Classification of XML Documents

Abdelhamid Bouchachia and Marcus Hassler
Dept. of Informatics-Systems
Alpen-Adria-Universität Klagenfurt, 9020, Austria
Email: hamid@isys.uni-klu.ac.at, marcus.hassler@uni-klu.ac.at

***Abstract*— With the explosion of XML-based online documents, the task of knowledge discovery from the web becomes highly significant. As an appropriate machinery, classification allows to categorize documents to facilitate that task. A classification approach is introduced in this paper. It is based on the k-nearest neighborhood algorithm that relies on an edit distance measure. The originality of the work lies in combining both the content and the structure of XML documents to compute the edit distance. The approach is empirically evaluated using real-world XML collections.**

I. Introduction

The eXtensible Markup Language (XML) has recently emerged as a standard for developing many web applications dealing with document storage and retrieval, e.g., digital libraries. XML was mainly developed to achieve an enriched representation of documents and more retrieval flexibility when searching information. Systems concerned with such web applications are mainly augmented traditional information retrieval systems. In contrast to traditional retrieval systems that deal with flat documents, XML retrieval systems take the logical structure of documents into account. In addition to the raw text (pure content), this structure is considered as a further valuable information source for document representation. It serves to refine the search process and to improve the quality of the retrieval results. Indeed, each document is represented as a tree of XML nodes, where each node is associated with a label and a content (called *XML component*). The goal of retrieval systems is, therefore, to retrieve only relevant components instead of the whole documents in response to the user queries. As a matter of consequence, the retrieval precision gets better.

As far as we are concerned with document classification, it is of high importance to exploit the structure of documents in order to devise a classification machinery. This latter is a very significant mechanism in the context of XML retrieval due to two reasons: (1) a user query can be satisfied by means of different possible answers; closely associated documents tend to be relevant to the same requests, and (2) retrieval systems and web mining tools are generally operational in the same environment.

Classifying (and clustering) XML documents can be basically done in three ways: (1) using exclusively the textual contents of documents as usually done in traditional text categorization (and clustering) systems, (2) using exclusively the structure of the XML documents, and (3) using both the contents and the structure in a hybrid manner. In this work we are interested in the latter approach. The aim is to discover structural and content patterns (characteristics) shared by XML documents of the same class. These patterns are mainly expressed in terms of their tags (node labels), contents, and inter-relationship.

To achieve this goal, we rely on the k-nearest neighborhood algorithm. The algorithm, which strongly relies on a distance measure, will be explained later. Since we are dealing with XML documents represented in the form of a tree, it is straightforward to adopt an *edit distance* to measure the distance between document trees. The edit distance algorithm computes the minimum cost to transform one document tree into another, taking content and structure into account. In this work, we show how the content can be embedded into the edit distance. To the best of our knowledge, no previous work has used edit distance taking the content of the XML tree into account.

The rest of the paper is organized as follows. Section II reviews related work on XML document classification. In Sec. III, the edit distance algorithm is discussed with respect to the content and structure before introducing a pure content-based approach in Sec. IV. In Sec. V we briefly explain the k-NN algorithm. Section VI discusses the experimental evaluation of the approaches. Finally, Sec. VII concludes the paper.

II. Related work

Although classification has been widely discussed in the framework of traditional information retrieval (with flat documents), it has not yet gained much attention in the field of structured documents. In the following, some of the research work dedicated to XML document classification are briefly summarized.

In [9], a classification approach is proposed that aims at using the structure and the contents to classify XML documents. It relies on a generative Bayesian classifier. Here, the generative model assumes that there are two types of belief, structural and textual, which will be combined to get one single evidence about a document assignment to classes:

$$P(doc|\theta) = P(str|\theta) \times P(cont|str, \theta)$$

where θ is a set of the model parameters, doc, str, and $cont$ designate document, structure, and content. The quantities $P(str|\theta)$ and $P(cont|str, \theta)$ are computed via a belief network as follows:

$$P(str|\theta) = \prod_{i=1}^{\#of\ tags} P(node_i|ancestor(node_i))$$

$$P(cont|str,\theta) = \prod_{i=1}^{\#of\ terms} P(term_i|node_j,\theta)$$

During the training phase, the parameters of the generative model are learned using the Expectation-Maximization method. Unseen XML documents are assigned to the class with the highest probability $P(doc|class_k)$.

In [3], a similar work to that described in [9] is developed. The method, however, is less general, since the assumption here is that all documents have the same structure across all classes. According to this method, documents are broken down into components, each of which contains either structured data or non-structured textual data. Let d_j, c_k, s_l, t_i, designate respectively a document j, a class k, a component l, and a term i, the total probability of a document is

$$P(d_j|c_k) = \prod_{s_l \in d_j} \prod_{t_i \in s_l} P_{s_l}(t_i|c_k)^{f^j(t_{is})}$$

where $f^j(t_{is})$ indicates the word frequency of term i in the s^{th} component of the document j. The first product is over all structural components s_l that are present in the document d_j. The probability P_{s_l} is obtained for each document component. From this, it is clear that each document is represented by a set of feature vectors, one for each component, so that word frequency $f^j(t_{is})$ is maintained on a per-component basis.

In [22] a classifier, called XRules, is developed. This classifier is structure oriented and aims at discovering a set of structural rules that define the individual classes. Basically, these rules, that reflect regular structural patterns of each class, are learned during the training phase.

Most of the classification related work in the area of structured data use *tree edit distance* to measure the distance between associated graphs/trees. Basically, edit distance algorithms compute the minimum cost to transform one document tree into another. Each transformation (e.g., insert, delete, alter) is associated with a certain cost. In addition to the edit distance measure, other approaches apply other types of distance measures. In the domain of XML classification, authors tried to consider XML peculiarities such as tags, parent-child relationships, root-leaf XPaths, arity of nodes, etc. [24], [4]. Many algorithms have been proposed to compute the edit distance between two trees [7], [8], [23], [18]. Most of these algorithms use dynamic programming techniques as was initially described in [13]. The aim is to find the cheapest sequence of transformations, called *delta script* or *edit script* [7]. The main difference of the various edit distance measures is the set of allowed edit operations and their associated costs. Early work [17] considered *insert* and *delete* of leaf nodes, and node *relabeling* of all nodes. Extensions were then proposed to allow insertion and deletion of nodes anywhere in the tree [19], [20], [18].

In general the problem of finding the edit distance between two trees is NP-hard [14], [2]. The work in [23] and [7] try to find an efficient algorithm for the reduced problem of ordered/binary trees, in which a left-to-right order among siblings is significant. Chawathe et al. [7] first applied the same edit operations and restrictions to detect changes in structured documents. In subsequent works he extended the approach to cover also move operations as basic edit operations [6] and defined later on operations for copying and gluing of subtrees [5].

Further edit distance methods dedicated to XML documents can be found in the literature related to structural mapping, e.g., change detection [8]. A good overview of existing algorithms for change detection together with their properties is given by Peters [16].

III. Tree Matching via Edit Distance

In this work, we will modify the algorithm proposed by Nierman and Jagadish [15] which allows to compute the edit distance between two trees using only the labels of the nodes. Whilst this algorithm is structure-oriented, in our research work, we are aiming at inducing a classifier that takes both the structure and the content into account. We kept the dynamical aspect of the algorithm, but we have improved it with respect to two dimensions: (1) the algorithm will rely on simplified edit operations; (2) the algorithm will take the content match into account when computing the edit distance. Concretely, we need to formulate a hybrid distance measure that combines structure-based and content-based distances. Each of these are described in the following sections.

A. Structure Matching

To formulate the distance, some definitions of the basic concepts are introduced.

Definition 1 (Tree node): A node n of a tree T is associated with a label $\lambda(n)$, a content $\gamma(n)$, a parent node $p \in T$, and a set of children nodes $ch(n)$. $\gamma(n)$ is empty ($null$), if a node does not have a content. The parent node p of the root node of T is $null$. The children nodes of n are uniquely identified as $n^1, n^2, \dots, n^k$, where k is the degree of n denoted as $deg(n)$.

Definition 2 (Ordered Tree): An ordered tree T is defined as a rooted tree, where a left-to-right order among the children nodes is significant. $root(T)$ denotes the root node of T. The children of T are identified as the children of the root node of T, denoted as $T^1, T^2, \dots, T^k$ ($k = deg(root(T))$). Further, $|T|$ represents the total number of nodes in tree T.

Based on the aforementioned definitions, the allowed basic edit operations can be described as follows:

Definition 3 (Insert operation): Given a tree T and a node $p \in T$, a node n can be inserted into T as the i^{th} child of the node p using the operation ins(T,n,p,i). The cost function of this operation is $C_{ins}(T,n,p)$.

Definition 4 (Delete operation): Given a tree T and a node $n \in T$ as the i^{th} child of node $p \in T$, n can be removed using the operation del(T,n,p,i). The cost function of deletion is $C_{del}(T,n,p)$.

Definition 5 (Alter operation): Given two nodes n_1 and n_2 whose labels are $\lambda(n_1)$ and $\lambda(n_2)$. The label of n_1 can be replaced with the label of n_2 using the operation $alt(n_1, n_2)(\equiv \lambda(n_1) \leftarrow \lambda(n_2))$. The cost function of this operation is $C_{alt}(n_1, n_2)$ defined as:

$$C_{alt}(n_1, n_2) = \begin{cases} 0 & \text{if } \lambda(n_1) = \lambda(n_2) \\ \beta & \text{otherwise} \end{cases} \tag{1}$$

Note that one can parameterize β so that altering a node lying at level l in the document tree costs $\beta(l)$. This seems reasonable for taking the depth of the tree and semantic closeness between tags into account.

Furthermore, if $C_{ins}(T, n, p) = C_{del}(T, n, p)$, the edit distance measure becomes symmetric. These operations are applied on nodes that can be either leaf nodes or inner nodes, where an inner node is the root of a subtree. The cost of applying an operation on an inner node is recursively computed by summing up the cost of its descendants. Hence, the following definitions of the cumulative costs:

Definition 6 (Recursive delete): The cumulative cost of deleting from the tree T a subtree, Sub, at node p is $C_{delCum}(T, Sub, p)$.

Definition 7 (Recursive insert): The cumulative cost of inserting into the tree T a subtree, Sub, at node p is $C_{insCum}(T, Sub, p)$

Both costs C_{delCum} and C_{insCum} are bottom-up computed as follows:

$$\begin{cases} C_{delCum}(T, Sub, p) = \sum C_{delCum}(T, Sub^j, root(Sub)) \\ \qquad\qquad + C_{del}(T, root(Sub), p) \\ \\ C_{insCum}(T, Sub, p) = C_{ins}(T, root(Sub), p) \\ \qquad\qquad + \sum C_{insCum}(T, Sub^j, root(Sub)) \end{cases} \tag{2}$$

In other words, at each node of the subtree Sub of the source (resp. destination) tree T_1 (resp. T_2), the cost of the recursive delete (resp. insert) operation is calculated by summing the cost for deleting (resp. inserting) the single node $root(Sub)$ with the cumulative cost of deleting (resp. inserting) each of its children Sub^j.

Having introduced some required variables, the edit distance between two trees T_1 and T_2 is computed using Alg. 1. Based on dynamic programming, this algorithm constructs a $deg(root(T_1)) \times deg(root(T_2))$ matrix of distance values between the nodes of the two trees. First, the algorithm compares the root nodes of T_1 and T_2. This corresponds to an alter operation (line 4). Then, as seen in lines 6 and 9, the algorithm computes the distance values of inserting or deleting all nodes given the roots of two trees. These values serve to trigger the dynamic computation of cumulative costs, where each child of T_1 is compared to each child of T_2 recursively. Indeed, a cell $distMat[i][j]$ ($i > 0$ and $j > 0$) is assigned a cost that is computed using the content of its neighboring three cells:

- the content of the upper left neighbor, $distMat[i-1][j-1]$, is added to the distance between the subtrees rooted at nodes n_i and n_j (i.e., T_1^i and T_2^j)) (line 14). This case corresponds to a match between node n_i and node n_j.
- the content of the left neighbor, $distMat[i][j-1]$, is added to the cost of inserting a subtree T_2^j to the source tree (line 15). This case corresponds to an insertion of a subtree rooted at node n_j.
- the content of the upper neighbor, $distMat[i-1][j]$, is added to the cost of removing a subtree T_1^i from the source tree (line 16). This case corresponds to a removal of an obsolete subtree rooted at node n_i.

The minimum cost of these three alternatives is retained and stored in $distMat[i][j]$.

Algorithm 1 $dist(T_1, T_2)$

```
1: int M = deg(root($T_1$))
2: int N = deg(root($T_2$))
3: int[][] distMat = new int[0..M][0..N]
4: distMat[0][0] = $c_{alt}(root(T_1), root(T_2))$
5: for $j = 1$ to $N$ do
6:     distMat[0][j] = distMat[0][j-1]
                + $C_{insCum}(T_2, T_2^j, root(T_2))$
7: end for
8: for $i = 1$ to $M$ do
9:     distMat[i][0] = distMat[i-1][0]
                + $C_{delCum}(T_1, T_1^i, root(T_1))$
10: end for
11: for $i = 1$ to $M$ do
12:     for $j = 1$ to $N$ do
13:         distMat[i][j] = min{
14:         distMat[i-1][j-1] + $dist(T_1^i, T_2^j)$,
15:         distMat[i][j-1] + $C_{insCum}(T_2, T_2^j, root(T_2))$,
16:         distMat[i-1][j] + $C_{delCum}(T_1, T_1^i, root(T_1))$
17:         }
18:     end for
19: end for
20: return distMat[M][N]
```

The original algorithm proposed by Nierman and Jagadish outputs only the edit distance between two trees. There is no way to reconstruct the optimal sequence of edit operations that led to the obtained final edit distance. To overcome that, we have to memorize all possible edit scripts that correspond to the minimal distance between the two trees at hand. We define an edit script as follows:

Definition 8 (Edit Script): An edit script δ is an ordered sequence of edit operations that transform T_1 into T_2. In general, there exist an infinite number $\delta^1, \delta^2, \ldots, \delta^m$ of edit scripts that correctly transform T_1 into T_2.

Definition 9 (Minimal Edit Script): Let $\delta^1, \delta^2, \ldots, \delta^m$ be a set of correct edit scripts transforming T_1 in T_2. A minimal edit script is then defined as:

$$\begin{aligned} Min_{k=1..m}\{\delta^k\} = \delta^i \iff \forall \delta^j \mid i \neq j, \\ dist(T_1, T_2)^{\delta^i} \leq dist(T_1, T_2)^{\delta^j} \end{aligned} \tag{3}$$

where $dist(T_1, T_2)^{\delta^j}$ indicates the distance between T_1 and T_2 obtained after applying the script δ^j. Note that there might be more than one minimal edit script for a pair of trees.

B. Content Matching

The second type of distance required is the content-based distance. To define it, we still rely on Alg. 1. However, the previously used cost functions have to be redefined in order to support the content match.

Let $sim(\gamma(n_1), \gamma(n_2))$ be an existing similarity function that compares the contents of two nodes n_1 and n_2. Therefore, this similarity can be any information matching function. Let us assume that this similarity measure is normalized so that it takes values in the unit interval [0,1] (where $sim = 0$ means no match, $sim = 1$ indicates full match, and $sim \in]0, 1[$ means partial match). Further, we define:

$$sim(null, null) = 1 \tag{4}$$

$$sim(null, \gamma(n_2)) = sim(\gamma(n_1), null) = 0 \tag{5}$$

Eq. 4 stipulates that the content-based similarity of two nodes with empty content is total (i.e., complete match), while Eq. 5 stipulates that content-based similarity between a node with a content and another node with empty content is 0.
The cost functions for **inserting** and **deleting nodes with their contents** are defined as:

$$C_{insCon}(n) = 1 - sim(null, \gamma(n)) = 1 - 0 = 1 \tag{6}$$

$$C_{delCon}(n) = 1 - sim(\gamma(n), null) = 1 - 0 = 1 \tag{7}$$

The cost of altering the content, C_{altCon} (Eq. 8) is the sum of the cost of changing the label $C_{alt}(n_1, n_2)$ (Eq. 1) and the cost of altering the content which is expressed as:

$$\rho * (1 - sim(\gamma(n_1), \gamma(n_2)))$$

The cost C_{altCon} is given by:

$$\begin{aligned} c_{altCon}(n_1, n_2) &= c_{alt}(n_1, n_2) \\ &+ \rho * (1 - sim(\gamma(n_1), \gamma(n_2))) \end{aligned} \tag{8}$$

The amount ρ is a cost factor that scales up the dissimilarity of contents (since $sim(\gamma(n_1), \gamma(n_2)) \in [0, 1]$ and $c_{alt}(n_1, n_2)$ can be larger than 1).

Furthermore, we can decide whether the alter operation is cheaper than the delete and the insert operations combined together. This can be tuned by a user-specified parameter, α ($0 \leq \alpha < 1$). Precisely, if $sim(\gamma(n_1), \gamma(n_2)) > \alpha$ then $C_{altCon} < (C_{delCon} + C_{insCon})$ (the alter is cheaper) and if $sim(\gamma(n_1), \gamma(n_2)) < \alpha$ then $C_{altCon} > (C_{delCon} + C_{insCon})$ (alter is more expensive); otherwise, alter has the same cost as that of both delete and insert combined. From this, we can determine the cost factor ρ as follows:

$$\rho * (1 - \alpha) = C_{delCon}(n_1) + C_{insCon}(n_2)$$

leading to:

$$\rho = \frac{C_{delCon}(n_1) + C_{insCon}(n_2)}{(1 - \alpha)} \tag{9}$$

Now, to take the content similarity into account $C_{alt}(root(T_1), root(T_2))$ in Alg. 1 (line 4) is substituted for $C_{altCon}(root(T_1), root(T_2))$.

IV. Tree Matching via Content Matrix

While the focus has been so far on explicit consideration of the structure to compare documents, in the following we suggest a more content-oriented matching procedure that uses the structure of documents only implicitly. More expressively, the idea is to measure the similarity between documents using only nodes with contents. Indeed, every node in the source tree is compared to all nodes in the destination tree relying on a content similarity matrix (denoted *Content Matrix*), where only nodes containing content are taken into account.

A cell $contSim[i][j]$ refers to the similarity degree (computed by means of a measure) between the contents of the node n_i in the source tree and the node n_j in the destination tree, i.e., $contSim[i][j] = sim(\gamma(n_i), \gamma(n_j))$. One might define a two-step process to perform the comparison: If the labels of the nodes to be compared are the same, then the contents of these nodes are compared. However, the first step of this comparison can be optional at wish of the user via a flag variable ζ. If ζ is set to $true$, the similarity of nodes with uneven labels is set to 0.

Once the content similarity matrix, $contSim[][]$, is filled, the distance between the corresponding documents can be computed using an algorithm that traverses that matrix in a single pass. Basically, three edit operations: insert, delete, and alter are applied. During the computation, these operations are labeled either $safe$ (certain) or $unsafe$ (uncertain) as shown in Alg. 2. Basically the algorithm proceeds mainly in two step: (1) computing the similarities, and (2) marking of nodes according to the edit operations as follows (item numbers correspond to lines in Alg. 2):

14: If $simMat[i][j] = 1$, then nodes $n_{1,i}$ and $n_{2,j}$ are both marked as $safe\_match$ with no additional cost.
15: A source node, $n_{1,i}$ having no matching destination nodes ($simMat[i][j] = 0, \forall j = 1 \ldots |dest|$) is marked as $safe\_delete$. The cumulative cost is increased by the weighted cost of $safe\_delete$.
16: A destination node, $n_{2,j}$ with no corresponding source nodes ($simMat[i][j] = 0, \forall i = 1 \ldots |source|$) is marked as $safe\_insert$. The cumulative cost is increased by the weighted cost of $safe\_insert$.
17: An unmarked source node $n_{1,i}$ is marked as $unsafe\_match$ along with one unmarked destination node $n_{2,j}$, if $n_{2,j}$ is the first node (minimal index j) fulfilling the condition $simMat[i][j] \geq \alpha$, where α is a user-specified similarity threshold.
18: Any remaining unmarked source node, $n_{1,i}$, ($simMat[i][j] < \alpha, \forall j = 1 \ldots |dest|$) are marked as $unsafe\_delete$.
19: Any remaining unmarked destination node, $n_{2,j}$, ($simMat[i][j] < \alpha, \forall i = 1 \ldots |source|$) are marked as $unsafe\_insert$.

As in the tree edit distance approach, each of the edit operations is associated with a certain cost. In addition, the labels $[safe]$ and $[unsafe]$ can be weighed. Since the distance is inversely proportional to the similarity, the transformation

Algorithm 2 $CM\_dist(T_1, T_2, \zeta)$

```
1: int M = $|T_1|$ /*only content nodes*/
2: int N = $|T_2|$ /*only content nodes*/
3: float[][] $simMat$ = new float[0..$M$][0..$N$]
4: for $i = 1$ to $M$ do
5:     for $j = 1$ to $N$ do
6:         if $\zeta = true$ and $\lambda(n_1) \neq \lambda(n_2)$ then
7:             $simMat[i][j] = 0$
8:         else
9:             $simMat[i][j] = sim(\gamma(n_{1,i}), \gamma(n_{2,j}))$
10:        end if
11:    end for
12: end for
13:
14: mark $safe\_match$
15: mark $safe\_delete$
16: mark $safe\_insert$
17: mark $unsafe\_match$
18: mark $unsafe\_delete$
19: mark $unsafe\_insert$
20:
21: return $\sum weighted\_costs$ (based on marks)
```

costs are calculated using the same formula applied in the tree edit distance approach taking $dist(n_{1,i}, n_{2,j}) = 1.0 - contSim[i][j]$ into account. The final distance between the documents is the sum of all of the (weighted) operations costs. Clearly, the computation of the edit script is done in linear time and space (since, there is no recursive computation).

V. Overview of k-NN

The k-NN algorithm [10] is based on the assumption that the classification of a sample is most similar to the classification of other samples that are nearby in the space. Compared to other learning methods such as probabilistic classifiers, k-NN does not rely on prior probabilities. Moreover, despite its efficiency problems, k-NN is known for its effectiveness [21]. The main computation task is that related to sorting the training samples in order to find the k-nearest neighbors of a given query. It is then straightforward to apply k-NN algorithm for classifying XML documents. To classify an unlabeled document (query) using k-NN, the algorithm finds the k documents in the reference samples (training documents) that are the closest to it. The label shared by the majority of these k nearest neighbors is assigned to the query.

VI. Evaluation

In this section, we will study several aspects like: How do different k values influence the classification? What is the impact of training size on the classification performance? How does content and structure matching perform compared to structure only matching?

Clearly, the first aspect only aims at finding the most reasonable value of k. Futhermore, all of these questions are discussed using some real-world XML collections (MovieDB -movie database-) which were proposed in INEX'05 [11]. Documents of these collections are assigned to 11 classes. Basically, the XML collections are of two types:

- Structure-only (SO) collections: These contain only the structure of the XML documents and include 4 collections: *m-db-s-0, m-db-s-1, m-db-s-2, m-db-s-3*, where the last 3 collections are noisy versions of the first one. The amount of noise increases from the first to the last collection. The collections originally come in the form of two sets (training and testing set) as follows: *m-db-s-0* (4824 4816), *m-db-s-1* (4818, 4814), *m-db-s-2* (4820, 4809), and *m-db-s-3* (4821, 4809).
- Content-and-structure (CAS) collection: This collection is called *m-db-cs-1* and consists of 2415 training and 2410 testing documents. Both training and testing sets are reasonably large and therefore sufficient to adopt the two standard evaluation stages, training and testing, separately.

To answer the questions formulated earlier, 3 sets of experiments are run. The first two deal with the structure only setting, while the last one is concerned with the content-and-structure setting. A comparison of our results against some available results from other authors is shortly highlighted at the end of this section.

In all experiments, we will rely on the classification accuracy is defined as:

$$\text{Accuracy} = \frac{\#\ \text{correctly classified testing docs}}{\#\ \text{testing docs}} \tag{10}$$

A. Experiment I : How Does k Affect the Accuracy?

As explained in Sec. V, the size of the neighborhood (k) is a key parameter in the k-NN algorithm. Therefore, one aspect to look at is to check the effect of k on the accuracy. For this purpose, we will experiment the values: 3, 5, 7, 9, 15, and 21. Note that only the SO collections are used and, due to time constraints, only a proportion (10%) of them is selected randomly and uniformly distributed over the 11 classes to show the effect of k.

Using Alg. 1 to run k-NN, and setting the required parameters: α (Eq. 9), C_{del} cost (Eq. 2), C_{ins} cost (Eq. 2), and β (Eq. 1), to 0.5, 1, 1, 2 ($\beta = C_{ins} + C_{del}$ to avoid permanent node relabeling) respectively, we obtain the results displayed in Tab. I.

TABLE I
Effect of k on the accuracy

Corpus	$k=3$	$k=5$	$k=7$	$k=9$	$k=15$	$k=21$
m-db-s-0	0.922	0.920	0.918	0.903	0.892	0.889
m-db-s-1	0.903	0.892	0.891	0.897	0.885	0.866
m-db-s-2	0.874	0.868	0.858	0.849	0.802	0.790
m-db-s-3	0.862	0.868	0.860	0.852	0.825	0.814

The outcome of the experiment is a 3-fold conclusion: **(i)** as k increases, the accuracy of the algorithm monotonically decreases independently of the collection used, **(ii)** the noise introduced in the m-db-s-1/2/3 has negatively impacted the accuracy (as the amount of noise in relation to m-db-s-0 increases, the accuracy decreases), and **(iii)** the maximum drop in the accuracy is only 3.3% when raising k. Therefore, we will continue using the different values of k in the remaining experiments since these results do not allow to convincingly consider a particular k-value better than the others.

More interesting, the classifier provide very high accuracy results, but this remains relative to the amount of documents used in this experiment.

B. Experiment II - How Does the Training Data Affect the Accuracy?

Furthermore, k-NN uses the entire set of training samples as a basis to label the query. Hence, it is clear that the size of the training set is crucial for the accuracy of the algorithm. To observe the effect of increasing the size of the training data set, we split the *m-db-s-0* collection into 5 ratios (10%, 30%, 50%, 70%, 100%). These ratios are randomly and uniformly selected among the whole training data so that every chunk contains all labels.

Using the same setting described in Sec. VI-A, we obtained the results shown in Tab. II. Unexpectedly, the size of the training set did not greatly impact the accuracy of the classifier. The reason might lie in the inter-document similarity, meaning that the classes are highly homogeneous. Furthermore, the accuracy remains in the same range of values (regardless the value of k) without noticeable fluctuations when increasing the size of the training data.

TABLE II
EFFECT OF THE TRAINING DATA ON THE ACCURACY

Size	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$
10%	0.922	0.920	0.918	0.903	0.892	0.889
30%	0.928	0.925	0.934	0.932	0.930	0.930
50%	0.924	0.929	0.928	0.925	0.925	0.923
70%	0.934	0.933	0.932	0.930	0.930	0.932
100%	0.934	0.934	0.932	0.932	0.929	0.931

C. Experiment III - How Does CAS Setting Affect the Accuracy?

To check the effectiveness of our approach taking both the content and the structure of XML documents into account, we will use the CAS collection (*m-db-cs-1*) described earlier and apply five (5) methods. These are in the following briefly described:

Meth.	Description
BM	A Boolean model [1] is applied as in traditional information retrieval, where documents are represented as a bag of words and where structure is neglected
TED_SO	Details are provided in Sec.III-A and the parameters are set as in Sec. VI-A
TED_CAS	Details are provided in Sec.III-B and the parameters are set as in Sec. VI-A
CM_match	Details are provided in Sec.IV where node labels are considered in the comparais
CM_any	Details are provided in Sec.IV where node labels are ignored
TED_CM	This is a combination of TED_SO and CM_any. Here, the final distance of two documents is $\alpha \cdot TED\_SO + (1-\alpha) \cdot CO\_any$, where α=0.5.

Note that the TED_SO method does not use the document contents, but it is included here for comparison purposes.

These methods are run on 20% of *m-db-cs-1* to obtain the results displayed in Tab. III.

TABLE III
EFFECT OF CAS ON THE ACCURACY

Method	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$
BM	0.327	0.352	0.352	0.331	0.335	0.313
TED_SO	0.916	0.907	0.895	0.895	0.856	0.860
TED_CAS	0.652	0.634	0.640	0.673	0.584	0.558
CM_match	0.352	0.360	0.305	0.309	0.296	0.272
CM_any	0.130	0.163	0.175	0.160	0.134	0.140
TED_CM	0.909	0.907	0.897	0.893	0.862	0.860

Although *m-db-cs-1* and *m-db-s-0* are different form each other, the accuracy of TED_SO on both collections (when using the structure only) is nearly the same and exceeds 90% accuracy. More surprising is the fact that the BM, CM_match, and CM_any methods perform worse. Furthermore, comparing the tree-edit methods *TED_CAS* and *TED_SO*, it is worth concluding that the content deteriorates the accuracy. Indeed the difference in the accuracy are significant: 26%, 27%, 25%, 22%, 23%, and 31%. This also true when ignoring entirely the structure, as with the BM method or when using the content matrix described in Alg. 2 and relying on CM_match and CM_any. The accuracy deterioration in this case is much worse. More consistent with our expectations, combining TED_SO and CM_any, which results in TED_CM, allows to obtain much better results since this combination enables to consider the content while assuring high classification accuracy of the TED_SO method. The accuracy in this case remains high.

From these preliminary experiments, one can see that the structure is a central aspect in the overall similarity between documents during classification. To further validate this result additional experiments on other document collections is certainly needed.

However, at this stage, our work only relies on the MovieDB collection available at our hand. As long as we are concerned with the accuracy of the proposed approach, we need to conduct comparative studies against other results from the literature. Unfortunately, the only ones found are related to MovieDB; hence our motivation for using this collection.

D. Comparison

Because we provided a range of methods, it is relevant to check how these methods compare to the state-of-the-art methods that have been applied on the same collection. To do that, we consider two references appearing in the INEX 2005 workshop. The first is by Hagenbuchner et al. [12] who applied contextual self-organizing maps for structured data (CSOM-SD) to classify XML documents. Actually, CSOM-SD are dedicated to clustering rather than to classification. However, in [12] they have been tested for classification purposes, using a meassure called "classification performance". The second is by Candillier et al. in [4] who applied inductive decision trees (IDT).

To compare these methods against ours, we need to use the same evaluation metrics, accuracy, recall, and precision (at the macro and micro levels) as shown in Tab. IV. Note that we consider just the best performance rates achieved by each method.

TABLE IV
CLASSIFICATION COMPARISON FOR *m-db-s-0*

Approach	Accuracy	Micro Recall	Macro Recall	Micro Precision	Macro Precision
CSOM-SD	0.873	-	-	-	-
IDT	-	0.968	0.960	-	-
TED_CM	0.934	0.934	0.934	0.937	0.911

'-': means value not available

The results illustrate that TED_CM largely outperforms CSOM-SD in terms of accuracy by a rate difference of 6%. However, when considering micro and macro-recall, the IDT approach performs better than TED_CM. Unfortunately, recall without precision is not much telling. The precision values achieved by TED_CM are very encouraging especially when taking recall values into account.

VII. CONCLUSIONS

This work introduces an XML classification approach based on k-nearest neighborhood algorithm which relies on edit distance measures. The originality of the approach comes from the fact that the edit distance considers both the content and structure of XML trees. Our initial results indicate that the proposed approach is very promising on both tasks: 'structure only' and 'content and structure'. Our finding articulates around the fact that the structure bears more weight than the content does. However, a combination of two methods among the proposed ones allows to tune the weight of both the content and the structure. Further empirical work using other document collections is certainly needed.

REFERENCES

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, ACM Press, 1999.

[2] D. Barnard, G. Clarke, and N. Duncan. Tree-to-tree correction for document trees. Technical Report 95-372, Department of Computing and Information Science, Queen's University, 1995.

[3] A. Bratko and B. Filipič. Exploiting structural information in semi-structured document classification. In *Proc. 13th Intl' Electrotechnical and CS Conf. (ERK)*, 2004.

[4] Laurent Candillier, Isabelle Tellier, and Fabien Torre. Transforming XML trees for efficient classification and clustering. In Fuhr et al. [11], pages 487–480.

[5] S. Chawathe. Comparing hierarchical data in external memory. In *Proc. 25th Intl' Conf. on Very Large Data Bases (VLDB)*, pages 90–101, 1999.

[6] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *ACM Intl' Conf. on the Management of Data (SIGMOD)*, pages 26–37, 1997.

[7] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proc. ACM SIGMOD Intl' Conf. on Management of Data*, pages 493–504, 1996.

[8] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proc. 18th Intl' Conf. on Data Engineering (ICDE)*, 2002.

[9] L. Denoyer and P. Gallinari. Bayesian network model for semi-structured document classification. *Information Processing and Management*, 40(5):807–827, 2004.

[10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., 2nd edition edition, 2001.

[11] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltn Szlvic, editors. *INEX 2004 Workshop Proceedings. Dagstuhl, Germany, December 15–17, 2003*. ERCIM, 2005.

[12] M. Hagenbuchner, A. Sperduti, A.C. Tsoi, F. Trentini, F. Scarselli, and M. Gori. Clustering XML documents using self-organizing maps for structures. In Fuhr et al. [11], pages 581–496.

[13] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[14] M. Mani, D. Lee, and M. Murata. Normal forms for regular tree grammars. Technical report, UCLA Computer Science Department, 2001.

[15] A. Nierman and H. Jagadish. Evaluating structural similarity in XML documents. In *Proc. 5th Intl' Workshop on the Web and Databases (WebDB)*, June 2002.

[16] L. Peters. Change detection in xml trees: A survey. In *4th Twente Student Conference on IT*, 2005.

[17] S. Selkow. The tree-to-tree editing problem. In *Information Processing Letters*, volume 6, pages 184–186, 1977.

[18] D. Shasha and K. Zhang. Approximate tree pattern matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.

[19] K. Tai. The tree-to-tree correction problem. *ACM Journal*, 26(3):422–433, 1979.

[20] J.T.L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 6(4):559–571, 1994.

[21] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proc. 22nd ACM SIGIR Intl' Conf. on Research and Development in IR*, pages 42–49. ACM Press, 1999.

[22] M. Zaki and C. Aggarwal. Xrules: An effective structural classifier for xml data. In *Proc. 9th Intl' Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[23] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *J. on Computing*, 18(6):1245–1262, 1989.

[24] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity metric for XML documents. In *Workshop on Knowledge and Experience Management (FGWM)*, 2003.