proc. the Intl. Conf. on Internet Computing*, CSREA Press, pp. 643-648.

Chen, Z. and Meng, X. (2000). Yarrow: A real-time client site meta search learner, *Proc. the AAAI 2000 Workshop on Artificial Intelligence for Web Search*, AAAI Press, pp. 12-17.

Chen, Z., Meng, X., Fowler, R.H., and Zhu, Z. (2001). FEATURES: Real-time adaptive feature learning and document learning, *J. the American Society for Information Science*, **52**, 8, pp. 655-665.

Chen, Z., Meng, X., Zhu, B. and Fowler, R.H. (2002). WebSail: From online learning to Web search, *J. Knowledge and Information Science*, **4**, 2, pp. 219-227.

Chen, Z. and Zhu, B. (2002). Some formal analysis of the Rocchio's similarity-based relevance feedback algorithm, *Information Retrieval*, 5, pp. 61-86, 2002.

Chen, Z., Zhu, B. and Meng, X. (2002). Intelligent Web Search through Adaptive Learning from Relevance Feedback, in Shi, N. and V.K. Murthy (eds.), *Architectural Issues of Web-Enabled E-Business*, Idea Group Publishing, , pp. 139-143.

Duda, R.O. and Hart, P.E. (1973). *Pattern Classification and Scene Analysis*, John Wiley.

Fishburn, P.C. (1970). *Utility Theory for Decision Making*, New York, Wiley, 1970.

Frakes, W.B. and Baeza-Yates, R.S. (1992). *Information Retrieval: Data Struc-*

tures and Algorithms, Prentice Hall.

Ide, E. (1971a). New experiments in relevance feedback, in Salton, G. (editor), *The Smart System - Experiments in Automatic Document Processing*, Prentice-Hall, pp. 337-354.

Ide, E. (1971b). Interactive search strategies and dynamic file organization in information retrieval, in Salton, G. (editor), *The Smart System - Experiments in Automatic Document Processing*, Prentice-Hall, pp. 373-393.

Kivinen, J., Warmuth, M.K. and Auer, P. (1997). The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant, *Artificial Intelligence*, pp. 325-343.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, *Machine Learning*, 2, pp. 285-318.

Meng, X. and Chen, Z. (2003). PAWS: Personalized Web Search with Clusters, *Proc. the 2003 Intl. Conf. on Internet Computing*, CSREA Press, pp. 46-52..

Meng, X. and Chen, Z. (2002). MARS: Applying multiplicative adaptive user preference retrieval to Web Search, *Proc. the Intl. Conf. on Internet Computing*, CSREA Press, pp. 643-648.

Meng, X., Chen, Z. and Spink, A. (2003). A multiplicative gradient descent search algorithm for user preference retrieval and its application to Web search, *Proc. of the IEEE Intl. Conf. on Information and Technology: Coding and Computing*, IEEE Press, pp. 150-154.

Raghavan, V.V. and Wong, S.K.M. (1986). A critical analysis of the vector space model for information retrieval, *J. the American Society for Information Science*, **37**, 5, pp. 279-287.

Roberts, F.S. (1976). *Measurement Theory*, Addison-Wesley, Readings (MS).

Rocchio, J.J. (1971). Relevance feedback in information retrieval, in Salton, G. (editor), *The Smart Retrieval System - Experiments in Automatic Document Processing*, Prentice-Hall, pp. 313-323.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, **65**, 6, pp. 386-407.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley.

Salton, G. and Buckley, C. (1990). Improving retrieval performance by relevance feedback, *J. the American Society for Information Science*, **41**, 4, pp. 288-297.

Salton, G., Wong, S.K.M. and Yang, C.S. (1975). A vector space model for automatic indexing, *Communications of ACM*, **18**, 11, pp. 613-620.

Spink, A. and Losee, R.M. (1996). Feedback in Information Retrieval, *Annual Review of Information Science and Technology (ARIST)*, **31**, pp. 33-77.

van Rijsbergen, C.J. (1979). *Information Retrieval.* Butterworths, London.

Wong, S.K.M., Yao, Y.Y. and Bollmann, P. (1988). Linear structures in information retrieval, *Proc. the 1988 ACM-SIGIR Conf. on Information Retrieval*, pp. 219-232.