A Quadratic Lower Bound for Rocchio's Similarity-Based Relevance Feedback Algorithm with a Fixed Query Updating Factor

Zhixiang Chen Bin Fu John Abraham

Abstract

Rocchio's similarity-based relevance feedback algorithm, one of the most important query reformation methods in information retrieval, is essentially an adaptive supervised learning algorithm from examples. In practice, Rocchio's algorithm often uses a fixed query updating factor. When this is the case, we strengthen the linear $\Omega(n)$ lower bound obtained in [9] and prove that Rocchio's algorithm makes $\Omega(k(n-k))$ mistakes in searching for a collection of documents represented by a monotone disjunction of k relevant features over the n-dimensional binary vector space $\{0,1\}^n$, when the inner product similarity measure is used. A quadratic lower bound is obtained when k is linearly proportional to n. We also prove an $O(k(n-k)^3)$ upper bound for Rocchio's algorithm with the inner product similarity measure in searching for such a collection of documents with a constant query updating factor and a zero classification threshold.

Keywords: Information retrieval, relevance feedback, vector space models, similarity, lower bounds.

Contact Information:

Zhixiang Chen

Department of Computer Science

University of Texas-Pan American

1201 W. University Drive

Edinburg, TX 78541, USA

Email: chen@cs.panam.edu

Phone: (56) 381-3520

Fax: (956) 384-5099

Affiliations of Authors:

Zhixiang Chen Bin Fu John Abraham

Department of Computer Science

University of Texas-Pan American

1201 W. University Drive

Edinburg, TX 78541, USA

Emails: chen@cs.panam.edu

binfu@cs.panam.edu

jabraham@panam.edu

1 Introduction

Research on relevance feedback in information retrieval has a long history [22, 2, 11, 13, 12, 17, 14, 19]. It is regarded as the most popular query reformation strategy [2]. The central idea of relevance feedback is to improve search performance for a particular query by modifying the query step by step, based on the user's judgments of the relevance or irrelevance of some of the documents retrieved. In the vector space model [19, 21], both documents and queries are represented as vectors in a discretized vector space. In this case, relevance feedback is essentially an adaptive supervised learning algorithm: A query vector and a similarity measure are used to classify documents as relevant and irrelevant; the user's judgments of the relevance or irrelevance of some the classified documents are used as examples for updating the query vector as a linear combination of the initial query vector and the examples judged by the user.

In his popular textbook [22], Keith van Rijsbergen describes the relevance feedback as a fixed error correction procedure and relates it to the linear separation problem. When the inner product similarity is used, relevance feedback is just a Perceptron-like learning algorithm [15]. It is known [14] that there is an optimal way for updating the query vector if the sets of relevant and irrelevant documents are known. Practically it is impossible to derive the optimal query vector, because the full sets of the relevant and irrelevant documents are not available. Wong, Yao and Bollmann [23] studied the linear structure in information retrieval. They designed a very nice gradient descent procedure to compute the coefficients of a linear function and analyzed its performance. In order to update the query vector adaptively, their gradient descent procedure must know the user preference which is in practice the unknown target to be learned by an information retrieval system.

There are many different variants of relevance feedback in information retrieval. However, in this paper we only study Rocchio's similarity-based relevance feedback algorithm [14, 12, 19]. As a first step towards formal analysis of Rocchio's similarity-based relevance feedback algorithm, the work in [9] establishes linear lower bounds on classification mistakes for the algorithm over the binary vector space $\{0,1\}^n$, when any of the four typical similarities (inner product, dice coefficient, cosine coefficient, and Jaccard coefficient) listed in [19] is used. The linear lower bounds obtained in [9] are independent of the query updating factor and the classification threshold that are used by the algorithm. A number of challenging problems regarding further analysis of the algorithm remain open [9]. In [5] we proved that the learning complexity of Rocchio's algorithm is $O(n + n^2(\log n + \log M))$ over the n-dimensional discretized vector space

$\{0, \ldots, M-1\}^n$, when the inner product similarity measure is used. Several general lower bounds were also obtained.

In practice, a fixed query updating factor and a fixed classification threshold are often used in Rocchio's similarity-based relevance feedback algorithm [2, 19]. When this is the case, one shall naturally ask whether the linear lower bounds obtained in [9] can be further strengthened? The main contribution of this paper is to give a positive answer to this question. We prove that Rocchio's algorithm makes $\Omega(k(n-k))$ mistakes in searching for a collection of documents represented by a monotone disjunction of k relevant features over the n-dimensional binary vector space $\{0,1\}^n$, when the inner product similarity measure is used. A quadratic lower bound is obtained when k is linearly proportional to n. In particular, when the zero classification threshold is used, the above quadratic lower bound also holds for the dice coefficient, cosine coefficient and Jaccard coefficient similarity measures. We also prove an $O(k(n-k)^3)$ upper bound for Rocchio's algorithm with the inner product similarity measure in searching for a collection of documents represented by a monotone disjunction of k relevant features with a constant query updating factor and a zero classification threshold. The general upper bound obtained in [5] for Rocchio's algorithm holds when arbitrary query updating factors are allowed. This upper bound does not apply to the case in this paper where only a fixed query updating factor is used.

It should be pointed out that the lower bounds established in [9, 5] and this paper for Rocchio's similarity-based relevance feedback algorithm is based on the following worst case considerations: The user acts as an adversary to the algorithm; the algorithm is required to precisely search for the collection of all the documents relevant to the given search query; and the algorithm is allowed to receive one document example judged by the user as relevance or irrelevant at each step. In practical applications, in contrast to the above worst case considerations, the user in general may not act as an adversary to the algorithm; the algorithm is usually required to search for a short list of top ranked documents relevant to the given search query; and at each step of the similarity-based relevance algorithm, the user may judge a few documents as relevance feedback to the algorithm. In other words, the appropriate situation in real-world information retrieval applications would be a kind of *"sympathetic oracle"* model, where the user is not an adversary to the information retrieval system but a *"sympathetic judge"* who provides the most useful possible information in order to help the system help him/her accomplish his/her work. Hence, our lower bounds proved in this paper as well as those in [9, 5] for Rocchio's similarity-based relevance feedback algorithm *may not affect* the algorithm's ef-

fective applicability to the real-world problems despite of their theoretical significance. The formal analysis of the algorithm helps us understand the nature of the algorithm well so that we may find new strategies to improve its effectiveness or design new algorithms for information retrieval. We have made some progress along this line in [3, 4, 8, 7, 10]. For example, in [3, 4], we have designed two types of multiplicative adaptive algorithms for user preference retrieval with provable better performance: One has better performance than Rocchio's algorithm in learning a class of linear classifiers over non-binary vector space. The other boosts the usefulness of an index term exponentially, while the gradient descent procedure in [23] boosts the usefulness of an index term linearly.

This paper is organized as follows. In section 2, we give a formal presentation of Rocchio's similarity-based relevance feedback algorithm. In section 3, we prove an $\Omega(k(n-k))$ lower bound over the binary vector space $\{0,1\}^n$ for Rocchio's algorithm with a fixed query updating factor when the classification threshold is zero. In section 4, we prove the $\Omega(k(n-k))$ lower bound holds for Rocchio's algorithm with a fixed query updating factor and a non-zero classification threshold over $\{0,1\}^n$. In section 5, we give an $O(k(n-k)^3)$ upper bound for Rocchio's algorithm with a constant query updating factor $\alpha = 1$ and a constant classification threshold $\theta = 0$. We conclude the paper in section 5.

2 Rocchio's Similarity-Based Relevance Feedback Algorithm

Let R be the set of all real values, and let R^+ be the set of all non-negative real values. Let n be a positive integer. In the binary vector space model in information retrieval [19, 21], a collection of n features (or terms) $T_1, T_2, \ldots, T_n$ are used to represent documents and queries. Each document $\mathbf{d}$ is represented as a vector $v_{\mathbf{d}} = (d_1, d_2, \ldots, d_n)$ such that for any i, $1 \leq i \leq n$, the i-th component of $v_{\mathbf{d}}$ is one if the i-th feature T_i appears in $\mathbf{d}$ or zero otherwise. Each query $\mathbf{q}$ is represented by a vector $v_{\mathbf{q}} = (q_1, q_2, \ldots, q_n)$ such that for any i, $1 \leq i \leq n$, the i-th component of $v_{\mathbf{q}} \in R$ is a real value used to determine the relevance (or weight) of the i-th feature T_i. Because of the unique vector representations of documents and queries, for convenience we simply use $\mathbf{d}$ and $\mathbf{q}$ to stand for their vector representations $v_{\mathbf{d}}$ and $v_{\mathbf{q}}$, respectively.

A similarity measure, or similarity for short, in general is a function m from $R^n \times R^n$ to R^+. A similarity m is used to determine the *relevance closeness* of documents to the search query and to rank documents according to such closeness. In the binary vector space model of information

retrieval [19, 21, 2], to retrieve relevant documents for a given query vector $\mathbf{q}$ with respect to a similarity m, the system searches for all the documents $\mathbf{d}$, classifies those with similarity values $m(\mathbf{q}, \mathbf{d})$ higher than an explicit or implicit threshold as relevant, and returns to the user a short list of relevant documents with highest similarity values. This information retrieval process is in fact determined by a linear classifier, as defined later in this section, which is composed of a query vector $\mathbf{q}$, a similarity m, and a real-valued threshold θ.

Unfortunately, in the real-world information retrieval applications, usually an ideal query vector cannot be generated due to many factors such as the limited knowledge of the users about the whole document collection. A typical example is the real-world problem of web search. In such a case, the user may use a few keywords to express what documents are wanted. However, it is nontrivial for both the user and a web search engine to *precisely* define the collection of documents wanted as a query vector composed of a set of keywords. The alternative solution to the query formation problem is, as stated in [19], to conduct searches iteratively, first operating with a tentative query formation (i.e., an initial query vector), and then improving formations for subsequent searches based on evaluations of the previously retrieved materials. This type of methods for automatically generating improved query formation is called relevance feedback, and one particular and well-known example is Rocchio's similarity-based relevance feedback [14, 12, 19].

Rocchio's similarity-based relevance feedback algorithm works in a step by step adaptive refinement fashion as follows. Starting at an initial query vector $\mathbf{q}_1$, the algorithm searches for all the documents $\mathbf{d}$ such that $\mathbf{d}$ is *very close* to $\mathbf{q}_1$ according to the similarity m, ranks them by $m(\mathbf{q}, \mathbf{d})$, and finally presents a short list of the top ranked documents to the user. The user examines the returned list of documents and judges some of the documents as relevant or irrelevant. At step $t \geq 1$, assume that the list of documents the user judged is $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$. Then, the algorithm updates its query vector as $\mathbf{q}_t = \alpha_{t_0}\mathbf{q}_1 + \sum_{j=1}^{t-1} \alpha_{t_j}\mathbf{x}_j$, where the coefficients $\alpha_{t_j} \in R$ for $j = 0, 1, \ldots, t-1$. At step $t+1$, the algorithm uses the updated query vector $\mathbf{q}_t$ and the similarity m to search for relevant documents, ranks the documents according to m, and presents the top ranked documents to the user. In practice, a threshold θ is explicitly (or implicitly) used to select the highly ranked documents. Practically, the coefficients α_{t_j} may be fixed as $1, -1$ or 0.5 [2, 19]. The following four typical similarities were listed in [19]: For any $\mathbf{q}, \mathbf{x} \in R^n$,

$$\textit{inner product}: \quad m_1(\mathbf{q}, \mathbf{x}) = \sum_{i=1}^{n} q_i x_i,$$

$$
\begin{aligned}
\textit{dice coefficient}: \quad m_2(\mathbf{q},\mathbf{x}) &= \frac{2m_1(\mathbf{q},\mathbf{x})}{m_1(\mathbf{q},\mathbf{q})+m_1(\mathbf{x},\mathbf{x})},\\
\textit{cosine coefficient}: \quad m_3(\mathbf{q},\mathbf{x}) &= \frac{m_1(\mathbf{q},\mathbf{x})}{\sqrt{m_1(\mathbf{q},\mathbf{q})}\sqrt{m_1(\mathbf{x},\mathbf{x})}},\\
\textit{Jaccard coefficient}: \quad m_4(\mathbf{q},\mathbf{x}) &= \frac{m_1(\mathbf{q},\mathbf{x})}{m_1(\mathbf{q},\mathbf{q})+m_1(\mathbf{x},\mathbf{x})-m_1(\mathbf{q},\mathbf{x})}.
\end{aligned}
$$

To make the above definitions valid for arbitrary $\mathbf{q}$ and $\mathbf{x}$, we define that the similarity between two zero vectors is zero, i.e.,

$$m_i(\mathbf{0},\mathbf{0}) \;=\; 0,\ for\ 1 \le i \le 4.$$

The similarity-based relevance feedback algorithm is essentially an adaptive supervised learning algorithm from examples [22, 20, 15]. The goal of the algorithm is to learn some unknown classifier 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 method for updating the query vector is similar to the Perceptron algorithm. We given the necessary formal definitions in the following.

Definition 1 *Let m from $R^n \times R^n$ to R^+ be a similarity. A classifier with respect to m over the n-dimensional binary vector space $\{0,1\}^n$ is a triple $(\mathbf{q},\psi,m)$, where $\mathbf{q} \in R^n$ is a query vector, and $\psi \in R$ is a threshold. The classifier $(\mathbf{q},\psi,m)$ classifies any documents $\mathbf{d} \in \{0,1\}^n$ as relevant if $m(\mathbf{q},\mathbf{d}) \ge \theta$ or irrelevant otherwise. The classifier $(\mathbf{q},\psi,m)$ is called a* linear classifier *with respect to the similarity m, if m is a linear function from $R^n \times R^n$ to R^+.*

Definition 2 *An adaptive supervised learning algorithm A for learning a target classifier $(\mathbf{q},\psi,m)$ over the n-dimensional binary vector space $\{0,1\}^n$ from examples is a game played between the algorithm A and the user in a step by step fashion, where the query vector $\mathbf{q}$ and the threshold θ are unknown to the algorithm A, but the similarity m is. At any step $t \ge 1$, A gives a classifier $(\mathbf{q}_t,\theta_t,m)$ as a hypothesis to the target classifier to the user, where $\mathbf{q}_t \in R^n$ and $\theta_t \in R$. If the hypothesis is equivalent to the target, then the user says "yes" to conclude the learning process. Otherwise, the user presents an example $\mathbf{x_t} \in \{0,1\}^n$ such that the target classifier and the hypothesis classifier differ at $\mathbf{x_t}$. In this case, we say that the algorithm A makes a mistake. At step $t+1$, the algorithm A constructs a new hypothetical classifier $(\mathbf{q}_{t+1},\theta_{t+1},m)$ to the user based on the received examples $\mathbf{x_1},\ldots,\mathbf{x_t}$. The learning complexity (or the mistake bound) of the algorithm A is in the worst case the maximum number of examples that it may receive from the user in order to learn some classifier.*

If the readers are familiar with on-line learning from equivalence queries [1, 16], then an adaptive supervised learning algorithm as defined above is a proper on-line learning algorithm for learning the class of classifiers from equivalence queries over the n-dimensional binary vector space. We now give the formal definition of Rocchio's similarity-based relevance feedback algorithm.

Definition 3 *Rocchio's similarity-based relevance feedback algorithm is an adaptive supervised learning algorithm for learning any classifier* $(\mathbf{q}, \theta, m)$ *over the* n*-dimensional binary vector space* $\{0,1\}^n$ *from examples. Let* $\mathbf{q}_1$ *be the initial query vector. At any step* $t \geq 1$*, the algorithm presents a classifier* $(\mathbf{q}_t, \theta_t, m)$ *as its hypothesis to the target classifier to the user, where* $\theta_t \in R$ *is the threshold, and the query vector* $\mathbf{q}_t$ *is modified as follows. Assume that at the beginning of step* t *the algorithm has received a sequence of examples* $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$*, then the algorithm uses the following modified query vector* $\mathbf{q}_t$ *for its next classification:*

$$\mathbf{q}_t \quad = \quad \alpha_{t_0}\mathbf{q}_1 + \sum_{j=1}^{t-1} \alpha_{t_j}\mathbf{x}_j, \tag{1}$$

where $\alpha_{t_j} \in R$*, for* $j = 0, \ldots, t-1$*, are called query updating factors.*

In particular, when a fixed query updating factor $\alpha > 0$ *and a fixed classification threshold* θ *are used, at any step* $t \geq 1$*, Rocchio's algorithm uses* $(\mathbf{q}_t, \theta, m)$ *as its hypothesis to the target classifier, and the query vector* $\mathbf{q}_t$ *is modified as*

$$\mathbf{q}_t \quad = \quad \mathbf{q}_1 + \sum_{j=1}^{t-1} \alpha(y_j^* - y_j)\mathbf{x}_j, \tag{2}$$

where y_j^* *is the binary classification value of the target classifier* $(\mathbf{q}, \psi, m)$ *on* $\mathbf{x}_j$*, and* y_j *is the binary classification value of the hypothesis classifier* $(\mathbf{q}_t, \theta, m)$ *on* $\mathbf{x}_j$*.*

Please note that our definition above is a generalized version of Rocchio's original algorithm. In our definition, any function m from $R^n \times R^n$ to R^+ can be used as a similarity; arbitrary real values can be used in computing the updated query vector; and finally our definition allows adaptive learning until the target is obtained.

3 When the Classification Threshold Is Zero

We will use a set of documents represented by a monotone disjunction of relevant features to study the mistake bounds of Rocchio's algorithm. The efficient learnability of monotone

disjunctions of relevant features (or attributes) has been extensively studied in machine learning (for example, [16]). Although very simple in format, monotone disjunctions are very common ways of expressing search queries, especially in the case of web search. All existing popular search engines support disjunctions of keywords as search query formations. For any k with $1 \leq k \leq n$, classifiers can be defined to precisely classify a monotone disjunction of k relevant features

$$X_{i_1} \vee \cdots \vee X_{i_k}. \tag{3}$$

i.e., to precisely classify whether any given document satisfies the monotone disjunction of (3) or not. If we choose a vector $\mathbf{u} \in R^n$ such that all its components are zero except that those at positions $i_1, \ldots, i_k$ are one, then it is easy to verify that for any $\mathbf{d} \in \{0,1\}^n$, each of the following four expressions is a necessary and sufficient condition for deciding whether $\mathbf{d}$ satisfies (3):

$$\begin{aligned} m_1(\mathbf{u},\mathbf{d}) &\geq \frac{1}{2}, \\ m_2(\mathbf{u},\mathbf{d}) &\geq \frac{2}{k+n}, \\ m_3(\mathbf{u},\mathbf{d}) &\geq \frac{1}{\sqrt{kn}}, \\ m_4(\mathbf{u},\mathbf{d}) &\geq \frac{1}{k+n-1}. \end{aligned}$$

This implies that $(\mathbf{u}, \frac{1}{2}, m_1)$, $(\mathbf{u}, 2/(k+n), m_2)$, $(\mathbf{u}, 1/\sqrt{kn}, m_3)$ and $(\mathbf{u}, 1/(k+n-1), m_4)$ are all respectively classifiers for (3).

From now on, we only consider Rocchio's algorithm with a fixed query updating factor $\alpha > 0$ and a fixed classification threshold θ.

Property 1 *Let $\mathbf{q}_{t+1}$ be the query vector of Rocchio's algorithm at step $t+1 \geq 2$. Then, for any s, $2 \leq s \leq 4$, and any $\mathbf{x} \in \{0,1\}^n$, we have:*
If $\mathbf{q}_1 = \mathbf{0}$, then

$$m_1(\mathbf{q}_{t+1}, \mathbf{x}) \geq 0 \iff m_1(\sum_{j=1}^{t}(y_j^* - y_j)\mathbf{x}_j, \mathbf{x}) \geq 0 \Longleftrightarrow m_s(\sum_{j=1}^{t}(y_j^* - y_j)\mathbf{x}_j, \mathbf{x}) \geq 0. \tag{4}$$

If $\mathbf{q}_1 \neq \mathbf{0}$, then

$$m_1(\mathbf{q}_{t+1}, \mathbf{x}_j) \geq 0 \Longleftrightarrow m_s(\mathbf{q}_{t+1}, \mathbf{x}_j) \geq 0. \tag{5}$$

By expression (2) in Definition 3, when $\mathbf{q}_1 = \mathbf{0}$, $\mathbf{q}_{t+1} = \sum_{j=1}^{t} \alpha(y_j^* - y_j)\mathbf{x}_j$, this yields the equivalence of (4) because $\alpha > 0$. (5) follows from the fact that the enumerator of m_s is m_1 and the denominator of m_s is positive.

Theorem 1 *Rocchio's similarity-based relevance feedback algorithm with similarity m_i, $1 \leq i \leq 4$, makes at least $3k(n-k)$ mistakes in searching for a collection of documents represented by a monotone disjunctions of k relevant features over the binary vector space $\{0,1\}^n$, when the initial query vector $\mathbf{q}_1 = \mathbf{0}$, and the algorithm uses a fixed query updating factor $\alpha > 0$ and a fixed classification threshold $\theta = 0$.*

Proof. By expression (4) of Property 1, we only need to consider the similarity m_1 and the query updating factor $\alpha = 1$. Without loss of generality, let us work on the monotone disjunction

$$F_k \quad = \quad X_1 \vee \cdots \vee X_k \tag{6}$$

of k relevant features. Other monotone disjunctions of k features can be handled with similarly.

The idea is that, for $j = 1, \ldots, k$, we construct a sequence of examples that allow the algorithm to focus on the learning of the relevant feature X_j exclusively. The algorithm gains no information for any other relevant features, hence it is unable to update the components of the query vector corresponding to relevant features other than X_j.

We consider the part of learning the relevant feature X_1. This part is divided into two steps: The preprocessing part and the learning part. We start with the preprocessing part. We construct examples $\mathbf{x}_t$, $t = 1, \ldots, n-k$, such that $x_{t,k+t} = 1$ and all its other components are 0. $\mathbf{x}_1$ does not satisfy F_k. But $m_1(\mathbf{q}_1, \mathbf{x}_1) = 0$, classifying x_1 as relevant. Hence, the algorithm makes one mistake, and the query vector $\mathbf{q}_2$ is updated as

$$\begin{aligned} &q_{2,k+1} = -1, \qquad \text{and} \\ &q_{2,j} = q_{1,j} = 0, \quad 1 \leq j \leq n \text{ and } j \neq k+1. \end{aligned}$$

Similarly, for $2 \leq t \leq n-k$, the algorithm makes a mistake on $\mathbf{x}_t$ and sets

$$\begin{aligned} &q_{t+1,k+t} = -1, \quad \text{and} \\ &q_{t+1,j} = q_{t,j}, \qquad \text{for } 1 \leq j \leq n \text{ and } j \neq k+t. \end{aligned}$$

At the end of this part, the query vector $\mathbf{q}_{n-k+1}$ is update as

$$\begin{aligned} &q_{n-k+1,j} = 0, \qquad \text{for } j = 1, \ldots, k, \text{ and} \\ &q_{n-k+1,k+j} = -1, \quad \text{for } j = 1, \ldots, n-k. \end{aligned}$$

We then begin the learning part. We construct examples $\mathbf{y}_t$ for $t = 1, \ldots, 2(n-k)$. We set

$$y_{t,1} = \begin{cases} 1, & \text{if } t \text{ is odd,} \\ 0, & \text{if } t \text{ is even.} \end{cases}$$
$$y_{t,l} = 0, \text{ for } 2 \le l \le k, \text{ and}$$
$$y_{t,k+l} = 1, \text{ for } 1 \le l \le n-k.$$

Obviously, $\mathbf{y}_t$ satisfies F_k if t is odd, and it does not if t is even. For $\mathbf{y}_1$, $m_1(\mathbf{q}_{n-k+1}, \mathbf{y}_1) = -(n-k) < 0$. This implies that the algorithm makes a mistake, and the query vector $\mathbf{q}_{n-k+2}$ is updated as

$$\begin{aligned} &q_{n-k+2,1} = 1, \\ &q_{n-k+2,l} = 0, \qquad \text{for } l = 2, \ldots, k, \text{ and} \\ &q_{n-k+2,k+l} = 0, \quad \text{for } l = 1, \ldots, n-k. \end{aligned}$$

For $\mathbf{y}_2$, $m_1(\mathbf{q}_{n-k+2}, \mathbf{y}_2) = 0$, implying that the algorithm makes another mistake, and the query vector $\mathbf{q}_{n-k+3}$ is updated as

$$\begin{aligned} &q_{n-k+3,1} = 1, \\ &q_{n-k+3,l} = 0, \qquad \text{for } l = 2, \ldots, k, \text{ and} \\ &q_{n-k+3,k+l} = -1, \quad \text{for } l = 1, \ldots, n-k. \end{aligned}$$

It follows from the similar analysis that the algorithm makes one mistake for each $\mathbf{y}_t$, and the query vector has the following property:

$$q_{n-k+t+1,1} = \begin{cases} (t+1)/2, & \text{for } t = 1, 3, \ldots, 2(n-k)-1, \\ t/2, & \text{for } t = 2, 4, \ldots, 2(n-k). \end{cases}$$
$$q_{n-k+t+1,l} = 0, \text{ for } l = 2, \ldots, k,$$
$$q_{n-k+t+1,k+l} = \begin{cases} 0, & \text{for } t = 1, 3, \ldots, 2(n-k)-1 \text{ and } 1 \le l \le n-k \\ -1, & \text{for } t = 2, 4, \ldots, 2(n-k) \text{ and } 1 \le l \le n-k. \end{cases}$$

In summary, both the preprocessing part and the learning part for the relevant feature X_1 force the algorithm to make at least $3(n-k)$ mistakes. Moreover, at the end the query vector $\mathbf{q}_{3(n-k)+1}$ is updated as

$$\begin{aligned} &q_{3(n-k)+1,1} = n-k, \\ &q_{3(n-k)+1,l} = 0, \qquad \text{for } l = 2, \ldots, k, \text{ and} \\ &q_{3(n-k)+1,k+l} = 0, \qquad \text{for } l = 1, \ldots, n-k. \end{aligned}$$

By simple induction, we can utilize the similar preprocessing part and learning part to force the algorithm to make $3(n-k)$ mistake to learn each of the remaining relevant features X_j,

$2 \leq j \leq k$. Therefore, the algorithm makes at least $3k(n-k)$ mistakes in searching for the collection of documents represented by F_k. □

Theorem 2 *Rocchio's similarity-based relevance feedback algorithm with similarity m_s, $1 \leq s \leq 4$, makes $\Omega(k(n-k))$ mistakes in searching for documents represented by a monotone disjunction of k relevant features over the binary vector space $\{0,1\}^n$, when the initial query vector $\mathbf{q}_1 \in \{0,1\}^n$ is not $\mathbf{0}$, the query updating factor $\alpha > 0$ is a constant and the classification threshold $\theta = 0$.*

Proof. By expression (5) of Property 1, we only need to consider the similarity m_1. Again, we work on the monotone disjunction F_k, defined in expression (6), of k relevant features. The idea of proof is similar to that of Theorem 1. That is, for each relevant feature, we construct examples that force the algorithm to focus on the learning of that feature exclusively.

We shall address two cases, the initial query vector $\mathbf{q}_1$ has either at least k zero components or fewer than k zero components.

Case 1: In this case, we assume without loss of generality that $q_{1,i} = 0$ for $i = 1, \ldots, k+m$, and $q_{1,k+m+j} = 1$ for $j = 1, \ldots, n-k-m$, where $0 \leq m \leq n-k$.

We start with the initial phase to set the $(k+j)$-th component of the query vector to $-\alpha$ for $j = 1, \ldots, m$. We construct examples $\mathbf{x}_t$ for $t = 1, \ldots, m$ such that $x_{t,k+t} = 1$ and all its other components are 0. $\mathbf{x}_t$ does not satisfy the given monotone disjunction F_k. By simple induction, $m_1(\mathbf{q}_t, \mathbf{x}_t) = 0$, implying that $\mathbf{x}_t$ is classified by the algorithm as relevant. Hence, the algorithm makes one mistake on $\mathbf{x}_t$, and the $(k+t)$-th component of the query vector $\mathbf{q}_{t+1}$ is updated as $q_{t+1,k+t} = -\alpha$ and all the other components are the same as those in $\mathbf{q}_t$. At the end, the query vector $\mathbf{q}_{m+1}$, denoted as $\mathbf{A}_1 = \mathbf{q}_{m+1}$ to simplify notation, is updated as

$$\begin{aligned} A_{1,j} &= 0, && \text{for } 1 \leq j \leq k, \\ A_{1,k+j} &= -\alpha, && \text{for } j = 1, \ldots, m, \\ A_{1,k+m+j} &= 1, && \text{for } j = 1, \ldots, n-k-m. \end{aligned}$$

The second phase is to manipulate the $n-k-m$ many components of $\mathbf{A}_1$ with value 1. Let T be the smallest integer satisfying $T > 1/\alpha$. We have

$$0 < T - 1/\alpha \leq 1. \tag{7}$$

For each j with $k+m+1 \leq j \leq n$, in order to force the algorithm to make mistakes at the j-th component of $\mathbf{A}_1$ we construct examples $\mathbf{y}_t$ such that $y_{t,j} = 1$ and all its other components

are 0, $t = 1, \ldots, T$. $\mathbf{y}_t$ does not satisfy the given monotone disjunction F_k. By simple induction, $m_1(\mathbf{A}_t, \mathbf{y}_t) = 1-(t-1)\alpha$, which is greater than or equal to zero because of the choice of T. This implies that the algorithm classifies $\mathbf{y}_t$ as relevant. Hence, the algorithm makes one mistake on $\mathbf{y}_t$, and the j-th component of the query vector $\mathbf{A}_{t+1}$ is updated as $A_{t+1,j} = 1 - t\alpha$ and all its other components remain the same as those in $\mathbf{A}_t$. At the end of this phase, the algorithm makes $T(n-k-m)$ mistakes. At this point, to simplify notation we let $\mathbf{B}_1 = \mathbf{A}_{T(n-k-m)+1}$, which is

$$B_{1,l} = 0, \text{ for } 1 \leq l \leq k, \tag{8}$$

$$B_{1,k+l} = -\alpha, \text{ for } j = 1, \ldots, m, \text{ and} \tag{9}$$

$$B_{1,k+m+l} = 1 - T\alpha, \text{ for } l = 1, \ldots, n-k-m. \tag{10}$$

The third phase is for learning the relevant feature X_1. We construct examples $\mathbf{z}_t$ for $t = 1, \ldots, 2(m + (n-k-m)(T - \frac{1}{\alpha}))$. We set

$$z_{t,1} = \begin{cases} 0, & \text{if } t \text{ is even,} \\ 1, & \text{if } t \text{ is odd,} \end{cases}$$

$$z_{t,l} = 0, \text{ for } 2 \leq l \leq k, \text{ and}$$

$$z_{t,k+l} = 1, \text{ for } 1 \leq l \leq n-k.$$

Obviously, $\mathbf{z}_t$ satisfies the given monotone disjunction F_k if t is odd, and it does not if t is even. By simple induction, when t is odd, we have $m_1(\mathbf{B}_t, \mathbf{z}_t) = (t-1)\alpha/2 - m\alpha + (n-k-m)(1-T\alpha) < 0$; and when t is even, we have $m_1(\mathbf{B}_t, \mathbf{z}_t) = t\alpha/2 + (n-k-m)\alpha + (n-k-m)(1-T\alpha) \geq (n-k-m)(\alpha + 1 - T\alpha) \geq 0$. The last step follows from expression (7). This means that the algorithm makes a mistake for each example $\mathbf{z}_t$, and the query vector is updated as follows: For $t = 1, 3, \ldots, 2(m + (n-k-m)(T - \frac{1}{\alpha})) - 1$, $B_{t+1,1} = \frac{(t+1)\alpha}{2}$, $B_{t+1,l} = 0$ for $l = 2, \ldots, k+m$, and $B_{t+1,l} = 1 - (T-1)\alpha$ for $l = k+m+1, \ldots, n$; for $t = 2, 4, \ldots, 2(m + (n-k-m)(T - \frac{1}{\alpha}))$, $B_{t+1,1} = \frac{t\alpha}{2}$, $B_{t+1,l} = 0$ for $j = 2, \ldots, k$, $B_{t+1,l} = -\alpha$ for $l = k+1, \ldots, k+m$, and $B_{t+1,l} = 1 - T\alpha$ for $l = k+m+1, \ldots, n$.

The above analysis implies that the algorithm makes $2(m+(n-k-m)(T-\frac{1}{\alpha}))$ mistakes in the phase of learning the relevant feature X_1. Comparing the query vector at the end of this phase with expressions (9) to (10), it is interesting to notice that the query vector at the beginning and at the end of this phase remains the same, except for its first component. Precisely, we have

$$B_{1,1} = 0, \text{ but } B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,1} = m + (n-k-m)(T - \frac{1}{\alpha}), \tag{11}$$

$$B_{1,j} = B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,j} = 0, \quad j = 2, \ldots, k, \tag{12}$$

$$B_{1,j} = B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,k+j} = -\alpha, \quad j = 1, \ldots, m, \quad \text{and} \tag{13}$$

$$B_{1,j} = B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,k+m+j} = 1 - T\alpha, \quad j = 1, \ldots, n-k-m. \tag{14}$$

We can apply a similar phase to the phase for learning the relevant feature X_1 for each of the remaining relevant features X_j, $2 \leq j \leq k$. That is, we can construct a new example $\mathbf{z}'_t$ from each $\mathbf{z}_t$. The new example $\mathbf{z}'_t$ is obtained via changing the j-th component of $\mathbf{z}_t$ to 1 if t is odd and to 0 if t is even, the i-th component to zero for $1 \leq i \leq k$ but $i \neq j$, and letting the other $n-k$ components remain the same as those in $\mathbf{z}_t$. By simple induction, the algorithm will make one mistake on each $\mathbf{z}'_t$, the query vector satisfies the similar invariant property as shown in expressions (12) to (14). Therefore, the algorithm makes at least $2k(m + (n-k-m)(T - \frac{1}{\alpha}))$ mistakes in all the k phases of learning the relevant feature X_i for $1 \leq i \leq k$. Combining with the first two phases, the algorithm makes in total at least $m + (n-k-m)T + 2k(m + (n-k-m)(T - \frac{1}{\alpha})) = \Omega(k(n-k))$ mistakes.

Case 2: In the second case, we assume without loss of generality that $q_{1,i} = 0$ for $i = 1, \ldots, m'$ where $0 \leq m' < k$, $q_{1,i} = 1$ for $i = m'+1, \ldots, k, \ldots, n$.

In this case, we do not need the initial phase in Case 1. For consistence with Case 1, we let $\mathbf{A}_1 = \mathbf{q}_1$. We first follow the second phase in Case 1 to manipulate, for each j, $1 \leq j \leq n-k$, the 1-component $A_{1,k+j}$ by constructing examples $\mathbf{y}_t$ for $t = 1, \ldots, T$. Like in the second phase in Case 1, each of such examples forces the algorithm to make a mistake, and to update $A_{t+1,j}$ to $1 - t\alpha$. At the end of this phase, the algorithm makes $(n-k)T$ mistakes, and the query vector $\mathbf{A}_{(n-k)T+1}$, denoted as $\mathbf{B}_1$ to simply notation, becomes $B_{1,l} = 0$ for $1 \leq l \leq m'$, $B_{1,l} = 1$ for $m'+1 \leq l \leq k$, and $B_{1,k+l} = 1 - T\alpha$ for $l = 1, \ldots, n-k$.

For i, $1 \leq i \leq m'$, we follow the same phase as that in Case 1 for learning the relevant feature X_i by constructing examples $\mathbf{z}_t$ for $t = 1, \ldots, 2(n-k)(T - \frac{1}{\alpha})$. The only difference is that here we have $m = 0$. Like in that phase in Case 1, these examples forces the algorithm to make $2(n-k)(T - \frac{1}{\alpha})$ mistakes, and the query vector maintains the similar invariant property as exhibited in expressions (12) to (14) in Case 1.

For i, $m'+1 \leq i \leq k$, we again follow the same phase as that in Case 1 for learning the relevant feature X_i by constructing examples $\mathbf{z}_t$ for $t = 1, \ldots, 2((n-k)(T - \frac{1}{\alpha}) - \frac{1}{\alpha})$. The two differences that we have here are $m = 0$ and $B_{1,i} = 1$. The fact of $m = 0$ will not change the process. But the fact of $B_{1,i} = 1$ will change the number of examples to $2((n-k)(T - \frac{1}{\alpha}) - \frac{1}{\alpha})$,

because the i-th component of the query vector B_{t+1} becomes $1+\frac{(t+1)\alpha}{2}$ for odd t, and $1+\frac{t\alpha}{2}$ for even t. Again, these examples forces the algorithm to make $2((n-k)(T-\frac{1}{\alpha})-\frac{1}{\alpha})$ mistakes, and the similar invariant property as exhibited in expressions (12) to (14) in Case 1. is maintained for the query vector.

Putting the above analysis together, the algorithm makes at least

$$L(\alpha) = (n-k)T + 2m'(n-k)(T-\frac{1}{\alpha}) + 2(k-m')((n-k)(T-\frac{1}{\alpha})-\frac{1}{\alpha})$$

mistakes in Case 2. If $\alpha = 1$, then $T = 2$, thus $L(\alpha) \geq 2k(n-k) + 2(n-2k+m')$. If $\alpha > 1$, then $T = 1$, thus

$$L(\alpha) \geq \frac{2k(n-k)(\alpha-1)}{\alpha} + n - 3k.$$

If $0 < \alpha < 1$, we consider two cases, either $1/\alpha = p$ for some integer $p > 1$, or $1/\alpha = p + r$ for some integer $p > 1$ and some decimal value r with $0 < r < 1$. When $1/\alpha = p$, we have $T = p+1$, and

$$L(\alpha) = (2k+1+\frac{1}{\alpha})(n-k) - \frac{2(k-m')}{\alpha}.$$

When $1/\alpha = p + r$, we have $T = p + 1$, and

$$L(\alpha) \geq 2k(n-k)(1-r) + (n-3k+2m')/\alpha.$$

The above analysis yields an $\Omega(k(n-k))$ lower bound for Rocchio's algorithm in Case 2. □

4 When the Classification Threshold Is Not Zero

One can introduce an auxiliary feature variable to deal with the threshold so that a linear classifier over the n-dimensional vector space with a non-zero threshold is equivalent to a linear classifier over the $(n+1)$-dimensional vector space with a zero threshold. Thus, it is tempting to use Theorems 1 and 2 to derive lower bounds for Rocchio's algorithm with a non-zero classification threshold over the $(n+1)$-dimensional binary vector space. However, one shall notice that the auxiliary feature variable will always maintain a value 1 in any example given to the algorithm and the threshold may have arbitrary values other than just 1 or 0. Therefore, the proofs of these two theorems cannot be applied here.

Theorem 3 *Rocchio's similarity-based relevance feedback algorithm with similarity m_1 makes $\Omega(k(n-k))$ mistakes in searching for a collection of documents represented by a monotone*

disjunction of k relevant features over the binary vector space $\{0,1\}^n$, when the query updating factor $\alpha > 0$ is a constant and the initial query vector is $\mathbf{q}_1 = \mathbf{0}$.

Proof. As in the previous section, let us consider the monotone disjunction F_k defined in expression (6), and for each i, $1 \leq i \leq k$, we construct examples that allow the algorithm to focus on the learning of the relevant feature X_i exclusively. We shall analyze four cases for the classification threshold θ.

Case 1: $\theta \geq (n-k)\alpha$. For each relevant feature X_i, $1 \leq i \leq k$, we construct a sequence of examples $\mathbf{x}_t, t = 1, \ldots, (n-k)$, such that $x_{t,i} = 1$ and $x_{t,j} = 0$ for $1 \leq j \leq n$ and $j \neq i$. Note that $\mathbf{x}_t$ satisfies F_k. Since $\mathbf{q}_1 = \mathbf{0}$, by simple induction, we have $m_1(\mathbf{q}_{(i-1)(n-k+1)+t}, \mathbf{x}_t) = (t-1)\alpha < \theta$. This implies that the algorithm makes a mistake on $\mathbf{x}_t$, and the query vector is updated as

$$\begin{aligned}
&\mathbf{q}_{(i-1)(n-k)+t+1,i} = t\alpha, \\
&\mathbf{q}_{(i-1)(n-k)+t+1,j} = (n-k)\alpha, \quad \text{for } 1 \leq j \leq i-1, \text{ and} \\
&\mathbf{q}_{(i-1)(n-k)+t+1,j} = 0, \qquad\qquad i+1 \leq j \leq n,
\end{aligned}$$

At the end, the algorithm makes at least $k(n-k+1)$ mistakes in learning all the k relevant features in F_k.

Case 2: $0 < \theta < (n-k)\alpha$. We choose the integer T such that $(T-1)\alpha < \theta \leq T\alpha$, $1 \leq T \leq (n-k)$. Let $N = \lfloor \frac{n-k}{T} \rfloor$. For the relevant feature X_1, we first construct an example $\mathbf{x}_1$ such that $x_{1,i} = 1$, $x_{1,k+j} = 1, j = 1, \ldots, NT$, and the remaining components are zero. This example satisfies F_k, but $m_1(\mathbf{q}_1, \mathbf{x}_1) = 0$ since $\mathbf{q}_1 = \mathbf{0}$. Hence, the algorithm makes a mistake, and the query vector is updated as $q_{2,1} = \alpha$, $q_{2,k+j} = \alpha$ for $j = 1, \ldots, NT$, and all the other components remain 0. Second, we construct examples $\mathbf{x}_t$ for $t = 2, \ldots, T$ such that $x_{t,1} = 1$ and all its other components are zero. This example satisfies F_k, but by simple induction $m_1(\mathbf{q}_t, \mathbf{x}_t) = (t-1)\alpha$, which is less than θ. Hence, the algorithm makes one mistakes on each $\mathbf{x}_t$, and the query vector is updated as $q_{t+1,1} = t\alpha$ and all the other components remain unchanged. Third, for each relevant feature X_i, $2 \leq i \leq k$, we construct examples $\mathbf{w}_t$ for $t = 1, \ldots, T$ such that $w_{t,i} = 1$ and all its other components are zero. This example satisfies F_k, but by simple induction $m_1(\mathbf{q}_{T(i-1)+t}, \mathbf{w}_t) = (t-1)\alpha < \theta$. Hence, the algorithm makes one mistakes on each $\mathbf{w}_t$, and the query vector is updated as $q_{T(i-1)+t+1,i} = t\alpha$ and all the other components remain unchanged. At the end of the above three steps, the algorithm makes kT mistakes, and the query vector $\mathbf{q}_{kT+1}$, denoted by $\mathbf{A}_1$ for simplicity, is updated as

$$A_{1,i} = T\alpha, \text{ for } 1 \leq i \leq k,$$

$$A_{1,k+j} = \alpha, \text{ for } 1 \le j \le NT, \text{ and}$$
$$A_{1,k+NT+l} = 0, \text{ for } 1 \le l \le n - NT - k.$$

Now, we construct examples $\mathbf{y}_t$ for $t = 1, \ldots, N-1$ such that $y_{t,k+T(t-1)+j} = 1, j = 1, \ldots, T$, and all its other components are zero. Each $\mathbf{y}_t$ does not satisfy F_k, but by simple induction $m_1(\mathbf{A}_t, \mathbf{y}_t) = T\alpha \ge \theta$. This forces the algorithm to make one mistake on $\mathbf{y}_t$ and to set $A_{t+1,k+T(t-1)+j} = 0$ for $j = 1, \ldots, T$, while keeping all the other components unchanged. For $t = N$, we set $y_{t,k+j} = 1$ for $j = 1, \ldots, NT$, and all the other components are zero. Again, this example does not satisfy F_k and forces the algorithm to make one mistake. The query vector $\mathbf{A}_{N+1}$, denoted as $\mathbf{B}_1$, is updated as

$$B_{1,i} = T\alpha, \text{ for } 1 \le i \le k,$$
$$B_{1,k+j} = -\alpha, \text{ for } j = 1, \ldots, (N-1)T, \text{ and}$$
$$B_{1,k+j} = 0, \text{ for } j = (N-1)T + 1, \ldots, NT.$$

At this point, we have finished the preprocessing phase and obtained the query vector $\mathbf{B}_1$. Next, we consider the step of learning the relevant feature X_i, for $1 \le i \le k$. We construct examples $\mathbf{z}_t$ for $t = 1, \ldots, 2(N-1)T$ such that $z_{t,i} = 1$ if t is odd and $z_{t,i} = 0$ if t is even, $z_{t,k+j} = 1$ for $j = 1, \ldots, NT$, and all the other components are zero. Notice that $\mathbf{z}_t$ satisfies F_k if and only if t is odd. By simple induction, the algorithm makes one mistake on each $\mathbf{z}_t$, and the query vector is updated as

$$B_{t+1,i} = \begin{cases} T\alpha + \frac{(t+1)\alpha}{2} & \text{for odd } t, \\ T\alpha + \frac{t\alpha}{2} & \text{for even } t, \end{cases} \tag{15}$$
$$B_{t+1,j} = B_{t,j} \text{ for } 1 \le j \le k \text{ and } j \ne i, \tag{16}$$
$$B_{t+1,k+j} = \begin{cases} 0 & \text{for odd } t \text{ and } 1 \le j \le (N-1)T, \\ -\alpha & \text{for even } t \text{ and } 1 \le j \le (N-1)T, \\ \alpha & \text{for odd } t \text{ and } (N-1)T + 1 \le j \le NT, \\ 0 & \text{for even } t \text{ and } (N-1)T + 1 \le j \le NT. \end{cases} \tag{17}$$
$$B_{t+1,j} = 0 \text{ for } k + NT + 1 \le j \le n. \tag{18}$$

At the end of this step, the algorithm makes $2(N-1)T + 1$ mistakes and, the query vector $\mathbf{B}_{2(N-1)T}$ maintains the following invariant property as exhibited in expressions (15) to (18): all its components are the same as those in $\mathbf{B}_1$, except that the i-component is updated successfully to learn the relevant feature X_i. This invariant property allows us to follow the similar step for

all the other relevant features X_j, $1 \le j \le k$ but $j \ne i$, to force the algorithm to make $2(N-1)T$ mistakes for each of those relevant features. Therefore, the algorithm makes $2k(N-1)T$ mistakes during these steps.

In summary, in this case the algorithm make at least $kT + N + 2k(N-1)T = \Omega(k(n-k))$ mistakes, since $1 \le T \le (n-k)$ and $N = \lfloor \frac{n-k}{T} \rfloor$.

Case 3: $\theta < -(n-k)\alpha$. We choose the integer $T' \ge (n-k)$ such that $-\alpha(T'+1) < \theta \le -\alpha T'$. In the initial step, for each j with $1 \le j \le n-k$, we construct examples $\mathbf{x}_t, t = 1, \ldots, T'+1$ such that $x_{t,k+j} = 1$, and $x_{t,l} = 0$ for $l \ne j$. These examples do not satisfy F_k. By simple induction, each $\mathbf{x}_t$ forces the algorithm to make a mistake and the query vector is updated as $q_{(j-1)(T'+1)+t+1,j} = -t\alpha$, and $q_{(j-1)(T'+1)+t+1,l} = q_{(j-1)(T'+1)+t,l}$ for $l \ne j$. At the end, the algorithm makes $(n-k)(T'+1)$ mistakes, and the query vector is updated as $q_{(n-k)(T'+1)+1,j} = -(T'+1)\alpha$ for $k+1 \le j \le n$, and $q_{(n-k)(T'+1)+1,l} = 0$ for $1 \le j \le k$. We let $\mathbf{A'}_1 = \mathbf{q}_{(n-k)(T'+1)+1}$ to simplify notation.

Now, for $i = 1, \ldots, k$, we consider the step of learning the relevant feature X_i. For X_i, we construct $(n-k)(T'+1)$ sequences of examples $\mathbf{y}_t$, $\mathbf{z}^t_s$, $s = 1, 2, \ldots, n-k$, $t = 1, \ldots, (n-k)(T'+1)$. Here, $y_{t,i} = 1$, $y_{t,j} = 0$ for $1 \le j \le k$ and $j \ne i$, and $y_{t,k+j} = 1$ for $1 \le j \le n-k$; $z^t_{s,k+s} = 1$ and all its other components are zero. Obviously, $\mathbf{y}_t$ satisfies F_k but $\mathbf{z}^t_s$ does not. By simple induction, each of $\mathbf{y}_t$ and $\mathbf{z}^t_s$ forces the algorithm to make a mistake, and at the end of the t-th sequence of examples, the query vector is updated as

$$\begin{aligned} A'_{t(n-k+1)+1,i} &= t\alpha, \\ A'_{t(n-k+1)+1,j} &= A'_{t(n-k+1),j}, \text{ for } 1 \le j \le k \text{ and } j \ne i. \\ A'_{t(n-k+1)+1,k+j} &= -(T'+1)\alpha, \text{ for } 1 \le j \le n-k. \end{aligned}$$

At the end of these steps for learning all the k relevant features, the algorithm makes $k(n-k)(T'+1)(n-k+1)$ mistakes. Adding the mistakes the algorithm made before, the total number of mistakes in this case is at least $k(n-k)(T'+1)(n-k+1) \ge k(n-k)^3$.

Case 4: $-(n-k)\alpha < \theta < 0$. we choose T'' such that $-(T''+1)\alpha < \theta \le -T''\alpha$ for some integer T'' with $0 \le T'' \le n-k-1$. For this T'', we have $-(n-k) \le -(T''+1)$. Initially, for any $1 \le j \le n-k$, we construct examples $\mathbf{x}_t, t = 1, \ldots, T''+1$, such that $x_{t,j} = 1$ and $x_{t,l} = 0$ for $l \ne j$. As in the initial phase of Case 3, each $\mathbf{x}_t$ forces the algorithm to make a mistake. After processing all j, $1 \le j \le n-k$, the algorithm makes $(n-k)(T''+1)$ mistakes, and the query vector $\mathbf{q}_{(n-k)(T''+1)+1}$, denoted as $\mathbf{A''}_1$ to simplify notation, will be $A''_{1,k+j} = -(T''+1)\alpha$

for $1 \leq j \leq n-k$, and $A''_{1,i} = 0$ for $1 \leq i \leq k$.

Next, for each i, $1 \leq i \leq k$, we consider the step of learning the relevant feature X_i. For $t = 1, \ldots, (n-k)(T''+1)$, we construct a sequence of examples $\mathbf{y}_t$, $\mathbf{z}^t_s$, $s = 1, 2, \ldots, n-k$. Here, $y_{t,i} = 1$ and $y_{t,j} = 0$ for $1 \leq j \leq k$ and $j \neq i$, and $y_{t,k+j} = 1$ for $1 \leq j \leq n-k$; $z^t_{s,k+s} = 1$ $z^t_{s,j} = 1$ for $1 \leq j \leq n$ and $j \neq k+k$. Notice that $\mathbf{y}_t$ is satisfies F_k, but those $n-k$ examples $\mathbf{z}^t_s$ does not. By simple induction, for each t, $\mathbf{y}_t$ and $\mathbf{z}_{t,s}$ forces the algorithm to make a mistake, and $\mathbf{y}_t$ is used to update the query vector by adding α to its i-th component and the last $n-k$ components while keeping all other components unchanged. Each of those $n-k$ examples $\mathbf{z}^t_s$ is used to update the query vector by adding $-\alpha$ to the $(k+s)$-th component while keeping all the other components unchanged. Thus, after processing $\mathbf{y}_t$ and $\mathbf{z}^t_s$, $1 \leq s \leq n-k$, $A''_{t(n-k+1)+1,i} = t\alpha$, $A''_{t(n-k+1)+1,j} = A''_{t(n-k+1),j}$ for $1 \leq j \leq k$ and $j \neq i$, and $A''_{t(n-k+1)+1,k+j} = -(T''+1)\alpha$ for $1 \leq j \leq n-k$. This invariant property remains the same for the other relevant features X_j, $1 \leq j \leq k$ and $j \neq i$. The total number of mistakes at the end of these steps for learning all the k relevant features is $k(n-k)(T''+1)(n-k+1)$ mistakes. Adding the mistakes the algorithm made before, the total number of mistakes in this case is at least $(n-k)(T''+1)+k(n-k)(T''+1)(n-k+1) \geq k(n-k)^2$. □

In the following, we consider that the constant query updating factor is 1.

Theorem 4 *Rocchio's similarity-based relevance feedback algorithm with similarity m_1 makes $\Omega(k(n-k))$ mistakes in searching for a collection of documents represented by a monotone disjunction of k relevant features over the binary vector space $\{0,1\}^n$, when the constant query updating factor $\alpha = 1$ and the initial query vector is $\mathbf{q}_1 \in \{0,1\}^n$ is not $\mathbf{0}$.*

Proof. As in the proof of Theorem 3, we consider the monotone disjunction F_k defined in expression (6), and analyze four cases for the classification threshold θ.

Case 1: $\theta \geq (n-k)$. For each relevant feature X_i, $1 \leq i \leq k$, we construct a sequence of examples $\mathbf{x}_t, t = 1, \ldots, n-k-1$, such that $x_{t,i} = 1$ and $x_{t,j} = 0$ for $1 \leq j \leq n$ and $j \neq i$. Note that $\mathbf{x}_t$ satisfies F_k. Since $\mathbf{q}_{1,i} = 0$ or 1, by simple induction, we have $t-1 \leq m_1(\mathbf{q}_t, \mathbf{x}_t) = t < n-k \leq \theta$. This implies that the algorithm makes a mistake on $\mathbf{x}_t$. At the end, the algorithm makes at least $k(n-k-1)$ mistakes in learning all the k relevant features in F_k.

Case 2: $0 < \theta < n-k$. If $\theta \geq (n-k)/10$, then following the same approach as for Case 1 we can prove that the algorithm makes at least $k((n-k)/10-1)$ mistakes in learning all the k relevant features in F_k. In the rest of this case we assume $0 < \theta < (n-k)/10$. We consider the

following two subcases of the initial query vector $\mathbf{q}_1$.

Subcase 2.1: The initial query vector $\mathbf{q}_1$ have at least k zero components. Without loss of generality, we assume that $q_i = 0$ for $1 \leq i \leq k+m$ and $q_j = 1$ for $m+k+1 \leq j \leq n$, where $0 \leq m \leq n-k$.

We choose the example $\mathbf{x}$ such that $x_j = 1$ for $j = 1$ or $k+1 \leq j \leq k+m$, and all its other components are zero. $\mathbf{x}$ satisfied F_k, but $m_1(\mathbf{q}_1, \mathbf{x}) = 0$. Hence, $\mathbf{x}$ forces the algorithm to make a mistake and the query vector is updated as $q_{2,1} = 1, q_{2,j} = 0$ for $2 \leq j \leq k$ and $q_{2,k+j} = 1$ for $1 \leq j \leq n-k$. Let T be the integer such that $T-1 < \theta \leq T$ and $N = \lceil (n-k)/T \rceil$. Since $0 < \theta < (n-k)/10$, $N \geq 10$. Let $\mathbf{x}_2$ be the example such that $x_{2,j} = 1$ for $j = k+1, k+1, \ldots, (N-1)T$ and all its other components are zero. This example does not satisfies F_k but $m_1(\mathbf{q}_2, \mathbf{x}_2) = T(N-1) > T \geq \theta$. Hence, it forces the algorithm to make a mistake and the query vector $\mathbf{q}_3$ becomes

$$
\begin{aligned}
&q_{3,j} = 1, \quad \text{for } j = 1, \\
&q_{3,j} = 0, \quad \text{for } 2 \leq j \leq k + (N-1)T, \\
&q_{3,j} = 1, \quad \text{for } j = k + (N-1)T + 1, \ldots, n.
\end{aligned}
$$

We choose the example $\mathbf{w}$ such that $w_j = 1$ for $k+1 \leq j \leq k+NT$ and all its other components are zero. This example does not satisfy F_k, but $m_1(\mathbf{q}_3, \mathbf{w}) = T \geq \theta$. Hence, the algorithm makes a mistake on $\mathbf{w}$ and query vector $\mathbf{q}_4$, denoted as $\mathbf{A}_1$ to simplify notation, becomes

$$A_{1,1} = 1, \text{ for } j = 1, \tag{19}$$

$$A_{1,j} = 0, \text{ for } 2 \leq j \leq k, \tag{20}$$

$$A_{1,k+j} = -1, \text{ for } 1 \leq j \leq (N-1)T, \tag{21}$$

$$A_{1,k+(N-1)T+j} = 0, \text{ for } 1 \leq j \leq T, \tag{22}$$

$$A_{1,k+NT+j} = 1, \text{ for } 1 \leq j \leq n-k-NT. \tag{23}$$

We now consider the step of learning the relevant feature X_i, $2 \leq i \leq k$. For X_i, we construct a sequence a examples $\mathbf{y}_t$, $t = 1, 2, \ldots, 2NT$ such that

$$
\begin{aligned}
&y_{t,i} = \begin{cases} 1, & \text{if } t \text{ is odd}, \\ 0, & \text{if } t \text{ is even}, \end{cases} \\
&y_{t,j} = 0, \text{ for } 1 \leq j \leq k \text{ and } j \neq i, \\
&y_{t,k+j} = 1, \text{ for } 1 \leq j \leq NT, \\
&y_{t,k+NT+j} = 0, \text{ for } 1 \leq j \leq n-k-NT.
\end{aligned}
$$

$\mathbf{y}_t$ satisfies F_k if and only if t is odd. By simple induction, if t is odd, we have $m_1(\mathbf{A}_t, \mathbf{y}_t) = -(N-1)T+(t-1)/2 \leq -(N-1)T+NT-1 = T-1 < \theta$; if t is even, we have $m_1(\mathbf{A}_t, \mathbf{y}_t) = T > \theta$; and the query vector is updated as

$$
\begin{aligned}
&A_{t+1,1} = 1;\\
&A_{t+1,i} = \begin{cases} (t+1)/2, & \text{if } t \text{ is odd,} \\ t/2. & \text{if } t \text{ is even,} \end{cases}\\
&A_{t+1,j} = A_{t,j}, \text{ for } 2 \leq j \leq k \text{ and } j \neq i,\\
&A_{t+1,k+j} = \begin{cases} 0, & \text{if } t \text{ is odd and } 1 \leq j \leq (N-1)T, \\ -1, & \text{if } t \text{ is even and } 1 \leq j \leq (N-1)T, \end{cases}\\
&A_{t+1,k+(N-1)T+j} = \begin{cases} 1, & \text{if } t \text{ is odd and } 1 \leq j \leq T, \\ 0, & \text{if } t \text{ is even and } 1 \leq j \leq T, \end{cases}\\
&A_{t+1,k+NT+j} = 1, \text{ for } 1 \leq j \leq n-k-NT.
\end{aligned}
$$

Hence, each $\mathbf{y}_t$ forces the algorithm to make a mistake. Thus, the algorithm makes at least NT mistakes for learning each of the relevant features X_i, $2 \leq i \leq k$. A similar sequence of examples can be applied the learning of X_1. Since query vector has a value 1 corresponding to X_1, we need $2(NT-1)$ examples to form the sequence. At the end, the algorithm makes in total at least $2(k-1)NT + 2(NT-1) + 3 = 2kNT + 1 \geq k(n-k)/5$ mistakes.

Subcase 2.2: The initial query vector $\mathbf{q}_1$ have at most k zero components. Without loss of generality, we assume that $q_i = 0$ for $1 \leq i \leq k-m$ and $q_j = 1$ for $k-m+1 \leq j \leq n$, where $0 \leq m \leq k$.

As in the Subcase 2.1, Let T be the integer such that $T-1 \leq \theta < T$ and $N = \lceil (n-k)/T \rceil$. Since $0 \leq \theta < (n-k)/10$, $N \geq 10$. Let $\mathbf{x}_1$ be the example such that $x_{1,j} = 1$ for $j = k+1, k+1, \ldots, (N-1)T$ and all its other components are zero. This example does not satisfies F_k but $m_1(\mathbf{q}_2, \mathbf{x}_2) = T(N-1) > T > \theta$. Hence, it forces the algorithm to make a mistake the query vector $\mathbf{q}_2$ becomes $q_{2,j} = 0$ for $1 \leq j \leq k-m$, $q_{2,j} = 1$ for $k-m+1 \leq j \leq k$, $q_{2,k+j} = 0$ for $1 \leq j \leq (N-1)T$, and $q_{2,j} = 1$ for $k+(N-1)T+1 \leq j \leq n$. We then choose $\mathbf{x}_2$ such that $x_{2,k+j} = 1$ for $1 \leq j \leq NT$ and all its other components are zero. This example does not satisfy F_k but $m_1(\mathbf{q}_2, \mathbf{x}_2) = T \geq \theta$. Hence, the algorithm makes a mistake at this example and the query vector $\mathbf{q}_3$, denoted by $\mathbf{A}_1$, becomes

$$A_{1,j} = 1, \text{ for } k-m+1 \leq j \leq k, \tag{24}$$

$$A_{1,j} = 0, \text{ for } 1 \leq j \leq k-m, \tag{25}$$

$$A_{1,k+j} = -1; \text{ for } 1 \le j \le (N-1)T. \tag{26}$$

$$A_{1,k+(N-1)T+j} = 0, \text{ for } 1 \le j \le T. \tag{27}$$

$$A_{1,k+NT+j} = 1, \text{ for } 1 \le j \le n-k-NT. \tag{28}$$

Comparing expressions (24) to (28) with expressions (19) to (23), the query vector $\mathbf{A}_1$ is almost same as that for subcase 2.1. The difference is that for the first k components of $\mathbf{A}_1$, the first m components have a value 1 in Subcase 2.2, while only the first component has a value 1 in Subcase 2.1. Therefore, we can follow the approach in Subcase 2.1 to construct a sequence of $2NT$ examples to force the algorithm to make $2NT$ mistakes for learning X_i, $m+1 \le i \le k$. We can also construct a sequence of $2(NT-1)$ examples to force the algorithm to make $2(NT-1)$ mistakes for learning X_j, $1 \le i \le m$. Hence, at the end the algorithm makes at least $2m(NT-1) + 2(k-m)NT + 2 \ge 2kNT - 2k \ge k(n-k)/5 - 2k$ mistakes.

Case 3. $\theta \le -(n-k)$. Let $T \ge (n-k)$ be the integer such that $-T-1 < \theta \le -T$. We consider the following two subcases of the query vector $\mathbf{q}_1$.

Subcase 3.1. $\mathbf{q}_1$ has at least k zero components. Without loss of generality, we assume that that $q_{1,i} = 0$ for $1 \le i \le k+m$ and $q_{1,j} = 1$ for $m+k+1 \le j \le n$, where $0 \le m \le n-k$.

For each $j = 1, 2, \ldots, m$, we choose a sequence of examples $\mathbf{y}_t$, $t = 1, 2 \ldots, T+2$, such that $y_{t,k+j} = 1$ and all its other components are zero. This example does not satisfy F_k and forces the algorithm to make a mistake. For each $\mathbf{y}_t$, the algorithm decreases the $(k+j)$-th component of the query vector by 1 while keeping all the other components unchanged. At the end the $(k+j)$-th component of the query vector become $-T-1 < \theta$. Similarly, we can choose a sequence of $T+3$ examples to change the $(k+m+j)$-th component of the query vector to $-T-1 < \theta$ for $1 \le j \le n-k-m$. That is, after processing $m(T+2)+(n-k-m)(T+3) = (n-k)T+3(n-k)+m$ examples, we have a query vector, denoted as $\mathbf{A}_1$ for simplicity, as follows:

$$\begin{aligned} A_{1,j} &= 0, && \text{for } 1 \le j \le k, \\ A_{1,k+j} &= -T-1, && \text{for } 1 \le j \le n-k. \end{aligned}$$

We now consider the learning of the relevant feature X_i, $1 \le i \le k$. In the step of learning X_i, for $1 \le t \le (n-k-1)(T+1)$, we construct a sequence of examples $\mathbf{y}_t, \mathbf{z}_s^t, s = 1, 2, \ldots, n-k$. Here, $y_{t,j} = 1$ for $j = i$ or $k+1 \le j \le n$, and $y_{t,j} = 0$ for $1 \le j \le k$ and $j \ne i$; $z_{s,j}^t = 0$ for $1 \le j \le k$, and $z_{s,k+j}^t = 1$ for $1 \le j \le n-k$. By simple induction, each of $\mathbf{y}_t$ and $\mathbf{z}_s^t$ forces the algorithm to make a mistake. $\mathbf{y}_t$ helps the algorithm to update the i-th component of the query vector to t and all the $(k+j)$-th components to $-T \ge \theta$, while the j-th component remains

unchanged for $1 \leq j \leq k$ and $j \neq i$. Each example $\mathbf{z}_s^t$ help the algorithm to reset the $(k+s)$-th component of the query vector back to $-T-1 \leq \theta$. At the end of this step, the i-component of the query vector becomes $(n-k-1)(T+1)$ and all the $(k+s)$-th components are $-T-1$ for $1 \leq s \leq n-k$, while the j-th components, for $1 \leq j \leq k$ and $j \neq i$, remain unchanged during this step. Thus, the total number of mistakes for learning those k relevant features is at least $k(n-k-1)(T+1)(n-k+1)+(n-k)T+3(n-k)+m = \Omega(k(n-k)^3)$, because $T \geq (n-k)$.

Subcase 3.2. $\mathbf{q}_1$ has at most k zero components. Without loss of generality, we assume that that $q_{1,i} = 0$ for $1 \leq i \leq m$ and $q_{1,j} = 1$ for $m+1 \leq j \leq n$, where $0 \leq m \leq k$.

As in Subcase 3.1, let $-T-1 < \theta \leq -T$ for the integer $T \geq n-k$, and we can use $(n-k)(T+3)$ examples to force the algorithm to make $(n-k)(T+3)$ mistakes and to change all the $(k+j)$-th components of the query vector to $-T-1$ for $1 \leq j \leq n-k$. For each relevant feature X_i, $1 \leq i \leq k$, the i-th component of the query vector has a value 0 or 1. As in the step of learning X_i in Subcase 3.1, we can use at least $(n-k-1)(T+1)-1$ sequences of examples, $\mathbf{y}_t, \mathbf{z}_s^t, s = 1, 2, \ldots, n-k$, for $1 \leq t \leq (n-k-1)(T+1)-1$ to learn X_i. The algorithm makes at least $k((n-k-1)(T+1)-1)(n-k+1)+(n-k)(T+3) = \Omega(k(n-k)^3)$, because $T \geq n-k$.

Case 4. $-(n-k) < \theta < 0$. If $\theta \leq -(n-k)/10$, then following the same approach as for Case 3 we can prove that the algorithm makes $\Omega(k(n-k)^3$ mistakes in learning all the k relevant features in F_k. In the rest of this case we assume that If $-(n-k)/10 < \theta < 0$. Let $0 \leq T < (n-k)/10$ be the integer such that $-T-1 < \theta \leq -T$. We consider the following two subcases of the query vector $\mathbf{q}_1$.

Subcase 4.1: The initial query vector $\mathbf{q}_1$ have at least k zero components. Without loss of generality, we assume that $q_i = 0$ for $1 \leq i \leq k+m$ and $q_j = 1$ for $m+k+1 \leq j \leq n$, where $0 \leq m \leq n-k$.

For each $j = 1, 2, \ldots, m$, we choose a sequence of examples $\mathbf{y}_t$, $t = 1, 2 \ldots, T+2$, such that $y_{t,k+j} = 1$ and all its other components are zero. This example does not satisfy F_k and forces the algorithm to make a mistake. It helps the algorithm to decrease the $(k+j)$-th component of the query vector by 1 while keeping all the other components unchanged. At the end the $(k+j)$-th component of the query vector become $-T-1 < \theta$ for $1 \leq j \leq m$. Similarly, we can choose a sequence of $T+3$ examples to change the $(k+m+j)$-th component of the query vector to $-T-1 < \theta$ for $1 \leq j \leq n-k-m$. That is, after processing $m(T+2)+(n-k-m)(T+3) =$

$(n-k)T + 3(n-k) - 4m$ examples, we have a query vector, denoted as $\mathbf{A}_1$ as follows:

$$\begin{aligned} A_{1,j} &= 0, && \text{for } 1 \le j \le k, \\ A_{1,k+j} &= -T-1, && \text{for } 1 \le j \le n-k. \end{aligned}$$

We now consider the learning of the relevant feature X_i, $1 \le i \le k$. In the step of learning X_i, for $1 \le t \le (n-k)T+9(n-k)/10$, we construct a sequence of examples $\mathbf{y}_t, \mathbf{z}_s^t, s = 1, 2, \ldots, n-k$. Here, $y_{t,j} = 1$ for $j = i$ or $k+1 \le j \le n$, and $y_{t,j} = 0$ for $1 \le j \le k$ and $j \ne i$; $z_{s,j}^t = 0$ for $1 \le j \le k$, and $z_{s,k+j}^t = 1$ for $1 \le j \le n-k$. By simple induction, each of $\mathbf{y}_t$ and $\mathbf{z}_s^t$ forces the algorithm to make a mistake. $\mathbf{y}_t$ helps the algorithm to update the i-th component of the query vector to t and all the $(k+j)$-th components to $-T \ge \theta$, while the j-th component remains unchanged for $1 \le j \le k$ and $j \ne i$. Each example $\mathbf{z}_s^t$ help the algorithm to reset the $(k+s)$-th component of the query vector back to $-T-1 \le \theta$ for $1 \le s \le n-k$. At the end of this step, the i-component of the query vector becomes $(n-k)T+9(n-k)/10$ and all the $(k+s)$-th components are $-T-1$ for $1 \le s \le n-k$, while the j-th components, for $1 \le j \le k$ and $j \ne i$, remain unchanged during this step. Thus, the total number of mistakes for learning those k relevant features is at least $k((n-k)T+9(n-k)/10)(n-k+1)+(n-k)T+3(n-k)-4m = \Omega(k(n-k)^3)$.

Subcase 4.2. $\mathbf{q}_1$ has at most k zero components. Without loss of generality, we assume that that $q_{1,i} = 0$ for $1 \le i \le m$ and $q_{1,j} = 1$ for $m+1 \le j \le n$, where $0 \le m \le k$.

As in Subcase 4.2, let $-T-1 < \theta \le -T$ for the integer $0 \le T < (n-k)/10$, and we can use $(n-k)(T+3)$ examples to force the algorithm to make $(n-k)(T+3)$ mistakes and to change all the $(k+j)$-th components of the query vector to $-T-1$ for $1 \le j \le n-k$. For each relevant feature X_i, $1 \le i \le k$, the i-th component of the query vector has a value 0 or 1. As in the step of learning X_i in Subcase 4.1, we can use at least $(n-k)T + 9(n-k)/10 - 1$ sequences of examples, $\mathbf{y}_t, \mathbf{z}_s^t, s = 1, 2, \ldots, n-k$, for $1 \le t \le (n-k)T + 9(n-k)/10 - 1$ to learn X_i. The algorithm makes at least $k((n-k)T+9(n-k)/10-1)(n-k+1)+(n-k)(T+3) = \Omega(k(n-k)^2)$ mistakes. □

We should point out that Theorem 4 holds for any constant query updating factor $\alpha > 0$. However, we observe that the proof for $\alpha \ne 1$ is complicated and involves tedious analysis of many cases.

5 An Upper Bound

When arbitrary query updating factors are allowed, we can follow [5] to derive an $O(n)$ upper bound for Rocchio's algorithm with similarity m_1 in searching for the collection of documents represented by a disjunction of k relevant features over the n-dimensional binary vector space. This upper bound can be further tightened to $O(n + k \log n)$ for small k with the help of Lemma 1 in [5] and the Winnow algorithm in [16]. However, such an upper bound does not hold when a fixed query updating factor is used. Using a fixed query updating factor in Rocchio's algorithm as described in [19, 2] has many merits, such as simplicity and efficiency. When allowing arbitrary updating factors in expressions (1) and (2), there might be an optimal set of factors for updating the query vector at each step, nevertheless finding them is not a trivial task. It should be pointed out that counting argument based on decision trees of all monotone disjunctions of k relevant features can only help us derive an $\Omega(k \log n)$ lower bound for Rocchio's algorithm.

Theorem 5 *When a fixed updating factor $\alpha = 1$, a zero classification threshold, and an initial query vector $\mathbf{q}_1 = \mathbf{0}$ are used, Rocchio's algorithm with similarity m_1 makes at most $O(k(n-k)^3)$ mistakes in searching for a collection of documents represented by a monotone disjunction of k relevant features over the n-dimensional binary vector space.*

Proof. As in the previous section, we consider the monotone disjunction F_k in expression (6). For the query vector $\mathbf{q}_t$ at step $t \geq 1$, each example $\mathbf{x}$ satisfying F_k, called a positive example, will increase at least one $q_{t,i}$, $1 \leq i \leq k$, by 1. This increment cannot be decreased at later steps. $\mathbf{x}$ can increase $q_{t,k+j}$ by 1, in the worst case, for all $j = 1, \ldots, n-k$. On the other hand, each example $\mathbf{y}$ that does not satisfy F_k, called a negative example, will decrease at least one $q_{t,k+j}$ by 1 but keep all q_i unchanged for $1 \leq i \leq k$. Eventually, all the increments of $q_{t,k+j}$ made by positive examples, for any $j = 1, \ldots, n-k$, shall be eliminated by negative examples.

We estimate in the worst case how small $q_{k+1} + \cdots + q_n$ can be. We have the following claim:

CLAIM 6 *For any $t \geq 1$, $q_{t,k+j} \geq -(n+k)$, $1 \leq j \leq n-k$.*

We prove this claim by induction on t. Since $\mathbf{q}_1 = \mathbf{0}$, this claim is true for $t = 1$. Assume that the claim is true for $t \geq 1$. That is, $q_{t,k+j} \geq -(n-k)$, $1 \leq j \leq n-k$. At step t, let $\mathbf{x}_t$ be the example received by the algorithm. If $\mathbf{x}_t$ is a positive example, then for $1 \leq j \leq n-k$

$$q_{t+1,k+j} = q_{t,k+j} + x_{t,k+j} \geq -(n+k) + x_{t,k+j} \geq -(n-k).$$

If $\mathbf{x}_t$ is a negative example, then $x_{t,i} = 0$ for $1 \leq i \leq k$, and $\mathbf{x}_t$ has, say, m components with a value 1 among the last $n-k$ components, where $1 \leq m \leq n-k$. Without loss of generality, we assume $x_{t,k+j} = 1$ for $1 \leq j \leq m$ and $x_{t,k+m+j} = 0$ for $1 \leq j \leq n-k-m$. Since $\mathbf{x}_t$ is a negative example, we have

$$m_1(\mathbf{q}_t, \mathbf{x}_t) = q_{t,k+1} + q_{t,k+2} + \cdots + q_{t,k+m} > 0, \tag{29}$$

and the query vector is updated as

$$\begin{aligned} q_{t+1,i} &= q_{t,i}, && \text{for } 1 \leq i \leq k, \\ q_{t+1,k+j} &= q_{t,k+j} - 1, && \text{for } 1 \leq j \leq m, \\ q_{t+1,k+m+j} &= q_{t,k+m+j}, && \text{for } 1 \leq j \leq n-k-m. \end{aligned}$$

By expression (29), we have $q_{t,k+j} > 0$ for at least one j with $1 \leq j \leq m$. Say, without loss of generality, $q_{t,k+s} > 0$, i.e., $q_{t,k+s} \geq 1$, for $1 \leq s \leq m' \leq m$, and $q_{t,k+m'+s} \leq 0$ for $1 \leq s \leq m-m'$. For $1 \leq s \leq m' \leq m$, $q_{t+1,k+s} = q_{t,k+s} - 1 \geq 0 \geq -(n-k)$. If $m' < m \leq n-k$, then for $1 \leq s \leq m-m'$, $q_{t+1,k+m'+s} = q_{t,k+m'+s} - 1 \geq -\sum_{1 \leq j \leq m'} q_{t,k+j} - \sum_{1 \leq j \leq m-m' \text{ and } j \neq s} q_{t,k+m'+j} - 1 \geq -\sum_{1 \leq j \leq m'} q_{t,k+j} - 1 \geq -m' - 1 \geq -(n-k-1) - 1 = -(n-k)$. For $1 \leq j \leq n-k-m$, $q_{t+1,k+m+j} = q_{t,k+m+j} \geq -(n-k)$. Similarly, For $1 \leq j \leq k$, $q_{t+1,j} = q_{t,j} \geq -(n-k)$. Hence, the claim is true at step $t+1$.

For any query $\mathbf{q}_t$ and for any example $\mathbf{x}_t$ satisfying the i-th relevant feature X_i, $1 \leq i \leq k$, by Claim 29 we have

$$\sum_{j=1}^{n} q_{t,j} x_{t,j} \geq q_{t,i} + \sum_{1 \leq j \leq n-k \text{ and } x_{t,j}=1} q_{t,j} \geq q_{t,i} - (n-k)^2.$$

If $q_{t,i} - (n-k)^2 > 0$, then the algorithm has learned the relevant feature X_i. Recall that each positive example $\mathbf{x}_t$ satisfying the relevant feature X_i will help the algorithm to set $q_{t+1,i} = q_{t,i} + 1$ and $q_{t,i}$ will never be decreased at any later steps. Hence, the algorithm needs at most $(n-k)^2 + 1$ positive examples satisfying X_i to increase the i-th component of the query vector to $(n-k)^2 + 1 > (n-k)^2$. Thus, the algorithm needs at most $k((n-k)^2+1)$ positive examples to learn all the k relevant features. Recall also that each positive example will add 1, in the worst case, to each of the last $n-k$ components of the query vector $q_{t,k+j}$ for $1 \leq j \leq n-k$. Each value 1 added to $q_{t,k+j}$ must be decreased, in the worse case, by one negative example. Thus, the algorithm needs at most $k((n-k)^2+1)(n-k)$ negative examples to learn those $n-k$ irrelevant features X_{k+j}, $1 \leq j \leq n-k$.

In summary, the algorithm needs at $k((n-k)^2+1)+k((n-k)^2+1)(n-k) = O(k(n-k)^3)$ examples to learn F_k. □

We should point out that Theorem 5 holds for any constant query updating $\alpha > 0$, but the proof for $\alpha \neq 1$ is tedious.

6 Conclusions

In this paper, we study the learning complexity of Rocchio's similarity algorithm with a constant query updating factor over the Boolean vector space $\{0,1\}^n$. Using a fixed query updating factor in Rocchio's algorithm has many merits, such as simplicity and efficiency. Usually, a constant query updating factor $\alpha = 1$ or 0.5, as described in [19, 2], is used in practice for Rocchio's algorithm. As the first step towards formal analysis of Rocchio's similarity-based algorithm, linear lower bounds on the learning complexity have been obtained in [9] for the algorithm in searching for a collection of documents represented by a monotone disjunction of at most k relevant features (or terms) over the n-dimensional binary vector space $\{0,1\}^n$. Those linear bounds hold when any of the four typical similarities (inner product, dice coefficient, cosine coefficient, and Jaccard coefficient) is used, no matter what value is used for the query updating factor. Several general lower bounds have also been obtained in [5] but those do not hold for monotone disjunctions of k relevant features.

We strengthen the linear lower bounds in [9] in this paper and prove that Rocchio's algorithm makes $\Omega(k(n-k))$ mistakes in searching for a collection of documents represented by a monotone disjunction of k relevant features over the n-dimensional binary vector space $\{0,1\}^n$, when the inner product similarity measure is used. This $\Omega(k(n-k))$ lower bound result is a combination of Theorems 1 to 4. A quadratic lower bound is obtained when k is proportional to n. In addition, this quadratic lower bound also holds for the dice coefficient, cosine coefficient, and Jaccard coefficient similarity measures, when the classification threshold is zero. We also have proved an $O(k(n-k)^3)$ upper bound for Rocchio's algorithm with a constant query updating factor $\alpha = 1$ and a constant classification threshold $\theta = 0$ in searching for a collection of documents represented by a monotone disjunction of k relevant features, when the inner product similarity measure is used. However, a gap between $O(k(n-k)$ and $\Omega(k(n-k)^3)$ remains open. Further study needs to carry out to close this gap.

Acknowledgment.

We thank three anonymous referees for their valuable comments and suggestions to help us revise the paper. An extended abstract of this paper was reported in [6]. Theorems 1 and 2 were proved for all the four similarities (inner product, dice coefficient, cosine coefficient, and Jaccard coefficient). Theorems 3 to 5 were proved for the inner product similarity. Research in this paper is supported in part by NSF CNS-0521585 and UTPA Faculty Start-up Research Fund.

References

[1] D. Angluin, Queries and concept learning, *Machine Learning* **2**(4): 319-432, 1987.

[2] R. Baeza-Yates and B. Ribeiro-Neto (eds.), Modern Information Retrieval, Addison-Wesley, 1999.

[3] Z. Chen, Multiplicative adaptive algorithms for user preference retrieval, Proceedings of the Seventh Annual International Computing and Combinatorics Conference, LNCS 2108, pp. 540-549, Springer-Verlag, 2001.

[4] Z, Chen, Multiplicative adaptive user preference retrieval and its applications to Web search, in: G. Zhang, A. Kandel, T. Lin and Y. Yao Y (eds.), Computational Web Intelligence: Intelligent Technology for Web Applications. World Scientific, 2004. pp. 303-328, 2004.

[5] Z. Chen and B. Fu, On the complexity of Rocchio's similarity-based relevance feedback algorithm, Journal of the American Society for Information Science and Technology, 58(10):1392-1400, 2007.

[6] Z. Chen and B. Fu, A Quadratic Lower Bound for Rocchio's Similarity-Based Relevance Feedback Algorithm, Proceedings of the Seventh Annual International Computing and Combinatorics Conference, LNCS 3595, pp. 955-964, Springer-Verlag, 2005.

[7] Z. Chen and X. Meng, Yarrow: A real-time client site meta search learner, Proceedings of the AAAI 2000 Workshop on Artificial Intelligence for Web Search, AAAI Press, pp. 12-17, 2000.

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

[9] Z. Chen and B. Zhu, Some formal analysis of Rocchio's similarity-based relevance feedback algorithm, Information Retrieval **5**: 61-86, 2002. (The preliminary version of this paper appeared in *Proceedings of the Eleventh International Symposium on Algorithms and Computation (ISAAC'00)*, LNCS 1969, pp 108-119, 2000.)

[10] Z. Chen, X. Meng, B. Zhu and R. Fowler, WebSail: From on-line learning to web search, Journal of Knowledge and Information Science, the special issue of WISE'00 **4**: 219-227, 2002.

[11] W. Frakes and R. Baeza-Yates (eds.), Information Retrieval: Data Structures and Algorithms, Prentice Hall, 1992.

[12] E. Ide, Interactive search strategies and dynamic file organization in information retrieval, in: Salton G, ed. The Smart System - Experiments in Automatic Document Processing. Prentice-Hall, Englewood Cliffs, NJ, 1971. pp. 373-393, 1971.

[13] E. Ide, New experiments in relevance feedback, in: G. Salton G (ed.) The Smart System - Experiments in Automatic Document Processing, Prentice-Hall, Englewood Cliffs, NJ, pp. 337-354, 1971.

[14] J. Rocchio, Relevance feedback in information retrieval, in: G. Salton (ed.) The Smart Retrieval System - Experiments in Automatic Document Processing, Prentice-Hall, Englewood Cliffs, NJ, pp. 313-323., 1971.

[15] D. Lewis, Learning in intelligent information retrieval, Proceedings of the Eighth International Workshop on Machine Learning, pp. 235-239, 1991.

[16] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, Machine Learning **2**:285-318, 1988.

[17] V. Raghavan and S. Wong, A critical analysis of the vector space model for information retrieval, Journal of the American Society for Information Science **37**(5):279-287, 1986.

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

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

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

[21] G. Salton, S. Wong and C. Yang, A vector space model for automatic indexing, Comm. of ACM **18**(11):613-620, 1975.

[22] C.J. Keith van Rijsbergen, *Information Retrieval*, Butterworths, 1979.

[23] S. Wong, Y. Yao and P. Bollmann, Linear structures in information retrieval, Proceedings of the 1988 ACM-SIGIR Conference on Information Retrieval, pp. 219-232, 1988.