Encoding XML in Vector Spaces

Vinay Kakade[1,⋆] and Prabhakar Raghavan[2]

[1] A9.com, Inc., USA
vkakade@a9.com
[2] Verity, Inc., USA
pragh@verity.com

Abstract. We develop a framework for representing XML documents and queries in vector spaces and build indexes for processing text-centric semi-structured queries that support a proximity measure between XML documents. The idea of using vector spaces for XML retrieval is not new. In this paper we (i) unify prior approaches into a single framework; (ii) develop techniques to eliminate special purpose auxiliary computations (outside the vector space) used previously; (iii) give experimental evidence on benchmark queries that our approach is competitive in its retrieval quality and (iv) as an immediate consequence of the framework, are able to classify and cluster XML documents.

1 Overview

1.1 Background and Motivation

We begin with three motivating examples. Consider a product search across multiple heterogeneous catalogs: *find red sweters* [sic] *and return their IDs ranked by price*; we seek matches even if a catalog entry uses *scarlet* instead of red and *pullover* instead of sweater. Text retrieval engines handle thesauri (e.g., for colors), stemming and misspelling, but cannot return specific elements within an XML document that best match the query.

Our second example comes from clustering semi-structured auto service records from multiple dealerships. There is value in discovering a cluster in which the free text contains words such as *blowout* and *rollover*, the `Make` field contains *Ford* or *Mercury* while the (child) `Model` node respectively contains *Explorer* or *Mountaineer*, while the `Parts Replaced` field includes terms such as *Firestone* and *tyre* – clearly this cannot be addressed by standard text clustering.

Our final example is of classification: consider an organization processing job applications. As each resume[3] comes in, we wish to route it to the job requisition best suited for that resume. For instance, computer science students take a wide variety of courses but few of the specialties survive into their work experience;

⋆ Work done while the author was a graduate student at Stanford University, USA.
[3] We cannot expect these resumes to conform to any single DTD.

D.E. Losada and J.M. Fernández-Luna (Eds.): ECIR 2005, LNCS 3408, pp. 96–111, 2005.

thus the word *networking* under the element `Work Experience` is likely to be a stronger feature than under `Education`, for resume routing.

Text retrieval systems use proximity metrics between documents while databases *select* using rigid range criteria; can we combine these ideas to provide semi-rigid proximity measures between semi-structured documents?

1.2 Main Contributions and Guided Tour

(1)We develop a framework for vector space XML indexing (Section 2) through the notion of *tree filters*. The choice of these filters governs the index size as well as its retrieval effectiveness. We measure index sizes for a class of tree filters derived from paths in a document (Section 2.2). (2) We benchmark our indexes on INEX *Content Only* queries (Section 3)[4]. We thus show that structure encoded in the vector space helps retrieval quality (Section 3.5), but at the expense of significantly larger indexes (Section 2.2). (3) We introduce *randomized indexes* (Section 2.2). Vector space encodings for XML are challenging in the absence of reliable DTD's. Whereas previous work handled this through additional calculations outside the vector space, randomized indexing lets us preserve the vector space framework. (4) We apply our framework to the classification and clustering of XML documents (Section 4) using standard vector space algorithms.

1.3 Some Technical Underpinnings

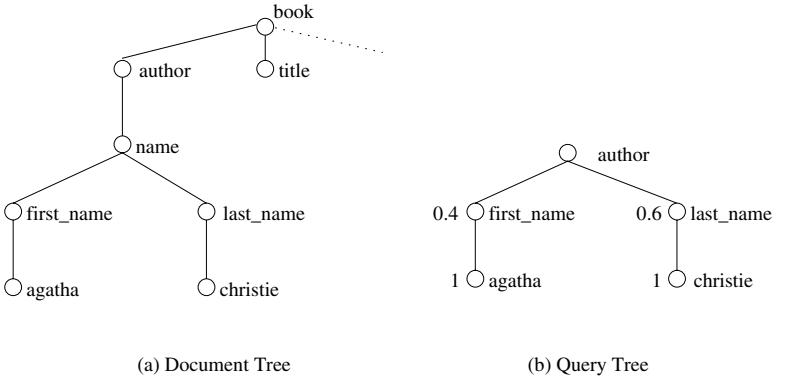
Vector spaces: The vector space paradigm has been a standard in text retrieval [28]. The research community has responded with a slew of techniques for improved vector space retrieval such as dimensionality reduction. The vector space paradigm gives us the full power of linear algebra and geometry. Given this background of effective and efficient vector space retrieval, a natural question arises: to what extent can XML retrieval exploit vector spaces?

Content-Centric Queries: We begin with an example:

Example 1. Consider a search for books whose title includes *mystery* and author includes *Agatha* and *Christie*. We should get results from different catalogs, one of which encodes books with the `author` element further split into `first_name` and `last_name` sub-elements, while the other does not. In Figure 1 we seek a partial match even though the path from `author` to each leaf is not strictly a match in the document tree.

A user with a semi-structured information need cannot be expected to conform to a rigid schema or query syntax for two reasons: (1) As argued in [22], end users (unlike applications) avoid detailed structure specification in their queries. (2) The majority of public XML documents have no DTD and only a minuscule

[4] The INEX data was made available to us by Tarragon Consulting Corp., USA as consultants to Tarragon and in accordance with all the Terms and Conditions of Tarragon's INEX data handling agreement.



(a) Document Tree (b) Query Tree

Fig. 1. Tree views of a document and a query

fraction have XML schema [24]. We view an XML document as a tree whose leaves are terms in the lexicon[5] (Figure 1).

Definition 1. *A query tree is a rooted tree in which each internal node is an element and each leaf is a term in the lexicon. Nodes of the query tree may have positive real weights associated with them, to assign a relative weight to the different elements in the query.*

This is the query abstraction used in much previous research on INEX queries [4, 22, 30]. In Figure 1, the query seeks an element with a weight of 0.6 on the match in the `last_name` element and a weight of 0.4 in the `first_name` element. Query processing assigns to a (query, document component) pair a real-valued score in $[0, 1]$. The system may return a `book`, or another element that appears to match the query (that could be a descendant or ancestor of a `book` element). In INEX the onus is on the engine to return the matching document component at the right level of specificity. Accordingly, each result for each INEX query is evaluated not only for the relevance of the match but also for the specificity (was the element returned too specific, too general or just right for the query).

1.4 Related Prior Work

Schlieder and Meuss [30] were among the earliest to adopt a vector space model for XML retrieval. The experimental results in [30] are modest in scale (22 documents) and there is no report of retrieval quality. The JuruXml engine [4, 5, 22] is perhaps the furthest developed vector space XML engine; our framework generalizes their work. They supplement their vector space with post-processing to handle cases such as the subpath match in Figure 1. Besides slowing down retrieval this makes the similarity computation unwieldy for classification/clustering. More significantly, post-processing robs us of a crucial tool:

[5] Henceforth *lexicon terms*. Our indexes will in general contain terms/axes that need not be terms in the lexicon.

linear algebra. This means that we lose access to linear algebraic techniques such as support vector machines and latent semantic indexing.

Grabs and Schek [14] index only certain *basic* elements into a vector space; the remainder are materialized depending on the query. The XIRQL language and HyREX engine [10, 12] represent the most substantial efforts at providing capabilities for XML motivated by information retrieval, such as relevance ranking and fuzzy matching. A drawback of XIRQL is that it requires a DTD/Schema. XQuery [6] is the W3C's draft specification for a query language for XML. While still evolving, it has favored a data-centric view focusing on highly structured queries. More recently TexQuery [2] addresses this gap by proposing a framework for adding scoring primitives based on a data model called a *fullmatch*. XRANK [16] extends link analysis to a method they call ElemRank, for hyperlinked XML corpora. XRANK works for keyword search rather than for the more general tree queries. The tutorial by Amer-Yahia et al. [1] provides a comprehensive review of XML query languages.

Doucet and Ahonen-Myka [8] initiate a study of XML document clustering. Treating text terms and element tags as separate feature sets in the INEX collection (Section 3.1), they run clustering algorithms and compare the results to a known partition of the documents.

2 The General Framework

We next argue that prior work relating XML to vector spaces falls within a framework in which four components specify an index.

1. **Index units** $\mathcal{IU}$: Which document elements are indexable as vectors? In a vector space text retrieval system, the index represents each document as a vector; what are the corresponding vectors here?
2. **Index terms** $\mathcal{IT}$: What are the axes of the vector space? In a text retrieval system, each lexicon term (possibly after stemming) becomes an axis.
3. **Retrievable units** $\mathcal{RU}$: What nodes in the documents can be returned as answers in a results list?
4. **Composition function** $\mathcal{CF}$: For a document d, let $v \in \mathcal{IU}$ be an index unit and $t \in \mathcal{IT}$ an index term. What is the weight of t in v? A composition function $\mathcal{CF}$ maps the triplet (d, v, t) to a non-negative real weight.

Note that composition functions capture two natural forms of weighting considered in prior literature. First, the notion of *inverse document frequency* (IDF) from text retrieval has been used by virtually all previous vector space formulations. Second, Fuhr *et al.* [10, 12] suggest a positive real *downgrade factor* $\gamma < 1$ as follows. For $v \in \mathcal{IU}$ and $t \in \mathcal{IT}$, let $\mathcal{CF}(v, t)$ be the weight of t in v. Then for $h > 0$, an index unit $a_h(v)$ that is the ancestor at height h above v, $\mathcal{CF}(a_h(v), t) = \gamma^h$. Intuitively, a lexicon term contributes a small but non-zero index entry even for an ancestor far above.

Schlieder and Meuss [30] use all nodes of each document for $\mathcal{IU}$ and $\mathcal{RU}$; they use all subtrees of each document for $\mathcal{IT}$. The elegance of this: the query-to-document score computation reduces to a form of tree pattern matching [18].

This has the disadvantage that the (possibly exponentially many) index terms cannot be pre-compiled into an index structure. This necessitates query-time index materialization that cannot exploit the many preprocessing techniques available in multidimensional retrieval.

While JuruXML [22] similarly uses all nodes as $\mathcal{IU}$ and $\mathcal{RU}$, it uses all root-to-leaf paths as $\mathcal{IT}$; this ensures a more tractable index size. Two additional ideas in JuruXML depart from a vector space formulation: (1) Dynamic programming for longest-common subsequence matching so that we have a match on the example in Figure 1. We invoke randomization in Section 2.2 to reduce this match problem to vector space retrieval. (2) A post-filtering step to enforce hard query constraints (+ and - operators to include/exclude specific content).

2.1 Tree Filters

In our indexes, $\mathcal{IU}$ and $\mathcal{RU}$ are as in [22, 30]: all nodes of all document trees are indexed and retrievable. For $\mathcal{IT}$ we introduce a notion of a *tree filters:* a graph property $\mathcal{P}$ that selects a subset of all subtrees of a document that satisfy $\mathcal{P}$. For instance [30] uses no tree filter at all, thus allowing all possible subtrees as index terms. In contrast JuruXML [22] uses $\mathcal{P}$ to select root-to-leaf paths. As another example, we could use a tree filter that selects all triplets of nodes (as vector space axes), one of which is the parent of the other two.

For query processing, the query tree is likewise expressed as a vector in the space of filtered index terms, normalized and scored using cosine similarity as in classic information retrieval. The class of tree filters used in a particular implementation determines (a) index space and retrieval time and (b) the quality of retrieved results. Below we study a particular class of simple tree filters.

2.2 ℓ-Path Filters and $VeXML_{\gamma,\ell}$

For a non-negative integer ℓ, consider all paths of exactly ℓ nodes of the document tree D *not* including the lexicon term nodes. Our index terms $\mathcal{IT}$ are as follows: to each such path $v_1, \ldots, v_\ell$, append each lexicon term t that appears in any sub-tree of D rooted at v_ℓ. Thus we would have one index term (which we call a *qualified term*) of the form $v_1, \ldots, v_\ell, t$ for each t and each path of the form $v_1, \ldots, v_\ell$. The notion of the appropriate set of document components for computing *inverse document frequencies (IDF)* is discussed in the prior work on vector space XML retrieval [4, 22, 30]; their ideas can easily be folded into our indexes so we do not discuss IDF further.

The case $\ell = 0$ and $\gamma = 1$ corresponds to using the lexicon terms as $\mathcal{IT}$. For convenience, we use $\ell = \infty$ to denote the case when $\mathcal{IT}$ consists of all root-to-leaf paths [22]. Below, we report empirical findings for index size as a function of ℓ, on the INEX 2002 corpus. For a real $\gamma \in [0, 1]$, denote by $VeXML_{\gamma,\ell}$ an index in our framework with the ℓ-path filter for selecting $\mathcal{IT}$, and downgrade factor γ [10, 12] in $\mathcal{CF}$. An algorithm to build $VeXML_{\gamma,\ell}$ is given in Algorithm 1. In this algorithm, $P = v_1, \ldots, v_k$, v is a root-to-leaf path from the root of document D to v; tag_j is the tag of the node v_j, $1 \leq j \leq k$.

Algorithm 1 Construction of $VeXML_{\gamma,\ell}$

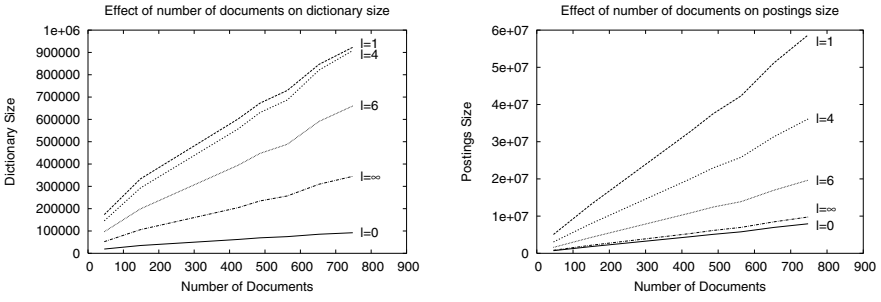
```
for each text node v in D do
  m = min(k, ℓ)
  if m = 0 then numberOfTerms = k
  else numberOfTerms = k − m + 1
  endif
  for each lexicon term t in v do
    for i = 1 to numberOfTerms in steps of 1 do
      if m = 0 then
        qualified_term = {t}
      else
        qualified_term = {tag_i, ..., tag_{i+m−1}, t}
      end if
      L = (possibly empty) postings list of qualified_term in VeXML_{γ,ℓ}
      for j = k to 1 in steps of −1 do
        weight = γ^{k−j}
        if ∃w such that ⟨v_j, w⟩ϵL then
          w = w + weight
        else
          L = L ∪ ⟨v_j, weight⟩
        end if
      end for
    end for
  end for
end for
Normalize VeXML_{γ,ℓ} so that each vector in it is a unit vector
```

Index Sizes: We study two size metrics for our $VeXML_{\gamma,\ell}$ indexes: *dictionary size* and *postings size*. The former, $|\mathcal{IT}|$, corresponds to the dimensionality of the vector space; this in turn affects the cost of elementary operations such as the inner product of two vectors. Postings size on the other hand depends on $|\mathcal{IU}|$. We study these as a function of the number of documents in the input.

The results are shown in Figures 2(a) and 2(b); for clarity we only show the representative cases $\ell = 0, 1, 4, 6, \infty$. Thus, we have a roughly linear increase in dictionary size, for all ℓ. (Note that the parameter γ does not affect the *number* of dictionary and postings entries, only the associated weights.)

A $VeXML_{\gamma,\ell}$ index does not address an important requirement for unreliable DTD's: subpath matching between query and document subtrees. In [22] this is addressed by computing a score based on substring matching *outside* of the vector space. To avoid such "outside" computation, we provide a randomized solution that approximately solves this substring matching problem.

Randomized Index Construction: We augment the deterministically constructed $VeXML_{\gamma,\ell}$ index with index terms of path length L, randomly generated from the documents. We give an intuitive description for the case $L = 2$ in the running text and a precise description in Algorithm 2. Let $v_1, \ldots, v_k, v$ be a root-to-leaf path (v being a lexicon term) in the document being indexed. For i, j



(a) Dictionary size vs. number of documents

(b) Postings size vs. number of documents

Fig. 2. Effect of number of documents on the dictionary size and postings size

Algorithm 2 Construction of randomized index

Let $VeXML_{\gamma,\ell}$ be constructed for D as per Algorithm 1.
for each root-to-leaf path $v_1, v_2, \ldots, v_k, v$ in D **do**
  Randomly select subsequences of length L of $v_1, \ldots, v_k$
  for each lexicon term t in v **do**
    for each randomly selected subsequence $s_1, \ldots, s_L$ of $v_1, \ldots, v_k$ **do**
      Let $tag_1, \ldots, tag_L$ be the corresponding tags.
      Add a qualified term $qualified\_term = \{tag_1, \ldots, tag_L, t\}$ to $VeXML_{\gamma,\ell}$, if it is not already there. The postings list of $qualified\_term$ is as in Algorithm 1.
    end for
  end for
end for

chosen randomly such that $1 \leq i < j \leq k$, we add index terms of the form $v_i v_j w$ for all lexicon terms w in v. The simplest such random choice would be uniformly from all $1 \leq i < j \leq k$; we could in fact weigh the distribution towards small values of i, j, to capture an effect suggested in [22]: matching structure close to the root is more important than structure deeper in the tree. This is repeated with independent random choices i, j.

Query Processing: Given a query tree we generate (in addition to the "deterministic" index terms corresponding to $VeXML_{\gamma,\ell}$) index terms for random ordered L-sequences of nodes in root-to-leaf paths. We then compute cosine similarity in this augmented vector space that includes $VeXML_{\gamma,\ell}$ together with random index terms. The net similarity score is the sum of two components: (a) an exact path match score from the axes of $VeXML_{\gamma,\ell}$ index as in Section 2.2 and (b) a subsequence match score resulting from the random sample.

Notes: In this method the index has random *terms*. Consequently, there is no absolute guarantee of identifying a specific subsequence match using our scheme, only a likelihood that increases as more randomized index terms are used. This

is defensible on several grounds: (1) our analysis below suggests that the scheme should perform reasonably well at modest index size increase, while eliminating the need for subsequence matching outside the vector space. (2) The entire vector space approach is predicated on several layers of approximation – how well the user's information need is expressed as a vector, how document semantics are expressed as a vector in a feature space, how well cosine similarity approximates the end-user's perception of quality, etc. Over a large sample of documents and queries, a well-designed vector space does well on many queries but could be mediocre on some. Our randomized approach is similar: document/query representations and match criteria are still approximate and (over an ensemble of documents and queries) will do well in the sense of the *expectation* of the match score. (3) Our method will never falsely assign a positive score when there is no subsequence match.

Analysis of Randomized Indexing. We sketch an analysis for $L = 2$; it can easily be extended for longer paths. Let d denote the length of a document path and q the length of a query path. Consider the case where the query and document path have a common subsequence of m elements. The probability that a sampled pair from the document and a sampled pair from the query are both in the common subsequence is $\binom{m}{2}/\binom{d}{2} \times \binom{m}{2}/\binom{q}{2}$. Consider such pair-sampling in conjunction with a base index $VeXML_{\gamma,2}$. Letting λ denote the number of lexicon terms in the document path, the number of index terms in $VeXML_{\gamma,2}$ from this document path is $\leq \lambda d$. A single sample in the randomized index will result in $\leq \lambda$ additional index terms. With S sample pairs, the increase in index size is $\leq \lambda S$. The probability of failing to detect a match is

$$\left[1 - \binom{m}{2}^2 / \binom{d}{2}\binom{q}{2}\right]^S \approx e^{-S\binom{m}{2}^2/\binom{d}{2}\binom{q}{2}}.$$

For $S \approx \Theta(d)$, we have on the one hand only a constant factor increase in index size over the underlying $VeXML_{\gamma,2}$ index, but a failure probability $\approx e^{-\binom{m}{2}^2/d\binom{q}{2}}$ that diminishes rapidly as the length m of the matching subsequence increases — as desired. Note that in practice q and m are likely to be very small (probably no more than 3 or 4) while d may be a little larger; the average for the INEX corpus is about 7. By suitably weighting the contribution of these randomized index terms to those from the underlying $VeXML_{\gamma,2}$ index, we can balance the contribution of subsequence matches to more strict path matches; this would depend on the application and its reliability of DTD's.

3 INEX Content-Only Queries

What is the tradeoff between index complexity and the quality of results retrieved? In Section 2 we examined the impact of ℓ on index size. Here we study the the impact of ℓ on the quality of the results. We begin (Section 3.2) by explaining our query formulation methodology. In Section 3.3 we review the INEX

2002 methodology for going from query results (furnished by an engine) to scores and precision-recall curves. Next (Section 3.4) we note some issues in applying this methodology to any engine (such as ours) that did not participate in INEX 2002, and our remedies. Finally in Section 3.5 we detail our results for varying γ and for $\ell = 0, 1$. Encouragingly, even for these cases we find that our quality is competitive; this suggests further experiments with $\ell > 1$.

3.1 INEX 2002 and Our Test Suite

The INEX 2002 corpus consisted of approximately 12000 articles from 12 magazines and 6 transactions published by the IEEE, for a total of approximately 500MB of data. On average a document contained over 1500 XML nodes at an average depth of nearly 7. The retrieval benchmark consists of 60 retrieval tasks, with 30 each of so-called *content-only* (CO) and *content-and-structure* (CAS) queries (see Figure 3 for an example). For each query, an engine had to return a ranked list of document components, each of which was then assessed by the INEX participants manually under two independent criteria: *relevance* and *coverage*. Based on the pair of scores assigned to each document component retrieved by an engine, the engine was assigned accuracy scores and precision-recall curves using ideas described in [13].

3.2 CO Topics Translation Methodology

The CO topics translation methodology we used is a slightly modified version of the one suggested by [22]. We apply the following translation rules for automatically constructing queries for $\ell = 0$:

- If there is only one word in the $\langle cw \rangle$ tag, we add it to the query with a weight of 1.0, along with all the terms in the $\langle Keywords \rangle$ tag with a weight of 0.3.
- If there are only two words in the $\langle cw \rangle$ tag, we add them to the query as a phrase with a weight of 1.0, along with all the terms in the $\langle Keywords \rangle$ tag with a weight of 0.3.
- If there are more than two words in the $\langle cw \rangle$ tag, we add them to the query with a weight of 0.3, and ignore the words in the $\langle Keywords \rangle$ tag (as they likely add noise).

For example, the query for $\ell = 0$ corresponding to Figure 3 is **1.0 "computational biology" 0.3 bioinformatics 0.3 genome 0.3 genomics 0.3 proteomics 0.3 sequencing 0.3 "protein folding"**.

The queries corresponding to $\ell = 1$ are constructed from those for $\ell = 0$. For each term t with weight w in a query for $\ell = 0$, we add three terms *bdy/t*, *fm/t*, *bm/t* with weight w to the query for $\ell = 1$. The reason is that most of the answers to the INEX queries lie in the subtrees of $\langle bdy \rangle$, $\langle fm \rangle$ and $\langle bm \rangle$ nodes.

3.3 INEX Evaluation Methodology

The reader should consult [13] for a detailed description of the INEX process; here we touch upon the salient points. We focus on the 30 content-focused in-

```
<INEX-Topic topic-id="31" query-type="CO" ct-no="003">
    <Title>
      <cw>computational biology</cw>
    </Title>
    <Keywords>
      computational biology, bioinformatics, genome, genomics, proteomics,
      sequencing, protein folding
    </Keywords>
    <Description>
      Challenges that arise, and approaches being explored, in the
      interdisciplinary field of computational biology.
    </Description>
    <Narrative>
      To be relevant, a document/component must either talk in general
      terms about the opportunities at the intersection of computer
      science and biology, or describe a particular problem and the ways
      it is being attacked.
    </Narrative>
</INEX-Topic>
```

Fig. 3. INEX Topic 31: Computational Biology

formation needs in the benchmark, referred to in INEX 2002 as the "CO query suite". Figure 3 gives an example, with only the relevant tags included. As in the TREC benchmark (http://trec.nist.gov/), each topic description is then turned into a query by the participant team, for its query engine. Next, the engine retrieves the top 100 results for each query, where a result is a document component from the collection (e.g., the abstract of a paper).

All components retrieved by any engine for a query are then put in a results pool; the typical such pool has about 2000 document components from 1000 articles [13]. Each result is evaluated by two criteria — its *relevance* and its *coverage*. The latter is motivated by the fact that an XML engine may retrieve a document component at any level — e.g., a whole paper, its abstract, a section within it or perhaps a definition. The evaluators assessed the relevance and coverage of each result, to determine whether it was too broad (say a whole book when a definition was sought) or too narrow. Relevance was assessed on a scale from Irrelevant (scoring 0) to Highly Relevant (scoring 3). Coverage was assessed on a scale with four levels: No Coverage (N: the query topic does not match anything in the document component retrieved), Too Large (L: the topic is only a minor theme of the component retrieved), Too Small (S: the component is too small to provide the information required) or Exact (E). At this point, every result returned by each engine has a pair of ratings from $\{0, 1, 2, 3\} \times \{N, S, L, E\}$ (although clearly some of these combinations would never arise).

3.4 Adapting the INEX Evaluations

To adapt the INEX 2002 assessments we first review the manner in which INEX combined the relevance/coverage assessments into scores. Define two f values

$$f_{strict}(rel, cov) = \begin{cases} 1 \text{ if } rel = 3 \text{ and } cov = E \\ 0 \text{ otherwise} \end{cases}$$

and

$$f_{generalised}(rel, cov) = \begin{cases} 1.00 \text{ if } rel, cov = 3E \\ 0.75 \text{ if } rel, cov \in \{2E, 3L\} \\ 0.50 \text{ if } rel, cov \in \{1E, 2L, 2S\} \\ 0.25 \text{ if } rel, cov \in \{1S, 1L\} \\ 0.00 \text{ if } rel, cov = 0N. \end{cases}$$

These f values allow us to combine the pair of assessments for a result into a single number in two ways — referred to respectively in INEX as *strict* and *generalised* quantization.

There is a difficulty in adopting this methodology: our engine could return some document components for which no assessments are available from the INEX pool. It is then impossible to average, over all query topics, our f value as a function of rank. Further, this makes it impossible for us to generate precision-recall curves as the INEX participants were able to. We circumvent this as follows: for each rank $r \in \{1, 2, \ldots, 100\}$ we average our f value over all results that we report at rank r, on any query, *for which INEX assessments are available.* Of the 30 INEX CO queries, six had no assessments at all; our results below are on the remaining 24 queries.

3.5 Results Quality for $VeXML_{\gamma,\ell}$

We give here the results for $VeXML_{\gamma,0}$ and $VeXML_{\gamma,1}$ for varying γ. Figure 4 shows the average f values for both strict and generalized quantization. For both, the best value of γ is about 0.9, for both $\ell = 0$ and $\ell = 1$. As noted above we could not generate precision-recall curves (the only published metrics for the INEX participants). However, we were able to obtain the actual retrieved document components from Tarragon, a team that was ranked 10th (out of 49) in strict quantization and 17th in generalized quantization for CO queries. We

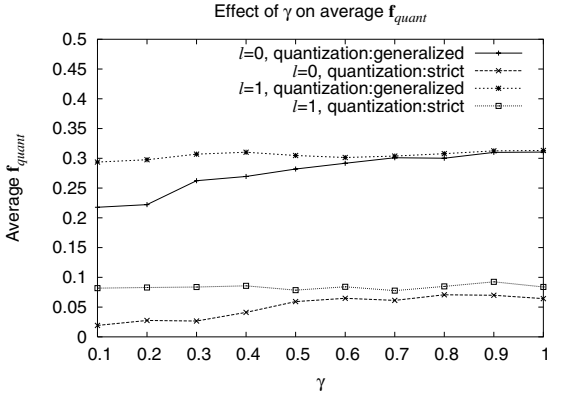


Fig. 4. Average f vs. γ

were thus able to average Tarragon's f values; they had an average of 0.0636 for f_{strict} and 0.243 for $f_{generalised}$. We clearly compare favorably on generalized quantization as well as on strict quantization. More interestingly, our f values are consistently higher for $\ell = 1$ than for $\ell = 0$, establishing that encoding XML structure in the vector space actually yields better retrieval quality.

4 Classification and Clustering

4.1 Classification

Because of our pure vector space approach, we can directly invoke any classification method that uses vector spaces. We demonstrate this with two classification methods – NN and CENTROID. We only classify the root of each document, for varying values of ℓ and γ. We ran our experiments on two datasets: INEX, and the CSLOG dataset [33] used in earlier work on XML classification.

For the INEX dataset we used as training documents articles four journals from the years 1996-98: IEEE Annals of the History of Computing, IEEE Computer Graphics and Applications, IEEE Computational Science & Engineering and IEEE Design & Test of Computers. As test documents, we used articles from the years 1999, 2000 and 2001 from the same four journals. NN and CENTROID were used to predict, for each test article, which journal it came from.

The CSLOG dataset contains documents that describe log reports at the CS department website of the Rensselaer Polytechnic Institute. Each document is in the Log Markup Language (LOGML) [26], which expresses the contents of a log file in XML, by modelling each user session as a graph. Each user session is given one of the two class labels: *edu* for users visiting the CS department from the "edu" or "ac" domains, and *other* for users from all other domains. We used the log reports of the first week as our training documents and used them to predict the class of each of the second week logs.Table 1 shows the number of features (i.e., $\mathcal{IT}$:) in our vector spaces for both these datasets. The column "Training features" shows the number of features in the training set, while the column "Total features" shows the number of features in the training as well as test sets.

Classification results for varying values of ℓ and γ are given in Table 2. They suggest that structure helps in classification for the CSLOG dataset, but does not help for the INEX dataset. In fact for $\ell = \infty$, the classifier fails to classify

Table 1. The number of features

ℓ	INEX		CSLOG	
	Training features	Total features	Training features	Total features
0	112434	189994	54953	71173
1	941264	1542911	219134	282537
2	910469	1608216	165490	213071
∞	371935	638416	73387	97986

Table 2. Classification accuracy (%) for the INEX and CSLOG dataset

γ	INEX Accuracy (1-NN)				INEX Accuracy (CENTROID)			
	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = \infty$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = \infty$
0.1	69.04	71.81	70.00	0	44.68	38.29	40.00	0.11
0.5	68.33	67.02	62.34	0	72.55	76.38	60.74	0.11
0.9	72.65	66.17	60.01	0	74.04	72.58	52.44	0.11
1.0	72.76	64.78	58.90	0	74.14	73.08	50.74	0.11

γ	CSLOG Accuracy (1-NN)				CSLOG Accuracy (CENTROID)			
	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = \infty$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = \infty$
0.1	96.77	96.81	97.11	98.61	96.86	97.30	96.93	97.84
0.5	78.90	78.61	78.59	79.71	71.80	71.97	71.96	75.65
0.9	77.78	77.64	77.42	77.80	72.15	73.95	73.41	73.08
1.0	77.72	77.54	77.24	77.54	73.06	74.22	73.22	72.80

the INEX documents, while it achieves near-perfect classification for CSLOG. The reason: the INEX corpus is richer in textual content than the LOGML server logs in the CSLOG dataset and likely demands more from the content for classification accuracy. So, we get good classification results for the INEX dataset for $\ell = 0$ and a high value of γ. The classifier fails to classify the documents for $\ell = \infty$, as the lexicon terms in the test collection become a lot more "qualified", and do not appear anywhere in the training collection.

On the other hand, the CSLOG dataset demands more from the structure than from the content. The class of each document in the CSLOG dataset turns out to be derived solely from the contents of the *name* attribute of the $\langle graph \rangle$ element — if the name attribute contains the token "edu" or "ac" in it, then the document is of type *edu*, otherwise it is of type *other*. A low γ effectively minimizes the contribution of all the descendants of the $\langle graph \rangle$ node; a high value of ℓ qualifies the lexicon terms occurring in the *name* attribute so that they will not match with the same lexicon terms occurring elsewhere in the document. Thus a low γ with high values of ℓ gives near-perfect classification results for the CSLOG dataset, somewhat better than the ones in [33]. Simple vector space classification thus gives near-perfect results here, without apparent need for tree mining [33].

4.2 Clustering

We evaluated our approach on the k-means clustering of XML documents. We clustered on the same subsets of the INEX and CSLOG datasets as the test documents used in classification (Section 4.1). The goal was to confirm our understanding of ℓ and γ in $VeXML_{\gamma,\ell}$. For the INEX dataset, we set the number of clusters (k) to 4, corresponding to the 4 journals. For the CSLOG dataset, we set the number of clusters to 2, corresponding to whether or not the documents are from the "edu"/"ac" domains. We compared our clusterings obtained with the ground truth using the *Variation of Information* (VI) measure [23]. The results are given in Table 3. These results confirm our intuition in Section 4.1

Table 3. k-means clustering quality (VI measure) for the INEX and CSLOG dataset

γ	INEX VI measure				CSLOG VI measure			
	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = \infty$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = \infty$
0.1	3.703	3.592	3.603	3.860	1.142	1.209	1.289	1.216
0.5	2.590	2.960	3.541	3.633	1.364	1.525	1.770	1.737
0.9	2.569	3.192	3.495	3.536	1.545	1.770	1.764	1.761
1.0	2.570	3.202	3.497	3.542	1.772	1.769	1.763	1.759

regarding the parameters γ and ℓ. A high value of γ with small ℓ gives good clustering results for INEX, while a low value of γ with a high ℓ gives good clustering results for CSLOG dataset, for the reasons outlined in Section 4.1.

5 Conclusion and Future Work

Our work validates that encoding XML in vector spaces can tap the wealth of techniques in vector space information retrieval. Several interesting directions open up as a result. First, there is the detailed empirical study of randomized indexing, as well as its potential application to other settings. Second, it would be interesting to understand the power of spectral techniques such as LSI for retrieval and support vector machines for classification. Third, our classification and clustering results show that index parameters are influenced by the nature of the XML content; what guidelines can we develop for these choices?

Acknowledgement

We thank Dr. Richard Tong of Tarragon Consulting Corp. for his suggestions and his help with the INEX dataset and the authors of [33] for their help with the CSLOG dataset.

References

1. S. Amer-Yahia, N. Koudas and D. Srivastava. Approximate matching in XML. `http://www.research.att.com/ sihem/publications/PART1.pdf`
2. S. Amer-Yahia, C. Botev and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. *WWW* 2004.
3. S. Amer-Yahia, L. V. S. Lakshmanan and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML *SIGMOD* 2004.
4. D. Carmel, N. Afraty, G. Landau, Y. Maarek and Y. Mass. An extension of the vector space model for querying XML documents via XML fragments. *XML and Information Retrieval Workshop at SIGIR*, 2002.
5. D. Carmel, Y. Maarek, M. Mandelbrod, Y. Mass, A. Soffer. Searching XML documents via XML fragments. *SIGIR* 2003.
6. D. Chamberlin, D. Florescu, J. Robie, J Siméon, M. Stefanescu. XQuery: A query language for XML. W3C Technical Report.

7. C.J. Crouch, S. Apte and H. Bapat. Using the extended vector model for XML retrieval. In [9], 95–98, 2002.
8. A. Doucet and H. Ahonen-Myka. Naive clustering of a large XML document collection. In [9], 81–88, 2002.
9. N. Fuhr, N. Gövert, G. Kazai and M. Lalmas. Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX), 2002.
10. N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Research and Development in Information Retrieval*, 172–180, 2001.
11. N. Fuhr and G. Weikum. Classification and Intelligent Search on Information in XML. *IEEE Data Engineering Bulletin* **25**(1), 2002.
12. N. Gövert, M. Abolhassani, N. Fuhr and K. Großjohann. Content-oriented XML retrieval with HyRex. In [9], 26–32, 2002.
13. N. Gövert and G. Kazai. Overview of INEX 2002. In [9], 1–17, 2002.
14. T. Grabs and H-J Schek. Generating vector spaces on-the-fly for flexible XML retrieval. Second SIGIR XML workshop, 2002.
15. D. Guillaume and F. Murtagh. Clustering of XML documents, *Computer Physics Communications*, **127**:215–227 (2000).
16. L. Guo, F. Shao, C. Botev and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. *SIGMOD* 2003.
17. Initiative for the evaluation of XML retrieval. http://qmir.dcs.qmul.ac.uk/INEX/
18. P. Kilpeläinen. Tree Matching Problems with Applications to Structured Text Databases. PhD thesis, Dept. of Computer Science, University of Helsinki, 1992.
19. G. Kazai, M. Lalmas, N. Fuhr and N. Gövert. A report on the first year of the INitiative for the Evaluation of XML Retrieval (INEX 02). *Journal of the American Society for Information Science and Technology* **54**, 2003.
20. R. Luk, H. Leong, T. Dillon, A. Chan, W. Bruce Croft, J. Allan. A survey in indexing and searching XML documents. *JASIST 53(6)* 415-437 (2002).
21. G. Kazai, S. Masood, M. Lalmas. A Study of the Assessment of Relevance for the INEX'02 Test Collection. *ECIR* 2004.
22. Y. Mass, M. Mandelbrod, E. Amitay, D. Carmel, Y. Maarek and A. Soffer. JuruXML – an XML retrieval system at INEX'02. In [9],73–80, 2002.
23. M. Meila. Comparing Clusterings. Technical Report 418, University of Washington Statistics Dept., 2002.
24. L. Mignet, D. Barbosa and P. Veltri. The XML Web: a First Study. *Proceedings of the 12th International World Wide Web Conference*, 2003. Evaluating Structural Similarity in XML Documents. *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002).*
25. N. Polyzotis, M. Garofalakis, Y. Ioannidis. Approximate XML Query Answers. *SIGMOD 2004.*
26. J. Punin, M. Krishnamoorthy, M. Zaki. LOGML: Log markup language for web usage mining. *In WEBKDD Workshop (with SIGKDD)*, 2001.
27. F. Rizzolo and A. Mendelzon. Indexing XML Data with ToXin. *Proceedings of Fourth International Workshop on the Web and Databases*, 2001.
28. G. Salton. The SMART Retrieval System – Experiments in automatic document processing. Prentice Hall Inc., Englewood Cliffs, 1971.
29. T. Schlieder. Similarity search in XML data using cost-based query transformations. *Proc. 4th WebDB*, 19–24, 2001.
30. T. Schlieder and H. Meuss. Querying and Ranking XML Documents. *Journal of the American Society for Information Science and Technology* **53**(6):489-503, 2002.

31. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang. D. DeWitt and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proc. VLDB* 1999.
32. M. Zaki. Efficiently Mining Frequent Trees in a Forest. *Proceedings of ACM KDD*, 2002.
33. M. Zaki and C. Aggarwal. XRULES: An Effective Structural Classifier for XML Data. *Proceedings of ACM KDD*, 2003.