A Multi-Ranker Model for Adaptive XML Searching

Ho Lam Lau and Wilfred Ng
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{lauhl, wilfred}@cse.ust.hk

Abstract

The evolution of computing technology suggests that it has become more feasible to offer access to Web information in a ubiquitous way, through various kinds of interaction devices such as PCs, laptops, palmtops, and so on. As XML has become a de-facto standard for exchanging Web data, an interesting and practical research problem is the development of models and techniques to satisfy various needs and preferences in searching XML data.

In this paper, we employ a list of simple XML tagged keywords as a vehicle for searching XML fragments in a collection of XML documents. In order to deal with the diversified nature of XML documents as well as user preferences, we propose a novel Multi-Ranker Model (MRM), which is able to abstract a spectrum of important XML properties and adapt the features to different XML search needs.

The MRM is composed of three ranking levels. The lowest level consists of two categories of similarity and granularity features. At the intermediate level, we define four tailored XML Rankers (XRs), which consist of different lower level features and have different strengths in searching XML fragments. The XRs are trained via a learning mechanism called the Ranking Support Vector Machine in a voting Spy Naïve Bayes Framework (RSSF). The RSSF takes as input a set of labeled fragments and feature vectors and generates as output Adaptive Rankers (ARs) in the learning process. The ARs are defined over the XRs and generated at the top level of the MRM.

We show empirically that the RSSF is able to improve the MRM significantly in the learning process that needs only a small set of training XML fragments. We demonstrate that the trained MRM is able to bring out the strengths of the XRs in order to adapt different preferences and queries.

1 Introduction

The evolution of computing technology suggests that it has become more feasible to offer access to Web information in a ubiquitous way, through various kinds of interaction devices such as PCs, laptops, palmtops, and so on. As XML is a de-facto standard for exchanging Web data, an interesting and practical research problem is the development of models and techniques to adapt XML information with respect to specific user needs.

In this paper, we exploit a list of keywords enclosed with tags, which we term a list of *key-tags* or *a key-tag search query*, for example "⟨journal⟩ VLDB ⟨/journal⟩", to search XML documents. A key-tag is a simple and flexible means to provide more accurate semantics for searching XML data than conventional IR searching; in this example, the key-tag means that VLDB is a journal in the search query. However, using an arbitrary combination of these two words as a traditional keyword search in a search engine may give rise to inaccurate interpretations when matching relevant fragments [1].

We focus on returning an effective ranked list of XML fragments as an answer to the query. Such searching concerns different dimensions of relevance as well as ranking effectiveness. Given a key-tag search query, the prime tasks are to devise an innovative way to estimate the relevance between the query and XML data and to rank the possible returned fragments in the search result. We demonstrate the viability and the benefits of using key-tags in searching XML data by proposing a Multi-Ranker Model (MRM), where the underlying idea is depicted in Figure 1.

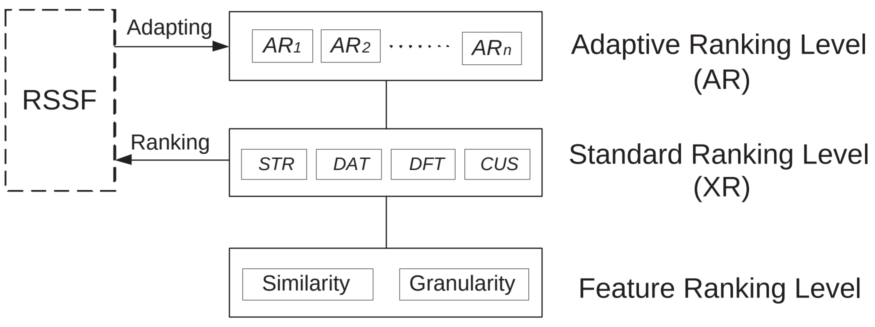


Figure 1: A three level Multi-Ranker Model (MRM) for adaptive XML searching

[1]We use the terms *fragments* (in documents) and *subtrees* (in DOM) interchangeably for XML data

As shown in Figure 1, the MRM is composed of three ranking levels. The lowest level consists of two categories of similarity and granularity features. At the intermediate level, we define four tailored XML Rankers (XRs), which consists of different low level features and have different strength in searching XML fragments. The XRs are trained via a learning mechanism called the Ranking Support Vector Machine in a voting Spy Naïve Bayes Framework (RSSF), whose conceptual diagram is shown in Figure 2. The RSSF takes as input a set of labeled fragments and feature vectors and generates as output Adaptive Rankers (ARs) in the learning process. The ARs are defined over the XRs and generated at the top level of the MRM.

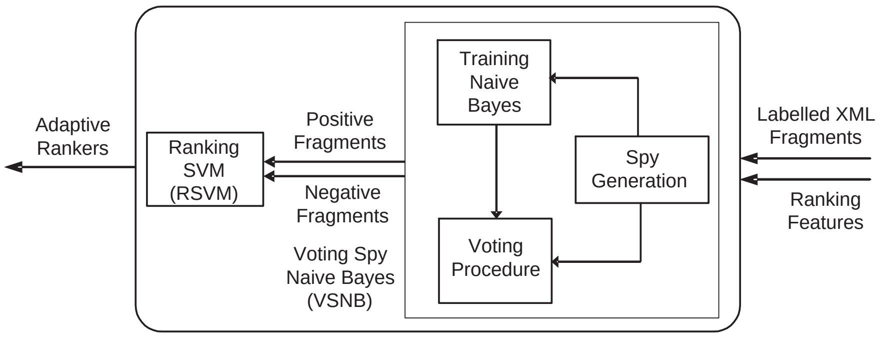


Figure 2: Ranking Support Vector Machine in a Voting SpyNB Framework (RSSF)

The RSSF is an enhancement of the Ranking Support Vector Machine (RSVM) algorithm proposed in [9, 27], which is essentially a machine learning technique that is applied here to optimize the performance of XML searching via key-tags. In order to discover preferences from analyzing the labeled fragments, we incorporate a novel *Voting Spy Naïve Bayes* (VSNB) algorithm into the RSSF, which generates the adaptive rankers at the top level. The RSSF analyzes the ranking results of the MRM by users' implicit feedback and categorizes the results into *labeled* and *unlabeled* datasets. The labeled dataset contains the result fragments that are classified as *relevant* and the unlabeled dataset contains the result fragments that have not been classified. Using VSNB we exploit the labeled (positive) fragments to discover the irrelevant (negative) fragments. The positive and negative fragments give rise to a large set of preference fragment pairs, which are employed to re-rank the search results and generate the AR rankers in the RSSF.

We empirically justify the choice of low level features with respect to important preferences and evaluate the proposed standard XML Rankers (XRs) in the MRM using a spectrum of real XML benchmark datasets. We show that each XML ranker has its individual strengths in attaining good precision and ranking quality in different XML preferences. In order to adapt the XML rankers to various kinds of XML databases and users' evaluations, we use a limited training dataset to train the MRM and generate the adaptive rankers for different users in the learning process. The RSSF is able to make the learning process efficient and the adaptive rankers effective, even when the amount of learning data is relatively small and sparse.

Our main contributions related to adaptive searching XML data are twofold.

- **A novel Multi-Ranker Model (MRM).** The MRM is able to cater to mixed searching needs from different users by providing a set of adapative rankers at the top level, when given a corpus of diversified XML datasets.
- **An effective training framework (RSSF).** The framework refines the rankers in a non-intervening manner by learning users' feedback on the returned search results. The required samples of labeled fragments are small and VSNB requires very low training cost in practice. The search performance of the trained ranker is shown to be adaptable to the users' preferences in XML searching.

Paper Organization. We discuss related work in Section 2. In Section 3, we clarify the fundamentals of key-tags. In Section 4, we explain the three-level ranking schemes of the MRM. In Section 5, we discuss VSNB training strategies for adaptive rankers and the techniques for matching XML fragments with key-tags. In Section 6, we present the experimental results that show the effectiveness of the RSSF and the MRM based on a wide spectrum of datasets. Finally, we offer concluding remarks in Section 7.

2 Related Work

There are two main areas of related work: XML searching and machine learning techniques for adaptive searching.

2.1 XML searching

XRANK [23] is an early proposed XML search engine for generating ranked results for keyword search queries of hyperlinked XML documents. This engine adopts a ranking formula based on the PageRank algorithm [42] used by an existing Web search engine.

As XRANK treats tag names and data values uniformly, it is not clear how the system is able to cater to the semantics of tagged data or fragments with different granularities in the search process. In contrast, we take into account the granularities of XML tags in ranking XML fragments. The referencing properties can also be captured as one of the feature components. We also aim to obtain quality ranking results via a novel application of the VSNB technique.

We share a similar spirit with [14, 12] in using a simple query language that is based on key-tags for XML searching. However, the datasets tested in the experimental results concerning the search quality are limited. There are only two XML datasets, SIGMOD and DBLP considered in XSEarch, which are typical data-centric documents. The performance of XSEarch when used to search more diverse XML datasets is not clear. Compared with XSEarch, we take into account a wide spectrum of XML data features and carry out experiments on an extensive set of XML documents and key-tag queries. More importantly, we develop a sophisticated ranking model that is adaptable to user preferences, whereas XSEarch only adopts a few ranking features and the interconnection relationship of XML nodes.

The recently proposed full-text search extension [6, 3, 32, 4] provides a sophisticated language syntax in XQuery expressions in order to support very fine searches. FleXPath [3] considers XPath queries of structures as a "template" and finds the best matching between the template and the full-text search. FleXPath provides ranging schemes for "top-K queries" that are interpreted in a formalized notion of relaxation semantics. The semantics essentially approximate a given query expression and retrieve the answers for a more general class of queries in the sense of query containment. However, our approach is not comparable to this work, since we do not aim to develop a soft interpretation of an existing class of precise queries such as XPath or XQuery but to develop an effective mechanism to adapt users' search preferences.

It is worth mentioning that the INitiative for the Evaluation of XML retrieval (INEX) [21, 36] has been addressing how to evaluate the effectiveness of XML search systems. Similar to TREC in IR the community, INEX establishes an infrastructure by providing a large set of test collection and scoring methods for evaluating content-oriented XML search systems. We also adopt some of their dataset [18] in Section 6.6 for comparison purposes.

2.2 Machine Learning Techniques for Adaptive Searching

There are two essential ways to collect training data, namely implicit and explicit feedback. Explicit feedback is to ask the users to provide relevance judgments on retrieved documents [16, 25, 45]. However, in many cases users tend to ignore the request when they do searching and thus explicit feedback is rather difficult to obtain in reality. Implicit feedback is to mine the user feedback from the log files without imposing burden on users (cf. the use of query log data in [22, 28, 29]); for example the clickthrough data, the time that a user has spent on reading a page, or the query keywords resubmitted by the user, all of them can be used to infer implicit feedback [15, 22, 28, 29, 30, 43, 44].

We employ various machine learning techniques to achieve adaptive XML searching. *Support Vector Machine* (SVM) [9] falls into the category of *supervised learning algorithms* in machine learning theory, which have been applied to improve web search engines. Joachims proposed a ranking SVM algorithm that uses clickthrough data to optimize the performance of a retrieval function in HTML search engines [27]. The limitation of Joachims' algorithm is that it requires a large set of training data to make the algorithm effective. In contrast, our work takes only a very small set of labeled XML fragments to train a combination of rankers and then generates adaptive rankers in a progressive manner. Naïve Bayes [13, 40] is a simple and efficient text categorization technique. However, conventional naïve Bayes requires both positive and negative examples as training data, while we only have positive examples. Thus, we employ a multi-spying technique to train naïve Bayes by incorporating unlabeled training examples [17]. In order to obtain more accurate *estimated negatives*. We further introduce a voting procedure to fully utilize the results from all spies, which generates accurate estimated negatives and thus establishes the adaptive ranking level. To our knowledge, this work is the first to apply VSNB as a learning approach in XML ranking.

Joachims and Granka analyze how users interact with the result page of a web search engine by carrying out a spectrum of user studies in [22, 28, 29]. They study the interpretation of clicks in relevance judgments using various strategies such as "(Click $>$ Skip Above)", "(Last Click $>$ Skip Above)", "(Click $>$ Earlier Click)", "(Last Click $>$ Skip Previous)" and "(Click $>$ No-Click Next)". The result indicates that the confidence level for the links which are "not-clicked" can be used as negative examples in web searching is up to 64.3% to 80.9% [28]. However, we do not make any assumption of skipped results for XML relevance judgment, since it is not clear to us whether the scan order in browsing web search results is applicable to the case of XML fragments. In our VSNB approach we infer estimated negative by utilizing all the positive samples (as spies) via a voting procedure, which does not assume any scan order for relevance judgment.

3 XML Key-Tags

In this section, we discuss how to search XML fragments via a list of tagged keywords (or key-tags). Our approach maintains the spirit of using a simple combination of key-tags, since complicated search functions in most information systems are largely ignored in reality. We now formalize the ideas related to key-tag queries.

Definition 3.1 (Key-Tag and Key-Tag Search Query) *Let* Π *be the set of tags or element names and* Σ *be the set of tagged data values (i.e., PCDATA) in an XML database. Let* $t \in \Pi \cup \{*\}$ *and* $w \in \Sigma \cup \{*\}$. *We define a* key-tag, $k = (t, w)$, *which can be viewed as a usual tagged form of an XML element "*$k = \langle t \rangle\ w\ \langle /t \rangle$*". A* key-tag search query, *denoted as*

Q (or simply a search query whenever no ambiguity arises), is a sequence of non-repeated key-tags.

The semantics of a search query is that a fragment, F, is considered as a result candidate if at least one key-tag, k, is found in the XML fragment. In this case we say that F contains k or k is contained in F. An XML fragment can be regarded as a subtree of a given XML document that is viewed as a DOM tree. If there is more than one subtree containing the same instance of k, we only choose the smallest subtree that contains k. The following definition describes the key-tag query semantics, which can be regarded as a special case of the relaxation query semantics that were recently proposed in [3].

Definition 3.2 (Search Query Semantics) *Let K be a non-empty subset of key-tags that are listed in the query, Q. We define an XML fragment, F, in a given document, to be a result candidate in the answer with respect to Q, if there exists some K such that all key-tags in K are contained in F. Let F_1 and F_2 be two fragments containing K. If F_1 is a subtree of F_2, we allow F_1 to be the only result candidate.*

For example, the search query in Figure 3 aims at finding the XML data of papers entitled "XML" written by "Mary" in year "2006". Note that the ordering of key-tags conveys a top-down view of searching. However, the query is still valid, even when the order does not match the hierarchy of the searched XML documents. We take into account the matching between the key-tag order in a given query and the usual order in an XML data tree such as the parent-child order or the sibling order, which is detailed in describing the feature ranking level of the MRM.

$$\boxed{\begin{array}{ll} & (\langle\text{author}\rangle\text{ Mary }\langle/\text{author}\rangle, \\ Q = & \langle\text{title}\rangle\text{ XML }\langle/\text{title}\rangle \\ & \langle\text{year}\rangle\text{ 2006 }\langle/\text{year}\rangle) \end{array}}$$

Figure 3: A key-tag search query

Although the language (the list of key-tags) we use has limited expressiveness, it is fundamental to an XML search engine and is independent of any declarative XML queries.

We emphasize that the aim of our study is not to integrate key-tags directly with XPath or XQuery. In fact, the key-tag search query offers much flexibility in terms of representation and users' search needs. Our approach supports a simple interface for entering key-tag words, which is not very different from the usual practice of entering simple keywords in existing search engines. For example, we are able to design a simple form-based interface as shown in Figure 4(a) for entering both tag names and textual data information. Some tags or words can be left empty (denoted as "$*$") as shown in Figure 4(b), which captures both "raw" text and required matching tags, in this case we say the tags and words are *incomplete*. We also say a key-tag is *complete* if it does not contain "$*$".

Tag	Word
author	Mary
title	XML
year	2006

(a)

Tag	Word
$*$	Mary
$*$	XML
$*$	2006

(b)

Tag	Word
author	$*$
title	$*$
year	$*$

(c)

Tag	Word
$*$	Mary
$*$	XML
year	$*$

(d)

Figure 4: Key-tag queries in a form-based interface

The semantics of "$*$" is that the data value (tag or word) exists but is unknown. The impacts of incomplete tags on the query result can be explained at two levels. First, when matching a key-tag k with the fragment F. If $k = (t, *)$, we say that k matches F if the tag of the form $\langle t \rangle . \langle /t \rangle$ is contained in F. If $k = (*, w)$, we say that k matches F if the tag of the form $\langle . \rangle w \langle /. \rangle$ is contained in F. However, we skip the checking of $K = (*, *)$, since it gives rise to the possibility of matching with the whole document space, which is impractical.

Second, when ranking the fragments resulting from matching incomplete key-tags with respect to the low level features, it may involve a comparison with the set of tags and keywords in Q and F. The effect of incomplete tags depends on the ranking features. But in general we do not take into account "$*$" in the evaluation in order to be fair to those exact matching components in ranking. For example, $Q_1 = \langle (a, 1), (b, 2) \rangle$ and $Q_2 = \langle (a, 1), (b, *) \rangle$. When we compute the keyword similarity, sim_K, the set of words in Q_1 is $\{1, 2\}$ and the set of words in Q_2 is $\{1\}$, in this case the incomplete tag has no effect on this feature. However, when we compute the access similarity, sim_A, the set of tag names in Q_1 and Q_2 are both $\{a, b\}$, in this case the incomplete key-tag $(b, *)$ contributes the same effect as $(b, 2)$.

4 Multi-Ranker Model

In this section, we introduce the Multi-Ranker Model (MRM) in order to handle diversified XML documents and user preferences in searching XML data.

4.1 Structure of the MRM

As already highlighted in Figure 1, the MRM consists of a set of XML features from which three ranking levels are defined. The lowest level of the MRM consists of two categories of similarity and granularity features, denoted as Sim and Grn, which essentially adapts the IR features of textual searching in an XML setting and will be detailed in Section 4.2.

On top of this feature ranking level we define the middle standard ranking level of four tailored XML Rankers (XRs), $STR, DAT, DFT,$ and CUS, which have different strengths in terms of the lower level features. The XRs are trained via a learning mechanism called the Ranking Support Vector Machine in a voting SpyNB Framework (RSSF), whose conceptual diagram is shown in Figure 2.

We now explain how the ranking schemes can be constructed at the adaptive ranking level (i.e. the AR level of the MRM in Figure 1). Let $E \in \{STR, DAT, DFT, CUS\}$ and an XML fragment F be ranked at position T_F from the top in E. We define the *Ranking Features*, ψ_E, with respect to F, as follows:

$$\psi_E = \begin{cases} \frac{11-T_F}{10} & \text{if } T_F < 10 \text{ in } E; \\ 0 & \text{otherwise.} \end{cases}$$

Essentially, we collect a set of high level features that indicate whether a ranking scheme, E, is capable of positioning a target fragment with high ranks that satisfy the user. There are totally four numerical ranking features. We restrict T_F to 10 or less in this feature, since we consider that the top-10 fragments are the most important returned results.

An *adaptive ranking feature vector*, denoted as Φ, contains the following eight high level features: (ϕ_{STR},ϕ_{DAT},ϕ_{DFT},ϕ_{CUS},ψ_{STR},ψ_{DAT},ψ_{DFT},ψ_{CUS}). The adaptive ranking of fragments can be calculated by using the vector equation, $\overrightarrow{W} \times \Phi$, where $\overrightarrow{W}$ is an adaptive weighting vector generated by VSNB training, which specifies the weight of different features of Φ.

We now devise four ranking schemes at the standard ranking level of the MRM, which is in order to optimize the ranking quality and to deal with diversified XML documents. The intuition is that, when a list of query results is shown to the user, without the schema knowledge, most users would judge the results either by the structure of the fragments and the textual similarity between the original query and the fragments.

- **Structure ranking (STR).** In this ranking scheme, we target the structural part of fragments and aim at supporting those users who have more interests in fragments with similar structures than data values. Since this ranking scheme gives preference to the structure of fragments, we implement the structure ranking, ϕ_{STR}, as (Sim_K, Sim_P, Sim_E, Sim_{AO}, Sim_{SO}, Grn_{Sib}, Grn_{Chi}, Grn_{Dis_+}, Grn_{Dis_-}).

- **Data ranking (DAT).** In this ranking scheme, we ignore the structure of the fragments, since the users care more about the textual similarity of the results. This ranking is similar to existing textual similarity ranking method but considers some relevant XML features such as Grn_{Tag} and Grn_{Att}. We define the data ranking, ϕ_{DAT}, as (Sim_K, Sim_A, Sim_E, Sim_C, Grn_{Tag}, Grn_{Att}).

- **System default ranking (DFT).** In this ranking scheme, we consider a union of the features sets of the STR and DAT schemes, that is, all similarity and granularity features. We define the system default ranking, ϕ_{DFT}, as (Sim_K, Sim_P, Sim_A, Sim_E, Sim_{AO}, Sim_{SO}, Sim_C, Grn_{Sib}, Grn_{Chi}, Grn_{Dis_+}, Grn_{Dis_-}, Grn_{Tag}, Grn_{Att}).

- **Customized ranking (CUS).** In this ranking scheme, the search engine administrator can define a customized ranking scheme, ϕ_{CUS}, by using a combination of low level ranking features.

4.2 Feature Ranking Level of the MRM

The low level features of the MRM are classified into the two categories of similarity (Sim) and granularity (Grn) features. These features are all essential to evaluating an XML fragment in the search result.

(I) Similarity Features:

Let $Q.\omega$ and $F.\omega$ be the sets of words (i.e., the textual values only) appearing in a query, Q, and an XML fragment, F, respectively, where $\omega \subseteq \Sigma$. Similarly, we define $Q.\tau$ and $F.\tau$ be the sets of tags in Q and F, respectively, where $\tau \subseteq \Pi$ (recall the meaning of Σ and Π in Definition 3.1).

1. Keyword similarity: $Sim_K(Q, F)$.

   Let N be the number of *non-stop words* occurring in $F.\omega$. Here, *stop words* are those words that have no meaning from the searching point of view, such as the definite and indefinite articles in English. Otherwise, words are *non-stop words*. We denote P_+ as the frequency of the words in $F.\omega$ belonging to $Q.\omega$ (positive samples) and P_- as the frequency of the terms in $F.\omega$ not belonging to $Q.\omega$ (negative samples) where $P_+ + P_- = 1$. We define the *keyword similarity*, denoted as $Sim_K(Q, F)$, between the query, Q, and the retrieved fragment, F, as follows:

$$Sim_K(Q, F) = \begin{cases} logN & \text{if } \forall w_i \in F.\omega, w_i \in Q.\omega; \\ -logN & \text{if } \forall w_i \in F.\omega, w_i \notin Q.\omega; \\ \frac{1}{2} log\frac{(1-P_-)P_+}{(1-P_+)P_-} & \text{otherwise.} \end{cases}$$

   The equation defined above is analogous to the formulae developed in text searching [8, 19]. There are three exclusive cases in the above formula. First, *all* the keywords in the fragment are found in the query. Second, *none* of the keywords in the fragment is in the query. Thus, we need to suppress this feature by a negative log formula. Finally, for the intermediate case, we compute the keyword similarity by using the log ratio measurement, which takes the frequency of both positive and negative samples into consideration. The formula is derived from Baye's theorem for probabilistic information retrieval in [20]. The log function is introduced to dampen the effect of the increase in the involved set size.

2. Access similarity: $Sim_A(Q, F)$.

   We maintain the set of n most frequently accessed key-tags, M, in the system. Let $M.\tau$ be the set of tag names in M. We define the *access similarity*, denoted as $Sim_A(Q, F)$, between Q, $M.\tau$, and F, as follows:

   $$Sim_A(Q, F) = \begin{cases} 1 & \text{if } F.\tau \text{ overlaps } (M.\tau \cap Q.\tau); \\ 0 & \text{otherwise.} \end{cases}$$

   This feature provides a Boolean variable that indicates if there exists a frequently accessed key-tag in Q that can also be found in the fragment.

3. Path similarity: $Sim_P(Q, F)$.

   Let $F.\rho$ be the set of all paths running from the root to leaf nodes in F. Let $Q.\rho = \{y \in F.\rho \mid \exists x \in Q.\tau$ such that x is a tag occurring in the path $y\}$. We define the *path similarity*, denoted as $Sim_P(Q, F)$, between the query, Q, and the fragment, F, as follows:

   $$Sim_P(Q, F) = \frac{|Q.\rho|}{|F.\rho|}.$$

   The $Sim_P(Q, F)$ is a simple ratio of the number of paths containing some key-tags of the query in the fragment to the total number of paths in the fragments. Essentially, this feature indicates the fraction of paths in the fragment that are related to the query.

4. Element similarity: $Sim_E(Q, F)$.

   We define the *element similarity*, denoted as $Sim_E(Q, F)$, between the query, Q, and the fragment, F, as the fraction of tags or words in F that overlaps those in Q:

   $$Sim_E(Q, F) = \frac{|(Q.\tau \cup Q.\omega) \cap (F.\tau \cup F.\omega)|}{|F.\tau| + |F.\omega|}.$$

   The feature treats both tags and keywords uniformly as normal words and computes a simple ratio of the number of common words in the query and fragment to the total number of words in the fragment.

5. Order similarity: $Sim_{AO}(Q, F)$ and $Sim_{SO}(Q, F)$.

   The order similarity can be further divided into the ancestor order similarity, AO, and the sibling order similarity, SO. Let B_Q be the set of all possible ordered pairs of tags extracted from Q, which matches the ordering of the key-tags given in Q (recall that Q is a sequence of key-tags by Definition 3.1), and let $F.AO$ and $F.SO$ be the sets of ancestor and the sibling ordered pairs of tags extracted from F. The ordered pairs in $F.AO$ and $F.SO$ match either the ancestor order or the sibling order of the searched document. We define the *ancestor order similarity* and *sibling order similarity*, denoted as $Sim_{AO}(Q, F)$ and $Sim_{SO}(Q, F)$, between the query, Q, and the fragment, F, as the fraction of $F.AO$ and $F.SO$ that overlaps those in B_Q:

$$Sim_{AO}(Q, F) = \frac{|B_Q \cap F.AO|}{|B_Q|},\ Sim_{SO}(Q, F) = \frac{|B_Q \cap F.SO|}{|B_Q|}.$$

   Both $Sim_{AO}(Q, F)$ and $Sim_{SO}(Q, F)$ features represent the fractions of ordered pairs of tags in the query that match up with their counterparts in their fragments according to *AO* and *SO* of the fragment tree. The extreme case happens when the fragment contains the query tags that are in the same order as the query. $Sim_{AO}(Q, F)$ and $Sim_{SO}(Q, F)$ then become one. On the other hand, if the fragment contains no tags of the query or the query tag order totally mismatches with that of the fragment, then $Sim_{AO}(Q, F)$ and $Sim_{SO}(Q, F)$ become zero.

6. Category similarity: $Sim_C(Q, F)$.

   We define n categories, $\{c_1, c_2, \ldots, c_n\}$, the set, c_i, is selected from the most common tags among the datasets. For example, a possible category of academic, denoted by $c_{academic}$, contains a set of tags such as { title, author, year, keyword, ... }. Recall that $Q.\tau$ and $F.\tau$ be the sets of tags in Q and F, respectively. We define a *query category vector*, v_Q, as $< \frac{|Q.\tau \cap c_1|}{|Q.\tau|}, \frac{|Q.\tau \cap c_2|}{|Q.\tau|}, \ldots, \frac{|Q.\tau \cap c_n|}{|Q.\tau|} >$ and a *fragment category vector*, v_F, as $< \frac{|F.\tau \cap c_1|}{|F.\tau|}, \frac{|F.\tau \cap c_2|}{|F.\tau|}, \ldots, \frac{|F.\tau \cap c_n|}{|F.\tau|} >$ according to C. We then define $Sim_C(Q, F)$ as the usual vector distance $|\ v_Q - v_F\ |$ between v_Q and v_F.

(II) Granularity Features:

Let $F.r$ be the root of F. We measure the *granularity* of a retrieved fragment, F, by the following *granularity features*:

- Sib : The order of occurrence of fragments whose roots are siblings of $F.r$.
- Chi : The order of occurrence of tags whose parent is $F.r$.
- Dis_+ : The distance from $F.r$ to the farthest leaf node.
- Dis_- : The distance from $F.r$ to the nearest leaf node.
- Tag : The order of occurrence of tags in $F.r$.
- Att : The order of occurrence of attributes of $F.r$.

The granularity measure of a feature, X, for a given fragment, F, denoted as $Grn_X(F)$, is defined as follows:

$$Grn_X(F) = \frac{X(F) - avg(X)}{avg(X)},$$

where X is one of the above granularity features and $avg(X)$ is the average value of the feature, X, in the XML document where F is embedded. The granularity feature is a simple ratio of various parameters to their average value of the fragments in the search result.

The following example helps to illustrate the low level features described above.

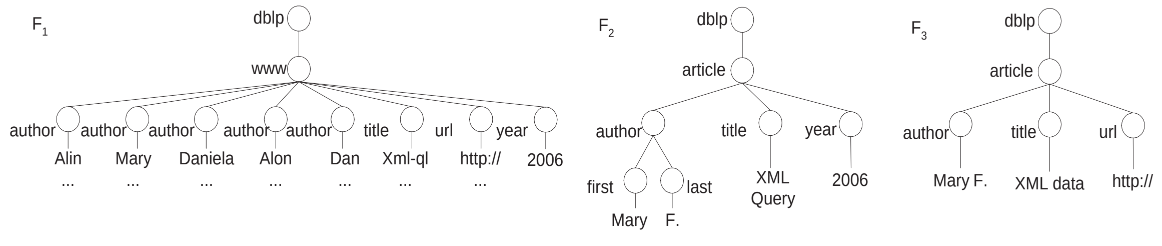


Figure 5: Three fragments, F_1, F_2 and F_3, returned by the query, Q'

Example 1 Assume F_1 be the DBLP fragment, which is shown in Figure 5. Let $Q = (\langle author \rangle Mary \langle /author \rangle, \langle title \rangle XML \langle /title \rangle)$. The keyword similarity, $Sim_K(Q, F_1) =$

$\frac{1}{2}log\frac{(1-0.25)0.75}{(1-0.75)0.25} = 0.4771$. Assume the tags $\langle title\rangle$ and $\langle author\rangle$ are the most frequently accessed tags, the access similarity, $Sim_A(Q, F_1) = 1$. There are 8 paths in F_1, where 6 of them contain tags from Q. The path similarity, $Sim_P(Q, F_1) = \frac{6}{8} = 0.75$. The element similarity, $Sim_E(Q, F_1) = \frac{10}{26} = 0.3846$. $Q.\tau = \{author, title\}$ and $B_Q = \{\langle author, title\rangle\}$. $F_1.\tau = \{dblp, www, author, title, url\}$, we have $F_1.AO = \{(dblp, www), (dblp, author), (dblp, title), (dblp, url), (www, author), (www, title), (www, url)\}$ and $F.SO = \{(author, title), (author, url), (title, url)\}$. The ancestor and sibling order similarities are $Sim_{AO}(Q, F_1) = \frac{0}{1} = 0$ and $Sim_{SO}(Q, F_1) = \frac{1}{1} = 1$.

We use F_1 again for illustrating the granularity features. Compared with other fragments, F_1 does not have any sibling while F_2 and F_3 are siblings to each other. So $Sib(F_1) = 0$, $Sib(F_2)$ and $Sib(F_3)$ are both 1. Thus, $Grn_{Sib}(F_1) = \frac{Sib(F_1)-(Sib(F_1)+Sib(F_2)+Sib(F_3)/3}{(Sib(F_1)+Sib(F_2)+Sib(F_3))/3} = \frac{0-(0+1+1)/3}{(0+1+1)/3} = -1$, which means that the number of siblings of F_1 is below average (i.e. positive Grn_{Sib} means relatively more siblings and negative means otherwise). As F_1 has 8 children, F_2 and F_3 have 3 children. We have $Grn_{Chi}(F_1) = \frac{8-(8+3+3)/3}{(8+3+3)/3} = 0.7143$. The distance from $F_1.r$ to the farthest leaf nodes is 2. Both $Dis_+(F_1)$ and $Dis_-(F_1)$ are 2. Similarly, $Dis_+(F_2)$ and $Dis_-(F_2)$ are 3 and 2, and $Dis_+(F_3)$ and $Dis_-(F_3)$ are both equal to 2. Thus, $Grn_{Dis_+}(F_1) = \frac{2-(2+3+2)/3}{(2+3+2)/3} = -0.1428$ and $Grn_{Dis_-}(F_1) = \frac{2-(2+2+2)/3}{(2+2+2)/3} = 0$. A negative $Grn_{Dis_+}(F_1)$ means that the farthest distance from leaf nodes to $F_1.r$ is below the average distance from the root to the leaf nodes, and zero represents that it is the average. The total number of tag occurrences in F_1, F_2, and F_3 are 10, 7, and 5. Thus, $Grn_{Tag}(F_1) = \frac{10-(10+7+5)/3}{(10+7+5)/3} = 0.3636$. The calculation for $Grn_{Att}(F_1)$ is similar to $Grn_{Tag}(F_1)$, whose value is 0, since all the fragments do not have attributes in this example. □

4.3 Changing Features and Preferences

The updating of the feature set such as adding and removing features is straightforward in our framework, since the low level features are encapsulated within various tailored standard rankers at the middle level of the MRM. We can update the features at the feature ranking level and then adjust the involved ranking scheme of the standard ranking level if necessary. It follows that the AR rankers redistributes the weight towards various standard rankings via learning at the adaptive ranking level. This is in fact one of the advantages of our system: the approach has the benefit of “ranking level independence”, which is analogous to the idea of data independence in the DBMS architecture.

We now discuss how our ranking schemes are updated when user preferences change, leading to a new adaptive weight vector trained by the VSNB module. Our framework supports continuous ranking updates with respect to the changes.

We take into account of past preferences and introduce an aging factor u to the previous weight vectors to handle the change in user preference. Informally, the weight of older preferences gradually reduces as the number of iterations in VSNB training. Our approach shares the same spirit of the the adaptive playout delay formula used in computer networking [31]. To simplify our explanation, we assume VSNB trains the AR for every submitted query.

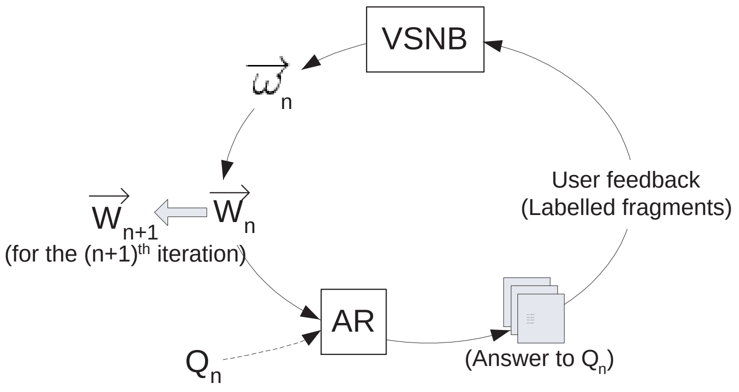


Figure 6: The underlying idea of updating the adaptive weight vector, W_n, at the n^{th} iteration

Figure 6 shows the conceptual flow of how the adaptive weight vector is updated. Suppose at the n^{th} iteration a user submits a query Q_n, the system ranks fragments according to the current adaptive weight vector, $\overrightarrow{W}_n$, which is obtained from the querying history of $(n-1)$ iterations, and finally returns a list of XML fragments as an answer. We simply use DFT to rank the answer to Q_1 at the very beginning. The user then provides his/her feedback (labelled fragments) to the system and thus VSNB generates a new *trained weight vector*, $\overrightarrow{\omega}_n$. After collecting the user feedback, the system updates the adaptive weight vector, $\overrightarrow{W}_{n+1}$, which is used by the adaptive ranker for the next submitted query Q_{n+1}. Formally,

$\overrightarrow{W}_{n+1}$ depends on the adaptive weight vectors of the past n queries and the current trained weight vector, $\overrightarrow{\omega}_n$, which is given by the following recursive formula:

$$\overrightarrow{W}_{n+1} = \begin{cases} \overrightarrow{\omega}_1, & \text{if } n = 1 \\ u\overrightarrow{\omega}_{n+1} + (1-u)\overrightarrow{W}_n, & otherwise. \end{cases}$$

In the above formula, we can see that the adaptive weighting vector of the current iteration is diminished by an *aging factor*, $0 < u < 1$, and that the weight vector representing past preferences is recursively reduced by the factor $(1-u)$. When we set the aging factor, u, to larger than 0.5, the weight given to the preference in the current training weight vector is more dominant than the past preferences. In general, for $n \geq 1$, the formula can be expanded as follows:

$$\overrightarrow{W}_{n+1} = u\sum_{j=1}^{n}(1-u)^{j-1}(\overrightarrow{\omega}_{n-j+2}) + (1-u)^n\overrightarrow{\omega}_1,$$

where $\overrightarrow{\omega}_j$ is the trained weight vector at the j^{th} iteration. We can see that $\overrightarrow{W}$ can be computed by using the trained weight vectors, which gives us flexibility to keep the feedback history for several iterations rather than computing $\overrightarrow{W}$ at each iteration. Note that when n tends to infinity (which can be regarded as a very large number of updates in practice), we have the formula given by

$$\overrightarrow{W}_\infty = \frac{u}{1-u}\sum_{j=0}^{\infty}(1-u)^j(\overrightarrow{\omega}_{n-j+2}).$$

We can see that the weight given to the past adaptive weight vectors decays exponentially and u, controls the rate of decay of the past vectors. Informally, the larger the u, the stronger the effect of the latest preference and the slower the decay. There is some consideration of choosing the u value. If u is too small (close to 0) then the system is too "rigid" to update the adaptive weight vector, since the past adaptive weight is still much retained in the formula. On the other hand, the system is "oversensitive" to the immediate and short-term preference change, if u is too high (close to 1). The performance of the system in fact lags behind the update a number of iterations of VSNB training, which is empirically studied in Section 6.4.

So far, we assume that an adaptive weight vector changes whenever the preference changes and is detected by the VSNB module. However, we do not intend to present a detailed evaluation of how often the VSNB should be invoked, since the optimization issues involved need the full space of another paper. The complication is that, if we update the AR for every submitted query for each user, it may impose much burden to the systems for handling heavy loads of queries and high number of users. A more feasible approach is to consider a trade-off between the system resources and the rate of updating the ARs. Obviously, in order to cater for those users who frequently change their preferences we need to invoke the training and update the rankers more often. Along this line of thought, we may impose a different periodic updating on ARs, in terms of time or the number of submitted queries for different users, which is more efficient from the point of view of resource utilization. A more sophisticated method is to monitor the precision of the user's implicit feedback on the returned answers; when the precision is lower than some threshold for a number of queries, we then invoke the VSNB module to train and update the adaptive rankers until the precision is back to the pre-defined level.

5 The RSSF Framework

In this section, we explain how RSSF infers the users' preference fragments for a small set of labeled data in order to generate the AR rankers as discussed in Section 4. We also present some technical details of the underlying indexing scheme and relevance score method, which are related to retrieving relevant XML fragments in our framework.

5.1 Searching in RSSF

Figure 7 shows the basic ideas of how we search XML documents by using key-tags within the RSSF framwork. When the user submits a query, the search engine will search the XML databases by matching the submitted key-tags. The query results (i.e. a set of XML fragments) are obtained based on the relevance scores which will be detailed in Section 5.6. The obtained results (i.e. a set of labelled XML fragments) are then passed to the MRM module for ranking and sent to the user. The user's feedbacks are then sent to the Voting SpyNB Training Module for adapting the standard rankings. For each fragment in the returned result, we measure the relevance scores and pass them to the MRM. The MRM collects user preferences and uses the RSSF algorithm to optimize the adaptive ranking functions in the top level by assigning different weights to the features of XRs in the middle level. The result of adaptive ranking constitute an adaptive ranking towards the preferences tailored to the user.

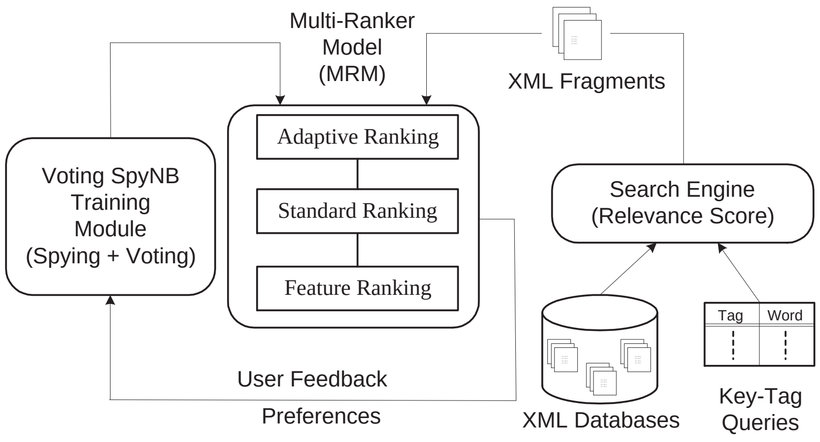


Figure 7: An overview of searching in RSSF

The search engine, MRM rankers and voting spyNB training module are implemented by Java language, and their connection with Oracle is via JDBC. We do not describe the implementation details of the XML databases and the associated searching process but remark that the query tools provided by the database vendor, such as the XML SQL Utility in Oracle [57], serve to build our prototype. We assume that low-level search operations are efficient and thus the searching mechanism is not in the scope of our work.

5.2 Preference Fragments

Given a query, Q, the returned list of ranked result is classified into two categories of labeled and unlabeled fragments. We use the set of labeled fragments as the training data. Formally, a labeled fragment is denoted as a triplet, (Q, R, P), where Q is the input search query, R is a list of ranked fragments, $(F_1, \ldots, F_n)$, and P is the set of labeled fragments that are considered to be relevant to the query.

Assume that a user "labels" three fragments, F_1, F_7, and F_{10}, which means that the fragments are considered to be relevant with respect to the user preference and thus the labeled fragments can serve as positive examples in RSSF. Let P denote the positive set, and U denote the unlabeled set. We proceed to analyze the elements of the positive fragments in VSNB. The objective is to identify which fragments in U are not similar to the positive fragments and thus we regard them as the *estimated negative* examples. Let $(EN \subset U)$ denote the estimated negative set. We denote the ranking from the sample data as r' and call the ordered pair deduced from the VSNB judgment *the preference fragment pair*, which are given as follows:

$$F_j <_{r'} F_i, \quad \forall F_i \in P,\ F_j \in EN. \tag{1}$$

For example, if $EN = \{F_2, F_4, F_6\}$ is obtained from VSNB. F_1 is more relevant than all the fragments in EN, according to VSNB's judgment. Thus, F_1 should rank ahead of these three links in the target ranking. Similarly, F_7 and F_{10} should also rank ahead of them. It is straightforward to check that the three sets of preference fragment pairs according to the three fragments, F_1, F_7, and F_{10}, can be obtained as shown in Figure 8. These three sets represent the relevance judgments collectively, where F_1, F_7 and F_{10} are incomparable with respect to $<_{r'}$.

Set of preference fragment pairs arising from F_1	Set of preference fragment pairs arising from F_7	Set of preference fragment pairs arising from F_{10}
$F_1 <_{r'} F_2$	$F_7 <_{r'} F_2$	$F_{10} <_{r'} F_2$
$F_1 <_{r'} F_4$	$F_7 <_{r'} F_4$	$F_{10} <_{r'} F_4$
$F_1 <_{r'} F_6$	$F_7 <_{r'} F_6$	$F_{10} <_{r'} F_6$

Figure 8: Sets of preference fragment pairs derived from the labeled data and learned EN

5.3 Ranking SVM Techniques

We now discuss how to apply *Ranking SVM* (RSVM) to train an AR ranker. The RSVM first takes the set of labeled fragment pairs as input and then returns a trained ranker. The RSVM algorithm needs to tolerate some ranking errors in the training process.

Suppose r^* is the target ranking in the search result of Q. Although r^* is optimal with respect to the documents, it is *not* fully observable in practice. However, we are able to obtain r' from the labeled data, which is, in fact, a subset of r^*. Given the training set, $\{(Q_1, r'_1), (Q_2, r'_2), \ldots, (Q_n, r'_n)\}$, we aim to find a rank that holds as many preference feedback fragment pairs in r' as possible.

The principle of achieving an optimal ranking with respect to a given training set is as follows. First, by extracting a feature vector, we can rank the documents in the search result by giving different weights to the features. Then, we find a weight vector, $\overrightarrow{\omega}$, that makes the set of inequalities given in (2) hold for $1 \leq k \leq n$:

$$\forall (F_i, F_j) \in r'_k : \overrightarrow{\omega}\phi(Q_k, F_i) \;\; > \;\; \overrightarrow{\omega}\phi(Q_k, F_j). \tag{2}$$

Here, $(F_i, F_j) \in r'_k$ is a fragment pair that corresponds to the preference pair, $(F_i <_{r'_k} F_j)$, with respect to the submitted query, Q_k; $\phi(Q_k, F_i)$ is a mapping that maps Q_k onto a sequence of features (or a *feature vector*) that describes the match between Q_k and F_i. Figure 9 illustrates how the weight vector, $\overrightarrow{\omega}$, determines the ordering of the three fragments, F_1, F_2, and F_3, in two dimensions. The documents are ordered as (F_1, F_2, F_3) according to $\overrightarrow{\omega_1}$ and as (F_2, F_1, F_3) according to $\overrightarrow{\omega_2}$. The former is better than the latter if the target ranking is $F_1 <_{r^*} F_2 <_{r^*} F_3$.

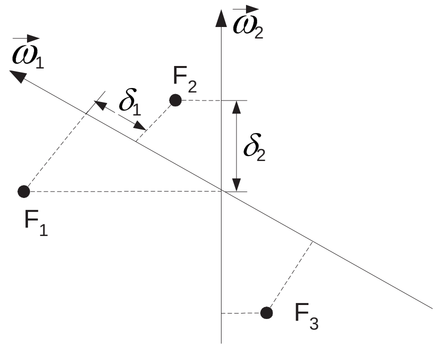


Figure 9: Ranking F_1, F_2, and F_3 according to the weight vectors, $\overrightarrow{\omega_1}$ and $\overrightarrow{\omega_2}$

The problem of solving $\overrightarrow{\omega}$ using the set of inequalities given in (2) is *NP-hard*. However, an approximated solution can be obtained by introducing a non-negative *slack variable*, ξ_{ijk}, to tolerate some ranking errors [9]. Recall that r'_k is a subset of the target ranking, r^*_k, for the search result of the query, q_k. Algorithm 1 outlines the RSVM algorithm based on the approximation. The basic idea is that if we consider that δ is the distance between the two closest projected fragments, then the larger the value of δ, the more definite the ranking, and hence the better the quality of the training result (see Figure 9). Parameter C is introduced here to allow for a trade-off between the margin size and the training errors in Algorithm 1.

Algorithm 1 Ranking SVM (RSVM) Algorithm

```
Input: A ranked list $r'_k$ $(1 \leq k \leq n)$ extracted from the set of labeled fragment pairs;
Procedure:
Minimize: $V(\overrightarrow{\omega}, \xi) = \frac{1}{2}\overrightarrow{\omega} \cdot \overrightarrow{\omega} + C\Sigma\xi_{ijk}$;
Subject to: for all $i, j$, and $k$,
    $\forall(F_i, F_j) \in r'_k : \overrightarrow{\omega}\phi(Q_k, F_i) > \overrightarrow{\omega}\phi(Q_k, F_j) + 1 - \xi_{ijk}$;
    $\xi_{ijk} \geq 0$;
Output: $\overrightarrow{\omega}$.
```

5.4 Voting Spy Naïve Bayes

Voting Spy Naïve Bayes (VSNB) consists of two main components as follows: a spying technique to obtain more accurate negative samples and a voting procedure to consider the information (posterior probability) discovered by the spies. Given a set of preference fragments, we need to categorize *unlabeled* fragments in order to discover the EN fragments. The challenge is that we need to make full use of all potential spies, since conventional naïve Bayes requires both positive and negative examples as training data, while we only have positive examples as indicated by the preference feedback as discussed in Section 5.2.

We first adapt conventional naïve Bayes in our context. Let "+" and "–" denote the positive and negative classes, respectively. Let $L = \{F_1, F_2, \ldots, F_N\}$ denote a set of fragments returned for a search query and $W = \{w_1, w_2, \ldots, w_M\}$ is a set of elements extracted from the labeled fragments in P, i.e. $W = \bigcup_{F_i \in P}(F_i.\tau \cup F_i.\omega)$. We proceed to count the occurrence of $w_i \in W$ appearing in $F_i \in L$ and train the naïve Bayes classifier by estimating the prior probabilities ($Pr(+)$ and $Pr(-)$), and likelihood ($Pr(w_j|+)$ and $Pr(w_j|-)$) as shown in Algorithm 2, where $Pr(w_j|+) + Pr(w_j|-) = 1$. In Algorithm 2, $\delta(+|F_i)$ indicates the class label of the fragment F_i. Its value is 1, if F_i is positive and 0 otherwise. $Num(w_j, F_i)$ is a function counting the number of times w_j appears in F_i. λ is the Laplace smoothing

Algorithm 2 Training Naïve Bayes Algorithm

```
Input:
    $L = \{F_1, \ldots, F_N\}$ and $W = \{w_1, \ldots, w_M\}$
Output:
    Prior probabilities: $Pr(+)$ and $Pr(-)$;
    Likelihoods: $Pr(w_j|+)$ and $Pr(w_j|-)$ $\forall w_j \in W$
Procedure:
1: $Pr(\pm) = \frac{\sum_{i=1}^{N} \delta(\pm|F_i)}{N}$;
2: for each $w_j \in W$ do
3:     $Pr(w_j|\pm) = \frac{\lambda + \sum_{i=1}^{N} Num(w_j, F_i)\delta(\pm|F_i)}{\lambda M + \sum_{k=1}^{M} \sum_{i=1}^{N} Num(w_k, F_i)\delta(\pm|F_i)}$;
4: end for
```

factor [37], which we set to $\lambda = 1$ to strengthen the naïve Bayes robustness.

When predicting unlabeled fragments, naïve Bayes calculates the *posterior probability* of a fragment F using the Bayes rule:

$$Pr(+|F) = \frac{Pr(F|+)Pr(+)}{Pr(F)},$$

where $Pr(F|+) = \prod_{j=1}^{M} Pr(w_j|+)$ is the product of the likelihoods of the elements in W. Then, F is predicted to belong to class "+", if $P(+|F)$ is larger than $P(-|F)$ and "–" otherwise.

A problem of using Algorithm 2 is that the training data in our context contains only positive and unlabeled examples. We therefore introduce a novel spying technique to train a naïve Bayes classifier. This idea is illustrated in Figure 10. First, a positive example p_i is randomly selected from P and put in U to act as a "spy". Then, the unlabeled examples in U together with p_i are regarded as negative examples with which to train a naïve Bayes. The trained classifier is then used to assign *posterior probability* $Pr(+|p_i)$ to each example in ($U \cup \{p_i\}$). After that, a threshold α_i is determined based on the *posterior probabilities* assigned to p_i. An unlabeled example in U is selected as an EN example if its probability is less than α_i. The fragment p_i acts as a spy, since it is positive but is put into U "pretending" to be a negative example.

We only need very few positive examples to be spies, since we incorporate a voting procedure in our classification process to reduce the bias towards a particular spy. First,

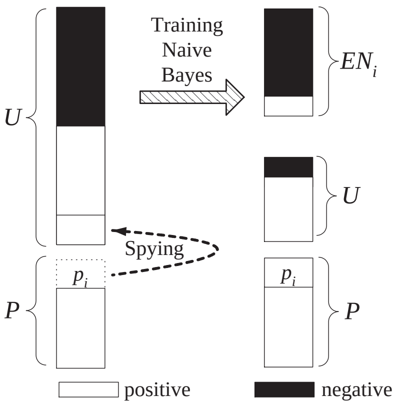


Figure 10: The underlying principle of the spying technique

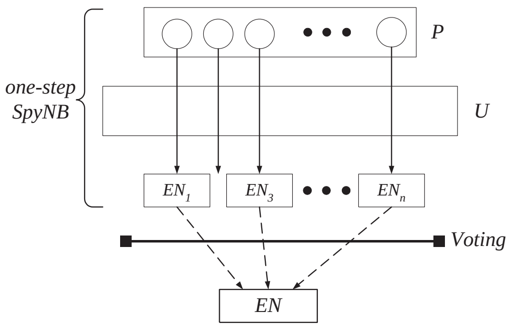


Figure 11: The voting procedure

the algorithm runs the spying technique n times, where $n = |P|$ is the number of positive examples. The probability $Pr(+|p_i)$ assigned to the spy p_i can be used as a threshold α_i to select a corresponding EN_i. That is, any unlabeled example $u_j \in U_i$ with a smaller probability of being a positive example than the spy is selected into EN_i (i.e. $Pr(+|u_j) < \alpha_i$). As a result, n candidate EN sets: $EN_1, EN_2, \ldots, EN_n$ are identified. Finally, a voting procedure is taken to combine all EN_i into the final EN. An unlabeled example is included in the final EN, if and only if, it obtains at least ($\beta \times |P|$) votes from all EN_i, where β is called the *voting threshold*.

We now present the VSNB algorithm in Algorithm 3, in which Steps 2 to 15 employ the spying technique $|P|$ times to generate $|P|$ candidate sets of EN_i. Steps 16 to 21 combine all EN_i into the final EN based on the voting result. To analyze the time complexity of VSNB, we let $|P|$ denote the number of labeled fragments (positive examples), $|U|$ denote the number of unlabeled fragments (unlabeled examples) and N denote the number of all

Algorithm 3 Voting SpyNB (VSNB) Algorithm

```
Input:
    $P$ – a set of positive examples; $U$ – a set of unlabeled examples; $\beta$ – a voting threshold;
Output:
    $EN$ – the set of estimated negative examples
Procedure:
 1: $EN_1 = EN_2 = \cdots = EN_{|P|} = \{\}$ and $EN = \{\}$;
 2: for each example $p_i \in P$ do
 3:     $P_i = P - \{p_i\}$;
 4:     $U_i = U \cup \{p_i\}$;
 5:     Assign each example in $P_i$ the class label 1;
 6:     Assign each example in $U_i$ the class label -1;
 7:     Train a naïve Bayes on $P_i$ and $U_i$ using Algorithm 2;
 8:     Predict each example in $U_i$ using trained NB;
 9:     Spy threshold $\alpha_i = Pr(+|p_i)$;
10:     for each $u_j \in U$ do
11:         if $Pr(+|u_j) < \alpha_i$ then
12:             $EN_i = EN_i \cup \{u_j\}$;
13:         end if
14:     end for
15: end for
16: for each $u_j \in U$ do
17:     $Votes =$ the number of $EN_i$ such that $u_j \in EN_i$
18:     if $Votes > \beta \times |P|$ then
19:         $EN = EN \cup \{u_j\}$;
20:     end if
21: end for
```

fragments. Training naïve Bayes in Algorithm 2 requires only one scan of all fragments. Thus, the time complexity of training is $O(N)$. The prediction of naïve Bayes costs $O(|U|)$ time, where $|U| < N$. Thus, Steps 2 to 15 of VSNB cost $O(|P| \times (N + |U|)) = O(|P| \times N)$ time. With a similar analysis, the time complexity of Steps 16 to 21 of VSNB is $O(|P| \times |U|)$, which is smaller than $O(|P| \times N)$. Overall, the time complexity of VSNB is $O(|P| \times N)$.

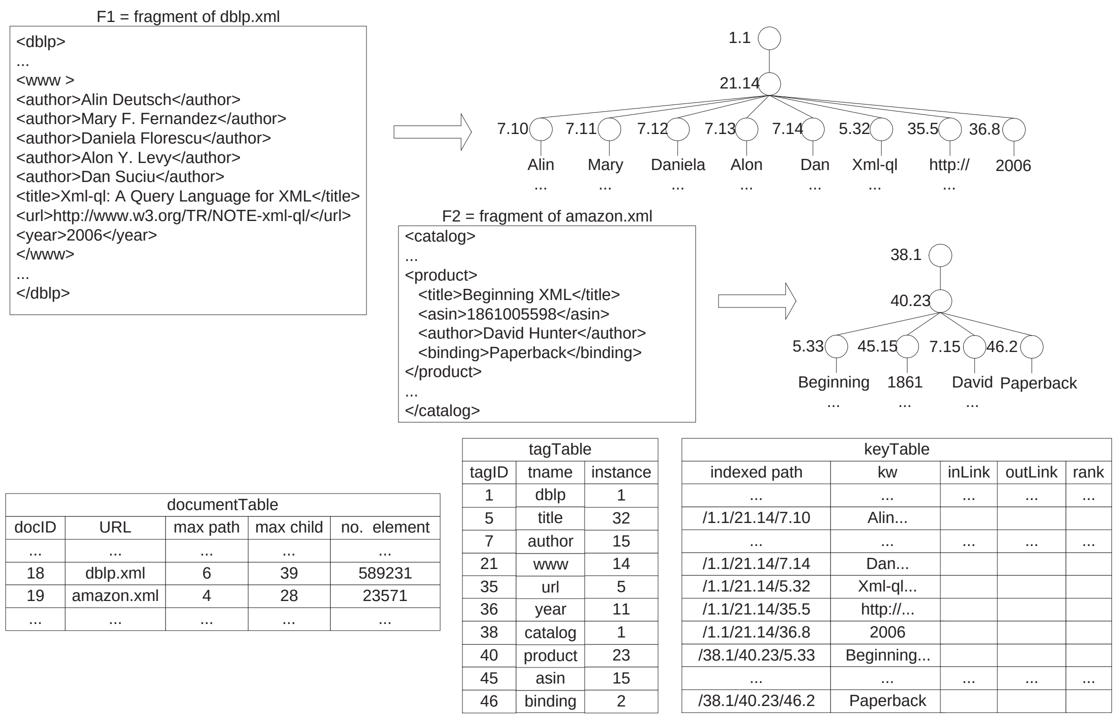
5.5 Indexed Contextual Paths and Fragments

Intuitively, a contextual path is a sequence of tags that represents a navigation through the tree structure of the fragment starting from the root r. A fragment, F, in the corpus can be regarded as a subtree of an XML document labeled by a contextual path in the corpus. A contextual path expression of length n is expressed as "$r/t_1/t_2/\cdots/t_n$". This path expression specifies finding a tag, t_1, anywhere in the document, and finding a tag t_2 nested in it, and so on until we find a tag t_n. We adopt a simple indexing scheme that numerically encodes the tag names in a depth-first search order of the corresponding fragment tree.

Definition 5.1 (Indexed Contextual Path and XML Fragment) *Let t be the tag name in an XML document tree, D, rooted at r. Let a_t and n_t be the corresponding numeric encoding of t and the order of occurrence of a tag in D. An indexed tag is denoted as $a_t.n_t$. Let "$p = r/t_1/t_2/\cdots/t_k$" be a contextual path consisting of k tags (r may be ignored if it is understood in the context). We define the indexed contextual path by $\rho =$ "$/a_{t_1}.n_{t_1}/a_{t_2}.n_{t_2}/\cdots/a_{n_k}.n_{t_k}$" to encode an occurrence of p in D. An XML fragment, F, specified by ρ, is the subtree of D rooted at the corresponding t_k node. We may also say that F is labeled by p, since a given indexed contextual path, ρ, corresponds to the occurrence of only one path, p.*

Following from Definition 5.1, an indexed contextual path can be as specific as a leaf element, t_l, using "$\rho = /a_{t_1}.n_{t_1}/a_{t_2}.n_{t_2}/\cdots/a_{n_k}.n_{t_l}$". In practice, a_t is a system-assigned identity of t and n_t is the same as the depth-first search order of the corresponding target node in the document tree. For example, in Figure 12 the tags "dblp", "www" and "author" are assigned with the encodings 1, 21, and 7, respectively. The indexed tag path, $\rho =$ "/1.1/21.14/7.14", encodes the path, $p =$ "/dblp/www/author", where the order of occurrence of the tags "dblp", "www", "author" in the document fragment indexed by ρ are 1, 14, and 14, respectively.

The index information is stored in the three relational tables. The underlying idea of using the index is that, when an XML document is loaded and parsed, a row is inserted into the `documentTable` to store the information. The SAX parser then extracts the tags and


F1 = fragment of dblp.xml

```
<dblp>
...
<www >
<author>Alin Deutsch</author>
<author>Mary F. Fernandez</author>
<author>Daniela Florescu</author>
<author>Alon Y. Levy</author>
<author>Dan Suciu</author>
<title>Xml-ql: A Query Language for XML</title>
<url>http://www.w3.org/TR/NOTE-xml-ql/</url>
<year>2006</year>
</www>
...
</dblp>
```

F2 = fragment of amazon.xml

```
<catalog>
...
<product>
  <title>Beginning XML</title>
  <asin>1861005598</asin>
  <author>David Hunter</author>
  <binding>Paperback</binding>
</product>
...
</catalog>
```

documentTable

docID	URL	max path	max child	no. element
...	...	...	...	...
18	dblp.xml	6	39	589231
19	amazon.xml	4	28	23571
...	...	...	...	...

tagTable

tagID	tname	instance
1	dblp	1
5	title	32
7	author	15
21	www	14
35	url	5
36	year	11
38	catalog	1
40	product	23
45	asin	15
46	binding	2

keyTable

indexed path	kw	inLink	outLink	rank
...	...	...	...	...
/1.1/21.14/7.10	Alin...			
...	...	...	...	...
/1.1/21.14/7.14	Dan...			
/1.1/21.14/5.32	Xml-ql...			
/1.1/21.14/35.5	http://...			
/1.1/21.14/36.8	2006			
/38.1/40.23/5.33	Beginning...			
...	...	...	...	...
/38.1/40.23/46.2	Paperback			

Figure 12: The index tables corresponding to two parsed XML fragments

converts the path into a corresponding indexed tag path based on the `tagTable`. When a tag, t, is encountered in the parser, we perform a search in the `tagTable`. If it is found with numeric encoding, a_t, and current_instance, n_t, we assign an indexed tag code, $a_t.n_t$, to the tag. If it is not found, we insert it into the `tagTable` and assign an indexed tag code, $a_t.0$, to the tag. Whenever the parser meets a text value, we store its indexed tag path, the text and link information in the `keyTable`. During the parsing, information, such as the number of elements or the maximum path depth, is collected and stored in the `documentTable`. Figure 12 presents an example that shows the relational tables immediately after two XML fragments have been parsed by the system.

5.6 Relevance Measure

The Vector Space Model (VSM) is extended to allow key-tags as an indexing unit, which is in order to determine the relevance between a search query and an XML fragment.

Definition 5.2 (Relevance Score) *Let Q and F be a query and an XML fragment, where F is specified by an index contextual path, ρ, and p is the corresponding contextual path. Let Ω be a weight function that maps a given key-tag, k, from Q (or F) into an indexing weight (a positive constant). The mapping takes into account the two components of $k = (t, w)$. The* relevance score *of F to Q, denoted as $Sim(Q, F)$, is evaluated by using a similarity between the weight vectors of F and Q given by the following expression:*

$$Sim(Q, F) = \frac{\Sigma_{k \in (Q \cap F)} \Omega(k, Q) \times \Omega(k, F) \times G(k, p)}{\mid Q \mid \times \mid F \mid}. \tag{3}$$

The weights associated with the fragment are calculated based on the product of two frequency parameters, $\Omega(k, Q)$ and $\Omega(k, F)$, which indicate the statistical importance, and one path parameter, $G(k, p)$, which indicates the importance of the fragment granularity.

1. The key-tag frequency, $\Omega(k_i, Q)$, represents the frequency of occurrence of a key-tag, k_i, within a query $Q = (k_1, k_2, \ldots, k_n)$, and is defined as follows:

   $$\Omega(k_i, Q) = \frac{2(n - i + 1)}{(n + 1)n},$$

   where n is the total number of key-tags in Q. Here the equation is simply a normalization of the position index, i, of the key-tag with respect to the sum of all position indexes, given by $\Sigma i = \frac{n(n+1)}{2}$. As shown in the denominator, the fraction demonstrates our consideration that a higher order of the occurrence of k_i with respect to Q, k_i, should then have a higher weight.

2. The fragment frequency, $\Omega(k, F)$, represents the content discrimination factor, which means that if a key-tag appears often in a fragment, then it describes the fragment contents well. However, if a key-tag appears in many fragments, then it is not useful for distinguishing a fragment.

   $$\Omega(k, F) = \begin{cases} (N_k / N_F) \times log(N / N_C) & \text{where } k \text{ is a key-tag in } F; \\ 0 & \text{otherwise,} \end{cases}$$

   where N_F is the total number of key-tags in F, N_k is the number of occurrences of k in F, N_C is the number of fragments in the collection that contain k, and N is the total number of fragments in the collection. Here the equation is analogous to the well-known *tf-idf* definition (cf. [8]) in order to represent our consideration that if a key-tag is frequent in the fragment (the first fraction) and infrequent in other fragments (the second fraction), then k_i should have a higher weight with respect to F.

3. The degree of granularity matching of k in p, $G(k,p) = t/l_p$, where t is the number of occurrences of tag in p and l_p is the length of p in F. Here, the equation is a simple ratio to represent the specificity of k in the path p.

Consider the Query Q in Figure 3 and the XML DBLP fragment F in Figure 12. We have $\Omega((author, Mary), Q) = \frac{2(3-1+1)}{(3+1)3} = 0.5$ and $\Omega((author, Mary), F) = 1/8 \cdot log(10/1) = 0.125$, assuming 10 fragments in the collection. The path, $p =$ "$/dblp/www/author$" contains the key-tag (author, Mary). The length of p is 3 and thus $G(k,p) = (1/3) = 0.3333$. It follows that the product of the three terms, $\Omega(k,Q)$, $\Omega(k,F)$, and $G(k,p)$, in the numerator of $Sim(Q,F)$ is $(0.5 \cdot 0.125 \cdot 0.3333) = 0.02083$. Similarly, we compute the numerator for the key-tags $(title, XML)$ as $(0.3333 \cdot 0.125 \cdot 0.3333) = 0.01389$ and for $(year, 2006)$ as $(0.1667 \cdot 0.125 \cdot 0.3333) = 0.006945$. Finally, we have the following relevance score.

$$Sim(Q,F) = \frac{0.02083 + 0.01389 + 0.006945}{3 \times 8} = 0.001736.$$

It is worth pointing out that if $G(k,p) = 1$ and we ignore the tag component, $k.t$, in the weight functions, then the cosine measure formula given in Definition 5.6 becomes a simple relevance measure of searching normal flat documents.

6 Experiments

In this section, we study various fundamental factors affecting the RSSF framework by a spectrum of XML datasets including INEX 2006 topics [18]. The experiments are conducted on a Solaris 2.8 with CPU 1x300Mhz Ultra30 and 256MB memory. We study the effect of varying the voting threshold in VSNB, the contribution of the low level features on the standard rankers, and the effectiveness of the MRM. We also examine the effect of updating the adaptive rankers when users switch their preferences in the framework. Finally, we discuss some overhead issues of the system.

6.1 Effect of Varying Voting Threshold

In order to study the impact of the voting threshold β in Algorithm 3 on the VSNB performance, we carried out an experiment to test various β values by using the *mix query set*, Q_{mix}, given in Appendix I(a), which contains key-tag search queries with respect to a spectrum of XML datasets as shown in the first column of Figure 13. An effective adaptive ranking function should give high ranking to the fragments users prefer. Thus, we first measure ranking quality based on *the average rank of labeled fragments*, denoted by υ. Intuitively it means the smaller the υ value, the better the ranking quality. Then, we show the actual improvement by the metric *"relative average rank of labeled fragments"*, which is given by $\Upsilon = \frac{\upsilon_a}{\upsilon_o}$, where υ_a is derived from an adaptive ranking function and υ_o is obtained from the original search result. If $\Upsilon < 1$, then it indicates that an actual improvement has been achieved.

Note that the study involves a preference judgement on the ranking results. We adopt five user preferences as given in the five right-hand columns of Figure 13, which are employed to judge which results are relevant according to a classification that assigns normalized weights to the XML datasets. If a retrieved fragment belongs to either of the sample XML documents, the probability of classifying it as positive in various preferences is labelled according to their respective normalized weights. For example, when an XML fragment in a returned result originates from the DBLP dataset, the fragment has a 30% chance of being labelled (i.e. the fragment is identified to be relevant) according to the academic preference, $P_{academic}$, and 11.11% the mixed preference, P_{mix}.

The result of the experiment is presented in Figure 14, which is based on a set of tested queries, Q_{mix}, which consists of ueries with respect to the XML datasets and is given in Appendix I(a).

As elaborated in Section 5.4, the β value reflects the confidence that VSNB has on a spy $p_i \in P$ in selecting the *estimated negative* (EN) examples. On the one hand, small β values

XML Datasets	$P_{academic}$	$P_{liteature}$	$P_{scientific}$	$P_{commerical}$	P_{mix}
DBLP	0.3	0.0	0.0	0.1	0.1111
Shakespeare	0.1	0.9	0.0	0.0	0.1111
Weblog	0.0	0.0	0.25	0.1	0.1111
Treebank	0.0	0.0	0.25	0.0	0.1111
Swissprot	0.0	0.0	0.25	0.0	0.1111
NASA	0.0	0.0	0.25	0.0	0.1111
Auction Data	0.0	0.0	0.0	0.8	0.1111
University Course	0.3	0.1	0.0	0.0	0.1111
SIGMOD Record	0.3	0.0	0.0	0.0	0.1111

Figure 13: Probability of selecting XML fragments for five user preferences.

(e.g. 10%) imply that VSNB may assign a fragment as an EN example based on the results of only very few spies, leading to large classification error. On the other hand, large β values (e.g. 100%) means that VSNB is too conservative, since it assigns a fragment as EN only when *all* the spies agree that the fragment is an EN. As a result, it is able to select very few EN examples to improve the precision.

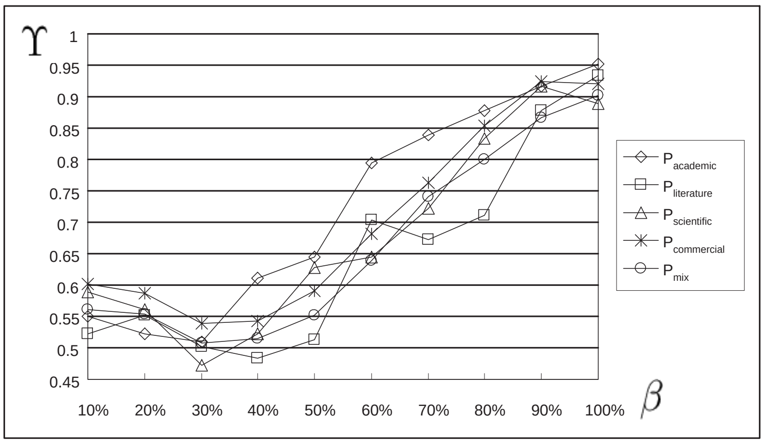


Figure 14: Performance of VSNB with varying β under five user preferences

Figure 14 shows that β indeed affects the performance of VSNB, since the curves are sloped and the optimal (minimal) values are generally around 30%. Large β values decrease the performance of VSNB, indicating that large β values make VSNB too conservative, which results in missing some real EN examples. Small β values may have the problem of admitting noisy EN examples, resulting in bad performance, which can also be observed in

Figure 14. Finally, it is worth noting that the optimal voting threshold, which is common to all preferences, gives flexibility to VSNB. The reason is that as users in reality have diversified interests and behavior, the optimal voting threshold can be used to adapt VSNB for different users. The theoretical analysis of this optimal voting threshold is also an interesting issue that deserves further study.

6.2 Effectiveness of Multi-Ranker Model

The RSSF algorithm is implemented on the RSVM, which is an open source code available in [47]. Besides this, the remaining work of the RSSF algorithm is mainly to deal with the input and output of the RSVM algorithm, which is indeed straightforward to implement. The challenging part of the evaluation is that much effort is required to carry out relevance justification with respect to preferences and compute the precision for all the rankers and the queries.

We analyze the results obtained in these experiments based on the metric of *k-precision*, which is defined as

$$k - precision = \frac{\text{Number of top } k \text{ relevant results}}{k},$$

where k is the number of top results returned by the rankers. If there are n relevant results in the top k results, the precision is n/k. We judge which results are relevant according to the five preferences given in the table of Figure 13. The relevance of the results is measured in an unbiased way by using precision. Precision and recall measures are common evaluation metrics used in information retrieval theory for searching textual documents [8]. However, we do not define the notion of the recall parameter in this context, since it is not practical to estimate the total number of relevant fragments on the XML datasets. Furthermore, the total number of fragments retrieved by the rankers is immaterial as the users are only concerned with a minority of fragments that is ranked at the top of a search result.

We compare the four standard rankers, XRs, regarding their particular strengths for handling different types of documents as follows. We test three different sets of queries, with each set consisting of 30 queries (see Appendix I(a) to (c)). The three sets of queries, Q_{dat},

Q_{doc} and Q_{mix}, have different characteristics in terms of tag labels and data values. For example, Q_{dat} has more complete tags, which targets more data-centric fragments. Q_{doc} has more complete keywords in the data values, which targets more document-centric fragments. Q_{mix} is a balanced set between Q_{dat} and Q_{doc}. The table below highlights some characteristics of the query sets. Here the terms complete tags and complete keywords refer to those key-tags that have no " $*$ " in t and w of a key-tag k. (cf. Definition 3.1)

Query Sets (30 each)	Average no. of key-tags	Complete tag (%)	Complete keyword(%)
$Q_{mix}(AppendixI(a))$	3.33	67%	70%
$Q_{dat}(AppendixI(b))$	3.33	100%	60%
$Q_{doc}(AppendixI(c))$	3.34	43.56%	100%

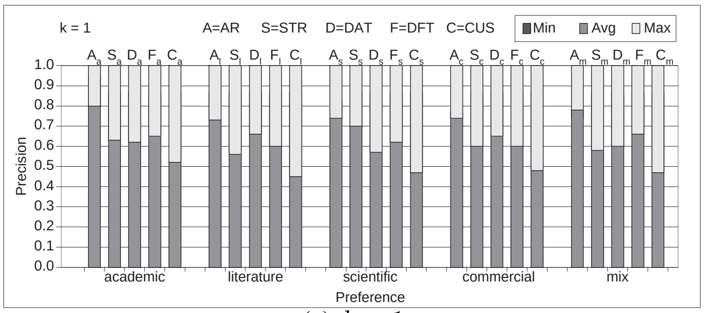
Figure 15: Statistics for the three query sets, Q_{dat}, Q_{doc} and Q_{mix}.

The three values of worst precision, average precision and best precision are superimposed on Figures 16(a) to 16(d) for Q_{mix}, Figures 17(a) to 17(d) for Q_{dat}, and Figures 18(a) to 18(d) for Q_{doc}, whose queries are detailed in Appendix I(a) to (c). In each diagram, there are altogether five bundles of bar charts such that each bar chart represents a ranking scheme. A bundle of five bar charts shows the performance of the five various ranking schemes with respect to user's preference. For the CUS ranking, we simply assign the weight for each query using random numbers.

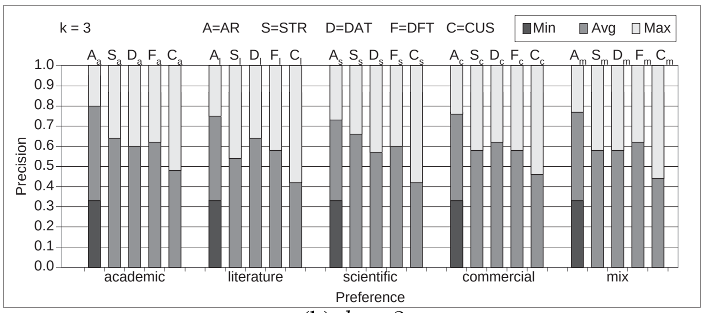
There are some important observations from Figures 16 to 18.

First, the performance of ARs (i.e. the first bar A_x for $x \in \{a, l, s, c, m\}$) are, in general, better than either of the standard rankers (i.e. the second to fifth bars XR$_x$ for XR $\in \{S, D, F, C\}$) in all preferences and query sets. This result is important, since the standard rankers have their respective strengthes, for example STR obtains better results for data-centric queries, Q_{dat}, in scientific preference than DAT while DAT performs better than STR for document-centric queries, Q_{doc}, in literature preference. However, the ARs are able to make use of the strengths of the standard rankers and obtain a better performance.

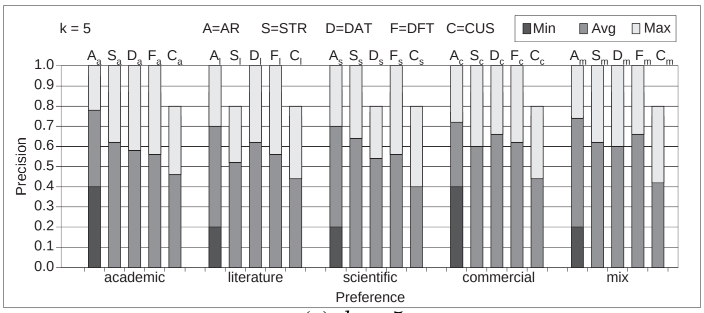
Second, an interesting observation is that, although DFT performs better than STR or DAT in some preferences, it is inferior to STR and DAT when it comes into the scope



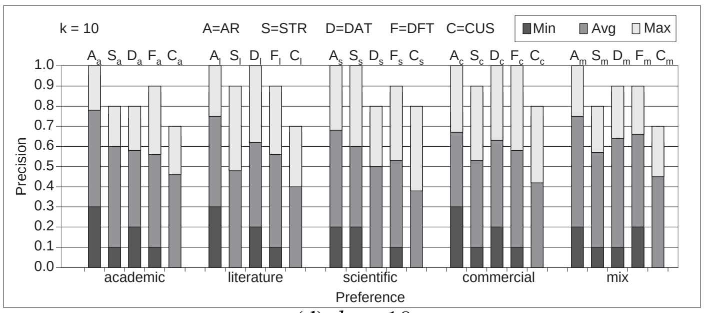
(a) $k = 1$



(b) $k = 3$



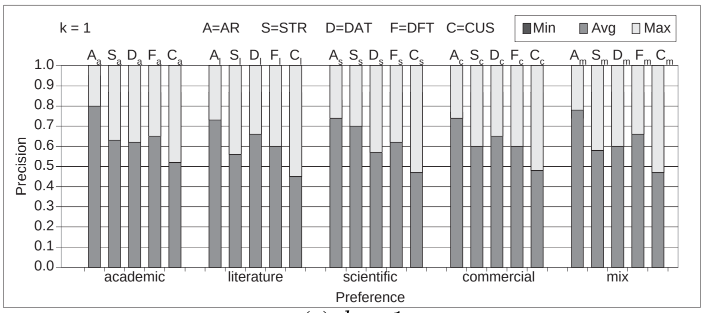
(c) $k = 5$



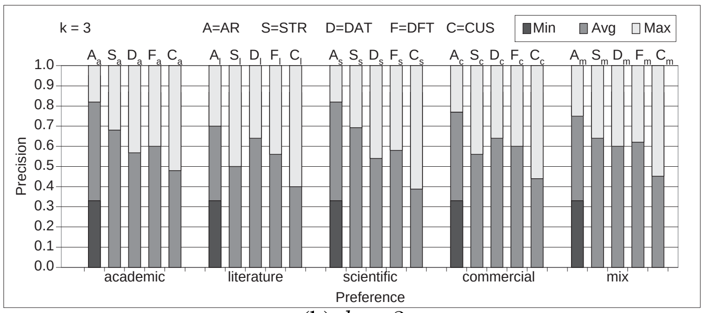
(d) $k = 10$

Figure 16: Comparison of the top-k precision of the XRs and AR for Q_{mix}

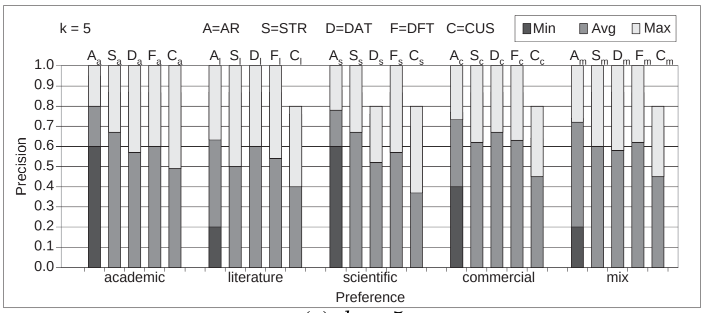
of scientific and literature preferences, where STR and DAT have a better strength than DFT. However, for Q_{mix} in mixed preferences in Figure 16, the performance of DFT becomes better than the counterparts of STR and DAT. For other query sets Q_{dat} and Q_{doc} in mixed preferences in Figures 17 and 18, the performance of DFT is also comparable to the counterparts of STR and DAT. We account for this phenomena by noting that DFT contains more low level features. Thus, it supports our intention that DFT is an effective *default* ranking scheme to initialize the adaptive ranking process in the MRM, since it is the best ranking choice when without any knowledge of user preference or VSNB training. Finally, CUS performs worst in all cases, which can be viewed as the baseline of ranking performance of other schemes, since CUS simply randomly assigns weight to all the features in the experiment. However, this ranker in practice is still important when fine tuning the system is needed.



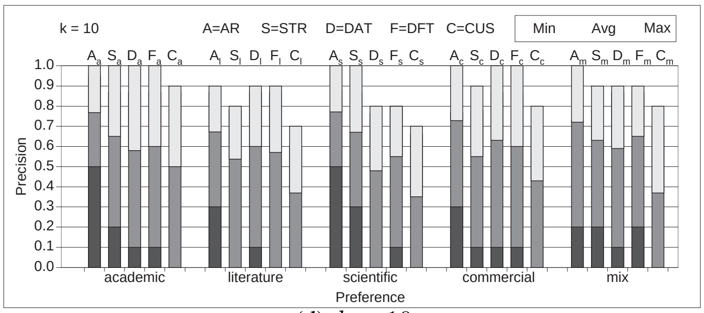
(a) $k = 1$



(b) $k = 3$

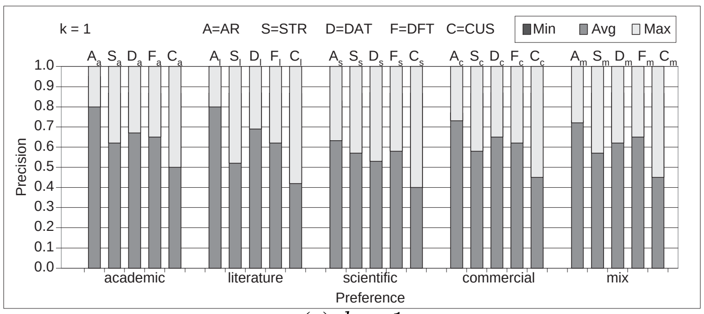


(c) $k = 5$

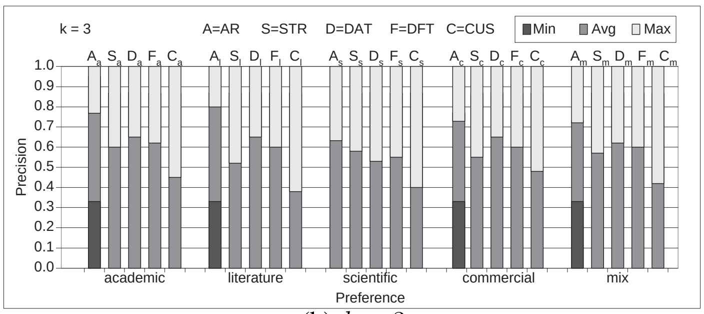


(d) $k = 10$

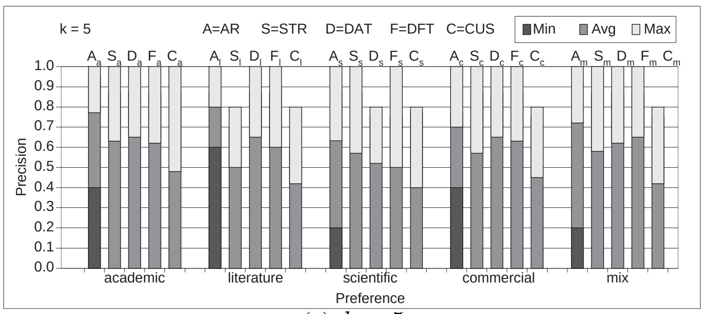
Figure 17: Comparison of the top-k precision of the XRs and AR for Q_{dat}



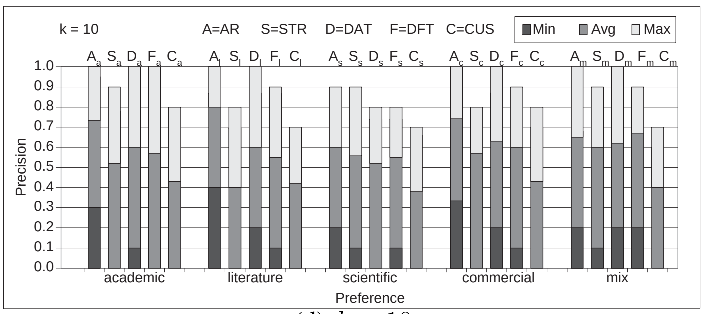
(a) $k = 1$



(b) $k = 3$



(c) $k = 5$



(d) $k = 10$

Figure 18: Comparison of the top-k precision of the XRs and AR for Q_{doc}

6.3 Effectiveness of Low Level Feature on Standard Rankers

In this section, we carry out an experiments on how individual low level features affect the standard rankers *STR* and *DAT*. Recall the ranking schemes for structure ranking, ϕ_{STR}, is defined as $(Sim_K,\ Sim_P,\ Sim_E,\ Sim_{AO},\ Sim_{SO},\ Grn_{Sib},\ Grn_{Chi},\ Grn_{Dis_+},\ Grn_{Dis_-})$ and for data ranking, ϕ_{DAT}, is defined as $(Sim_K,\ Sim_A,\ Sim_E,\ Sim_C,\ Grn_{Tag},\ Grn_{Att})$.

In this experiment, we use five different sets of queries: $Q_{academic}$, $Q_{liteature}$, $Q_{scientific}$, $Q_{commercial}$ and $Q_{mixdata}$, each set contains 20 queries. Each query consists of 1 to 5 key-tags, which are randomly chosen from a collection XML datasets according to Figure 19. The five sets of queries can be found in Appendix II(a) to (e). For example, the query in $Q_{academic}$, (<course>Geometry</course>, <year>02</year>, <*> Lam</*>), is aiming at selecting fragments from DBLP, University and SIGMOD Record datasets. The reason for using these five query sets, which have clear target on XML datasets, is that the influence of diversified XML datasets on the low level features can be directly observed and better analyzed.

Query Sets	XML Data
$Q_{academic}(AppendixII(a))$	DBLP, University Course, SIGMOD Record
$Q_{literature}(AppendixII(b))$	Shakespeare
$Q_{scientific}(AppendixII(c))$	Weblog, Treebank, Swissprot, NASA
$Q_{commercial}(AppendixII(d))$	Auction Data
$Q_{mixdata}(AppendixII(e))$	All of the above datasets

Figure 19: The five query sets

We test the precision of the queries by removing one (low level) feature at a time and record the change of the precision of query answer. The result are shown in Figures 20 and 21.

First, we compute the *average top-10 precision* ρ of the queries in a query set according to ϕ_{STR} or ϕ_{DAT}, then we measure ρ when removing one of the low level features and denote it as ρ'. The change in precision, $\Delta\rho$, is given by $\frac{(\rho'-\rho)}{\rho} \times 100\%$. For example, in the column of Sim_K, it shows that $\Delta\rho$ is decreased by 15.15% in $Q_{academic}$ and by 13.51% in $Q_{literature}$, when we remove Sim_K from ϕ_{STR}.

From the last row of the above results, we can see that removing any of the low level features from either ϕ_{STR} or ϕ_{DAT} on average decreases the precision, except Grn_{Att} in ϕ_{DAT}, which increases the overall precision by 0.8%. This anomaly is due to the fact that only a few fragments in our datasets contains attribute values. As a result, a slight increase in

Queries	Sim_K	Sim_P	Sim_E	Sim_{AO}	Sim_{SO}	Grn_{Sib}	Grn_{Chi}	Grn_{Dis+}	Grn_{Dis-}
$Q_{academic}$	-15.15%	-4.55%	-6.82%	-0.76%	-3.03%	-12.12%	+6.06%	-6.82%	+2.27%
$Q_{literature}$	-13.51%	+5.41%	-12.61%	-3.60%	-6.31%	-5.41%	-8.12%	+0.90%	+2.70%
$Q_{scientific}$	-4.76%	-10.20%	-5.44 %	-4.08%	-1.36%	+6.80%	+2.04%	-6.80%	-6.12%
$Q_{commercial}$	-9.34%	-5.61%	-6.54%	+3.74%	+0%	+0.93%	-2.80%	-1.87%	+0.93%
$Q_{mixdata}$	-7.87%	-3.94%	-5.51%	-2.36%	+2.36%	-1.57%	-0.79%	-3.15%	-0.79%
Average	-10.13%	-7.09%	-8.39%	-1.41%	-1.67%	-2.27%	-0.72%	-3.55%	-0.20%

Figure 20: Precision change of removing low level features on the structure ranker, ϕ_{STR}.

Queries	Sim_K	Sim_A	Sim_E	Sim_C	Grn_{Tag}	Grn_{Att}
$Q_{academic}$	-12.50%	0%	-5.47%	-15.63%	-3.13%	-0.78%
$Q_{literature}$	-11.64%	-1.37%	-3.42%	-9.59%	-2.74%	-1.37%
$Q_{scientific}$	-6.72%	+1.68%	-5.04%	-8.40%	+5.88%	+2.52%
$Q_{commercial}$	-6.16%	-1.37%	-5.48%	-7.53%	-0.68%	+1.37
$Q_{mixdata}$	-9.70%	-0.75%	-5.22%	-10.45%	-0.75%	+2.24%
Average	-9.35%	-0.36%	-4.93%	-10.32%	-0.28%	+0.80%

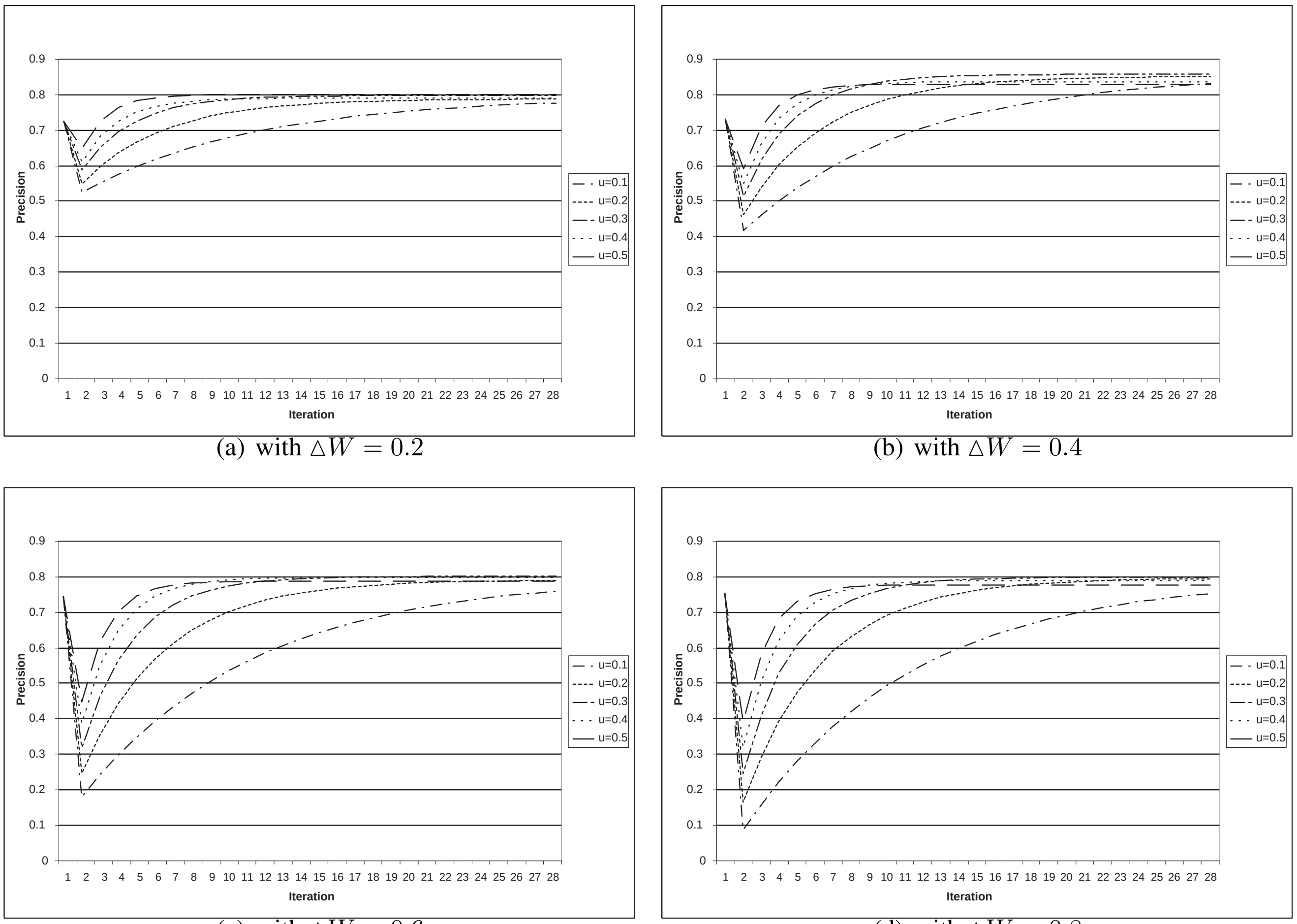
Figure 21: Precision change of removing low level features on the data rankers, ϕ_{DAT}.

some querying results from $Q_{academic}$ and $Q_{liteature}$ has a very large impact on the precision.

We also observe that the low level feature, Sim_K plays an important role in both standard rankers, which decreases the precision from 9 to 11% when removing it from the rankers. From Figure 20, we find that removing Sim_P decreases the precision for the query set $Q_{scientific}$ by 10.2%, however, it increases the precision for the query set $Q_{liteature}$ by 5.41%. This interesting result is due to the different structure for the XML datasets. The XML datasets for $Q_{scientific}$ are often deeply nested with different tags, removing low level features, such as Sim_K, Grn_{Dis+} and Grn_{Dis-}, decreases the precision significantly. The XML datasets related to $Q_{liteature}$ are only nested by 2 to 3 levels with regular tags. Removing these low level features does not decrease the precision, on the contrary it enhances the precision, since the weights of other less relevant features are relatively increased. Admittedly, this study only examines the effect of missing a single feature against the queries. It would be interesting, from a machine learning perspective, to extend the study of the impact on rankers under multiple missing features, since the interaction between features may exist in the vector. We do not carry out this extension as it is quite involved in terms of the number of possible feature combinations and in fact, the result is only peripherals to our main ideas.

6.4 Updating Adaptive Rankers

In this section, we run experiments on how the aging factor, u, affects the precision of the queries. We introduce a variable, $\triangle W$, which represents the degree of changing preferences. $\triangle W$ is computed as the average difference of the weights between two adaptive weighting vectors. For example, a user's preference generated by the nth query, which is represented by the adaptive weight vector $\overrightarrow{W}_n$, is (0.7, 0.1, 0.05, 0.05, 0.08, 0.01, 0.005, 0.005), and he/she switches the preference, $\overrightarrow{W}_{n+1}$, to (0.1, 0.7, 0.05, 0.05, 0.01, 0.08, 0.005, 0.005), then $\triangle W_{n+1} = \frac{|(0.7-0.1)|+|(0.1-0.7)|+\ \cdots\ +|(0.005-0.005)|}{8} = 0.1675$.



(a) with $\triangle W = 0.2$

(b) with $\triangle W = 0.4$

(c) with $\triangle W = 0.6$

(d) with $\triangle W = 0.8$

Figure 22: Changing preferences and the effect of AR updates

Figures 22(a) to 22(d) show the top-10 precisions when the preference changes from $u = 0.1$ to $u = 0.5$, and $\triangle W_1 = 0.2$ to 0.8 respectively. The experiment is carried out using the mix query set, Q_{mix} from Appendix I(a). We can see that the system adapts to the change in $\overrightarrow{W}$ much faster than other u values when $u = 0.5$, which takes about five iterations to obtain 95% of maximum precision in the four cases of $\triangle W$. When $u = 0.1$, it takes more than 28 iterations to adapt to the change after updating AR in the second iteration.

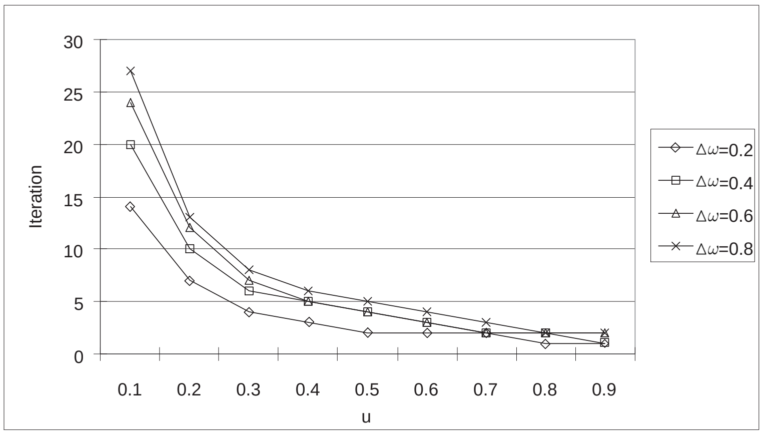


Figure 23: The number of iterations for the adaptive ranker to obtain a stable precision

Figure 23 shows the number of iterations required for the adaptive ranker to obtain 95% of the maximum precision. The results show the higher the value of u, the higher the effectiveness of the weighting vector of the current iteration. Thus, it seems that the system adapts faster under higher u values. However, using high u values makes the system too sensitive to the recent change and thus its performance may become very unstable, especially when a user changes his/her preference frequently. As discussed in Section 4.3, the optimized performance under continuous AR updates needs more fundamental analyses, which is not the main focus of this paper.

6.5 Some Overhead Issues

The total processing time in the MRM depends on both the searching and ranking times. The computation of the ranking incurs some overheads ranging from 7% to 13% of the total processing time as shown in the last column in Figure 24. As we can see in the second and third columns, the search time does not simply depend on the size of the document in the corpus. For example, Weblog requires much less search time than does Treebank, although they are similar in size. This is due to the fact that Weblog is more regular in structure but

Treebank has many more distinct elements and a deeper structure for indexing. Eventhough the data sets Auction Data, University Courses and SIGMOD Record are small in document size, the system still has to read the tag information before we actually search the database and thus gives rise to a search time of over 1000ms. It is also worth mentioning that the high ranking overhead for NASA is due to the fact that although its smaller size reduces the search time, the returned fragments are large and thus more time is needed for ranking than in the cases of Weblog and SwissProt.

XML Data	Doc Size (MB)	Searching time (ms)	Ranking time (ms)	Total time (ms)	Ranking Overhead (%)
DBLP	127	3833	461	4294	10.74
NASA	23	2879	398	3277	12.15
Shakes	32	3608	305	3913	7.79
Weblog	89.8	3340	266	3606	7.38
Treebank	82	4275	438	4713	9.29
SwissProt	109	3232	279	3511	7.95
Auction Data	0.1	1276	106	1382	7.67
University Courses	3.3	1864	175	2039	8.58
SIGMOD Record	0.5	1433	114	1547	7.37

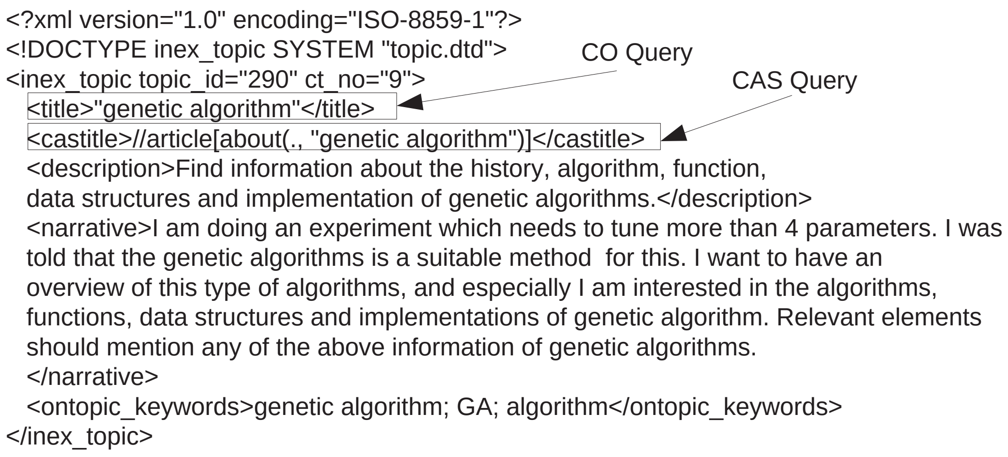
Figure 24: Searching Time and Ranking Overhead

6.6 Experiments using INEX Datasets

In this section, we further examine the effectiveness of adaptive ranking based on XRs and low level features. We adopt the INEX 2006 collection [18], which is the Wikipedia XML Corpus that provides a set of 659,388 XML documents together with a set of standard queries and their relevance assessments for the retrieval runs (called INEX Topics). There are totally 125 INEX 2006 topics. In each INEX topic, two types of queries are provided as the means for different information retrieval experiments: the Content-Only (CO) and the Content and Structure (CAS) queries. The CO queries ignore the document structure and contain only content-related conditions and the CAS queries explicitly state the structural constraints in the form of NEXI queries [53, 54].

An example of INEX topic, which is labelled as 290.xml, is presented in Figure 25. It consists of the standard titles, description and narrative fields. We can find the corresponding

CO and CAS queries described within the <title> and <castitle> elements, which are "genetic algorithm" and "`//article[about(., "genetic algorithm")]`" respectively. A fragment is considered as relevant if it matches the "description", "narrative" and "ontopic_keywords" elements in this INEX topic.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="290" ct_no="9">
  <title>"genetic algorithm"</title>
  <castitle>//article[about(., "genetic algorithm")]</castitle>
  <description>Find information about the history, algorithm, function,
  data structures and implementation of genetic algorithms.</description>
  <narrative>I am doing an experiment which needs to tune more than 4 parameters. I was
  told that the genetic algorithms is a suitable method  for this. I want to have an
  overview of this type of algorithms, and especially I am interested in the algorithms,
  functions, data structures and implementations of genetic algorithm. Relevant elements
  should mention any of the above information of genetic algorithms.
  </narrative>
  <ontopic_keywords>genetic algorithm; GA; algorithm</ontopic_keywords>
</inex_topic>
```

Figure 25: An Example INEX topic (290.xml)

We conduct two sets of experiments for examining the effectiveness of adaptive ranking. One is to compare the AR ranker with the four XR rankers in the MRM and another is to compare the adaptive ranker based on XRs with the one based on low level features.

In our experiments, we need to convert the CAS queries into key-tag queries to adapt our system. For example, one CAS query in topic 289, "`//*[about(., emperor "Napoleon I" Polish)]`", is written as the key-tag query, {`<*>emperor</*>`, `<*>Napoleon I</*>`, `<*>Polish</*>`} , and another query in topic 290, "`//article[about(., "genetic algorithm")`", is written as the key-tag query, `<article> genetic algorithm </article>`. Other key-tag queries converted from the topics can be found in Appendix III [58].

In order to compare our search results with other relevance models [35, 48, 38], we adopt the two parameters of *Mean Average Precision* (MAP) and the precision at 10 as the searching effectiveness measure [41]. We compute MAP for each topic using the average precision over 100 recall points (0.01 to 1.00) in the query result and then taking the average of all such topic-wise precisions (cf. the same approach used in [48]). In addition to MAP values, we also measure the value of precision at 10, denoted as $prec@10$, for each run, which is also used in [35, 48].

The baseline for all the experiments is a run for all 125 INEX topics using our adapted SVM model discussed in Section 5.6. We examine the precision without considering the user feedback for the 1500 results of each INEX topic.

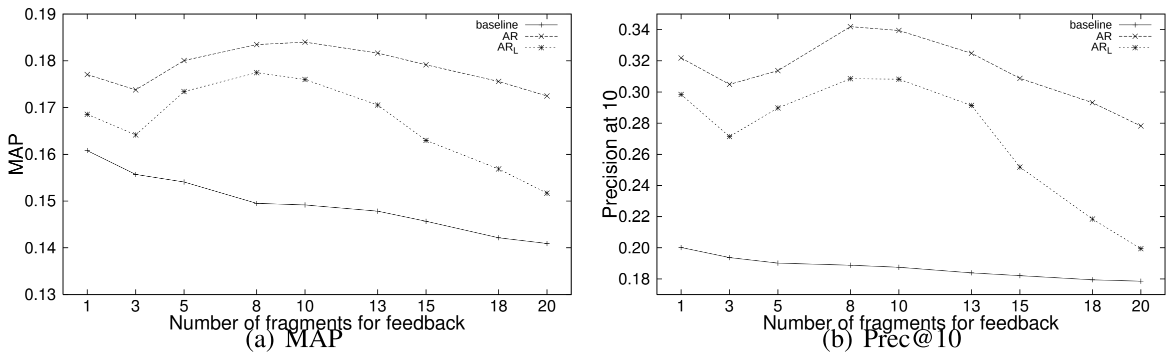
Rankers	MAP	prec@1	prec@3	prec@5	prec@10
Baseline	0.1630	0.2478	0.2242	0.2212	0.2204
AR	0.1801	0.3486	0.3363	0.3298	0.3188
STR	0.1705	0.2470	0.2549	0.2465	0.2315
DAT	0.1755	0.3221	0.3186	0.3009	0.2914
DFT	0.1704	0.2832	0.2950	0.2973	0.2684
CUS	0.1677	0.2081	0.1976	0.1929	0.1922
$TopX$	0.1053	0.3097	0.2448	0.2407	0.2106

Figure 26: MAP and prec@k for the baseline, AR, four XRs and TopX.

Figure 26 shows the results of the MAP and precision at $k \in \{1, 3, 5, 10\}$ of the baseline, AR, and the four XRs using the INEX 2006 collection and topics. It can be checked that the AR ranker yields the best result, and the performance of DAT is comparable to the AR ranker. The result of STR is less satisfactory compared to AR and DAT, which is mainly because the structure part of the CAS queries of INEX 2006 topics are mostly “article” and “section” tags. This makes the judgement of STR less effective since it takes no advantage of the structural aspect of the fragments. The result is consistent to our findings in Section 6.2 (cf. Q_{doc} in Figure 18). The performance of DFT is comparable to the AR ranker, and the performance of CUS is the least impressive.

We also compare our rankers with TopX engine [51], which is an XML search engine accessible in [52] for top-k searching. TopX determines the k top-ranked result elements of documents according to their aggregated scores with respect to all query conditions about content and structure. The results in Figure 26 show that AR outperforms TopX. The MAP of TopX is relatively low, since TopX engine stops processing as soon as it can safely determine k top-ranked result elements. We also find that DAT also outperforms TopX. This can be explained by the fact that the ranking of TopX engine depends on both content and structure. However, the structure of the dataset in INEX 2006 collection is quite regular and thus structure features do not help much to improve TopX ranking, which also explains the less satisfactory performance found in STR. However, the tailored ranker DAT is more effective in ranking this kind of XML fragments.

We now study the difference in performance of the adaptive ranker that is based on the set of the low level features, named AR_L, and on the XRs developed in MRM (i.e. the original AR). We also compare the results with our baseline run. The generation of AR_L is similar to that of using the four XRs. However, AR_L adapts to the twelve low level ranking features presented in Section 4.2 in the adaptive ranking feature vector Φ given in Section 4.1, rather than to the high level features related to XRs. Using the relevant feedback we compute the values of MAP and prec@10 against different numbers of feedback fragments. The top-k relevant fragments are selected as our feedback in this experiment. To access the quality of feedback algorithms, we use the residual collection techniques [46] that is also used in the INEX Relevance Feedback Track [35, 48]. In this technique, all XML fragments that are used by the feedback algorithm, (i.e. those whose relevance is known to the algorithm), must be removed from the list of returned fragments before evaluation of the results with feedback takes place. This includes all k elements “seen” or used in the feedback process regardless of their relevance.



(a) MAP (b) Prec@10

Figure 27: Comparison of the MAP and prec@10 of the the baseline, AR_L and AR.

Figure 27 shows the MAP and prec@10 of the AR_L and the AR rankers with the baseline against the number of feedback fragments running from 1 to 20.

When the number of feedback fragments is small (around 1 to 3), the MAP and prec@10 values decrease. This is because both adaptive rankers are not able to predict the preference accurately. As the number of feedback fragments increases, it can be seen that the peak MAP and prec@10 are obtained when there are 8 to 10 feedback fragments. However, when the number of feedbacks increases further, the performance decreases again. The can be explained by the fact that using residual collection techniques we need to remove used feedback fragments from the collection. Gradually removing relevant fragments from the dataset gives rise to a decrease in the MAP and prec@10. A further increase in the number of feedback fragments does not help improve the prediction accuracy.

From Figure 27, we also see that the AR ranker outperforms the AR_L ranker in general. Although the peak performance of AR_L is comparable to AR, the performance of AR_L degrades more rapidly and falls to the baseline after the peak. We account for this behavior by noting that each fragment contributes some noise scores for the low level features in AR_L. Thus, the distribution of the features in AR_L becomes "flat" in the weight vector as the number of feedback fragments becomes large, and thus degrades the adaptive performance. However, XRs consist of the features adapted as a whole and thus individual features are less affected by the small noise in learning.

7 Concluding Remarks

This study involves cross disciplinary techniques in IR, databases and machine learning. We have presented an effective approach to performing an adaptive XML search. Our proposed approach deals with the diversity of XML data in reality and the need for specifying target information in simple and direct search queries. We suggest that an XML search query can be expressed as a list of key-tags, which is a natural generalization of keywords in traditional IR searching. We have also presented an extension of the vector space model that integrates various similarity and granularity measures between a search query and XML fragments.

As XML documents are diverse, we consider four standard ranking schemes based on different combinations of low level features that are related to XML fragments and search queries. The novel multi-ranker model is developed to cater to different search needs and adapt the user preference further. The adaptive ranking schemes can be trained in a framework called RSSF, which is able to improve the retrieval quality via learning from the user's preference feedback. Based on the voting SpyNB framework, our RSSF algorithm requires only a small set of labeled data for training and does not intervene in the search process. We demonstrate that the MRM indeed improves the retrieval quality when comparing the adaptive rankers to the individual standard rankers.

In this study the adaptive search engines are examined from a technical perspective. It is worth carrying out a user study to check if the simple text search via key-tag queries in practice is adequate for an XML search. For example, we can provide subjects with a list of search tasks against a given XML corpus by using key-tag queries and mixed mode XML query processing [24]. In addition, the interaction within the feature set for ranking and an analytical model of updating adaptive rankers with respect to a changing preference are interesting issues that deserve further study.

References

[1] E. Agichtein, E. Brill and S. Dumais *Improving Web Search Ranking by Incorporating User Behavior.* In: Proc. of SIGIR, 2006

[2] E. Agichtein, E. Brill, S. Dumais and R. Ragno *Learning User Interaction Models for Predicting Web Search Result Preferences.* In: Proc. of SIGIR, 2006

[3] S. Amer-Yahia, L. Lakshmanan, P. Shashank. *FleXPath: Flexible Structure and Full-Text Querying for XML.* In: Proc. of SIGMOD, 2004.

[4] S. Amer-Yahia, C. Botev and J. Shanmugasundaram. *TeXQuery: A FullText Search Extension to XQuery.* In Proc. of WWW, 2004.

[5] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava and D. Toman. *Structure and Content Scoring for XML.* In: Proc. of VLDB, 2005.

[6] S. Amer-Yahia, E. Curtmola, A. Deutsch. *Flexible and Efficient XML Search with Complex Full-text Predicates.* In: Proc. of SIGMOD, 2006.

[7] S. Amer-Yahia and M. Lalmas. *XML Search: Languages, INEX and Scoring.* In: SIGMOD Record, Vol. 35, No. 4, December 2006.

[8] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-wesley-Longman, 1999.

[9] K. Bennet and A. Demiriz. *Semi-supervised Support Vector Machines.* Advances in Neural Information Processing Systems, Vol. 11, pp. 368–374, 1998.

[10] J. Bosak. *Shakespeare in XML.* http://www.ibiblio.org/xml/examples/shakespeare/, 2004.

[11] Y. Cao, J. Xu, T. Liu, H. Li, Y. Huang and H. Hon *Adapting Ranking SVM to Document Retrieval.* In: Proc. of SIGIR, 2006

[12] D. Carmel, Y. S. Marrek, M. Mandelbrodand, Y. Mass, A. Soffer. *Searching XML documents via XML fragments.* In: Proc. SIGIR, 2003.

[13] X. Chai, L. Deng, Q. Yang and C. X. Ling *Test-Cost Sensitive Naive Bayes Classification.* In: Proc. of ICDM, 2004.

[14] S. Cohen, J. Mamou, Y. Kanza, Y. Sagiv. *XSEarch: A Semantic Search Engine for XML.* In: Proc. of VLDB, 2003.

[15] S. Cohen, R. Shapire, and Y. Singer. *Learning to Order Things.* Journal of Artificial Intelligence Research, Vol. 10, pp. 243–270, 1999.

[16] K. Crammer and Y. Singer. *Pranking with Ranking.* In: Proc. of NIPS, 2001.

[17] L. Deng, X. Chai, Q. Tan, W. Ng, D.L. Lee. *Spying Out Real User Preferences for Metasearch Engine Personalization.* In: Proc. of ACM WEBKDD, 2004.

[18] L. Denoyer and P. Gallinari. *The Wikipedia XML Corpus.* In: SIGIR Forum, 2006.

[19] N. Fuhr. *Optimum Polynomial Retrieval Functions Based on the Probability ranking principle.* ACM Trans. on Info. Sys., Vol. 7, Issue 3, pp. 183–204, 1989.

[20] N. Fuhr. *Probabilistic Models in Information Retrieval.* The Computer Journal, Vol. 35, No. 3, pp. 243–255, 1992.

[21] A. Trotman, M. Lalmas and N. Fuhr. *INitiative for the Evaluation of XML Retrieval (INEX).* http://inex.is.informatik.uni-duisburg.de/2007/index.html.

[22] L. Granka, T. Joachims, and G. Gay. *Eye-Tracking Analysis of User Behavior in WWW-Search.* In: Proc. of SIGIR, 2004.

[23] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. *XRANK: Ranked Keyword Search over XML Documents.* In: Proc. of SIGMOD, 2003.

[24] A. Halverson, et al. *Mixed Mode XML Query Processing.* In: Proc. of VLDB, 2003.

[25] R. Herbrich, T. Graepel, and K. Obermayer. *Large Margin Rank Boundaries for Ordinal Regression.* In A. S. et al., editor, Advances in Large Margin Classifiers, pp. 115–132, 2000.

[26] T. K. Ho, J. J. Hull, and S. N. Srihari. *Decision Combination in Multiple Classifier Systems.* In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16. No. I, January 1994.

[27] T. Joachims. *Optimizing Search Engines using Clickthrough Data.* In: Proc. of SIGKDD, 2002.

[28] T. Joachims L. Granka and B. Pan. *Accurately Interpreting Clickthrough Data as Implicit Feedback.* In: Proc. of SIGIR, 2005.

[29] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski and G. Gay. *Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search.* ACM Transactions on Information Systems (TOIS), Vol. 25, No. 2 (April), 2007.

[30] C. Kemp and K. Ramamohanarao. *Long-term Learning for Web Search Engines.* In T. E. et al., editor, PKDD, pp. 263–274, 2003.

[31] J. Kurose and K. Ross. *Computer Networks: A Top Down Approach Featuring Internet.* 3rd Edition, Addison Wesley, 2004

[32] Y. Li, C. Yu and H.V. Jagadish. *Schema-Free XQuery.* In: Proc. of VLDB, 2004.

[33] X. Li and B. Liu. *Learning to Classify Text using Positive and Unlabeled Data.* In: Proc. of IJCAI, 2003.

[34] B. Liu, W. S. Lee, P. S. Yu and X. Li. *Partially Supervised Classification of Text Documents.* In: Proc. ICML, 2002.

[35] Y. Mass and M. Mandelbrod. *Relevance Feedback for XML Retrieval.* In: Proc. of INEX, 2004.

[36] Y. Mass and M. Mandelbrod. *Using the INEX Environment as a Test Bed for Various User Models for XML Retrieval.* In: Proc. of INEX, 2005.

[37] A. McCallum and K. Nigam. *A Comparison of Event Models for Naive Bayes Text Classification.* In Proc. of AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48, 1998.

[38] V. Mihajlovic, G. Ramirez, A. P. de Vries and D. Hiemstra. *TIJAH at INEX 2004 Modeling Phrases and Relevance Feedback.* In: Proc. of INEX, 2004.

[39] G. Miklau. *UW XML Repository.* http://www.cs.washington.edu/research/xmldatasets/, 2006.

[40] T. Mitchell. *Machine Learning.* McGraw Hill, Inc., 1997.

[41] NIST. *Common Evaluation Measures.* appendix in Special Publication 500-250 (TREC 2001), NIST, Gaithersburg, MD.

[42] L. Page, S. Brin, R. Motwani and T. Winograd. *The Pagerank Citation Ranking: Bringing Order to the Web.* Technical report, Stanford Digital Library Technologies Project, 1998.

[43] F. Radlinski and T. Joachims. *Query Chains: Learning to Rank from Implicit Feedback.* In: Proc. of KDD, 2005

[44] F. Radlinski and T. Joachims. *Evaluating the Robustness of Learning from Implicit Feedback.* In: Proc. of ICML, 2005.

[45] S. Rajaram, A. Garg, Z. S. Zhou, and T. S. Huang. *Classification Approach Towards Ranking and Sorting Problems.* In Lecture Notes in Artificial Intelligence, volume 2837, pp. 301–312, September 2003.

[46] I. Ruthven and M. Lalmas. *A survey on the Use of Relevance Feedback for Information Access Systems.* Knowledge Engineering Review, 18(1), 2003.

[47] SVM^{light}. *Support Vector Machine.* http://svmlight.joachims.org/, Ver 6.01 2004.

[48] R. Schenkel and M. Theoblad. *Feedback-Driven Structural Query Expansion for Ranked Retrieval of XML Data.* In: Proc. of EDBT 2006.

[49] Q. Tan, X. Chai, W. Ng, and D. L. Lee. *Applying Co-training to Clickthrough Data for Search Engine Adaptation.* In: Proc. of DASFAA, 2004.

[50] A. Theobald and G. Weikum. *The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking.* In: Proc. of EDBT, 2002.

[51] M. Theobald, R. Schenkel and G. Weikum. *An Efficient and Versatile Query Engine for TopX Search.* In: Proc. of VLDB, 2005.

[52] M. Theobald, R. Schenkel and G. Weikum. *TopX Search.* http://infao5501.ag5.mpi-sb.mpg.de:8080/topx.

[53] A. Trotman and B. Sigurbjornsson. *Narrowed Extended XPath I (NEXI).* In: Proc. of INEX, 2004.

[54] A. Trotman and M. Lalmas. *The Interpretation of CAS.* In: Proc. of INEX, 2005.

[55] World Wide Web Consortium. *XQuery 1.0: An XML Query Language.* http://www.w3.org/TR/xquery/, W3C Working Draft 22 August 2003.

[56] World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Full-Text.* http://www.w3.org/TR/2005/WD-xquery-full-text-20050404/, W3C Working Draft 4 April 2005.

[57] *XML SQL Utility in Oracle.* http://www.oracle.com/index.html, 2004.

[58] *Full List of Queries Used in Experiments.* http://www.cse.ust.hk/∼wilfred/mrm/, 2007.

Note: we only select samples of queries in each category of the following appendices. For the full list of queries, the readers can consult from the web pages in [58].

Appendix I: Sample Test Queries for Multi-Ranker Model for Adaptive XML Searching.

(a) The query set: Mix queries, Q_{mix}

1. <name>xml</name>
2. <*>281</*>, <table>comp</table>
3. <sub>art</sub>, <info>*</info>
4. <lab>*</lab>, <start>8</start>, <course>data</course>
5. <*>magic</*>, <*>fr</*>, <para>*</para>,<name>*</name>
6. <bldg>*</bldg>,<credit>*</credit>,<re>*</re>,<page>*</page>, <hour>*</hour>
7. <table>comp</table>, <entry>33</entry>, <*>system component</*>, <*>IV</*>, <*>new</*>
8. <limit>0</limit>, <time>09</time>, <start>09</start>, <end>12</end>
9. <*>1995</*>, <*>engine</*>, <*>computer science</*>, <*>course</*>, <*>university</*>
10. <speech>duke</speech>,<line>86</line>,<pop>10</pop>,<instruct>ge</instruct>,<limit>0</limit>

(b) The query set: Data-centric queries, Q_{dat}

1. <author>La</author>
2. <title>*</title>, <name>e</name>
3. <author>a</author>, <initial>d</initial>, <line>7</line>
4. <speech>you</speech>, <descr>*</descr>, <day>5</day>, <i>*</i>
5. <title>search</title>, <year>9</year>, <nn>*</nn>, <sen>*</sen>
6. <cpu>*</cpu>, <city>la</city>,<name>j</name>, <info>*</info>
7. <vv>*</vv>,<cou>data</cou>,<month>6</month>,<title>*</title>
8. <section>e</section>,<lab>*</lab>,<start>08</start>,<end>12</end>,<title>*</title>
9. <editor>na</editor>,<keyword>data</keyword>,<history>8</history>,<x>e</x>
10. <act>active</act>,<name>sp</name>,<key>data</key>,<nn>*</nn>,<sell>*</sell>

(c) The query set: Document-centric queries, Q_{doc}

1. <author>mary</author>
2. <persona>lord</persona>,<weblog>html</weblog>, <*>10</*>
3. <bldg>916</bldg>, <*>management</*>
4. <history>Chile</history>, <*>Dr.</*>, <*>star</*>
5. <play>coriolanus</play>, <x>entlo</x>, <*>let</*>
6. <*>ng</*>, <line>phoenix</line>, <*>edu</*>, <*>submit</*>
7. <info>http</info>, <url>gov</url>, <*>asia</*>, <*>British</*>
8. <*>magic</*>, <*>fr</*>, <para>black</para>, <*>saint</*>, <*>knight</*>
9. <*>Paris</*>, <*>UK</*>, <*>tube</*>, <*>German</*>, <*>action</*>
10. <*>1995</*>, <*>engine</*>, <*>computer science</*>, <*>course</*>, <*>university</*>

Appendix II: Sample Test Queries for Effectiveness of Low Level Features on Standard Rankers.

(a) The query set: $Q_{academic}$

1. <*>mary</*>
2. <title>*</title>, <year>19</year>
3. <*>Univ</*>, <name>US</name>
4. <course>Geometry</course>, <year>02</year>, <*>Lam</*>
5. <bldg>g</bldg>, <credit>3</credit>,<re>*</re>, <page>9</page>, <hour>4</hour>
6. <page>30</page>, <issue>2</issue>, <title>*</title>, <*>math</*>
7. <date>July</date>, <*>science</*>, <*>art</*>, <title>human</title>, <author>Jack</author>
8. <author>ng</author>, <room>10</room>, <*>basic</*>, <course>*</course>, <date>*</date>
9. <*>asia</*>, <editor>*</editor>, <title>query</title>, <section>*</section>, <year>19</year>
10. <title>*</title>, <author>*</author>, <year>*</year>, <issue>*</issue>, <note>*</note>

(b) The query set: $Q_{liteature}$

1. <*>Rowland</*>
2. <speech>duke</speech>,<line>86</line>
3. <grpdescr>ero</grpdescr>, <act>His brother</act>, <line>senior</line>
4. <act>*</act>, <speech>*</speech>, <persona>*</persona>
5. <*>crown</*>, <*>my power</*>, <*>mighty power</*>
6. <speech>false reports</speech>, <speech>wound the world</speech>, <*>warkworth</*>, <psersona>enobarbus</persona>
7. <*>chorus</*>, <prologue>prologue</prologue>, <title>*</title>, <stagedir>enter</stagedir>, <*>hampton</*>
8. <act>sounds confused</act>, <line>*</line>, <*>king at hampton</*>, <speech>*</speech>, <*>royality</*>
9. <speech>false reports</speech>, <act>sounds confused</act>, <*>rumour</*>, <*>Ham</*>, <prologue>prologue</prologue>
10. <*>Ham</*>, <*>King</*>, <*>castle</*>, <*>chorus</*>, <*>rumour</*>

(c) The query set: $Q_{scientific}$

1. <dat>10</dat>
2. <*>999</*>
3. <id>4</id>, <l_>5</l_>
4. <gen>A</gen>, <tran>*</tran>
5. <x>o</x>, <rm>8</rm>,<fix>r</fix>
6. <history>Chile</history>, <*>Dr.</*>, <*>star</*>
7. <trans>*</trans>, <dat>2000</dat>, <swiss>*</swiss>, <*>gg</*>
8. <x-1>*</x-1>, <hash>5</hash>, <_quotes_>*</_quotes_>, <dollar>32</dollar>
9. <*>99</*>, <*>ocean</*>, <*>==</*>, <*>1yh07</*>, <*>2004</*>

10. <river>*</river>, <mountain>*</mountain>, <desert>*</desert>,<island>*</island>,<lake>*</lake>

(d) The query set: $Q_{commerical}$

1. <cpu>900</cpu>
2. <cpu>800</cpu>, <*>20GB</*>
3. <desc>dual</desc>, <bidder>Cleveland</bidder>
4. <bid>san francisco</bid><seller>WG3</seller>
5. <info>np</info>, <listing>Pavilion</listing>, <none>*</none>
6. <name>Los</name>, <num>3</num>, <closed>Nov</closed>
7. <description>Dual processing</description>, <cpu>900</cpu>, <bid>$2450</bid>, <bidder>Cleveland</bidder>
8. <mem>128</mem>, <cpu>800</cpu>,<brand>Hewlett</brand>, <name>Los</name>, <amount>2000</amount>
9. <mem>*</mem>, <cpu>*</cpu>,<brand>*</brand>, <name>*</name>, <amount>*</amount>
10. <*>128</*>, <*>30GB</*>, <*>800</*>, <*>Clara</*>, <*>Nov</*>

(e) The query set: $Q_{mixdata}$ (This is a subset of Q_{Mix} in Appendix I(a))

1. <name>xml</name>
2. <para>17</para>, <time>09</time>
3. <author>*</author>, <n>*</n>
4. <descr>cost</descr>, <para>database</para>, <author>olive</author>
5. <*>Paris</*>, <*>UK</*>, <*>tube</*>
6. <play>ham</play>, <s>4</s>,<pro>w</pro>,<mem>8</mem>
7. <title>search</title>, <year>9</year>, <nn>*</nn>, <sen>*</sen>
8. <limit>0</limit>, <time>09</time>, <start>09</start>, <end>12</end>
9. <*>sea</*>, <name>baltic</name>, <*>carri</*>, <entry>disney</entry>
10. <speech>duke</speech>,<line>86</line>,<pop>10</pop>,<instruct>ge</instruct>,<limit>0</limit>

Appendix III: Sample Test Queries converted from INEX 2006 Datasets.

Topic 289: <*>emperor</*>, <*>Napoleon I</*>, <*>Polish</*>

Topic 290: <article>genetic algorithm</article>

Topic 291: <article>Olympian</article>, <article>god</article>, <article>goddess</article>, <figure>Olympian</figure>, <figure> figure, god</figure>, <figure> figure, goddess</figure>

Topic 292: <article>Renaissance</article>, <article>painting</article>, <article>Italian</article>, <figure>Flemish</figure>, <figure>Renaissance</figure>, <figure>painting</figure>, <figure>Italian</figure>, <figure>Flemish</figure>

Topic 293: <article>wifi</article>, <section> wifi</section>, <section>security</section>, <section>encryption</section>

Topic 294: <article>user interface</article>, <section>design</section>, <section>usability</section>, <section>guidelines</section>

Topic 295: <article>software</article>, <section>intellectual property</section>, <section>patent license</section>

Topic 296: <*>Borussia Dortmund</*>, <*>European Championship Intercontinental Cup</*>

Topic 297: <article>cool jazz</article>, <article>west coast</article>, <section>musician</section>

Topic 298: <article>George Orwell</article>, <article>Eric Arthur Blair</article>, <section>George Orwell books</section>, <section>George Orwell essays</section>, <section>1984</section>, <section>Animal Farm</section>