Exploiting Structure, Annotation, and Ontological Knowledge for Automatic Classification of XML Data

Martin Theobald
Saarland University
D-66041 Saarbruecken
Germany
mtheobald@cs.uni-sb.de

Ralf Schenkel
Saarland University
D-66041 Saarbruecken
Germany
schenkel@cs.uni-sb.de

Gerhard Weikum
Saarland University
D-66041 Saarbruecken
Germany
weikum@cs.uni-sb.de

ABSTRACT

This paper investigates how to automatically classify schemaless XML data into a user-defined topic directory. The main focus is on constructing appropriate feature spaces on which a classifier operates. In addition to the usual text-based term frequency vectors, we study XML twigs and tag paths as extended features that can be combined with text term occurrences in XML elements. Moreover, we show how to leverage ontological background information, more specifically, the WordNet thesaurus, for the construction of more expressive feature spaces. For efficiency our implementation computes features incrementally and caches ontology entries. Our experiments demonstrate the improved accuracy of automatic classification based on the enhanced feature spaces.

1. INTRODUCTION

Despite the great advances on XML data management and querying, the currently prevalent XPath- or XQuery-centric approaches face severe limitations when applied to XML documents in large intranets, digital libraries, federations of scientific data repositories, and ultimately the Web. In such environments, data has much more diverse structure and annotations than in a business-data setting and there is virtually no hope for a common schema or DTD that all the data complies with. Without a schema, however, database-style querying would often produce either empty result sets, namely, when queries are overly specific; or way too many results, namely, when search predicates are overly broad, the latter being the result of the user not knowing enough about the structure and annotations of the data. This calls for applying information retrieval (IR) technology, in particular, the ranked retrieval paradigm, but at the same time adding structural search conditions to the traditional IR repertoire of keyword queries [6, 7, 14, 8].

An important IR technique is automatic classification for organizing documents into topic directories based on statistical learning techniques [10, 3, 12, 4]. Once data is labeled with topics, combinations of declarative search, browsing, and mining-style analysis is the most promising approach to find relevant information, for example, when a scientist searches for existing results on some rare and highly specific issue. This paper explores automatic classification for schemaless XML data. The anticipated benefit is a more explicit, topic-based organization of the information which in turn can be leveraged for more effective searching. The main problem that we address towards this goal is to understand which kinds of features of XML data can be used for high-accuracy classification and how these feature spaces should be managed by an XML search tool with user-acceptable responsiveness.

This paper systematically explores the design space outlined above by investigating features for XML classification that capture annotations (i.e., tag-term pairs), structure (i.e., twigs and tag paths), and ontological background information (i.e., mapping words onto word senses). Particularly challenging issues are how to deal with a very small training basis, on one hand, and how to handle documents that include multiple topics and a correspondingly heterogeneous vocabulary, on the other hand. For both issues exploiting structural and ontological features that go beyond the standard bag-of-words model (i.e., context-free terms and their frequencies) seems to be a promising direction.

2. FEATURE SPACES FOR XML DATA

Using only text terms (e.g., words, word stems, or even noun composites) and their frequencies (and other derived weighting schemes such as tf*idf measures [10]) as features for automatic classification of text documents poses inherent difficulties and often leads to unsatisfactory results because of the noise that is introduced by the idiosyncratic vocabulary and style of document authors. The key lies in a more precise document characterization for semistructured data such as XML by exploiting its structure and annotation. For XML data we postulate that tags (i.e., element names) will be chosen much more consciously and carefully than the words in the element contents and expect a certain awareness of authors for the need of meaningful and reasonably precise annotations and structuring.

So we view tags as high-quality features of XML documents. When we combine tags with text terms that appear in the corresponding element contents, we can interpret the resulting (tag, term) pairs almost as if they were *(concept, value) tuples* in the spirit of a database schema with at-

Copyright is held by the author/owner.
International Workshop on the Web and Databases (WebDB).
June 12–13, 2003, San Diego, California.

tribute names and attribute values. For example, pairs such as (programming_language, Java) or (lines_of_code, 15000) are much more informative than the mere co-occurrence of the corresponding words in a long text (e.g., describing a piece of software for an open source portal). Of course, we can go beyond simple tag-term pairs by considering entire *tag paths*, for example, a path "university/department/chair" in combination with a term "donation" in the corresponding XML element, or by considering *structural patterns* within some local context such as *twigs* of the form "homepage/ teaching ∧ homepage/research" (the latter could be very helpful in identifying homepages of university professors).

An even more far-reaching option is to map element names onto an ontological knowledge base. This way, tags such as "university" and "school" or "car" and "automobile" could be mapped to the same semantic concept, thus augmenting mere words by their *word senses*. We can generalize this by mapping words to semantically related broader concepts (hypernyms) or more narrow concepts (hyponyms), if the synonymy relationship is not sufficient for constructing strong features. And of course, we could apply such mappings not just to the element names, but also to text terms that appear in element contents.

In doing this, we encounter a number of problems that need to be solved towards a viable and high-quality XML document classifier:

- *High dimensionality:* Combining tags with terms leads to a feature space of much higher dimensionality than the usual vector spaces used for document representation in IR and classification. Consequently, it will be populated much more sparsely than a simpler term vector space, and effectively training a classifier will be more challenging.

- *Ambiguity:* When mapping words onto ontological concepts the ambiguity of natural language often prevents a unique mapping. So we need some approach for disambiguating word senses based on the context of the word occurrence.

- *Noise:* We do not believe in semantically perfect annotations or perfect ontologies; terminology, even when chosen very carefully, is naturally diverse and evolves over time. So there will always be a fair amount of noise in the enhanced feature spaces.

- *Efficiency:* The high dimensionality of the richer feature spaces and also the mapppings onto ontological concepts incur significant computational overhead; so an efficient implementation is all but straightforward.

2.1 Combining Terms and Tags

When parsing and analyzing an XML document we extract characteristic words from element contents by means of simple stopword filtering or noun recognition, and we combine these text terms with the corresponding element name; these pairs are encoded into a single feature of the form *tag$term* (e.g., "car$Passat").

We include all tags in this combined feature space, but for tractability and also noise reduction only a subset of these features is selected in the input vectors of the classifier. For each topic and its competing topics in the classification (i.e., the sibling nodes in the topic directory), we compute, from the training documents, the *Mutual Information* [1, 10] (MI, aka. relative entropy or Kullback-Leibler divergence) between the topic-specific distribution of the features and a distribution in which the features of documents are statistically independent of the topics:

$$\begin{aligned} &relevance\ of\ feature\ X_i \\ &for\ discriminating\ topic\ C_k: \\ MI(X_i, C_k) \quad &= \sum_{X \in \{X_i, \bar{X_i}\}, C \in \{C_k, \bar{C_k}\}} P[X \wedge C] log \frac{P[X \wedge C]}{P[X]P[C]} \end{aligned}$$

Sorting the features in descending order of this measure gives us a ranking in terms of the discriminative power of the features, and we select the top m features for the classifier (usually, with m being in the order of 500 up to 5000). Thus, we obtain a ranking for the complete set of tag-term pairs that occur in the training data, because different element types (i.e., with different tags) usually imply different sets of specifically characteristic terms. For example, in digital library documents terms such as "www" or "http" are more significant within elements tagged "content", "section", or "title" rather than elements tagged "address" where they may simply indicate the author's homepage URL.

2.1.1 Feature Weighting

Term-based features are usually quantified in the form of tf*idf weights [1, 10] that are proportional to the frequency of the term (tf) in a given document and to (the logarithm of) the inverse document frequency (idf) in the entire corpus (i.e., the training data for one topic in our case). So the highest weighted features are those that are frequent in one document but infrequent across the corpus. Analogously to the arguments for feature selection, we compute tf and idf statistics for tag-term pairs. The weight $w_{ij}(f_i)$ of feature f_i in document j is computed as $w_{ij}(f_i) = tf_{ij}\ idf_i$, where idf_i is the logarithm of the *inverse element frequency* of term t_i in the corpus.

This way, the *idf* part in the weight of a term is implicitly computed for each element type separately without extra effort. For example, in a digital library with full-text publications, the pair (journal_title, transaction) would have low *idf* value (because of ACM Transactions on Database Systems, etc.), whereas the more significant pair (content, transaction) would have high *idf* value (given that there have been relatively fewer papers on transaction management in the recent past).

2.2 Exploiting Structure

Using tag paths as features gives rise to some combinatorial explosion. However, we believe that this is a fairly modest growth, for the number of different tags used even in a large data collection should much smaller than the number of text terms and we expect real-life XML data to exhibit typical context patterns along tag paths rather than combining tags in an arbitrarily free manner. These characteristic patterns should help us to classify data that comes from a large number of heterogeneous sources. Nevertheless, efficiency reasons may often dictate that we limit tag-path features to path length 2, just using (parent tag, tag) pairs or (parent tag, tag, term) triples.

Twigs are a specific way to split the graph structure of XML documents into a set of small characteristic units with respect to sibling elements. Twigs are encoded in the form

"left child tag $ parent tag $ right child tag"; examples are "research$homepage$teaching", with "homepage" being the parent of the two siblings "research" and "teaching", or "author$journal_paper$author" for publications with two or more authors. The twig encoding suggests that our features are sensitive to the order of sibling elements, but we can optionally map twigs with different orders to the same dimension of the feature space, thus interpreting XML data as unordered trees if this is desired.

For tag paths and twig patterns as features we apply feature selection to the complete structural unit. MI is computed on *distinct feature sets* each for tag-term pairs and twig features. The final proportion of both kinds of features in the resulting topic-specific feature spaces is scaled by empirical consideration of the given data and depending on their degree of structural diversity.

2.3 Exploiting Ontological Knowledge

Rather than using tags and terms directly as features of an XML document, an intriguing idea is to map these into an ontological concept space. In this paper we have used WordNet [5, 11] as underlying ontology, which is a fairly comprehensive common-sense thesaurus carefully handcrafted by cognitive scientists. WordNet distinguishes between *words* as literally appearing in texts and the actual *word senses*, the concepts behind words. Often one word has multiple senses, each of which is briefly described in a sentence or two and also characterized by a set of synonyms, words with the same sense, called *synsets* in WordNet. In addition, WordNet has captured hypernym (i.e., broader sense), hyponym (i.e., more narrow sense), and holonym (i.e., part of) relationships between word senses.

In the following we describe how we map tags onto word senses of the ontology; the same procedure can be applied to map text terms, too. The resulting feature vectors only refer to word sense ids that replace the original tags or terms in the structural features. The original terms are no longer important while the structure of each feature remains unchanged. Obviously this has a potential for boosting classification if we manage to map tags with the same meaning onto the same word sense.

2.3.1 Mapping

A tag usually consists of a single word or a composite word with some special delimiters (e.g., underscore) or a Java-style use of upper and lower case to distinguish the individual words. Consider the tag word set $\{w_1, ..., w_k\}$. We look up each of the w_i in WordNet, or actually a special database constructed from WordNet's contents (see Section 3.2), and identify possible word sense $s_{i_1}, \ldots, s_{i_m}$. For example, for a tag "goal" we would find the word senses:

1. goal, end – (the state of affairs that a plan is intended to achieve and that (when achieved) terminates behavior to achieve it; "the ends justify the means")

2. goal – (a successful attempt at scoring; "the winning goal came with less than a minute left to play")

and two further senses. By looking up the synonyms of these word senses, we can construct the synsets {goal, end, content, cognitive content, mental object} and {goal, score} for the first and second meaning, respectively. For a composite tag such as "soccer_goal", we look up the senses for both "soccer" and "goal" and form the cross product of possible senses for the complete tag and represent each of these senses by the union of the corresponding two synsets. Obviously, this approach would quickly become intractable with growing number of words in a composite tag, but more than two words would seem to be extremely unusual.

Now the key question is: which of the possible senses of a tag is the right one? Our disambiguation approach is based on word statistics for some local context of both the tag that appears in some document and the candidate senses that are extracted from the ontology. For a tag t_i, we consider the full text content (including the tag name) of the corresponding element and all its subordinate elements as a *local context* $con(t_i)$ of tag t_i in document d_j. For a candidate word sense s_j, we extract synonyms, all immediate hyponyms, holonyms, and hypernyms, and optionally the hyponyms of the hypernyms (i.e., the siblings of s_j in the relationship graph). Each of these has a synset and also a short explanatory text. We form the union of the synsets and corresponding texts, thus constructing a *local context* $con(s_j)$ of sense s_j for $j \in \{1, \ldots, p\}$. As an example, the context of sense 1 of the word "goal" (see above) corresponds to the bag of words {goal, end, state, affairs, plan, intend, achieve, ..., content, cognitive content, mental object, perceived, discovered, learned, ..., aim, object, objective, target, goal, intended, attained, ...}, whereas sense 2 would be expanded into {goal, successful, attempt, scoring, winning, goal, minute, play, ..., score, act, game, sport, ...}.

The final step towards disambiguating the mapping of a tag onto a word sense is to compare the tag context $con(t_i)$ with the context of candidates $con(s_1)$ through $con(s_p)$ in terms of a similarity measure between bags of words (note that the context construction may add the same word multiple times, and this information is kept in the word bag). Our implementation uses the cosine similarity between the tf*idf vectors of $con(t_i)$ and $con(s_j)$. Finally, we map tag t_i onto that sense s_j whose context has the highest similarity to $con(t_i)$. Denote this sense as $sense(t_i)$ and the set of all senses of tags that appear in the training data as $senses_{train} = \{sense(t_i) | t_i \; appears \; in \; training \; data\}$.

2.3.2 Quantified Relationships

The feature space constituted by the mapping of tags onto word senses is appropriate when training documents and the test documents to be classified have a major overlap in their word senses (i.e., the images of their tag-to-word-sense mappings). In practice, this is not self-guaranteed even for test documents that would indeed fall into one of the trained topics. Suppose, for example, that we have used training documents with tags such as "goal", "soccer", and "shot", which are all mapped to corresponding word senses, and then a previously unseen test document contains tags such as "Champions_League", "football", and "dribbling", which correspond to a disjoint set of word senses. The classifier would have no chance to accept the test document for topic "sports"; nevertheless we would intellectually rate the test document as a good candidate for sports.

To rectify this situation, we define a similarity metric between word senses of the ontological feature space, and then map the tags of a previously unseen test document to the word senses that actually appeared in the training data and are closest to the word senses onto which the test document's tags would be mapped directly.

For defining a sense-to-sense similarity metric, we pursue a pragmatic approach that exploits term correlations in natural language usage. We consider the synsets $synset(s_1)$ and $synset(s_2)$, and estimate the statistical correlation between these two word sets in a very large text corpus. As the underlying corpus we use the Web itself, by performing a large crawl that starts with topic-specific training data (not only XML, but also HTML documents that are characteristic for the given topic). From the crawl result we compute the Dice coefficient [10]:

$$dice(s_1, s_2) = \frac{2 \cdot df(synset(s_1) \cup synset(s_2))}{df(synset(s_1)) \cdot df(synset(s_2))}$$

To avoid combinatorial explosion and the computational overhead in the critical path of the classifier, we have precomputed these similarity measures for all neighboring word senses in a graph whose edges reflect hypernym/hyponym relationships. At run-time the word sense s' that is most similar to a given word sense s is determined by finding the shortest path between the two senses in the relationship graph and averaging the similarity measures of the corresponding edges.

Putting everything together, a tag t_i in a test document d_j is first mapped to a word sense $s := map(t_i)$, then we find the closest word sense $s' := argmax_{s'}\{sim(s', s) | s' \in senses_{train}\}$ that is included in the feature space of the training data, and scale the weight of the feature containing s in document d_j by $sim(s, s')$ based on concept similarities estimated from large-scale crawling. Fig. 1 depicts the architecture of our method.

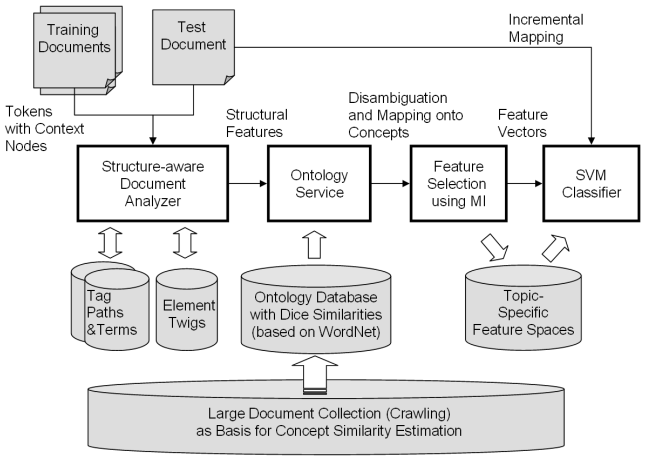

Figure 1: Exploiting Structure and Ontological Knowledge for Classification.

3. IMPLEMENTATION

3.1 Classifier

Document classification consists of a training phase for building a mathematical decision model based on intellectually preclassified documents, and a decision phase for classifying new, previously unseen documents fetched by the crawler. We chose *Support Vector Machines (SVM)* [2, 15, 9] as a leading-edge classification method. More specifically, we use the SVM-light V5.00 software, restricting ourselves to linear SVMs.

The *hierarchical multi-class* classification problem for a tree of topics is solved by training a number of binary SVMs, one for each topic in the tree. For each SVM the training documents for the given topic serve as positive samples, and we use the training data for the tree siblings as negative samples. SVM computes a maximum-margin separating hyperplane between positive and negative samples in the feature space; this hyperplane then serves as the decision function for previously unseen test documents (by computing a simple scalar product). A test document is recursively tested against all siblings of a tree level, starting with the root's children, and assigned to all topics for which the classifier yields a positive decision (or, alternatively, only to the one with the highest positive classification confidence).

3.2 Ontology Service

The internal structure of our ontology service is derived from the WordNet graph $G = (S, E_t)$ and stored in a set of database relations, i.e., a relation for the nodes of the ontology graph that yield all known synsets S, and a relation R_t for each of the supported edge types *hypernym*, *holonym*, and *hyponym* that connect these synsets nodes. The ontology graph provided by WordNet is enriched by edge-weights. For each edge type, all edges of the ontology graph are stored as triples $R_t = \{(id_s, id_t, dice(s,t)\}$, where $id_s, id_t \in E_t$, $dice(s,t) \in [0,1]$ and $t \in \{hyponym, hypernym, holonym\}$, including the ids of the source and the target synset and the quantified similarity estimated by the Dice coefficient (see Section 2.3.2).

Our implementation of the ontology service provides the following basic functions (these functions can be invoked as a Java class API, via RMI to a Java bean, or as Web Service methods using SOAP):

- `findConceptByTerm`: For a document tag or term t, find all synsets S that contain t. Since a term may have different meanings depending on the context that it occurs in (*polysemy*), this method often returns more than one synset.
- `disambiguateConcepts`: For a given tag or term t_i along with its context $con(t_i)$ in a document, find the most suitable synset from a set of synsets S by comparing the cosine between $con(t_i)$ and all synset contexts $con(s_i)$ with $s_i \in S$. This method aims to eliminate the polysemy of terms that appear in multiple synsets.
- `findConceptSimilarity`: Query the ontology graph to detect the shortest path p between two concepts s_1 and s_2 and return the average edge weight $sim_p(s_1, s_2)$ between the source and the target node.
- `findBestMatch`: This method subsequently compares the similarity $sim_p(s, s_i)$ for $s_i \in senses_{train}$ and returns the training synset that maximizes $sim_p(s, s_i)$ (more details in Section 3.3).
- `getDiceCoeff`: This method computes the Dice coefficient for two arbitrary concepts s_1, s_2 by accessing the document frequencies $df(synset(s_1))$, $df(synset(s_2))$, and $df(synset(s_1) \cup synset(s_2))$ in the initial large-scale crawl directly.

To improve performance over frequently repeated queries, all result sets and retrieved similarities are cached. To implement the similarity search between two concepts s_i and

s_j, the ontology service searches for the shortest path between s_i and s_j. With regard to performance issues, we decided to avoid a breadth search over the complete ontology graph starting at each concept, and rather exploit the DAG-like structure of WordNet to use the transitive hypernym relations only. At run-time the sense's similarity $sim_p(s_i, s_j)$ is determined by finding the nearest common hypernym h_{s_i,s_j} that minimizes the sum of the lengths of the two paths $p_1 = \{s_i, \ldots, h_{s_i,s_j}\}$ and $p_2 = \{s_j, \ldots, h_{s_i,s_j}\}$ that connect s_i and s_j via h_{s_i,s_j} and determine the average edge weight of p_1 and p_2.

$sim_p(s_i, s_j) =$

$$\frac{1}{n+m}\left(\sum_{k=1}^{n-1} dice(s_{ik}, s_{i,k+1}) + \sum_{l=1}^{m-1} dice(s_{jl}, s_{j,l+1})\right)$$

If there is more than one common hypernym that minimizes the path length, we choose the one with highest average edge similarity sim_p.

3.3 Incremental Mapping for Classification

In the *classification phase* we aim to extend the test vector by finding approximate matches between the concepts in feature space and the formerly unknown concepts of the test document. Like the training phase, the classification identifies synonyms and replaces all tags/terms with their disambiguated synset ids. 1) If there is a direct match between a test synset s and a synset in the training feature space (i.e., $s \in senses_{train}$), we are finished and put the respective concept-term-pair in the feature vector that is generated for the test document. 2) If there is no direct match between the test synset s and any synset derived from the training data (i.e., $s \notin senses_{train}$), we replace the actual test synset with its most similar match s' using the 'findBestMatch' method. Features which were removed in the training documents through feature selection are also skipped in the test documents. The weight $w_{ij}(f_i)$ of feature f_i in document j is now scaled by $sim_p(s, s')$, i.e., $w_{ij}(f_i) = sim_p(s, s')\ tf_{ij}\ idf_i$, to reflect the concept similarities of approximate matches in the feature weights.

To avoid topic drift, we limit the search for common hypernyms to a depth of 2 in the ontology graph. Concepts that are not connected by a common hypernym within this threshold are considered as dissimilar and obtain a similarity value of 0.

4. PRELIMINARY EXPERIMENTS

4.1 Setup

The following preliminary experiments where designed to shed initial light into the specific advantages and disadvantages of our approaches, using two different data sets in comparison to a standard term-based text classifier built from the element names and textual contents using standard tf*idf weights. All tests where run for two-topic directories, which is the simplest case from the classification point-of-view and directly corresponds to the binary SVM classification schema. To measure classification quality, we used the *F-measure* [1, 10] which is the harmonic mean of precision and recall, a standard quality measure in IR. Training and tests were performed on disjoint document sets.

4.2 The Internet Movie Database

The first data set consists of an XML version of the IMDB (Internet Movie Database) collection (www.imdb.org). The IMDB is available as a plain text database with structured records that contain movie titles, actors, genres, etc. We generated one XML document for each movie and automatically annotated the resulting XML elements according to the attributes of the corresponding records (e.g, title, actor, role, summary, goof, trivia, quotes, etc.). An XML tag was generated for each attribute. This resulted in a maximum nesting depth 4 (e.g., movie/casting/character/actor). To test the classifier we choose a training set with $n = 50$ movies for each of the two topics '*Action*' and '*Western*'. We used this genre label as the target of the classifier and removed the genre attribute from all test documents. The aim of this test is to study the precision and recall of tag-term pairs and twigs compared to traditional text classification. We varied the number m of features from $m = 10$ to $m = 10.000$ per topic using the MI criterion for both the pure textual and the structure-conscious classifiers. Since there are variations in tag ancestor/descendant/sibling occurrences, we dedicated 5 percent of the overall feature space dimensions to the most specific twig features, again using MI as selection criterion for the best twigs. So a large fraction of the feature space dimensions capture the best tag-term pairs, and ontology lookups were restricted to tags only. Before feature selection, the (maximum) structural feature space contained 25543 distinct tag-term pairs and twigs versus 12062 distinct text terms extracted from a total of 100 training documents.

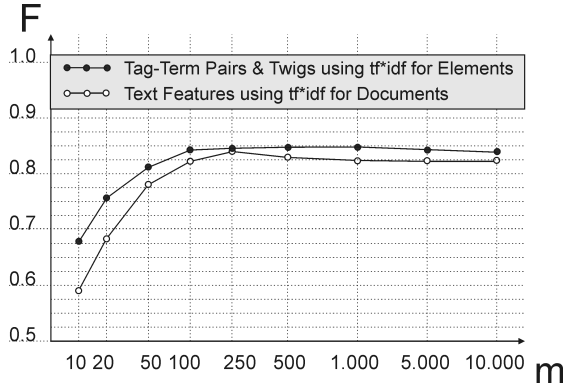

Figure 2: Classification of '*Action*' versus '*Western*' for the IMDB movie collection using very few features.

Figure 2 shows the improved accuracy (in terms of the F-measure) of tag-term pairs and twigs over mere text features for less than 250 features. For low-dimensional feature spaces the structure-aware features yield a significantly better representation of the training data than pure text feature spaces can achieve. Interestingly, the best structural features after MI turn out to be format descriptions as depicted in table 1 (with stemmed terms).

4.3 The Reuters-21578 Dataset

The second data set consists of the Reuters-21578 collection from the Reuters newswire which is part of the TREC standard benchmark collection (trec.nist.gov). The Reuters collection has a fairly restricted vocabulary (only 7783 distinct terms are contained in our largest training collection

Action	Western
release_country$usa	sound_mix$mono
video_standard$ntsc	filmed_in$white
sound_encoding$analog	filmed_in$black
disc_format$clv	writer$alphabet
master_format$film	summary$longhorn
sound_encoding$cx	summary_by$abilen
aspect_ratio$close	summary_by$le
aspect_ratio$ld	summary$ride
aspect_ratio$teletext	summary_by$adam
aspect_ratio$caption	role$henchman

Table 1: Top 10 features for '*Action*' versus '*Western*' using MI. Synset ids are replaced by original tags/terms for better readability.

with 1000 documents). The articles are authored by professional writers according to guidelines and so they are uniformly structured and hardly exhibit any annotations (merely line breaks, but no semantically valuable tagging which makes it analogous to processing simple HTML). So this experiment focused on studying the benefits of ontological features without exploiting the document structure. We applied the mapping of terms into ontological concepts only to *nouns* in the text contents for a very small set of training documents for the two Reuters topics '*Acquisition*' and '*Earnings*'. Beginning with just 1 example per topic, we stepwise increased the training basis to 500 samples per topic (starting with the longest documents and adding documents in descending order of length, i.e., their amount of terms).

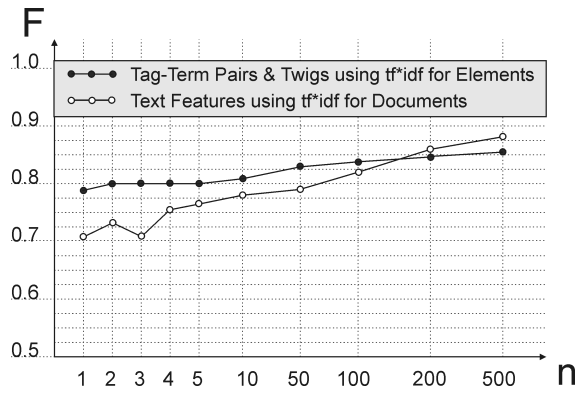

Figure 3: Classification of '*Acquisition*' versus '*Earnings*' for the Reuters-21578 collection using very few training documents.

Figure 3 shows that for less than 10 training samples, the classifier with tag-term pairs using ontology lookups on all known nouns clearly outperformed the standard text classifier. Moreover, the classifier with the ontology mapping seemed to be less sensitive to noise, whereas the pure text classifer showed some fluctuations created by noise. For more than 100 training samples we again observe a saturation effect in the text feature space; at this point both classifiers perform equally well and adding more samples does not lead to any richer feature spaces. The phenomenon that the ontology classifier is finally outperformed by the text classifier can be attributed to the fact that the disambiguation of text terms does not always work correctly. At some point, a non-negligible number of terms is mapped to wrong concepts in the ontology (as we could verify intellectually), and this results in reduced accuracy.

5. CONCLUSIONS

This paper is a first step towards understanding the potential of exploiting structure, annotation, and ontological features of XML documents for automatic classification. Our experiments are fairly preliminary, but the results indicate that the direction is worthwhile to be explored in more depth. Our current and near-future work includes more comprehensive experimental studies and making the ontological mapping (namely, IR-style word sense disambiguation) more robust. In particular, we are working on incorporating other sources of ontological knowledge into our system to go beyond the information provided by WordNet. This ongoing work is part of the BINGO! project in which we are building a next-generation focused crawler and comprehensive toolkit for automatically organizing and searching semistructured Web and intranet data [13].

6. REFERENCES

[1] R. Baeza-Yates, B. Ribeiro-Neto: Modern Information Retrieval. Addison Wesley, 1999.

[2] C.J.C. Burges: A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2(2), 1998.

[3] S. Chakrabarti, M. Berg, B. van den and Dom: Focused Crawling: A New Approach to Topic-specific Web Resource Discovery. WWW Conference, 1999.

[4] R. Duda, P. Hart, D. Stork: Pattern Classification, Wiley, 2000.

[5] C. Fellbaum: WordNet: An Electronic Lexical Database. MIT Press, 1998.

[6] N. Fuhr, K. Grossjohann: XIRQL: A Query Language for Information Retrieval in XML Documents. ACM SIGIR, 2001.

[7] T. Grabs, H.-J. Schek: Generating Vector Spaces On-the-fly for Flexible XML Retrieval. XML and Information Retrieval Workshop - ACM SIGIR, 2002.

[8] S. Guha, H.V. Jagadish, N. Koudas, D. Srivastava, T. Yu: Approximate XML Joins, ACM SIGMOD, 2002.

[9] T. Joachims: The Maximum-Margin Approach to Learning Text Classifiers. Ausgezeichnete Informatikdissertationen 2001, D. Wagner et al. (Hrsg.), GI-Edition - Lecture Notes in Informatics (LNI), Koellen Verlag, Bonn, 2002.

[10] C.D. Manning, H. Schuetze: Foundations of Statistical Natural Language Processing. MIT Press, 1999.

[11] G. Miller: Wordnet: A Lexical Database for English. Communications of the ACM 38(11), 1995.

[12] T. Mitchell: Machine Learning. McGraw Hill, 1996.

[13] S. Sizov, G. Weikum, et al: The BINGO! System for Information Portal Generation and Expert Web Search. CIDR, 2003

[14] A. Theobald, G. Weikum: The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. EDBT, 2002.

[15] V. Vapnik: Statistical Learning Theory. Wiley, 1998.