Multi-label classification and extracting predicted class hierarchies

Florian Brucker, Fernando Benites *, Elena Sapozhnikova

Department of Computer and Information Science, Box M712, University of Konstanz, 78457 Konstanz, Germany

A R T I C L E   I N F O

Article history:
Received 4 January 2010
Received in revised form 8 July 2010
Accepted 13 September 2010

Keywords:
Multi-label classification
Hierarchy extraction
Text mining
Adaptive resonance theory (ART)

A B S T R A C T

This paper investigates hierarchy extraction from results of multi-label classification (MC). MC deals with instances labeled by multiple classes rather than just one, and the classes are often hierarchically organized. Usually multi-label classifiers rely on a predefined class hierarchy. A much less investigated approach is to suppose that the hierarchy is unknown and to infer it automatically. In this setting, the proposed system classifies multi-label data and extracts a class hierarchy from multi-label predictions. It is based on a combination of a novel multi-label extension of the fuzzy Adaptive Resonance Associative Map (ARAM) neural network with an association rule learner.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

With the exponential growth of digital content on the Internet, in intranets and databases, it is becoming increasingly difficult to find and manage relevant materials. A successful way to cope well with the enormous information flow is to classify documents into hierarchically structured topics automatically. This task can be performed by various data mining and machine learning techniques, which learn to classify previously unseen instances from a set of labeled training instances.

Despite its importance, hierarchical classification has only gained more attention in the data mining community in the last decade. A number of approaches have been developed to take advantage of hierarchies in automatic document categorization [1–4]. One type of methods uses a divide-and-conquer approach to decompose the hierarchical problem into a set of smaller possibly non-hierarchical or at least easier-to-solve problems corresponding to splits in the hierarchy which can then be tackled using standard “flat” algorithms [2,4]. Each of these sub-problems is solved by a local classifier and the outputs of such classifiers are then combined. Another possibility is to incorporate the hierarchical information into a single “hierarchical” classifier [5].

Hierarchical classifiers have been shown to slightly outperform flat ones even in the single-label setting when each instance belongs to exactly one category [2]. However, hierarchical classification problems are often of the multi-label classification (MC) type when instances are labeled by multiple classes rather than just one. In such a classification task the goal is to predict a label set of an unknown size for each unseen instance. The combination of hierarchical classification and MC is referred to as hierarchical multi-label classification (HMC) and deals with classes which encode a set inclusion relation: instances of some class belong also to all its ancestors. Common examples include the organization of news paper articles into hierarchical categories (where, for example, “soccer” is a sub-category of “sports”) and the hierarchical arrangement of patent categories, cf. the datasets presented in Section 4. In the last few years interest in HMC grew due to the increasing importance of complex datasets, particularly in text mining and functional genomics, where an instance inherently belongs to more than one of the hierarchically organized classes. Recently, hierarchical multi-label classifiers have been developed both by using divide-and-conquer approaches [3,6–8] and by incorporating the class hierarchy information into a single classifier [5,9], the latter is also referenced as big bang approach [10].

Usually, hierarchical classification methods work with predefined class hierarchies, which are provided by domain experts. However, it is often difficult to maintain such a hierarchy when the number of classes is large or when the data are changing frequently. Therefore, there is a need for data-driven automatic hierarchy learning in many applications. For this reason we are interested in a more challenging HMC task where the hierarchy itself is unknown being part of the information to be found. The proposed data mining system was developed within the framework of the project DAMIART (Data Mining with Adaptive-Resonance-Theory-based neural networks) and consists of a multi-label classifier and an association rule learner. It produces class hierarchies by analyzing the relationships between labels.

* Corresponding author. Tel.: +49 7531 88 4765; fax: +49 7531 88 4805. E-mail addresses: florian.brucker@uni-konstanz.de (F. Brucker), fernando.benites@uni-konstanz.de (F. Benites), elena.sapozhnikova@uni-konstanz.de (E. Sapozhnikova).

0031-3203/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.09.010
predicted by the classifier. In contrast to techniques that extract hierarchies directly from the data, the use of a multi-label classifier for data mining enables hierarchy extraction (HE) even when available multi-label data are sparse or many original instances remain single-labeled. Thus a relatively small training set can be used for learning or the classifier can be trained online and then a large number of multi-label predictions is available for the association rule generator to derive a hierarchy from the output classes. An additional motivation point is that if a hierarchy can be extracted successfully from the data, it can also be used to improve the classification performance—for example, by filtering the labels that do not fit into the hierarchy.

To date there have been only few attempts to tackle HMC with unknown class hierarchies. In [4,11], methods based on hierarchical clustering were proposed for the automatic identification of hierarchical relationships between classes from single-label data. These works are closely related to a more general field of concept hierarchy and ontology learning from data [12] and more specifically from text (see e.g. [13–17]). In a recent work [18], the ART network was used for hierarchical clustering. To the best of our knowledge, the only study of HE from multi-label predictions is presented in [19]. In our work we followed the latter approach but extended it in several ways. First, in contrast to [19], where the default ARTMAP [20] neural network is used, our approach utilizes ML-ARAM [21], a multi-label extension of fuzzy adaptive resonance associative map (ARAM) [22]. The motivation for replacing default ARTMAP by ML-ARAM is that the postprocessing training step used in [19] excludes online learning, which is the main advantage of the ART networks. The online learning seems well suited when dealing with frequently changing data as, for example, in the newsgroups. Second, we examined the impact of using different multi-label classifiers on our system: two other multi-label classifiers, namely ML-kNN [23] and BoostTexter [24], were used to empirically evaluate system performance. Third, our approach extended [19] in the sense that datasets with predefined class hierarchies were used in order to assess the quality of extracted hierarchies. They can be quantitatively compared with the original hierarchies by means of special hierarchy proximity measures. The use of three such measures for comparing extracted hierarchies is investigated: in addition to the existing measures constrained tree edit distance (CTED) [25] and taxonomic overlap (TO) [26], we introduce a new distance measure named lowest common ancestor path distance (LCAPD), which is shown to be better suited for our needs. Another contribution of this work is the development of a quality estimator indicating how well the extracted hierarchy fits the data. It can be used for the quantitative assessment of the HE quality even if no original hierarchy is available. It should be noted that despite the fact we chose the classifier and the HE algorithm of a certain type, the parts of the proposed system are inherently independent and can be effectively combined with any other suitable method.

The paper is organized as follows: After introducing a general description of the problem to be solved in Section 2, we present our approach in Section 3. Experimental results evaluating the proposed system on multi-label datasets with known hierarchies are described and discussed in Section 4. Finally, Section 5 concludes the paper.

2. Problem description and notation

2.1. Multi-label classification

Traditional (single-label) classification is the task of learning a mapping from a set of instances $X$ into a set of labels $L = \{1, \ldots, q\}$ from $n$ given instance-label pairs $(x^{(i)}, l^{(i)})$ with $x^{(i)} \in X$, $l^{(i)} \in L$ for $s = 1, \ldots, n$. Therefore, each instance is associated with exactly one label. In MC, each instance is allowed to have one or more than one label. Thus the task is to learn a mapping from $X$ into the power set $2^L$ of $L$ from the training data $(x^{(i)}, m^{(i)})$ with $x^{(i)} \in X$ and $m^{(i)} \subseteq L$. In the following $q \geq 2$ will be assumed. Any subset $m \subseteq L$ is called a multi-label. To simplify notation, multi-labels are often represented as binary vectors: Given a multi-label $m$ and a label $i \in L$, set $m_i = 1$ if and only if $i \in m$, and $m_i = 0$ otherwise.

2.2. Hierarchical multi-label classification

HMC extends MC by imposing a hierarchy structure on the multi-labels: A hierarchy $H$ on $L$ is considered as a rooted labeled directed tree, i.e. an arrangement of $L$ in tree form.

A hierarchy $H$ is required to contain each label $i \in L$ exactly once, and to simplify presentation it is further assumed that the root of the tree has the label $0 \not\in L$. Since there is a one-to-one mapping between hierarchy nodes and labels we usually do not distinguish between them. Thus $H$ can be interpreted as a set of edges $H \subseteq (L \cup \{0\}) \times L$ where $(i, j) \in H$ if and only if $i$ is the parent of $j$. The parent of $i$ in $H$ is denoted by $p_H(i)$, and the set of all ancestors of $i$ in $H$ (excluding $i$) is denoted by $A_H(i)$.

The connection between multi-labels and a hierarchy $H$ is as follows: If a multi-label $m$ contains a certain label $i \in L$, it should also contain its parent and thus all its ancestors:

$$p_H(i) \in m \text{ for all } i \in m \text{ if } p_H(i) \neq 0. \quad (1)$$

In this example, $p(4) = 2$, $p(2) = 1$, and $p(1) = 0$, thus the multi-label $m'$ is consistent with $H_1$ since it satisfies (1). $m'$, on the other hand, is not consistent with the hierarchy, since $p(3) = 1 \notin m'$.

It is important to note that the prediction task is still the same as in traditional MC, and that the hierarchy itself is not part of the training data. Thus, while the original multi-labels in an HMC dataset are assumed to be consistent with the corresponding hierarchy, this cannot, in general, be true for predicted hierarchical multi-labels. How well the predicted multi-labels adhere to the hierarchy depends heavily on the used classification algorithm.

2.3. Hierarchy extraction (HE)

Given a family of multi-labels $M$ which is consistent with a certain hierarchy, if and how that hierarchy can be reconstructed from the multi-labels is an obvious question. It is also interesting how can this be accomplished if the multi-labels do not strictly adhere to the hierarchy, as in the case of predicted multi-labels? This task of finding the “proper” hierarchy for $M$ is called HE. It is important to point out that in most cases there exist many different hierarchies for the same set of multi-labels: For example, the trivial hierarchy $(\{0\}, H \subseteq L)$ is consistent with any family of multi-labels. This motivates the following definitions: A generalization operation on a hierarchy $H$ consists in replacing a link $(p, c) \in H$ satisfying $p \neq 0$ with a link $(q, c)$ where $q = p_H(p)$. If $H'$ can be obtained from $H$ via a non-empty sequence of generalization operations, $H'$ is called a generalization of $H$. In that case $H'$ is also said to be more general than $H$, and $H$ to be more specific than $H'$. We write $H \ll H'$ if and only if $H' = H$ or $H' \ll H$. A hierarchy that is more specific than another includes stricter rules about the relationships of the labels. This implies...
that more general hierarchies are, in general, flatter. Consider, for example, the hierarchies shown in Fig. 1. H3 can be generated from H1 by replacing the link (2, 3) with the link (1, 3), which constitutes a generalization operation since 1 ≈ 2 in H1. Therefore, H1 < H3, and similarly H2 < H3 and H2 < H3 < H2. On the other hand, 2 is an ancestor of 3 in H3, while in H2 it is the other way round. This means that neither of H2 and H4 is more general than the other since generalization operations never create new ancestors.

Since the goal is to find a consistent hierarchy that captures as much of the information contained in the multi-labels as possible, it is natural to ask if and when there is a single consistent hierarchy that is more specific than any other consistent hierarchy. This is formalized as follows: If a hierarchy H is consistent with M and there is no other hierarchy H’ that is also consistent with M while satisfying H ≺ H’ then H is said to be maximally consistent with M. In the event that there is exactly one hierarchy H that is maximally consistent with M, H is said to represent H.

The requirements for M to represent a (not necessarily known) hierarchy are reasonable. For a label i ∈ L define the set of multi-labels containing i as M_i = \{m ∈ M | i ∈ m\}. M is said to partition two labels i, j ∈ L, i ≠ j, if M_i ≠ M_j, i.e. if there is at least one multi-label m ∈ M that contains exactly one of i and j. M partitions L if M partitions any two labels i, j ∈ L, i ≠ j. If i, j, k ∈ L are three different labels satisfying M_i ⊆ M_j and M_j ⊆ M_k while neither M_j ⊆ M_i nor M_k ⊆ M_j holds, then (i, j, k) is called an undetermined triple in M with respect to L.

The following basic result is easily verified:

**Lemma 1.** M represents a hierarchy on L if and only if it partitions L and there are no undetermined triples in M with respect to L.

It turns out that in this case the corresponding hierarchy can actually be reconstructed, for example by using the algorithm presented in Section 3.2. The conditions are sensible: If M does not partition two labels i, j ∈ L both labels are, from the multi-labels’ perspective, identical and one of them can be removed. The second condition ensures that two possible parents for a label i ∈ L can be arranged within a hierarchy in a consistent way such that both are ancestors of i.

The following example illustrates this result. Assume we are given three multi-label families

\[ M = ((1),(2,3),(1,2,3,4)), \]
\[ M' = ((1,3),(1,2,3,4)), \]
\[ M'' = ((1,2),(1,3),(1,2,4)) \]

over the label set \( L = \{1, 2, 3, 4\} \). As M’ does not partition labels 2 and 3, it does not represent a hierarchy on L, therefore at least two different hierarchies on L exist that are both maximally consistent with M’ (Figs. 1c and d). Note that M’ does satisfy the second condition of Lemma 1. M’ does partition L, but does not satisfy the second condition of Lemma 1: Although \( M'_2 \subseteq M'_3 \) and \( M'_2 \subseteq M'_3 \) hold, neither \( M'_2 \subseteq M'_4 \) nor \( M'_2 \subseteq M'_4 \) does. Two different hierarchies that are both maximally consistent with M’ are shown in Figs. 1a and b. Finally M’’ fulfills both conditions of Lemma 1 and therefore represents a hierarchy on L, which is shown in Fig. 1a.

Lemma 1 is a satisfying result for multi-labels that are consistent with a certain hierarchy, but in the case of predicted multi-labels, seeking maximal consistency may lead to inferior results: The problem is that predicted multi-labels are not expected to be fully consistent with the original hierarchy. Searching for a hierarchy that is maximally consistent with the predicted labels would therefore produce a hierarchy in which many labels are positioned too close to the virtual root. This is because even a single predicted multi-label that is not consistent with a certain parent-child relationship by mistake will prevent that relationship from being included in the hierarchy. In other words: An algorithm that produces maximally consistent hierarchies has no error tolerance. A good HE algorithm therefore has to trade off consistency for level of detail: The more details the constructed hierarchy has, the higher the number of multi-labels not consistent with it usually is.1

---

1 In the algorithm presented in Section 3.2, this is controlled by the minimum confidence parameter.
Methods for evaluating the performance of an HE algorithm usually require that the original hierarchy is known. In that case performance can be measured, for example, by calculating the distance between the extracted and original hierarchies using measures like those presented in Section 3.4. If the original hierarchy is unknown, heuristics can be applied to estimate the quality of extracted hierarchies. One such heuristic is given by the quality estimator QE as described in Section 3.2.

Instead of using labeled trees to represent hierarchies it is also possible to use labeled directed acyclic graphs. The concepts and techniques described in this article can easily be extended to this more general case. Labeled trees were chosen here because they simplify presentation and because most hierarchical multi-label datasets have tree-like hierarchies.

3. Method

The proposed data mining system consists of an ART-based multi-label classifier ML-ARAM and an association rule learner, but it is not restricted to the use of a certain multi-label classifier and HE algorithm. For performance comparison two other state-of-the-art multi-label classifiers, namely BoosTexter and ML-kNN were used.

3.1. Multi-label classifiers

3.1.1. Fuzzy ARAM

ARAM belongs to the family of Adaptive Resonance Theory (ART) neural networks [27] and performs incremental supervised learning of pattern pairs. It can be visualized as two overlapping Fuzzy ART modules ART_a and ART_b consisting of two input fields \( F_a \) and \( F_b \) of neurons connected through weights \( W^a \) and \( W^b \) by a common category field \( F_2 \) (Fig. 2). During learning, given an input pattern \( \tilde{A} \) presented at \( F_a \) and an output pattern \( \tilde{B} \) presented at \( F_b \), an \( F_2 \) category node is selected to encode the pattern pair. In classification problems, \( F_a \) serves as the input field for the feature vectors and \( F_b \) for the class vectors. When performing a classification task, ARAM incrementally creates prototypes for input patterns, and associates each prototype with its respective class. Only the basic concepts of the algorithm are discussed below, more details can be found in [28].

At the beginning of the training process, all weight vectors \( \tilde{W}_k \) \((k = 1, \ldots, N)\) are set to unity and said to be uncommitted. When learning an \( M^a \)-dimensional feature vector \( \tilde{A} \) and its \( M^b \)-dimensional class vector \( \tilde{B} \), the vectors are first converted into the complement-codified vectors \( \tilde{A} = (a, a') = (A^1, \ldots, A^{M^a}, 1-A^1, \ldots, 1-A^{M^a}) \), \( \tilde{B} = (b, b') = (B^1, \ldots, B^{M^b}, 1-B^1, \ldots, 1-B^{M^b}) \). (2)

This results in an equal-length normalization of the input vectors because the used \( l_1 \) norm of complement-coded vectors is constant (analogically for \( \tilde{B} \)):

\[
|\tilde{A}| = \sum_{i=1}^{M^a} a_i + \left( M^a - \sum_{i=1}^{M^a} a_i \right) = M^a.
\]

ARAM then computes the activation function \( T_k \) for each of the \( F_2 \) nodes:

\[
T_k(\tilde{A}) = \frac{|\tilde{A} \land \tilde{W}^a|}{\tilde{A} + |\tilde{W}^a|}.
\]

Here, \( \land \) denotes the fuzzy AND, element-wise min operator, and \( \alpha > 0 \) is called the choice parameter. Then the maximally activated node \( K \) is selected to be a winner according to the winner-take-all (WTA) choice rule

\[
T_K = \max(T_k : k = 1, \ldots, N).
\]

Before node \( K \) can be used for learning, the winner choice should be confirmed by the pair of match criteria checking that its weight vectors are sufficiently close to their respective feature and class vectors:

\[
|\tilde{A} \land \tilde{W}^a| \geq \rho_a, \quad |\tilde{B} \land \tilde{W}^b| \geq \rho_b,
\]

where \( \rho_a, \rho_b \in [0,1] \) are the user-defined vigilance parameters. A match of an input and a prototype stored in the weight vector of \( K \) is sufficient if their similarity is above the value of the vigilance parameter. Selecting lower vigilance values leads to building broader prototypes while larger values cause the creation of more specific prototypes. Additionally, lower values lead to higher memory compression than large values. If any of the inequalities fails, the system inhibits the winning node \( K \) and enables the category with next highest activation to be selected. This search process continues until the input is either assigned to an existing (committed) node that satisfies both match criteria or to a new (uncommitted) neuron. Thus uncommitted neurons are added if required. A successful end of search is followed by the learning process according to

\[
\tilde{W}^a_k \text{new} = \beta_a (\tilde{A} \land \tilde{W}^a_k \text{old}) + (1-\beta_a) \tilde{W}^a_k \text{old},
\]

\[
\tilde{W}^b_k \text{new} = \beta_b (\tilde{B} \land \tilde{W}^b_k \text{old}) + (1-\beta_b) \tilde{W}^b_k \text{old},
\]

where \( \beta_a, \beta_b \in [0,1] \) is the learning rate. During learning the selected \( F_2 \) node \( K \) learns to encode the input and output vectors by adjusting its weight vectors \( \tilde{W}^a_k \) and \( \tilde{W}^b_k \). The fast learning mode corresponds to setting \( \beta_a, \beta_b = 1 \). It immediately encodes an input vector into the weights as opposite to more general slow learning over lengthy training cycles which is common for neural networks. Fast learning is a typical feature of ART-based networks which allows them to add new information on the fly and to be trained online.

3.1.2. ML-ARAM

Although ARAM enables multi-label learning, its direct use for solving an MC task can be ineffective due to the WTA choice rule. It allows only the most highly activated category to be selected. Though justified for mutually exclusive classes, this can lead to poor performance in the multi-label setting. Thus, it would be advantageous to utilize distributed activation during the classification stage as in [20]. In the case of MC, a better prediction can be made by joining the class information of those categories that are approximately maximally activated. This can be achieved by combining the individual predictions of several most activated nodes as follows: First, a set of \( N_b \) best categories with the largest activation values is chosen according to the rule: a category \( k \) is included in the set, if the relative difference \( (T_k - T_k) / T_K \) is within
a predefined fraction \( t \) of the activation range \( r = T_K - T_{\text{min}} \), where \( T_{\text{min}} \) is the minimum activation of committed neurons. To simplify presentation, assume that the categories 1, \ldots, \( N_b \) are the result of this selection. Then the activations of the \( N_b \) categories are normalized:

\[
    u_k = \frac{T_k}{\sum_{i=1}^{N_b} T_i}.
\]

This content addressable memory (CAM) rule [29] differs from the power rule used in [30] and is computationally less expensive. The resulting distributed output pattern \( \vec{P} \) is made by calculating a weighted sum of the \( N_b \) predictions \( W_{x,b}^i \):

\[
    \vec{P} = \sum_{i=1}^{N_b} u_i W_{x,b}^i.
\]

Thus, \( \vec{P} \) contains a score for each label that is proportional to the frequency of predicting this label among the \( N_b \) best categories. Since the fast learning of ART networks leads to varying performance with the input presentation order, voting across several networks trained with different orderings of the training set is usually utilized. During a test stage, several networks trained on different sequences of training patterns provide their predictions for a given test pattern.

Voting typically improves classification performance [31] and reduces its variability. It also presents a useful option for averaging multi-label predictions. So, multiplying the number of training instances closer to \( x \) to \( \{1, \ldots, N_b\} \) (called \( A \)) and assigning it to \( \{1, \ldots, N_b\} \) (called \( B \)) enables the sum to be built over all voters and produces a collective distributed output pattern \( \vec{P} \), which may then be used to determine the predicted classes.

An additional modification is made in the ART\( _b \) weights, which now count label frequencies for each ART\( _b \) category during learning. After commitment they are increased by 1 each time the corresponding label occurs. The weight matrix \( W_{x,b} \) then contains information about the frequency with which each label was coded by a node \( k \).

Finally, a postprocessing filter method [19] is applied: the signals \( \vec{P} \) are sorted in descending order, and all corresponding output classes are included in the multi-label up to the point of maximum decrease in the signal size from one class to the next.

3.1.3. ML-kNN

In single-label classification, the basic \( k \)-nearest-neighbor (kNN) classifier works as follows: To classify an instance \( x \), first search for its \( k \) nearest neighbors \( N_k(x) \) in the set of training instances \( \mathcal{I} \). Among these neighbors, check which label occurs most often, and assign it to \( x \). A possible refinement of the algorithm is to weight the labels of the neighbors according to their inverse distance from \( x \), thereby increasing the influence of training instances closer to \( x \).

In [23] ML-kNN is presented, which is an MC algorithm derived from the traditional kNN classifier. For an instance \( x \), denote its multi-label by \( m(x) \subseteq L \). Based on the \( k \) nearest neighbors of \( x \) in the training set \( N_k(x) \subseteq \mathcal{I} \) define a membership counting vector \( C(x) \) by

\[
    C(x)_i = \sum_{y \in N_k(x)} m(y)_i \quad \text{for } i \in L.
\]

That is, for each \( i \in L \), \( C(x)_i \) is the number of instances in \( N_k(x) \) whose multi-labels contain \( i \). For each label \( i \in L \), one then defines the two events “the multi-label of \( x \) contains \( i \)” and “the multi-label of \( x \) does not contain \( i \)”, which are denoted by \( h^i \) and \( h^i \), respectively. Denote by \( e^i_r \) the event that exactly \( r \) of the \( k \) neighbors in \( N_k(x) \) have label \( i \), where \( r \in \{0, \ldots, k\} \). To decide whether \( i \) will be included in the predicted multi-label of \( x \), ML-kNN computes the posteriori probabilities of \( h^i \) and \( h^i \), given the multi-label distribution among the neighbors \( N_k(x) \). These probabilities are denoted by \( P(h^i | e^i_r(x)) \) and \( P(h^i | e^i_r(x)) \), respectively. The predicted multi-label \( \hat{m}(x) \) for \( x \) is then determined using the following maximum likelihood principle:

\[
    \hat{m}(x)_i = \arg\max_{b \in \{0,1\}} P(h^i | e^i_r(x)) \quad \text{for } i \in L.
\]

That is, \( i \) is included in \( \hat{m}(x) \) if and only if \( P(h^i | e^i_r(x)) > P(h^i | e^i_r(x)) \). Using Bayes’ Law, this can be rewritten as

\[
    \hat{m}(x)_i = \arg\max_{b \in \{0,1\}} P(h^i)^{P(\epsilon | m(x))} \quad \text{for } i \in L.
\]

The prior probabilities \( P(h^i) \) and posterior probabilities \( P(\epsilon | m(x)) \) are estimated from the computation set \( \mathcal{I} \) based on frequency counting:

\[
    P(h^i) \approx \frac{|\mathcal{I}_i|}{|\mathcal{I}|} \quad \text{for } i \in L,
\]

\[
    P(h^i) = 1 - P(h^i) \quad \text{for } i \in L,
\]

\[
    P(\epsilon_i | m(x)) \approx \frac{|\{y \in \mathcal{I} | C(y)_i = r\}|}{|\mathcal{I}|} \quad \text{for } i \in L, \ r \in \{0, \ldots, k\},
\]

\[
    P(\epsilon_i | h^i) \approx \frac{|\{y \in \mathcal{I} | C(y)_i = r\}|}{|\mathcal{I}|} \quad \text{for } i \in L, \ r \in \{0, \ldots, k\}.
\]

Here, \( \mathcal{I}_i \) is the set of training instances whose multi-labels contain \( i \): \( \mathcal{I}_i = \{y \in \mathcal{I} | i \in m(y)\} \).

3.1.4. BoosTexter

Boosting is a technique to combine multiple weak classifiers into a single strong classifier with the objective of outperforming the classifiers it has been built from. BoosTexter [24] is an implementation of this method for the field of text categorization, based on the well-known AdaBoost algorithm [32]. It linearly combines simple linear classifiers into a strong learner treating each label independently.

During learning the algorithm creates on each iteration step a new weak hypothesis focusing on samples which were incorrectly classified by the already existing hypotheses. This focus is given by a distribution calculated for each document (sample) \( x^{(s)} \) and label \( i \) of the training data based on its classification by the last weak hypothesis produced and the actual distribution representing indirectly all the other created hypotheses. This distribution has higher values for incorrectly classified samples and small values otherwise. The new hypothesis should rank better those samples which have not been properly learned by the existing hypotheses by paying more “attention” to high values of the distribution. The weak hypotheses are represented as one-level decision trees also called decision stumps. These simple rules have two outcome values indicating that a single input feature of the document is conform to the rule or not.

At the first iteration the distribution is uniform: \( D_1 = 1/nq \). The distribution \( D_k \) is then passed at each iteration \( k \) to the weak learner, which creates a new hypothesis \( h_k \). Then the distribution is updated for the next iteration:

\[
    D_{k+1}(s,i) = \frac{D_k(s,i) \exp(-y^{(s)}_i h_k(x^{(s)},i))}{Z_k}
\]

The true label \( y^{(s)}_i \) of the document \( s \) is defined in this context as

\[
    y^{(s)}_i = \begin{cases} 
        +1 & \text{if } i \in m^{(s)}, \\
        -1 & \text{if } i \notin m^{(s)}.
    \end{cases}
\]

The weak hypothesis \( h_k(x^{(s)}, i) \) will be explained next, followed by the definition of \( Z_k \). To obtain \( h_k(x^{(s)}, i) \), there are some
intermediate steps: For \( u \in \{0,1\} \) define
\[
W^{\text{init}}_u = \sum_{i=1}^{n} D_b(s)(R_b(x^{(s)}) = u \land x^{(s)} = b),
\]
where \( b \in \{-1, +1\} \), and \( \{\Phi\} = 1 \) if \( \Phi \) is true, and \( \{\Phi\} = 0 \) otherwise. \( R_b(x^{(s)}) = 1 \) if the document \( x^{(s)} \) meets the requirements of the rule \( R_b \), and 0 otherwise. In our case it means that the value of a specific feature \( x^{(s)}_j \) of a document \( x^{(s)} \) is above a threshold \( v_k \) or not. The next step is to calculate the \( c^{\text{init}} \), i.e. the weights for each label for the hypothesis \( k \) for each (both) outcome(s) of the rule \( R_k \):
\[
c^{\text{init}}_k = \frac{1}{2} \ln \left( \frac{W^{\text{init}}_{\text{pos}} + e}{W^{\text{init}}_{\text{neg}} + e} \right),
\]
where \( u \in \{0,1\}, u = R_k(x^{(s)}) \) indicates if the rule \( R_k \) is obeyed, and \( e = 1/nq \) is a “smoothing” term to avoid over-fitting. Without \( e \), \( c^{\text{init}} \) could grow arbitrarily large since \( W^{\text{init}}_{\text{pos}}, W^{\text{init}}_{\text{neg}} \in \{0,1\} \). Then \( h_k(x^{(s)}, j) \) is chosen as
\[
h_k(x^{(s)}, j) = \begin{cases} c^{\text{init}}, & \text{if } R_k(x^{(s)}) = 0, \\ c^{\text{init}}, & \text{otherwise}. \end{cases}
\]
This means that if the document complies with the rule \( R_k(x^{(s)}) \), the hypothesis \( h_k(x^{(s)}, j) \) will be the constant \( c^{\text{init}} \), otherwise it will be \( c^{\text{init}} \). The feature and the threshold \( v_k \) for the rule \( R_k \) are chosen so that it minimizes \( Z_k \):
\[
Z_k = 2 \sum_{u \in \{0,1\}} \sum_{s=1}^{S} \sqrt{W^{\text{init}}_{\text{pos}} + 1 W^{\text{init}}_{\text{neg}}}. 
\]
\( Z_k \) is also a normalizing factor for the distribution \( D_b \) as in (5). The generation of hypotheses is iterated until the maximum number of iterations (boosting rounds) \( K \) defined by the user is reached. This parameter influences the generalization of the classifier, increasing it will make BoosTexter creates more hypotheses covering every time better the training data, which also increases the danger of over-fitting.

After learning the strong learner predicts for a document \( x^{(s)} \) the label \( i \) as
\[
\hat{y}^{(s)}_i = \text{sign} \left( \sum_{k=1}^{K} h_k(x^{(s)}, j) \right).
\]
**Summarizing the classification procedure:** After learning BoosTexter has a set of hypotheses for each label, each hypothesis \( k \) has two values \( c_k \) and a rule \( R_k \). Each rule is based on a specific feature of the input vector \( x^{(s)} \) and a threshold \( v_k \). The value \( c^{\text{init}} \) is then chosen as \( h_k(x^{(s)}, j) \) by the rule \( R_k \) if \( x^{(s)}_j \) is above \( v_k \) else \( c^{\text{init}} \) is chosen. Then all \( h_k(x^{(s)}, j) \) are summed and the label \( i \) is taken into the predicted multi-label \( \hat{m}^{(s)} \) if the sum is positive.

### 3.2. Hierarchy extraction

#### 3.2.1. Algorithm

Similarly to [19], an association rule learner related to the Apriori-algorithm [33] was used for extracting hierarchies from multi-label predictions. However, our HE algorithm is essentially different from that of [19]: First, it produces trees instead of directed acyclic graphs. Second, it does not contain a mechanism for detecting equivalent labels because in our experiments the labels are known to be distinct.

The proposed HE algorithm works as follows: Let \( M \) be a family of multi-labels and \( 0 \leq c < 1 \) a minimum confidence parameter.

1. For each pair of labels \( ij \in L \), where \( i \neq j \) and \( M_i \neq \emptyset \), a rule \( i \rightarrow j \) with confidence \( c_{ij} = |M_i \cap M_j|/|M_i| \) is created.

2. Any rule \( i \rightarrow j \), for which \( c_{ij} < c \) holds, is discarded.

3. For any two labels \( ij \in L \), \( i \neq j \), the one rule of \( i \rightarrow j \) and \( j \rightarrow i \) which has the lower confidence value is discarded.

4. To each label \( i \in L \), a level \( l_i \) is assigned according to the following procedure: level 1 is assigned to all labels not appearing as consequents in any of the retained rules. All rules \( j \rightarrow k \) having \( l_j = 1 \) are marked. The next level is assigned to all labels not appearing as consequents in any unmarked rule. All rules \( j \rightarrow k \), where \( j \) is in that level, are marked. This procedure is repeated until a level has been assigned to each label or no unmarked rules remain. The next level is then assigned to any labels to which no level has been assigned yet. This procedure ensures that the more often a label appears as a consequent, the higher its level is. Therefore, labels with a lower level are located closer to the leaves of the hierarchy.

5. A set of parent candidates \( C_i \) is assigned to each label \( i \in L \), consisting of all labels \( j \in L \setminus \{i\} \) whose level \( l_j \) is higher than that of \( i \):
\[
C_i = \{ j \in L \setminus \{i\} \mid l_j > l_i \}.
\]

By only selecting labels with a higher level, the extracted hierarchy is guaranteed to be acyclic. From these candidates, the ones with the lowest level are selected, which ensures that no grand-parent or higher ancestor is accidentally chosen as a parent:
\[
C^* = \left\{ j \in C_i \mid l_j = \min_{k \in C_i} l_k \right\}.
\]

6. To construct the tree, each label \( i \) is assigned to a unique parent. The most promising candidate is chosen as a parent, i.e. that label \( j \in C^*_i \) for which the confidence \( c_{ij} \) of the rule \( i \rightarrow j \) is maximal:
\[
p(i) = \arg \max_{j \in C^*_i} c_{ij}.
\]

If \( C^*_i = \emptyset \) (and therefore \( C^*_i = 0 \)), the virtual root 0 is chosen as the parent of \( i \): \( p(i) = 0 \).

Note that the choice of \( j \in C^*_i \) in Step 6 might not be unique, i.e. there may be more than one label \( j \in C^*_i \) such that \( c_{ij} = \max_{k \in C^*_i} c_{ik} \). From a theoretic point of view, all such labels are equally suited as a parent of \( i \), and none of them is an “inherently better” choice. Analog to Lemma 1, such a situation occurs if the “ideal” hierarchy is not uniquely determined.2 Similarly, it may be the case that there are two labels \( ij \in L \) for which both rules \( i \rightarrow j \) and \( j \rightarrow i \) have the same confidence. In that event, it is not clear which one should be removed in Step 3. Again, either choice of removing \( i \rightarrow j \) or \( j \rightarrow i \) is, from a theoretic viewpoint, equally justified. For the sake of reproducibility, our implementation chooses among all labels in \( C^*_i \) with maximal confidence that label which has the smallest index; if \( c_i = c_j \geq c \) and \( l_i < l_j \), the rule \( j \rightarrow i \) is discarded.

The following simple example illustrates the influence of the parameter \( c \). Assume that for a dataset featuring the label set \( L = \{1, 2, 3, 4\} \) a classification algorithm produced the following predicted multi-labels:
\[
\{(1,2), (1,2,4), (1,3), (1,2,3,4), \} \text{ and } (1,2,3,4).
\]

Table 1 shows the confidence values \( c_{ij} \) for these multi-labels. A setting of \( c = 4/5 \) produces a hierarchy that is consistent with the

---

2 One has to be careful not to argue along the lines of maximal consistency here, since hierarchies extracted by the algorithm are usually not maximally consistent due to the usage of rules with a confidence lower than 1. As noted in Section 2.3, this is often desirable in the case of non-consistent (e.g. predicted) multi-labels.
multi-labels. In this case, only the rules with confidence 1, namely 2→1, 3→1, 4→1, and 4→2, remain. Due to the sorting in Step 4, the result of the algorithm for c>4/5 is thus the hierarchy H_1, depicted in Fig. 1a. The same hierarchy is extracted for 2/3<c<4/5, since in that case the additional remaining rule 1→2 is removed in Step 3. If c=2/3, then the additional remaining rule 3→2, which is not removed in Step 2, causes 2 to be made a child of 2, resulting in the hierarchy H_2 as depicted in Fig. 1c. Again the same result is obtained for 1/2<c<2/3, since the additional remaining rule 1→3 is removed in Step 3. Finally, for a setting of c≤1/2, all rules except those removed in Step 3 remain, producing a strictly lower triangular matrix, and the algorithm returns H_3 as depicted in Fig. 1c.

Thus even on this extremely simple example the algorithm yields three different results, depending on the choice of c. When comparing these results, one notices that, as c is decreased, the extracted hierarchies get more specific. On the other hand, the higher c is, the more consistent the multi-labels and the extracted hierarchies are.

3.2.2. Quality estimation

A useful addition to the presented HE algorithm is the automatic calculation of a quality estimator: If the true hierarchy of a dataset is not known it can be used to estimate how well the extracted hierarchy fits the data. The idea is that for consistent multi-labels, the rule that links a label i to its parent p(i) should have confidence c_{p(i)} = 1. For inconsistent multi-labels this will usually not hold. Thus a low value of (1-c_{p(i)}) means that the "label → parent" link fits the data well, whereas a high value indicates a poor correspondence. Additionally, the parents coming from the next level of the hierarchy should be preferred to the parents coming from higher levels, this is measured by the level difference the parent of a label ldi has to its parent. The quality estimator (QE) for the whole extracted hierarchy is defined as

\[ QE = 1 - \frac{1}{q} \sum_{i=1}^{q} \frac{c_{p(i)}}{1+ldi} \]

where c_{0}=0 was set for any label i for which the virtual root 0 was chosen as parent in order to penalize shallow hierarchies, in which nodes are linked directly to the root. Our experiments show that for datasets with known hierarchies, the value of QE for the extracted hierarchy is highly correlated with the tree distances between the original and the extracted hierarchy (see Table 2, Section 3.4 and Fig. 5). The quality estimator is thus a valuable heuristic, useful, for example, for selecting an appropriate value for the minimum confidence parameter.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classifier</th>
<th>Correlation of QE with</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LCAPD</td>
</tr>
<tr>
<td>20 newsgroups (all)</td>
<td>ML-ARAM</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
<tr>
<td>20 newsgroups (20%)</td>
<td>ML-ARAM</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
<tr>
<td>20 newsgroups (30%)</td>
<td>ML-ARAM</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
<tr>
<td>20 newsgroups (40%)</td>
<td>ML-ARAM</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
<tr>
<td>20 newsgroups (50%)</td>
<td>ML-ARAM</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
<tr>
<td>RCV1-v2</td>
<td>ML-ARAM</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
<tr>
<td>WIPO-alpha</td>
<td>ML-ARAM</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>ML-kNN</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>BoostTexter</td>
<td>0.99</td>
</tr>
</tbody>
</table>

For each dataset and classifier, hierarchies were extracted from the corresponding prediction, and compared to the true hierarchy (using the distance measures from Section 3.4) for 100 linear separated settings of the minimum confidence parameter in the range of [0,1]. This table shows the correlation between the resulting QE values and the tree distances.

3.3. Multi-label classification performance measures

MC experiments of the next section were evaluated by means of nine performance measures. First, we adopted two most commonly used example-based measures for multi-label predictions: Accuracy (A) and F-measure (F). Given a family S = \{(x^{(i)}, m^{(i)})\}_{i=1}^{n} of n test examples where m^{(i)} is the true multi-label for an instance x^{(i)}, let \hat{m}^{(i)} be the predicted multi-label for x^{(i)}. Accuracy measures how many of the predicted labels are actually present:

\[ A = \frac{1}{n} \sum_{i=1}^{n} \frac{|m^{(i)} \cap \hat{m}^{(i)}|}{|m^{(i)} \cup \hat{m}^{(i)}|} \]

The larger the A value, the better the MC performance. F-measure is the harmonic mean of precision and recall calculated on a per-instance basis:

\[ F = \frac{1}{n} \sum_{i=1}^{n} \frac{2|m^{(i)} \cap \hat{m}^{(i)}|}{|m^{(i)}| + |\hat{m}^{(i)}|} \]

The closer the F value is to 1, the better the performance.

Since these measures are based on the comparison of multi-labels, they are threshold dependent. Additionally, ranking-based performance measures, namely one-error (OE), ranking loss (RL), coverage (C), and average precision (AP) were used. One-error evaluates how many times the top-ranked label is not in the set of proper labels of the instance. The performance is perfect when OE equals to 0. Ranking loss is defined as the average fraction of pairs of labels that are ordered incorrectly. Coverage evaluates how far we need, on average, to go down the list of labels in order to cover all the proper labels of the instance. The smaller its value, the better the performance. Average precision evaluates the average fraction of labels ranked above a particular label i in m^{(i)} which actually are in m^{(i)}. If it is equal to 1, the performance is perfect.

Besides the above discussed example-based performance measures, label-based ones are also often utilized. They are calculated using binary evaluation measures for each label: the
counts for true positives (tp), false negatives (fn), and false positives (fp). The most popular of them are recall, precision, and F1-measure. We used the micro-averaged version of F1 with the binary evaluation measures counted on the whole test set:

$$F1 = \frac{2 \sum_{i \in L} tp_i}{\sum_{i \in L} tp_i + \sum_{i \in L} fp_i + \sum_{i \in L} fn_i}.$$

The perfect performance corresponds to F1 = 1. Moreover, micro-averaged precision and recall were used for computing the area under a precision-recall curve (AUPRC) [5]. Micro-averaged precision corresponds to the proportion of predicted labels in the test set that are correct and recall to the proportion of labels that are correctly predicted. AUPRC has been claimed to be a well-suit performance measure for MC tasks where the number of negative instances significantly exceeds the number of positive instances [5]. Another advantage of AUPRC is its global nature and independence of a certain threshold value. The closer the AUPRC value is to 1, the better the performance.

And finally, the special hierarchical loss function proposed in [6] was used for performance evaluation. Hierarchical loss (H-loss) does not consider mistakes made in subtrees of incorrectly predicted labels and penalizes only the first mistake along the path from the root to a node:

$$H-loss = \frac{1}{n} \sum_{i=1}^{n} \sum_{j \in L} c_i m_j^{(i)} - m_j^{(i)} \land \forall k \in A_i(j) : m_k^{(j)} = m_k^{(i)}.$$

where $c_i$ are label-dependent costs and for a predicate $\Phi$ the term $[\Phi]$ is set to 1 if $\Phi$ is true, and $[\Phi]=0$ otherwise. Following [7], normalized costs were used: $c_i = 1/(c(p(i)), (i \in L)$, where $c(i)$ is the set of all direct children of $i$. The smaller the $H$-loss value, the better the performance.

3.4. Proximity measures for hierarchies

In order to evaluate the quality of extracted hierarchies, a proper proximity measure is required. Measuring the proximity of graphs (also known as graph matching [35]) and trees is a common problem in many fields of science and engineering. Many algorithms have been proposed to solve it, often adapted to the special needs of a given field. More generic approaches use tools from graph theory, like the concept of maximum common subgraphs or the generalization of the well-known string edit distance to graphs [35] and trees (called the tree-to-tree correction problem) [36]. Specialized algorithms, on the other hand, draw on knowledge of the underlying models represented by the graphs. Examples include comparing object-oriented case representations for case-based reasoning [37], conceptual graphs for natural language processing [38] and ontology comparison [39], as well as phylogenetic trees in molecular evolution studies [40].

We use three different proximity measures: the constrained tree edit distance (CTED) [25] as a representative of the popular edit distances, taxonomic overlap (TO) [26] from the field of concept representation and lowest common ancestor path distance (LCAPD). The latter is a new tree distance measure, developed especially for the type of hierarchies common in HMC. A description of the mutual differences between the three measures is given at the end of this section.

The depth $d_L(i)$ of a label $i \in L$ is defined recursively by $d_L(0) = 0$ and $d_L(i) = d_L(p(i)) + 1$ for $i \neq 0$. The degree of a node in a graph is the number of edges incident to the node. In the case of a tree, this is the number of the node’s children plus one.

3.4.1. Constrained tree edit distance

CTED [25] is based upon the idea of transforming one hierarchy into another via certain local editing operations, each of which has a cost attached to it. The distance between two hierarchies is the minimum total cost of transforming the one into the other. Three modification operations are considered: A node $i$ is deleted by transmitting its children to its parent and removing $i$ from the hierarchy. Inserting is the complement of deleting: Inserting $i$ as a child of $j$ makes $i$ the parent of some subset of the children of $j$. Changing a node means changing its label. A uniform cost of 1 for each operation was used in our experiments. Since computing the distance in this form is MAX SNP-hard for unordered labeled trees [41,25] uses a constrained version which is computable in polynomial time. The constraint is that disjoint subtrees have to be mapped to disjoint subtrees.

The original definition does not include any normalization. For two hierarchies with $q$ labels each, $2q$ is the highest possible CTED – corresponding to deleting all labels of the first hierarchy and inserting all labels of the second hierarchy. In this work, CTED is normalized by $\frac{q}{2}$ which results in distances in $[0,1]$.

In our case the complexity of calculating the CTED is $O(q^2 \cdot (\text{deg}(H_1) + \text{deg}(H_2))) - \log_2(\text{deg}(H_1) + \text{deg}(H_2)))$, where $q$ is the number of labels and $\text{deg}(H)$ refers to the maximum degree of any node in $H$ in [25].

3.4.2. Taxonomic overlap

TO [26] compares the nodes of two hierarchies according to how many ancestors and descendents they share. We describe a version adapted to the situation in our experiments. The semantic cotopy $SC(i,H)$ for a label $i \in L$ in a hierarchy $H$ is the set of its ancestors and all its descendents:

$$SC(i,H) = A(i) \cup \{j \in H \mid i \in A(j)\}.$$

Given two hierarchies $H_1$ and $H_2$, the corresponding overlap for $i \in L$ is defined as

$$O(i,H_1,H_2) = \frac{|SC(i,H_1) \cap SC(i,H_2)|}{|SC(i,H_1) \cup SC(i,H_2)|}.$$

Note that this is a symmetric expression in $H_1$ and $H_2$ with values in $[0,1]$. The TO of $H_1$ and $H_2$ is the average of all overlaps:

$$TO(H_1,H_2) = \frac{1}{q} \sum_{i \in L} O(i,H_1,H_2).$$

For two identical hierarchies, TO is 1, while it is 0 if $H_1$ and $H_2$ are totally unrelated. Since the two other hierarchy proximity measures used in the experiments are distances (with smaller values indicating higher similarity), we use $TO' = 1-TO$ to simplify comparisons.

In our situation the complexity of $TO'$ is $O(q^2 \cdot \text{max}(d(H_1), d(H_2)))$, where $q$ is the number of labels and $d(H)$ is the maximum depth of any node in $H$. Thus $TO'$ has a high worst-case complexity for very deep hierarchies (i.e. when $\text{max}(d(H_1), d(H_2)) = O(q)$), whereas for CTED this is the case for very broad hierarchies (i.e. when $\text{deg}(H) = O(q)$).

3.4.3. LCA-path tree distance

Since in a hierarchy a parent is a more general concept than any of its children the locations of the same label $i$ in two hierarchies $H_1$ and $H_2$ can be compared by how far one has to travel up the chain of ancestors in both hierarchies until a common ancestor is found. LCAPD is based on the following generalization of the Lowest Common Ancestor (LCA): For a label $i \in L$ denote with $a_1, \ldots, a_m$ the ancestors of $i$ in $H_1$, such that
LCAPD(i, H1, H2) = dφj, H1j, H2j i.

Since 0 is an ancestor of all labels in all hierarchies, the LCA always exists. Note that this definition is not symmetric: Usually LCA(i, H2, H1) ≠ LCA(i, H1, H2).

The LCA is used to measure how far apart the positions of a label in both hierarchies are: If a label i has the same parent in both hierarchies it is regarded as being in the same position, label in both hierarchies are: If a label i has the same parent in both hierarchies it is regarded as being in the same position, otherwise the distances of i to its LCA in both hierarchies are used to measure how far apart both locations are. The cost for i is defined as

\[ D(i, H1, H2) = \begin{cases} 0 & \text{if } p_{H1}(i) = p_{H2}(i), \\ \sum_{k=1,2} d_{Hk}(i) - d_{Hk}(\text{lca}(i, H1, H2)) & \text{otherwise,} \end{cases} \]

This can be seen as the length of the path from i to lca(i, H1, H2) in H1 and from lca(i, H1, H2) to i in H2.

As D is not symmetric, both combinations are averaged. Therefore, the complete LCA-path distance between H1 and H2 is

\[ D(H1, H2) = \frac{1}{W} \sum_{i=1}^{n} D(i, H1, H2) + D(i, H2, H1), \]

where W is a normalization factor: The worst case for a misplaced label i is that lca(i, H1, H2) = 0, since then (6) reduces to \( D(i, H1, H2) = d_{H1}(i) + d_{H2}(i) \). The same holds accordingly for \( D(i, H2, H1) \). Thus choosing \( W = \sum_{i=1}^{n} d_{H1}(i) + d_{H2}(i) \) ensures that \( D(H1, H2) \in [0, 1] \).

The complexity of a straightforward implementation of LCAPD is \( O(q \cdot d(H1) \cdot d(H2)) \), where \( d(H) \) is the maximum depth of any node in H. As TO*, it has a high worst-case complexity for very deep hierarchies.

### 3.4.4. Comparison of hierarchy proximity measures

The different hierarchy proximity measures reflect the dissimilarities between hierarchies in different ways. When interpreting the values of graph proximity measures it is thus important to consider how well the types of differences detected by the measures match the model represented by the graph (e.g. hierarchies, conceptual graphs, etc.). This is especially true for measures developed with the more abstract problem of graph or tree matching in mind (like CTED). For all but the most simple hierarchies quantifying the differences between two of them is usually a highly non-intuitive task. We therefore give examples of the differences between CTED, TO* and LCAPD illustrated by small hierarchies for which an approach based on intuition is still possible.

**Fig. 3 a)** to the former one is larger than that to the latter one. Nevertheless, the CTED is 0.25 for both of them. The corresponding LCAPD values 0.33 and 0.20, and the TO* values 0.24 and 0.16 match the intuitive view of the difference better. This comes as no surprise, since both LCAPD and TO* are based on the idea that the

| a | ibm.hardware | motorcycles | hockey | christian | recreation | sport | vehicles | politics | guns | mideast | misc | computer | graphics | ms-windows | windows.x | hardware | science | crypt | electronics | med | space |
|---|-------------|-------------|--------|-----------|------------|-------|----------|---------|-----|---------|------|----------|---------|-----------|-----------|---------|---------|-------|--------|--------|------|-------|
| forsage | athiesm | crypt | med | faith | christian | religion.misc | recreation | motorcycles | hockey | recreation | sport | vehicles | politics | guns | mideast | misc | computer | graphics | ms-windows | windows.x | hardware | science | crypt | electronics | med | space |

Fig. 3 Hierarchies for the 20 newsgroups dataset: (a) original hierarchy; (b) and (c) as extracted from different predictions. Incorrectly placed labels are shown in bold, whereas labels printed in italics correspond to intermediate nodes in the original hierarchy.
hierarchies encode parent-child relationships, whereas CTED
simply views the hierarchies as abstract trees. On the other hand,
while LCAPD and TO* match our situation better, CTED has the
advantage of being a metric [25] (i.e. it satisfies the triangle
inequality), which is neither the case for LCAPD nor for TO*.

TO* is designed for conceptual graphs and is thus more
adapted to our use of trees than CTED. The following example,
however, shows that it is still too general to fully capture our use
of trees as hierarchies. Consider the two hierarchies shown in
Fig. 4: Both of them contain three labels, but describe entirely
different relationships between them. Their distance as measured
by CTED, TO* and LCAPD is 0.33, 0.44 and 1.00, respectively. Thus,
only LCAPD describes them as being as different as possible, while
the results of the other two measures are contrary to our
tuition.

Some of the hierarchies of the datasets used in our experi-
ments contain nodes with exactly one child node (single-child
labels). One can show that TO* does not distinguish between such
labels and their children, in the following sense: Assume we are
given a hierarchy \(H\) and a label \(l\) that has exactly one child, \(j\), in \(H\).
Modify \(H\) by swapping \(i\) and \(j\), thereby creating a hierarchy \(R\)
where \(j\) is the parent of \(i\), \(P_{R}(i) = P_{R}(j)\), and the children of \(j\) in \(R\)
become children of \(i\) in \(H\). Then \(TO^*(H,R) = 0\). This is not the case
for either LCAPD or CTED.

4. Experiments

We tested the proposed data mining system based on ML-ARAM
by real-world data from the text-mining field and compared its
performance to the performance obtained by replacing ML-ARAM
with two other multi-label classifiers: ML-kNN and BoosTexter.
Three datasets with increasing complexity were used in our
experiments: 20 newsgroups—a dataset comprising newsgroup
postings, the Reuters Corpus Volume 1 version 2 (RCV1-v2)
dataset [42] composed of newswire stories written by Reuters
documents categorized by the international patent classification
scheme. To examine the impact of multi-label classifiers on HE,
the original hierarchies were compared with hierarchies extracted
from the test labels as predicted by the classifiers.

The experimental setup of ML-ARAM had the following
parameter values: choice parameter \(\kappa = 0.0001\); learning rates
\(\beta_{a,b} = 1.0\) for fast learning; vigilance parameters \(\rho_{a} = 0.9\)
and \(\rho_{b} = 1.0\). The parameter \(t\) was chosen to be 0.05 for the 20
newsgroups dataset and 0.02 for the large datasets. The number of
voters \(V\) was set to 9.

Following [23], ML-kNN was used with 10 nearest neighbors
\(k = 10\) and Laplace smoothing \((s = 1)\).

BoosTexter was trained using 500 boosting rounds as in [23].
The original implementation of BoosTexter only produces rank-
ings, not true multi-labels. However, since the corresponding
threshold is 0 for all BoosTexter rankings [24] these are easily
convertible into multi-labels.

4.1. Twenty newsgroups dataset

Due to the lack of publicly available multi-label datasets with
class hierarchies, in the first part of experiments the popular
single-label dataset 20 newsgroups [4,43] was enhanced by
considering additional labels corresponding to the levels of the
hierarchy shown in Fig. 3a. This dataset is a collection of almost
20 000 postings from 20 newsgroups sorted by date into training
and test sets containing 60% and 40% of the documents,
respectively. The data were preprocessed by discarding all words
appearing only in the test documents and all words found in the
stop-word list from [44]. Afterwards, all but the 2%-most-frequent
words were eliminated to reduce the dimensionality, and empty
records were removed. Documents were represented using the
well-known TF-IDF (term frequency-inverse document frequency)
word weighting scheme [43]. The idea behind this representation
is that words that occur in most documents are assumed to be less
useful in representing individual documents. The TF-IDF weights
were then normalized to the range of \([0,1]\). Conversion to TF-IDF
and normalization were performed separately for training and test
data. This resulted in the 1070-dimensional dataset with 11 256
training and 7493 test instances as well as 28 labels.

This dataset was used for classification and extraction of class
hierarchies in several experiments. First, the complexity of the
data was checked by extracting the class hierarchy from the
tree multi-labels of the test set. This hierarchy was perfect, which
indicates that the dataset is of good quality and the multi-
labels describe the hierarchy completely. Next, we studied the
performance of the multi-label classifiers and their ability to
infer the class hierarchies in the presence of only partly
available hierarchical information. A series of HE experiments
was performed, with training multi-labels having a decreasing
number of inserted non-leaf labels describing the levels in the
hierarchy. These non-leaf labels were removed from 20%, 30% and
40% randomly selected training multi-labels, leaving the multi-
lables single-labeled. Additional experiments were made to find
out the influence of the minimum confidence parameter \(c\) on
the quality of the extracted hierarchies. As stated in the literature,
reasonable default values for \(c\) are in the range of 0.5–0.7 [19].

In our experiments, hierarchies were extracted for 100 different \(c\)
values ranging from 0 to 1.0 and compared them to the original
hierarchy. Fig. 5 shows some of the results. Experiments
confirmed that the heuristic \(QE\) is strongly correlated with the
distance measures (Table 2). One can also see that the \(QE\) curve
still slightly falls when the distances stay constant. The reason is
the use of weighted level difference, which rewards the rules with
higher confidence as statistically more important. \(QE\) can thus be
used to estimate the optimal \(c\) value without knowledge of the
original hierarchy. The results are presented in Table 3 together
with the corresponding MC performance measures described in
Section 3.3.

ML-ARAM, being able to learn rare patterns and to process
single-label data along with multi-label data, solved this task
better than two other classifiers. The obtained performance
significantly exceeds the results reported in [19] where extracted
hierarchies were much smaller in size and consisted of only 8
nodes. Thus, the experiments confirm the applicability of the
network to middle-scale hierarchies. Taking into account the
performance on all datasets, ML-ARAM is superior to the other
classifiers on at least 6 and at most 7 out of 9 evaluation measures
creating on average about 228 prototypes (e.g., compare to 11 256
prototypes stored by ML-kNN). BoosTexter has the second best
performance, but its predictive power degraded more quickly than that of ML-ARAM, when increasing the number of single-label instances. With 30% and 40% removed labels, ML-ARAM has a clear majority of wins and is able to produce accurate multi-label predictions. The poorest MC results were shown by ML-kNN. Moreover, any reduction in the number of multi-labels led to a dramatic decrease in its performance, which is best demonstrated by such measures as F1 and F. It should be noted that different evaluation metrics provide different views of the relative performance of the classifiers. Additionally, when trained on the dataset with 40% removed labels, ML-ARAM significantly outperformed ML-kNN trained on the original dataset with all labels.

Fig. 5. Influence of the minimum confidence parameter on the quality of the hierarchy extracted from predicted labels of the 20 newsgroups dataset (after training with all labels). Note the high correlation between the heuristic QE and the distance measures: (a) ML-ARAM, (b) ML-kNN and (c) BoosTexter.

Table 3
Classification and HE performance of ML-ARAM, ML-kNN and BoosTexter on the 20 newsgroups dataset evaluated by the measures of Section 3.3 for training with all labels, as well as with 20%, 30% and 40% removed non-leaf labels.

<table>
<thead>
<tr>
<th>Measure</th>
<th>All</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.635</td>
<td>0.429</td>
<td>0.549</td>
<td>0.613</td>
</tr>
<tr>
<td>F1</td>
<td>0.694</td>
<td>0.565</td>
<td>0.677</td>
<td>0.675</td>
</tr>
<tr>
<td>F</td>
<td>0.691</td>
<td>0.480</td>
<td>0.605</td>
<td>0.671</td>
</tr>
<tr>
<td>OE</td>
<td>0.221</td>
<td>0.336</td>
<td>0.222</td>
<td>0.259</td>
</tr>
<tr>
<td>RL</td>
<td>0.500</td>
<td>0.124</td>
<td>0.073</td>
<td>0.108</td>
</tr>
<tr>
<td>AP</td>
<td>0.789</td>
<td>0.677</td>
<td>0.778</td>
<td>0.774</td>
</tr>
<tr>
<td>AUPRC</td>
<td>0.775</td>
<td>0.618</td>
<td>0.749</td>
<td>0.743</td>
</tr>
<tr>
<td>H-loss</td>
<td>0.219</td>
<td>0.239</td>
<td>0.210</td>
<td>0.232</td>
</tr>
</tbody>
</table>

The bold values mark the best classifier with respect to the particular training set. All hierarchy proximity measures correspond to hierarchies extracted with the value of the minimum confidence parameter c chosen for each set of predicted multi-labels using the optimal value of QE.
The hierarchy proximity measures help to quantify dissimilarities between obtained hierarchies, an inevitable task for comparing large hierarchies. So, the distance values of Table 3 quantitatively illustrate the good ability of ML-ARAM to produce class hierarchies: It correctly derived the whole class hierarchy of 28 nodes even with 40% removed labels. BoosTexter showed the same result, but ML-kNN could not extract the correct hierarchy in the absence of 40% multi-labels. The presented results show that ML-ARAM and BoosTexter performed better than ML-kNN on this dataset.

4.2. RCV1-v2 dataset

The next experiment was based on the tokenized version of the RCV1-v2 dataset introduced in [42]. RCV1-v2 contains three label sets, of which only the topics set is used here. It consists of 103 labels arranged in a hierarchy of depth four. The original training set of 23149 documents was used in a 10-fold cross-validation experiment (removing the last 9 entries). The cross-validation prediction chunks were concatenated then for performing HE.

This dataset has a more complex class hierarchy than the previous dataset. So, for example, two labels in this dataset are not partitioned therefore predicted many empty multi-labels in contrast to ML-kNN. The MC performance measures presented in Table 4 are given as mean values with their corresponding standard deviations. The paired t-test was used to identify statistically significant performance improvements. As evaluated at the level of 0.05, the differences between the classifiers in each MC metric were significant. One can see that BoosTexter outperforms ML-ARAM with respect to rankings measures, whereas ML-ARAM is much better on MC measures. ML-kNN has the worst results both in solving the MC task and in HE. Even though ML-ARAM has poorer performance values based on rankings than BoosTexter, its hierarchy is superior in terms of all distances. It is not surprising that the MC performance measures do not properly reflect the quality of predicted multi-labels from the viewpoint of HE, but it does reveal the need for development of a new performance measure especially suited to take HE potential into account.

### Table 4
Classification and HE performance of ML-ARAM, ML-kNN and BoosTexter on the RCV1-v2 dataset evaluated by the measures of Section 3.3.

<table>
<thead>
<tr>
<th>Measure</th>
<th>ML-ARAM</th>
<th>ML-kNN</th>
<th>BoosTexter</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.757 ± 0.009</td>
<td>0.667 ± 0.006</td>
<td>0.702 ± 0.006</td>
</tr>
<tr>
<td>P1</td>
<td>0.603 ± 0.008</td>
<td>0.747 ± 0.005</td>
<td>0.776 ± 0.006</td>
</tr>
<tr>
<td>F</td>
<td>0.813 ± 0.008</td>
<td>0.732 ± 0.006</td>
<td>0.778 ± 0.006</td>
</tr>
<tr>
<td>OE</td>
<td>0.069 ± 0.006</td>
<td>0.100 ± 0.007</td>
<td>0.062 ± 0.003</td>
</tr>
<tr>
<td>RL</td>
<td>0.076 ± 0.004</td>
<td>0.025 ± 0.001</td>
<td>0.014 ± 0.001</td>
</tr>
<tr>
<td>C</td>
<td>10.764 ± 0.033</td>
<td>8.216 ± 0.256</td>
<td>5.708 ± 0.289</td>
</tr>
<tr>
<td>AP</td>
<td>0.877 ± 0.005</td>
<td>0.847 ± 0.004</td>
<td>0.878 ± 0.004</td>
</tr>
<tr>
<td>AUPRC</td>
<td>0.841 ± 0.008</td>
<td>0.822 ± 0.004</td>
<td>0.847 ± 0.005</td>
</tr>
<tr>
<td>H-loss</td>
<td>0.128 ± 0.006</td>
<td>0.166 ± 0.005</td>
<td>0.166 ± 0.005</td>
</tr>
<tr>
<td>Wins</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

The bold values mark the best classifier. The hierarchy proximity measures correspond to hierarchies extracted with the value of the minimum confidence parameter c chosen for each set of predicted multi-labels using the optimal value of QE.

To represent the obtained results more thoroughly, we summarized some statistics describing the predicted multi-labels of the three classifiers in Tables 5 and 6. Here one can see, for example, that ML-kNN generally has a lower number of true positive predictions in all hierarchy levels than both other classifiers. Comparing ML-ARAM and BoosTexter, it can be stated that despite being able to produce fewer not-partitioned labels than ML-ARAM, BoosTexter generated a very large number of inconsistent multi-labels, which prevented it from extracting a proper class hierarchy. It is important to note that ML-ARAM produced no inconsistent multi-labels at all. The reason is that multi-labels are processed in ML-ARAM as whole and thus none of the ancestor can be lost. Benefiting from the use of instance-based postprocessing for converting rankings into labels, ML-ARAM also produced no empty multi-labels in contrast to ML-kNN and BoosTexter, which utilize a global threshold parameter and therefore predicted many empty multi-labels.

### Table 5
Distribution of true positives (tp) / false positives (fp) in the predictions for the RCV1-v2 dataset by label depth.

<table>
<thead>
<tr>
<th>Depth</th>
<th>Classifier</th>
<th>ML-ARAM</th>
<th>ML-kNN</th>
<th>BoosTexter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tp</td>
<td>fp</td>
<td>tp</td>
<td>fp</td>
</tr>
<tr>
<td>First level</td>
<td>24005</td>
<td>2112</td>
<td>22219</td>
<td>2878</td>
</tr>
<tr>
<td>Second level</td>
<td>20888</td>
<td>4306</td>
<td>17671</td>
<td>3551</td>
</tr>
<tr>
<td>Third level</td>
<td>10116</td>
<td>2086</td>
<td>8575</td>
<td>1491</td>
</tr>
<tr>
<td>Fourth level</td>
<td>200</td>
<td>97</td>
<td>169</td>
<td>37</td>
</tr>
</tbody>
</table>

To represent the obtained results more thoroughly, we summarized some statistics describing the predicted multi-labels of the three classifiers in Tables 5 and 6. Here one can see, for example, that ML-kNN generally has a lower number of true positive predictions in all hierarchy levels than both other classifiers. Comparing ML-ARAM and BoosTexter, it can be stated that despite being able to produce fewer not-partitioned labels than ML-ARAM, BoosTexter generated a very large number of inconsistent multi-labels, which prevented it from extracting a proper class hierarchy. It is important to note that ML-ARAM produced no inconsistent multi-labels at all. The reason is that multi-labels are processed in ML-ARAM as whole and thus none of the ancestor can be lost. Benefiting from the use of instance-based postprocessing for converting rankings into labels, ML-ARAM also produced no empty multi-labels in contrast to ML-kNN and BoosTexter, which utilize a global threshold parameter and therefore predicted many empty multi-labels.

### Table 6
Statistics over multi-labels predicted for RCV1-v2, summed over all chunks.

<table>
<thead>
<tr>
<th>Number of</th>
<th>ML-ARAM</th>
<th>ML-kNN</th>
<th>BoosTexter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-partitioned labels</td>
<td>10</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>Inconsistent multi-labels</td>
<td>0</td>
<td>952</td>
<td>2454</td>
</tr>
<tr>
<td>Empty multi-labels</td>
<td>0</td>
<td>635</td>
<td>266</td>
</tr>
</tbody>
</table>

To represent the obtained results more thoroughly, we summarized some statistics describing the predicted multi-labels of the three classifiers in Tables 5 and 6. Here one can see, for example, that ML-kNN generally has a lower number of true positive predictions in all hierarchy levels than both other classifiers. Comparing ML-ARAM and BoosTexter, it can be stated that despite being able to produce fewer not-partitioned labels than ML-ARAM, BoosTexter generated a very large number of inconsistent multi-labels, which prevented it from extracting a proper class hierarchy. It is important to note that ML-ARAM produced no inconsistent multi-labels at all. The reason is that multi-labels are processed in ML-ARAM as whole and thus none of the ancestor can be lost. Benefiting from the use of instance-based postprocessing for converting rankings into labels, ML-ARAM also produced no empty multi-labels in contrast to ML-kNN and BoosTexter, which utilize a global threshold parameter and therefore predicted many empty multi-labels.

### 4.3. WIPO-alpha dataset

The WIPO-alpha dataset[^1] is a collection of patent documents made available for research by the World Intellectual Property Organization (WIPO). The original hierarchy consists, from top to bottom, of 8 sections, 120 classes, 630 subclasses and about 69000 groups. In our experiment, the hierarchy was only used down to and including subclasses. Each document in the collection has one so-called main code and any number of secondary codes, where each code describes a subclass to which the document belongs. Both main and secondary codes were used in our experiment. We removed subclasses with fewer than 50 training and 50 test instances (and any documents that only belonged to such “small” subclasses). The remaining training and test documents were then combined to form a single dataset. From each document, the title, abstract and claims text parts were taken, stop-words were removed using the list from [44] and words were stemmed using the Snowball stemmer [45]. All stems with less than 5000 occurrences in the data were removed; the remaining stems were converted to TF-IDF weights and these weights were normalized to the range of [0, 1]. The final dataset consists of

62 587 records with 1173 attributes each and a label set of size 273. In this dataset there are 32 single-child labels and therefore 64 labels cannot be partitioned. MC was performed by means of 10-fold cross-validation (removing the last seven entries). Predicted hierarchies were extracted again from the whole dataset utilizing multi-labels of all cross-validation chunks.

The results obtained on the WIPO-alpha dataset are shown in Table 7. As evaluated by the paired t-test at the level of 0.05, the differences between the classifiers in each MC metric were significant. BoosTexter has in this dataset a much better performance than ML-ARAM, only in three MC performance was ML-ARAM better. Here, a disagreement between the classification quality and the HE quality is especially obvious when comparing the results of ML-ARAM and BoosTexter. Although the performance measures of BoosTexter are mostly better than those of ML-ARAM, the distances between the original and the extracted hierarchies are much smaller in the case of ML-ARAM, which indicates its better HE performance. ML-kNN had again the worst classification results and HE performance. The effect of single-child labels on different distance measures can be seen here: The TO* values are significantly lower than the LCAPD and CTEP values because TO* does not penalize incorrect assignment of such labels in contrast to the other distances as discussed at the end of Section 3.4.

Tables 8 and 9 show statistics describing the predicted multi-labels of three classifiers. This time ML-kNN again has fewer true positive predictions in all hierarchy levels than both other classifiers. But the difference between ML-ARAM and BoosTexter in the number of false positives is much higher on this dataset. BoosTexter has fewer false positives and therefore significantly better classification results. The minimum possible number of not-partitioned labels was obtained by ML-ARAM and BoosTexter, ML-kNN had 75 not-partitioned labels. The number of inconsistent multi-labels of BoosTexter was again very large.

4.4. Improving classification results using predicted hierarchies

The datasets were also used in order to test the idea of possible improvement of classification performance by means of available hierarchy information. When MC algorithms classify HMC datasets, they suffer from a lack of hierarchy information and their multi-label predictions are often inconsistent, i.e. some ancestors for predicted labels are not predicted. When a label is predicted correctly while some of its ancestors are not, the classification performance is low. In such a case, the prediction quality could be improved by adding the missing ancestors to the predicted multi-label using available hierarchy information. To this end, we propose utilizing a hierarchy extracted from the predicted multi-labels. However, there is some inherent uncertainty with this approach: It is generally impossible to know whether a label is predicted correctly or not. Adding missing ancestors to a label that was incorrectly predicted to be in the multi-label obviously worsens the prediction quality in most cases. For this problem, there is no solution within reach, but it turns out that this is not as severe a problem as one might imagine. Since multi-labels predicted by ML-ARAM are always consistent, the described postprocessing is not applicable to its predictions.

Table 10 shows the changes in performance measures and the number of changed labels after postprocessing of the results with the use of hierarchy information on all datasets. In this experiment, test set predictions for the 20 newsgroups dataset as well as predictions for the whole datasets of RCV1-v2 and WIPO-alpha were used for each classifier. Several postprocessing variants were tested: For P∗P and P−P, the predicted multi-labels were made consistent with the hierarchy extracted from them by either adding missing ancestors (in the case of P∗P) or by removing inconsistent labels (in the case of P−P). P+C and P−C use the same mechanism, but with respect to the true hierarchy. The differences between P∗P and P+C as well as between P−P and P−C, show how much could be gained by better hierarchy estimation. P±P/C uses the correct multi-labels to remove labels which are both inconsistent and incorrect, while adding the missing ancestors of inconsistent but correct labels according to the extracted hierarchy. Thus P±P/C shows how much could be gained by using a perfect heuristic for telling correct and incorrect labels apart. Finally, P±C/C equals P±P/C but uses the true hierarchy: This is the optimal case showing how much room for improvement there is for this kind of postprocessing.

It is worth pointing out that P∗P and P+C usually perform better than their “removing counterparts” P−P and P−C. More importantly, P∗P (and P+C) performs always at least as good as
the correct multi-labels, namely $P+P$ and $P$. For $A^{7}$, the improvements obtained by $P+P$ and $P+C$ are usually similar, since missing ancestors were still correct themselves most of the time. This is because, in our experiments, labels with improvements resulting from $P$ tactics based on the true multi-labels. It is also noteworthy that, for $A^{7}$, improvements over the non-postprocessed predictions are marked in bold.

F. Brucker et al. / Pattern Recognition 44 (2011) 724–738 737

The improvements obtained by $P+P$ and $P+C$ are usually similar, since missing ancestors were still correct themselves most of the time. This is because, in our experiments, labels with improvements resulting from $P$ tactics based on the true multi-labels. It is also noteworthy that, for $A^{7}$, improvements over the non-postprocessed predictions are marked in bold.

F. Brucker et al. / Pattern Recognition 44 (2011) 724–738 737

the original predictions, whereas $P/C$ (and $P/C^0$) usually reduces prediction quality. This is because, in our experiments, labels with missing ancestors were still correct themselves most of the time. The improvements obtained by $P+P$ and $P+C$ are usually similar, which shows that the extracted hierarchies are indeed a good heuristic for this kind of postprocessing. In the case of ML-kNN, the improvements obtained by $P+P$ are also close to the ones obtained for $P \pm P/C$, which indicates that the “always add missing ancestors” strategy does not perform much worse than a selective add/remove tactic based on the true multi-labels. It is also noteworthy that, for ML-kNN, $P+P$ often accomplishes at least 50% of the possible improvements resulting from $P/C$. While predictions were also improved for BoosTexter, the effect is not as evident as with ML-kNN and the differences between $P+P$ and $P \pm C$ show that a lot of potential improvements were not accomplished.

Comparing the number of changed labels for both classifiers, one can see that it is higher for BoosTexter due to its large number of inconsistent multi-labels (see Tables 6 and 9).

5. Conclusion

This paper has studied multi-label classification (MC) in the presence of class hierarchies with the objective of automatically discovering hierarchical relationships between output classes from predicted multi-labels. This is a more challenging task than standard Hierarchical MC (HMC) when classifiers rely on a known class hierarchy. We have proposed an approach for learning and extracting class hierarchies combining a novel multi-label extension of the ARAM neural network ML-ARAM with a hierarchy extraction (HE) algorithm based on an association rule learner. We have also developed a new tree distance measure and a quality estimator for quantitative comparison of hierarchies, which appear to be useful tools in HE. Although based on ML-ARAM, the proposed approach is general enough to be used with any other multi-label classifier. In extensive experiments made on three text mining real-world datasets, ML-ARAM has been compared within the presented framework against two state-of-the-art multi-label classifiers, namely ML-kNN and BoosTexter. The experimental results verify that the proposed approach is suitable for extracting middle and large-scale class hierarchies from predicted multi-labels. Furthermore, they show that ML-ARAM significantly outperforms both other classifiers in HE, assessing the quality of the extracted hierarchies by compari-

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classifier</th>
<th>ML-kNN</th>
<th>Boostexter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Postprocessing</td>
<td>Measure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A$</td>
<td>$F_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 newsgroups</td>
<td>$P+P$</td>
<td>0.006</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>$P$</td>
<td>–0.003</td>
<td>–0.002</td>
</tr>
<tr>
<td></td>
<td>$P+C$</td>
<td>0.006</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>$P$</td>
<td>–0.003</td>
<td>–0.002</td>
</tr>
<tr>
<td></td>
<td>$P \pm P/C$</td>
<td>0.010</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>$P \pm C/C$</td>
<td>0.010</td>
<td>0.011</td>
</tr>
<tr>
<td>RCV1-v2</td>
<td>$P+P$</td>
<td>0.006</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>$P$</td>
<td>–0.005</td>
<td>–0.004</td>
</tr>
<tr>
<td></td>
<td>$P+C$</td>
<td>0.006</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>$P$</td>
<td>–0.006</td>
<td>–0.005</td>
</tr>
<tr>
<td></td>
<td>$P \pm P/C$</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td></td>
<td>$P \pm C/C$</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>WIPO-alpha</td>
<td>$P+P$</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>$P$</td>
<td>–0.008</td>
<td>–0.008</td>
</tr>
<tr>
<td></td>
<td>$P+C$</td>
<td>0.007</td>
<td>0.006</td>
</tr>
<tr>
<td></td>
<td>$P$</td>
<td>–0.007</td>
<td>–0.006</td>
</tr>
<tr>
<td></td>
<td>$P \pm P/C$</td>
<td>0.007</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>$P \pm C/C$</td>
<td>0.009</td>
<td>0.010</td>
</tr>
</tbody>
</table>

For $A, F_1, F$ and $H$-loss, differences are shown between postprocessed and non-postprocessed predictions. Thus, in the case of $A, F_1$ and $F$, positive values denote improvements, whereas for $H$-loss this is the case for negative values. For the two postprocessing versions which do not require knowledge of either the correct hierarchy or the correct multi-labels, namely $P+P$ and $P$, improvements over the non-postprocessed predictions are marked in bold.

One important issue to be explored in future work is the development of an HMC ARAM extension. In addition we seek to generalize the approaches presented in this paper in order to apply them to information fusion tasks with the objectives of knowledge extraction from multiple data sources. There is often a variety of interrelated data, which can all be used to describe the same problem. By integrating evidence from different sources it is possible to infer new knowledge. A potential application of the proposed system is hierarchy merging or mapping when, for example, the same data exist, but are labeled differently in respect to alternative class hierarchies. In such a case, the system will be able to extract the underlying hierarchies and to relate them by either creating a common class hierarchy or extracting the rules describing relationships between classes of different hierarchies. This can be very useful for combining evidence from multiple sources decreasing human effort in creating structured knowledge representations.
References