# 10

# Classification and Focused Crawling for Semistructured Data

Martin Theobald, Ralf Schenkel, and Gerhard Weikum

## 10.1 Introduction

Despite the great advances in 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.

An important IR technique is automatic classification for organizing documents into topic directories based on statistical learning techniques [218, 64, 219, 102]. 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. 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 work 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). With respect to the tree structure of XML documents, we study XML twigs and tag paths as extended features that can be combined with text term occurrences in XML elements. XML twigs

H. Blanken et al. (Eds.): Intelligent Search on XML Data, LNCS 2818, pp. 145–157, 2003.

are triples of the form *(ancestor element, left sibling element, right sibling element)* which allow a shallow structure-aware document representation, while tag paths merely describe linear ancester/descendant relationships with no regard for siblings. Moreover, we show how to leverage ontological background information, more specifically, the WordNet thesaurus, for the construction of more expressive feature spaces.

The various options for XML feature spaces are implemented within the BINGO! [281] focused crawler (also known as *thematic crawler* [63]) for expert Web search and automated portal generation. BINGO! has originally been designed for HTML pages (including various formats of unstructured text like PDF, Word, etc.), and is now extended not only to extract contents from XML documents but to exploit their structure for more precise document representation and classification.

## 10.2 Overview of the BINGO! System

The BINGO! [1] focused crawling toolkit consists of six main components that are depicted in Figure 10.1: the multi-threaded crawler itself, an HTML document analyzer that produces a feature vector for each document, the classifier with its training data, the feature selection as a "noise-reduction" filter for the classifier, the link analysis module as a distiller for topic-specific authorities and hubs, and the training module for the classifier that is invoked for periodic retraining.

Topic-specific bookmarks play a key role in the BINGO! system. The crawler starts from a user's bookmark file or some other form of personalized or community-specific topic directory [22]. These intellectually classified documents serve two purposes: 1) they provide the initial seeds for the crawl (i.e., documents whose outgoing hyperlinks are traversed by the crawler), and 2) they provide the initial contents for the user's topic tree and the initial training data for the classifier. The classifier, which is the crucial filter component of a focused crawler, detects relevant documents on the basis of these bookmark samples, while it discards off-the-topic documents and prevents their links from being pursued. The following subsections give a short overview of the main components of BINGO!. For more details see [282, 281].

### 10.2.1 Crawler

The crawler processes the links in the URL queue using multiple threads. For each retrieved document the crawler initiates some analysis steps that depend on the document's MIME type (e.g., HTML, XML, etc.) and then invokes the classifier on the resulting feature vector. Once a crawled document has been successfully classified, BINGO! extracts all links from the document and adds

[1] **B**ookmark-**In**duced **G**athering **o**f **!**nformation

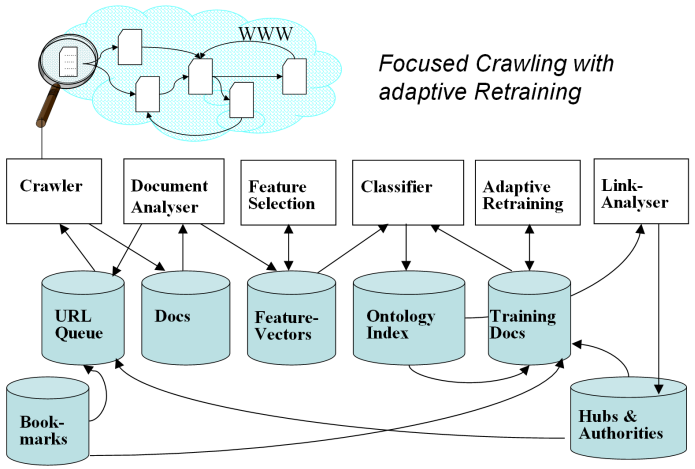


**Fig. 10.1.** The BINGO! architecture and its main components

them to the URL queue for further crawling. The ordering of links (priority) in the crawler queue is based on the classification confidence provided by the specific classification method that is used. This confidence measure is derived from either statistical learning approaches (e.g., Naive Bayes [19, 209]) or the result of regression techniques (e.g., Support Vector Machines [57, 306, 180]). All retrieved documents are stored in our database index including the features, links and available metadata like URL(s), title, authors, etc.

### 10.2.2 Document Analyzer

BINGO! computes feature vectors for documents according to the standard bag-of-words model, using stopword elimination, Porter stemming, and $tf*idf$ based term weighting [19, 209]. We consider our local document database as an approximation of the corpus for *idf* computation and recompute it lazily upon each retraining.

The standard document analyzer for unstructured data can address a wide range of content handlers for different document formats (in particular, PDF, MS Word, MS PowerPoint etc.) as well as common archive files (zip, gz) and converts the recognized contents into simple HTML. So these formats can be processed by BINGO! like usual web pages (except for the link extraction).

For semistructured data like XML, document analyzation offers challenging possibilities to generate more meaningful features under the consideration of annotation and structure which is a main issue of our recent work (see section 10.3). Because of the inherent differences between structured and

unstructured feature sets, training documents of these two kinds cannot be merged under the same classification model. Hence, classification could only take place with two different classifiers for structured and unstructured data. Therefore, our recent efforts are directed towards a unified view for structured and unstructured data including the automatic transformation and annotation of HTML to embed XML-style document handling into crawling over the Web.

### 10.2.3 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)* [57, 306, 180] as a leading-edge classification method. More specifically, we use Thorsten Joachim's SVM-light V5.00 software restricting ourselves to linear SVMs.

The SVM algorithm is a learning algorithm that consists of a supervised training phase for a *binary set of topics*. A separating hyperplane defined as $\mathbf{w} \cdot \mathbf{x} + b = 0$, where $w$ denotes the normal vector of this hyperplane, and $|b|/||w||$ is the perpendicular distance from the hyperplane to the origin, is computed such that it maximizes the margin between a set of positive and negative feature vectors in the $m$-dimensional feature space. These feature vectors represent a user-specific document collection with positive samples for the user's topic of interest as well as explicit negative samples [281] in the vector space model.

The classification step for a test document $\mathbf{y}$ then simply consists of the computation of the algebraic sign of a decision function of the form $\mathbf{w} \cdot \mathbf{y} > 0$, the sign denotes on which side of the hyperplane the test vector is determined to be, and the value of the result yields the SVM classification confidence in form of the distance of the test vector to the separating hyperplane.

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. The computed hyperplane that separates these samples 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 by the root's children, and it is assigned to all topics for which the classifier yields a positive decision (or alternatively, only to the one with the highest positive classification confidence) [103, 68, 306].

### 10.2.4 Topic-Specific Feature Spaces

The feature selection algorithm provided by the BINGO! engine yields the most characteristic features for a given topic; these are the features that are

used by the classifier for testing new documents. A good feature for this purpose discriminates competing topics from each other, i.e., those topics that are at the *same level of the topic tree*. Therefore, feature selection has to be topic-specific; it is invoked for every topic in the tree individually.

We use the *Mutual Information* (MI) measure to build topic-specific feature spaces. This technique, which is a specialized case of the notions of cross-entropy or Kullback-Leibler divergence [209], is known as one of the most effective methods [326, 325] that is slightly in favor of rare terms (i.e., the ones with a high *idf* value) which is an excellent property for classification.

Mutual information can be interpreted as a measure of how much the joint distribution of features and topics deviate from a hypothetical distribution in which features and topics are independent of each other (hence the remark about MI being a special case of the Kullback-Leibler divergence which measures the differences between multivariate probability distributions in general).

The root node of the taxonomy tree yields the unification of all topic-specific feature spaces and provides a simple dictionary data structure for one-to-one mapping of features (i.e., terms) to dimensions (i.e., integers) in the vector space which is used to generate the input vectors of the SVM.

### 10.2.5 Using the Link Structure for Topic Distillation

The link structure between documents in each topic is an additional source of information about how well they capture the topic. The BINGO! engine applies Kleinberg's link analysis method, coined HITS [189], iteratively to each topic in the taxonomy tree (i.e., HITS is invoked after a certain amount of new documents has been visited). Analogously to Google's Page Rank, the HITS algorithm uses the documents adjacency matrix $A_c$ to exploit principle Eigenvectors of each topic-specific sub graph $G_c = (V_c, E_c)$ of the Web. While Page Rank uses $A_c$ directly, HITS divides the task into the approximation of the principle Eigenvectors of the matrices $A_c^T A_c$, and $A_c A_c^T$ respectively, and thus produces a ranking for the best *hubs* (link collections) and the best *authorities* (most frequently referenced sources). Despite a numerical solution of this Eigenvalue problem, the HITS algorithm yields an approximation by iteratively summing up $x_p = \sum_{(q,p)\in E_c} y_q$ and $y_p = \sum_{(p,q)\in E_c} x_q$ and normalizing these values until they converge to $x^* = A_c^T y^*$ and $y^* = A_c x^*$ which provides an essential efficiency improvement with regard to the application of this method for Web crawling.

In a first approach, this method may be applied to XLinks [95] and XPointers [144] as well, however, without regarding the XPointer information and using XLinks as whole document pointers that aggregate the hub and authority weights for the root elements only. A more fine grained approach might overcome these limitations and exploit the tree sub-structure of XML documents by regarding the elements themselves as mini-documents, and thus,

splitting documents into micro-hubs and micro-authorities to produce a more precise ranking for large, heterogeneous XML DOM [311] trees.

### 10.2.6 Retraining

Building a reasonably precise classifier from a very small set of training data is a challenging task. Effective learning algorithms for highly heterogeneous environments like the Web would require a much larger training basis, yet human users would rarely be willing to invest hours of intellectual work for putting together a rich document collection that is truly representative of their interest profiles. To address this problem we distinguish two basic crawl strategies:

- The *learning phase* serves to automatically identify the most characteristic documents of a topic, coined *archetypes*, and to expand the classifier's knowledge base among the bookmarks' neighbor documents with highest classification confidence and best authority score.
- The *harvesting phase* then serves to effectively process the user's information demands with improved crawling precision and recall.

BINGO! repeatedly initiates retraining of the classifier when a certain number of documents have been crawled and successfully classified with confidence above a certain threshold. At such points, a new set of training documents is determined for each node of the topic tree. For this purpose, the best archetypes are determined in two complementary ways. First, the link analysis is initiated with the current documents of a topic as its base set. The best authorities of a tree node are regarded as potential archetypes of the node. The second source of topic-specific archetypes builds on the confidence of the classifier's yes-or-no decision for a given node of the ontology tree. Among the automatically classified documents of a topic those documents whose yes decision had the highest confidence measure are selected as potential archetypes. The *intersection* of the top authorities and the documents with highest SVM confidence form a new set of candidates for the promotion to the training data.

After successfully extending the training basis with additional archetypes, BINGO! retrains all topic-specific classifiers and switches to the harvesting phase now putting emphasis on recall (i.e., collecting as many documents as possible) where the crawler is resumed with the best hubs from the link analysis.

## 10.3 Feature Spaces for XML Data

### 10.3.1 HTML to XML

The BINGO! engine can crawl both HTML and XML documents. However, there is surprisingly little Web data available in XML format, mostly because

of the extra burden that XML (or some even more advanced Semantic-Web style representation such as RDF [320] or DAML+OIL [166]) would pose on non-trained users in terms of authoring and maintaining their Web pages. So, unless XML authoring is significantly improved [151], simply typing text and adding simple HTML-style markup is likely to remain the favorite format on the Web. Ironically, most of the dynamically generated information that can be obtained via Web portals (e.g., Amazon, eBay, CNN, etc.) is actually stored in backend databases with structured schemas but portal query results are still delivered in the form of almost unstructured HTML.

Following prior work on HTML wrapper generators and information extraction tools (see, e.g., [256, 82, 23, 265]), we have developed a toolkit for automatically transforming HTML documents into XML format. Our tool first constructs a valid XML document for input in plain text, HTML, PDF, etc., and then uses rules based on regular-expression matching to generate more meaningful tags. For example, keywords in table headings may become tags, with each row or cell of the table becoming an XML element. Our framework is currently being extended to use also machine-learning techniques such as Hidden Markov Models for more elaborated annotation. Since we are mostly interested in focused crawling and thematic portal generation, the tool has been designed for easy extensibility to quickly add domain specific rules.

### 10.3.2 Features from Tag Paths

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 [209]) 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. 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. We do not expect authors to be as careful as if they designed a database schema, but there should be high awareness of the need for meaningful and reasonably precise annotations and structuring. Furthermore, we expect good XML authoring tools to construct tags in a semi-automatic way, for example, by deriving them from an ontology or a "template" library (e.g., for typical homepages) and presenting them as suggestions to the user.

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) pairs* in the spirit of a database schema with attribute 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 and take advantage of the semantics of a term within its respective document context. 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.

### 10.3.3 Combining Terms and Tags

When parsing and analyzing an XML document we extract relevant terms 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 the terms. For each topic and its competing topics in the classification (i.e., the sibling nodes in the topic directory), we again compute from the training documents, the 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. Note, that also for structure-aware feature sets the above argumentation for using cross-entropy remains valid.

Sorting the features in descending order of the MI measure gives us a ranking in terms of the discriminative power of the features, and we select the top $n$ features for the classifier (with $n$ being in the order of 500 up to 5000). We have two options for applying this selection procedure: 1) first ranking and selecting terms alone and then combining the selected ones with all tags, or 2) ranking the complete set of tag-term pairs that occur in the training data. We have chosen the second option, for different element types (i.e., with different tags) often seem to exhibit 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. For efficiency, we introduce a pre-filtering step and eliminate all terms whose total frequency within the given topic is below

some threshold; only the remaining terms, usually in the order of 10000, are considered for (tag, term) feature selection.

Term-based features are quantified in the form of $tf * idf$ weights 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. For XML data we could compute $tf$ and $idf$ statistics either for tags and terms separately or for tag-term pairs. Analogously to the arguments for feature selection, we have chosen the option with combined tag-term statistics. 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 $t_i$. From the viewpoint of an individual term this approach is equivalent to interpreting every XML element as if it were a mini-document. 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 few papers on transaction management in the recent past). Achieving the same desired effect with separate statistics for tags and terms would be much less straightforward to implement.

### 10.3.4 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 be 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.

## 10.4 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 work we have used WordNet [114, 216] 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 word 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.

### 10.4.1 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 an ontology database 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.

The final step towards disambiguating the mapping of a tag onto a word sense is to compare the tag context $con(t)$ with the context of candidates $con(s_1)$ through $con(s_p)$ in terms of a similarity measure between bags of words. The standard IR measure for this purpose would be the cosine similarity between $con(t)$ and $con(s_j)$, or alternatively the Kullback-Leibler divergence between the two word frequency distributions (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 $tf$ vectors of $con(t)$ and $con(s_j)$ for its simpler computation. Finally, we map tag $t$ onto that sense $s_j$ whose context has the highest similarity to $con(t)$ which is similar to the disambiguation strategy that the XXL [296] engine applies. Denote this word sense as $sense(t)$ and the set of all senses of tags that appear in the training data as $senses_{train} = \{sense(t_i) | t_i \; appears \; in \; training \; data\}$.

Putting everything together, a tag $t$ in a test document $d$ is first mapped to a word sense $s := map(t)$, then we find the closest word sense $s' := argmax_{s'}\{sim(s', s) | s' \in F\}$ that is included in the feature space F of the training data, and set the weight of $s'$ in $d$ to $sim(s', map(t))$.

### 10.4.2 Incremental Mapping

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 a detailed discussion how the semantic similarity $sim(s, s')$ of two concepts $s$ and $s'$ in the ontology graph can be estimated see Chapter 8.

If the training set itself yields rich feature spaces that already contain most of the classification-relevant terms we will ever encounter in our test documents, we cannot expect vast improvements by ontology lookups and

thus leave out this step using only given tag-term pairs and twigs. With increasingly sparse training feature spaces (in comparison to the test set), we can stepwise add lookups for the strongest (most relevant) features (i.e. the tags) and/or terms, where infrequent nouns among the element contents would be the next source of potentially relevant features. Of course, full lookups for each term using the ontology service with its disambiguation function is best suited for very few training data but also has the highest costs and decreases performance.

### 10.4.3 Ontological Classification in the Vector Space

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. The weight $w_{ij}(f_i)$ of feature $f_i$ in document $j$ is now scaled by $sim(s, s')$, i.e., $w_{ij}(f_i) = sim(s, s')\ tf_{ij}\ idf_i$, to reflect the concept similarities of approximate matches in the feature weights.

In praxis, we limit the search for common hypernyms to a distance of 2 in the ontology graph. Concepts that are not connected within this threshold are considered as dissimilar for classification and obtain a similarity value of 0. To improve performance over frequently repeated queries, all mappings of similar synsets within the same document (i.e., the same context) and their retrieved similarities are cached.

Unlike related techniques using ontologies for query expansion, we do not change the length of the feature vectors (i.e., the amount of a document's features). Our approach is merely adjusting the weights of unknown test concepts according to their nearest match in the feature space with regard to correlations defined by the ontology; adding new features to the vector could produce more matches in the classification step and would distort the result. This ontology-based evaluation can be regarded as a "similarity kernel" for the SVM that goes beyond syntactic matching.

## 10.5 Conclusions

Preliminary experiments were run on XML representations of the IMDB (Internet Movie Database) collection and the Reuters-21578 data set taking into consideration the XML structure using simple tag-term pairs (only regarding a single ancestor tag) and twigs. These tests indicate a clear improvement

of the classifiers' F-measure [19, 209] (the harmonic mean of precision and recall) for the structure aware feature sets over a purely text-based feature space. For the IMDB the structure-aware classifier improved the F-measure from approximately 0.7 that the text classifier achieved to about 0.8 when very few (5-20) documents were used for training. For the Reuters data mapping of text terms onto ontological concepts led to a similar improvement when few training documents were used. In both cases the text classifiers continuously converged to (but never ouperformed) the F-measure of the structural classifiers when we were feeding more training documents (up to 500 per topic) into the classifiers knowledge base which led to a saturation effect where ontology lookups could not replace any formerly unknown training concepts any more.

These tests show the potential of structural features like tag-term pairs and twigs towards a more precise document characterization and the computation of a structure-aware classifier. Ontology lookups are a promising way for disambiguating terms in a given document context and for eliminating polysemy of text terms.

Our approach 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 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.