A semi-dependent decomposition approach to learn hierarchical classifiers

J. Díez[a] J.J. del Coz[a,*] and A. Bahamonde[a]

[a]*Artificial Intelligence Center, University of Oviedo at Gijón, E33271 Gijón, Asturias, Spain*
http://www.aic.uniovi.es/MLGroup

Abstract

In hierarchical classification, classes are arranged in a hierarchy represented by a tree or a forest, and each example is labeled with a set of classes located on paths from roots to leaves or internal nodes. In other words, both multiple and partial paths are allowed. A straightforward approach to learn a hierarchical classifier, usually used as a baseline method, consists in learning one binary classifier for each node of the hierarchy; the hierarchical classifier is then obtained using a top-down evaluation procedure. The main drawback of this naïve approach is that these binary classifiers are constructed independently, when it is clear that there are dependencies between them that are motivated by the hierarchy and the evaluation procedure employed. In this paper, we present a new decomposition method in which each node classifier is built taking into account other classifiers, its descendants, and the loss function used to measure the goodness of hierarchical classifiers. Following a bottom-up learning strategy, the idea is to optimize the loss function at every subtree assuming that all classifiers are known except the one at the root. Experimental results show that the proposed approach has accuracies comparable to state-of-the-art hierarchical algorithms and is better than the naïve baseline method described above. Moreover, the benefits of our proposal include the possibility of parallel implementations, as well as the use of all available well-known techniques to tune binary classification SVMs.

Key words: Hierarchical classification, Multi-label learning, Structured output classification, Cost-sensitive learning, Support Vector Machines

* Corresponding author. Phone: +34 985 18 2501, Fax: +34 985 18 2125
Email addresses: `jdiez@aic.uniovi.es` (J. Díez), `juanjo@aic.uniovi.es` (J.J. del Coz), `antonio@aic.uniovi.es` (A. Bahamonde).

1 Introduction

Many real-world domains require automatic systems to organize objects into known taxonomies. For instance, a news website, or a news service in general, needs to classify the latest articles into sections and subsections of the site [1–6]. This learning task is usually called *hierarchical classification*. Although most of its applications deal with textual information, there are other fields in which hierarchical classification can be useful. The authors of [7,8] described an algorithm to classify speech data into a hierarchy of phonemes. A system was presented in [9] in which a robot can infer the similarity between different tools using a learned taxonomy. Another interesting task is related to biological terms: the Gene Ontology [10] is a controlled vocabulary used to represent molecular biology concepts and is the standard for annotating genes/proteins. This task has recently been addressed using hierarchical classification [11,12].

Hierarchical classification differs from multiclass learning in that: i) the whole set of classes has a hierarchical structure usually defined by a tree, and ii) each object must be labeled with a set of classes consistent with the hierarchy: if an object belongs to a class, then it must belong to any of its ancestors. In multi-label learning tasks, see for instance [13,14], training examples belong to a subset of labels too, but the output space does not necessarily have any hierarchical structure.

The aim of hierarchical classification algorithms is to learn a model that can accurately predict a set of classes; notice that these subsets of classes generally have more than one element and are endowed with a subtree structure. In the more general case, see Figure 1, these subtrees may have more than one branch (we then say that there are *multipaths* in the labels) and subtrees may not end on a leaf (i.e. they include *partial paths*). In this paper we will present a learning algorithm for hierarchical classification able to deal with multiple and partial paths.

1.1 Related work

As in multiclass classification, the algorithms available in the literature used to solve hierarchical classification can be arranged into two main groups: those that take a decomposition approach, and those that learn a hierarchical classifier in a single process. Decomposition algorithms learn a model for each node of the hierarchy using different methods; a hierarchical classification of an object is then obtained by combining, in some way, the predictions of these individual classifiers. The algorithms presented in [1–3,11] belong to this group. Hierarchical classification can, however, be seen as a whole rather than a series

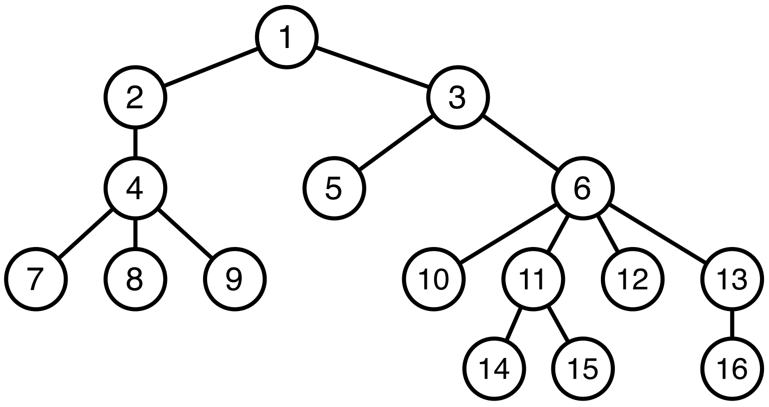

Fig. 1. Our approach can deal with examples that belong to multiple and partial paths; for instance an example can belong to classes {1,2,4,3,6,12}.

of local learning tasks; the idea being to optimize the global performance all at once. This approach is adopted in [4–8].

In [1], Koller and Sahami employ a Bayesian classifier at each internal node of the hierarchy to distinguish between its children. In the learning stage, they only use those instances that belong to the class as training instances. Their approach does not permit multipath or partial paths in the labels: the examples must belong to exactly one class at the bottom level of the hierarchy and the algorithm always predicts a single leaf.

In [2], a classifier is trained at each node and the outputs of all classifiers are combined by integrating scores along each path. After training the support vector machines (SVM) classifiers, the authors fit a sigmoid to the output of the SVM using regularized maximum likelihood fitting. The SVM thus produces posterior probabilities that are directly comparable across categories.

In [3], Cesa-Bianchi et al. presented an algorithm able to work with multipaths and partial paths. Essentially it constructs a conditional regularized least squares estimator for each node. This is an on-line algorithm and in each iteration an instance is presented to the current set of classifiers, the predicted labels are compared to the true labels, and regularized least squares estimators are updated.

A two-step approach was presented in [11]: first, an SVM model is learned for each node in an attempt to distinguish whether an instance belongs to that node, and then a Bayesian network is used to ensure that the predictions are consistent with the hierarchy.

Cai and Hofmann [4,5] presented two algorithms based on the large margin principle. These authors also derive a novel taxonomy-based loss function between overlapping categories that is motivated from real applications. The difference between both papers is that in the former their algorithm is only able to predict one category, while in the latter, they employ the category ranking approach proposed in [15] to deal with the additional challenge of multipaths.

In [6], Rousu et al. presented a kernel-based method in which the classification model is a variant of the maximum margin Markov network framework. This algorithm relies on a decomposition of the problem into single-example subproblems and conditional gradient ascent for optimisation of these subproblems. They propose a loss function that decomposes into contributions of edges so as to marginalize the exponential-sized problem into a polynomial one.

An on-line algorithm and a batch algorithm were presented in [7,8] combining ideas from large margin kernel methods and Bayesian analysis. The authors associate a prototype with each label in the hierarchy and formulate the learning task as an optimization problem with varying margin constraints. They impose similarity requirements between the prototypes corresponding to adjacent labels.

Finally, Vens et al. [16] compare three decision tree algorithms on the task of hierarchical classification: i) an algorithm that learns a single tree that predicts all classes at once, ii) one that learns a separate decision tree for each class, and iii) an algorithm that learns and applies such single-label decision trees in a hierarchical way. The first one outperforms the others in all aspects: predictive performance, model size and efficiency.

1.2 Our approach

Some of the papers cited above, for instance [3,6], compare their methods with two baseline algorithms: a kind of ”flat” one-vs-rest multiclass SVM and a hierarchical classifier based on SVM, usually called H-SVM. Both algorithms consist in learning a binary classifier for each node (class) of the hierarchy to predict whether an example belongs to the class at that node or not. The difference between both methods is that H-SVM constructs each binary classifier using only training examples for which the ancestor labels are positive, while multiclass SVM uses all examples. In order to make a fair comparison with hierarchical approaches and to guarantee consistent predictions with respect to the hierarchy, in the prediction phase of both baseline algorithms, the set of models are applied to an instance using a top-down evaluation procedure until a classifier fails to include that node in its predicted classes. This evaluation process also means that both algorithms are able to deal with multipath and partial path predictions.

In the experimental results reported in the literature, H-SVM is very competitive with respect to the proposed hierarchical algorithms and outperforms flat multiclass SVM. The only reason explaining the latter result is that H-SVM employs the predefined hierarchy to select the training examples used to build each SVM classifier. H-SVM takes into account the fact that, given the evaluation procedure used, the binary classifier of each node will be applied *after* its ancestors.

However, the main drawback of H-SVM is that binary classifiers are still constructed independently. As in multiclass classification, the advantage of direct methods over decomposition approaches is that the formers can capture some dependencies between individual classifiers. In the context of hierarchical classification, the presence of such dependencies is even clearer. They are motivated by the hierarchy and, in the case of H-SVM , also by the evaluation procedure.

In this paper, we shall present a new decomposition method that aims to improve the performance of H-SVM . Let us remark that H-SVM takes into account the hierarchical dependencies between the classes to select the training examples for each binary classifier. We want to exploit these *dependencies* even more. In our approach, binary classifiers are not independent: each node classifier is learned considering the predictions of other classifiers, its descendants, and the loss function used to measure the goodness of hierarchical classifiers. Following a bottom-up learning strategy, the idea is to optimize the loss function at every subtree assuming that all classifiers are known except the one at the root. We shall show that the performance of the two baseline methods described in this section can be improved using this learning method. The aim is to prove that a decomposition approach for hierarchical classification can be as successful as in multiclass classification [17].

In addition to the performance obtained, the advantages of decomposition algorithms for hierarchical classification are derived from their modularity. They can be straightforwardly implemented in a parallel platform to obtain a very fast learning method. They are simple and can be built, with some easy adaptations, with the user's favorite binary classifier; for instance, SVM. Moreover, the overall performance of the classifier can be improved using well-known techniques available for tuning binary classifiers, as occurs with SVM.

1.3 Outline of the paper

The paper is organized as follows. The next section formally introduces hierarchical learning, including appropriate loss functions, and the notation used throughout the rest of the paper. The third section is devoted to explaining the proposed decomposition method in detail. We present the main idea and show how it can be easily implemented using cost-sensitive binary SVM. Finally, the last section reports some experiments on benchmark data sets conducted to compare the approach presented here with other state-of-the-art algorithms in the context of hierarchical classification.

2 Hierarchical classification

In hierarchical classification, we have a set of classes arranged according to a known taxonomy. Formally, we have a tree $\mathcal{T}$ with r nodes, one for each class. In fact, we could start from a forest of trees $\mathcal{F}$, but then we would add an artificial root node to join the whole set of classes in a tree. Therefore, in what follows, we shall consider our hierarchy to be represented by a tree $\mathcal{T}$. In this context, hierarchical classification tasks are defined by a training set $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$, in which each example is described by an entry represented by a vector $\mathbf{x}_i$ of an input space $\mathcal{X}$, and a vector $\mathbf{y}_i$ of an output space $\mathcal{Y} \subset \{-1, +1\}^r$. We shall interpret each output $\mathbf{y}_i$ as a subset of the set of classes $\{1, \ldots, r\}$: $y_{ij} = +1$ if and only if the i^{th} example belongs to the j^{th} class. In the following, we shall use the symbol $\mathbf{y}_i$ both as a vector and as a subset of classes when no confusion can arise. We shall assume that all elements of $\mathcal{Y}$ observe the underlying hierarchy defined by $\mathcal{T}$ in the sense that

$$\forall \mathbf{y}_i \in \mathcal{Y}, \quad y_{ij} = -1 \Rightarrow \forall k \in des(j), \quad y_{ik} = -1,$$

in which $des(j)$ stands for the set of *descendants* of node or class j, but not including j.

A straightforward approach to learn tasks of this kind may consist in learning a family of binary models $\{\mathbf{w}_1, \ldots, \mathbf{w}_r\}$, one for each node (class) of $\mathcal{T}$. For instance, using linear classifiers, an entry $\mathbf{x}$ will be assigned to all classes j such that $(+1 = \operatorname{sign}(\langle \mathbf{w}_j, \mathbf{x} \rangle))$. However, this procedure may lead to inconsistent predictions with respect to $\mathcal{T}$. To avoid these, a top-down prediction procedure can be used, as in [3,6]. Thus, an entry can only be assigned to a class j if it was previously classified into its parent class, $par(j)$; therefore, an entry not assigned to one class, will *automatically* not be assigned to any of its descendants. This approach is followed, for instance, by the two baseline methods described in Section 1.2.

2.1 Loss functions

To complete the specification of the hierarchical learning task, we need to decide which loss function will be used to measure the goodness of the hypothesis learned. A first option may be to employ the zero-one loss function:

$$l_{0/1}(\mathbf{y}_i, \mathbf{y}'_i) = [\mathbf{y}_i \neq \mathbf{y}'_i]. \tag{1}$$

The problem is that it is not possible using this loss function to capture any difference between very wrong predictions and nearly correct ones.

In a real-world application, hopefully, an expert in the field could provide us with the costs of having false positives ($fp(j)$) and false negatives ($fn(j)$) for each class j [4]. Then, we can define

$$l_{\mathcal{T}}(\mathbf{y}_i, \mathbf{y}'_i) = \sum_{j \in \mathbf{y}_i - \mathbf{y}'_i} fn(j) + \sum_{j \in \mathbf{y}'_i - \mathbf{y}_i} fp(j). \tag{2}$$

Nevertheless, in the experiments reported below, as in [4–6], we always assume that all costs have value 1, and so we obtain a loss function that only reflects the cardinality of the symmetric difference of a pair of subsets of classes: the number of different elements. In symbols:

$$l_{\Delta}(\mathbf{y}_i, \mathbf{y}'_i) = \sum_{j=1}^{r} [y_{ij} \neq y'_{ij}] = |(\mathbf{y}'_i - \mathbf{y}_i) \cup (\mathbf{y}_i - \mathbf{y}'_i)| = |\mathbf{y}'_i \ominus \mathbf{y}_i|. \tag{3}$$

In [6], Rousu et al. proposed other loss functions, weighting the classes according to the proportion of the hierarchy that is in the subtree $\mathcal{T}_j$ rooted by node j, or sharing the relevance of each node between its siblings starting with 1 for the root. It is easy to see that these loss functions are particular cases of the general framework, $l_{\mathcal{T}}$, presented here.

3 A semi-dependent decomposition hierarchical classifier: H-SVM $_{l_{\mathcal{T}}}$

We shall now describe our approach to build hierarchical learners based on the use of binary classifiers and a straightforward implementation using SVM. Following the notation of the previous section, we assume that we have before us a learning task specified by a training set $\mathcal{S}$ and two real positive functions, fp and fn, to compute the costs of false positives and false negatives of classes, respectively.

The aim of this paper is to discover how to design a decomposition approach to learn a competitive hierarchical classifier, in the same way that decomposition methods are used to successfully solve multiclass classification tasks. Our goal is to improve the performance of the two most popular decomposition algorithms used as baseline methods in hierarchical classification papers: a kind of flat SVM and H-SVM , see Section 1.2. Recall that the only difference between both methods is the set of examples used to learn each model $\mathbf{w}_j$. Formally,

- Flat SVM: *all* entries of $\mathcal{S}$ will be considered and, as in multiclass learning when we are using the one-vs-rest strategy, the subset of positive ($\mathcal{S}_j^+$) and negative ($\mathcal{S}_j^-$) examples will be given by

$$\begin{aligned} \mathcal{S}_j^+ &= \{(\mathbf{x}_i, \mathbf{y}_i) : y_{ij} = +1\}, \\ \mathcal{S}_j^- &= \{(\mathbf{x}_i, \mathbf{y}_i) : y_{ij} = -1\}. \end{aligned} \tag{4}$$

- H-SVM: learns to distinguish between those examples that belong to j's *parent*,

$$\begin{aligned} \mathcal{S}_j^+ &= \{(\mathbf{x}_i, \mathbf{y}_i) : y_{ij} = +1\}, \\ \mathcal{S}_j^- &= \{(\mathbf{x}_i, \mathbf{y}_i) : y_{i,par(j)} = +1 \wedge y_{ij} = -1\}. \end{aligned} \tag{5}$$

According to these definitions, both options only differ in the $\mathcal{S}_j^-$ set. For example, considering the tree depicted in Figure 1, using H-SVM, $\mathcal{S}_7^-$ would only contain examples that belong to class 4 and do not belong to class 7, while in flat SVM it would contain all the examples that do not belong to class 7. Therefore, H-SVM is faster because it uses less examples, i.e. only those that belong to the parent of the node which model is being calculated, and more effective, as has been proven in several experimental results reported in previous papers. The main drawback of flat SVM is that individual models are built considering examples of a kind that it will probably not classify so often given the evaluation procedure used. For instance, using all examples to learn classifier $\mathbf{w}_7$, the learning process considers examples that do not belong to any of its ancestors but the root, i.e. instances that only belong to the subset of classes $\{1, 3, 5, 6, 10, 11, 12, 13, 14, 15, 16\}$. In the prediction procedure, however, said model ($\mathbf{w}_7$) will only classify those examples if its ancestor models, $\mathbf{w}_2$ and $\mathbf{w}_4$, fail (classifying those instances as positive). So, $\mathbf{w}_7$ considers some objects that are not so useful for the main task that this model has to perform: classify examples of class 4 and its descendants, the very examples that are used by H-SVM.

3.1 Main ideas

Our proposal is based on two main ideas:

(1) individual classifiers must be dependent on those that are related to them according to the hierarchy, and
(2) individual classifiers must be learned by locally optimizing our hierarchical loss function (Eq. 2), instead the binary zero-one loss function.

Let us examine these two ideas. Obviously, the only way to build a hierarchical classifier, made up of a set of dependent individual classifiers $\{\mathbf{w}_1, \ldots, \mathbf{w}_r\}$, is using an algorithm that learns all these models at once. However, these methods present the disadvantage of their computational complexity, which depends on the number of classes. This number is usually much bigger in hierarchical classification tasks than in multiclass problems. We can significantly reduce this complexity by following a decomposition approach, but we have to pay a price: we cannot build each classifier depending on all others. We have to restrict ourselves to learn each classifier depending only on *some* of the others.

Given the hierarchy described by $\mathcal{T}$, we basically have two options: each classifier can depend on its ancestors, using a top-down learning strategy, or on its descendants, using a bottom-up algorithm. But it makes more sense for each classifier to depend on its descendants. The main reason is that, depending on the top-down prediction procedure used to ensure hierarchically consistent responses, parent models can change the predictions of their descendants: whenever they classify an example as negative, it is also classified as negative in all of their descendant classes. Therefore, a model has more influence on the overall performance of the learner when it is closer to the root. Taking into account these circumstances, it is preferable for models near to the root to be computed later, using the information of their descendants: they can know the predictions of their descendant models and use these predictions to build a classifier adapted to them, thus improving the overall performance of the local hierarchical learner placed at that subtree. Following this reasoning, our approach uses a bottom-up learning strategy, models are calculated from leaf nodes to the root of the tree, and each classifier depends on the classifiers of its descendant classes.

The second idea is to change the loss function optimized to learn each individual classifier. As in multiclass tasks, the individual classifiers of the proposed hierarchical decomposition methods optimize the binary zero-one loss function. In hierarchical classification, however, this setting is not appropriate. Moreover, if we consider that in our approach each binary classifier $\mathbf{w}_j$ will be built when all other classifiers of the subtree rooted at node j ($\mathcal{T}_j$) are known, then we can optimize any hierarchical loss function instead the binary zero-one loss function.

In fact both ideas are complementary: locally optimizing the hierarchical loss function described in Eq. 2 requires that the different subsets of binary classifiers must be considered together. Combining both, our method is based on building each binary classifier optimizing the hierarchical loss function used, and considering that all classifiers of the subtree rooted at that node have been learned before. Our experimental results will show that a better decomposition method can be obtained using this bottom-up learning procedure.

3.2 Using cost-sensitive learning to optimize $l_{\mathcal{T}}$ at every subtree

To the best of our knowledge, all the new loss functions proposed to deal with hierarchical classification, including our proposal (Eq. 2), are example-based functions, i.e., they decompose linearly into a combination of the individual classification errors. In order to optimize such functions, we need only bear in mind that it will not be the same to classify each training example incorrectly; in other words, each example can make a *different* contribution to the overall loss. These are the kind of tasks that cost-sensitive learning methods [18] solve. Our approach is therefore based on assigning different *costs* to each example during learning; these costs depend on the hierarchy of classes, the loss function used and, as we shall see, the top-down prediction procedure. Before learning all our binary classifiers, we have to calculate the cost that each example must have and then apply any binary cost-sensitive learner able to assign different costs to each example.

In the trivial case, when we are learning a model $\mathbf{w}_j$ of a leaf node j, for instance $\mathbf{w}_7$ in the learning task depicted in Figure 1, the loss of each example $\mathbf{x}_i$ is

$$l_{\mathcal{T}}(y_{ij}, y'_{ij}) = \begin{cases} 0 & \text{if} \quad y_{ij} = y'_{ij}, \\ fn(j) & \text{if} \quad y_{ij} = +1, y'_{ij} = -1, \\ fp(j) & \text{if} \quad y_{ij} = -1, y'_{ij} = +1, \end{cases}$$

since we only have to consider that model. From the point of view of the classifier $\mathbf{w}_j$, the maximum cost or loss caused by an example is the difference between the loss when the classification of the example is wrong ($fn(j)$ or $fp(j)$) and when the example is classified correctly (always 0). Then the cost assigned to an example $\mathbf{x}_i$ in order to compute a leaf model $\mathbf{w}_j$ is:

$$c_{ij} = \begin{cases} fn(j) - 0 & \text{if} \quad y_{ij} = +1, \\ fp(j) - 0 & \text{if} \quad y_{ij} = -1. \end{cases} \tag{6}$$

In the case of positive examples, this difference is $fn(j)$, and $fp(j)$ if the example is negative. Notice that, if functions fp and fn always return 1 (l_Δ loss function), every example will have the same cost. Obviously, this also occurs when fp and fn always return another constant value. In such situations, the leaf classifier learned by a cost-sensitive method and by its counterpart learner that optimizes the binary zero-one loss function, will be the same. This will be the reason why our algorithm optimizing l_Δ loss function (Sections 3.3 and 3.4) will only differ from H-SVM in those classifiers attached to internal nodes; classifiers at leaf nodes will be exactly the same because there is no sub-hierarchy involved in that subproblems.

In the more general case, when we are learning an internal node classifier (for instance, $\mathbf{w}_4$ in our example), things become more complicated. First, we must bear in mind that, following our bottom-up learning strategy, its descendant models ($\mathbf{w}_7$, $\mathbf{w}_8$ and $\mathbf{w}_9$ for model $\mathbf{w}_4$) have already been learned, and we can know the consistent top-down predictions of that models ($\mathbf{y}'_{i,des(j)}$) for all examples $\mathbf{x}_i$ of the data set $\mathcal{S}_j$ used to build that classifier (predictions $\{y'_{i7}, y'_{i8}, y'_{i9}\}$ for the examples of $\mathcal{S}_4$). In this case, the loss of each example in all different situations is:

$$l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) =$$
$$\begin{cases} \sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) + \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) & \text{if } y_{ij} = y'_{ij} = +1, \\ 0 & \text{if } y_{ij} = y'_{ij} = -1, \\ fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) & \text{if } y_{ij} = +1, y'_{ij} = -1, \\ fp(j) + \sum_{k \in \mathbf{y}'_{i,des(j)}} fp(k) & \text{if } y_{ij} = -1, y'_{ij} = +1. \end{cases} \tag{7}$$

This definition requires some explanations. First, notice that the loss function now also considers the labels of the descendant classes of node j: $\mathbf{y}_{i,des(j)}$ and $\mathbf{y}'_{i,des(j)}$. The only situation in which there is no loss is when the example is labeled as negative by model $\mathbf{w}_j$ and $y_{ij} = -1$. This is due to the fact that, since labels are consistent with the hierarchy, the true labels $\mathbf{y}_{i,des(j)}$ and predicted labels $\mathbf{y}'_{i,des(j)}$ are all -1. Let us recall that in this case the predicted labels are negative because we are applying the top-down prediction procedure discussed through the paper: if y'_{ij} is -1, then all its descendant labels must also be negative.

Perhaps the most surprising part of this definition is that, even when a positive example ($y_{ij} = +1$) is correctly classified by model $\mathbf{w}_j$ ($y'_{ij} = +1$), there can be some loss. This loss is caused by its descendant classifiers and is the sum of its false positives and false negatives; i.e., the expression in Eq. 2 excluding the root node. In the two other cases, when predictions of $\mathbf{w}_j$ are wrong, the loss is the sum, for all the nodes of $\mathcal{T}_j$ (including the root), of false negatives for positive examples and the sum of false positives for negative examples. Here we apply the expression of Eq. 2 considering that one of the subsets, $\mathbf{y}_{i,des(j)}$ or $\mathbf{y}'_{i,des(j)}$, is empty. As before, this is guaranteed by the top-down prediction procedure.

In this general case, the cost of each example in $\mathcal{S}_j$ is computed again as the difference between the loss when the example is incorrectly classified and when the example is correctly classified by $\mathbf{w}_j$. For negative examples, this expression is:

$$c_{ij} = fp(j) + \sum_{k \in \mathbf{y}'_{i,des(j)}} fp(k) - 0. \tag{8}$$

Notice that, as fp and fn are positive functions, this cost is always greater than 0. In the case of positive examples, the cost of each example is given by the following expression:

$$c_{ij} = fn(j) \quad + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) - \sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) - \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k). \quad (9)$$

It should be noted that this expression can be negative for some examples, whenever

$$\left(fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) \right) < \left(\sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) + \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) \right),$$

or 0 when both terms are equal. In the former case, this means that it is preferable to fail those examples at node j because the loss in its descendant classes is greater. When the cost is 0, it does not matter whether the classification of these examples is right or wrong as the loss is the same in both situations. In fact, we can remove these examples from the $\mathcal{S}_j$ data set.

These two cases explain why it is important for our approach to learn, at each internal node, binary models that depend on their descendant classifiers. Our method not only assigns a different relevance to each example during learning using the costs described previously, but can also change its class, when the cost is negative, or not considering it when its cost is 0.

3.3 Optimizing l_Δ: an example

In order to better explain the previous section, we shall now describe an example to learn the binary classifier for class 4 of the hierarchical classification problem depicted in Figure 1. The data set used, $\mathcal{S}_4$, is shown in Table 1. The table includes the true labels ($\mathbf{y}_{i,des(4)}$) and the predicted labels ($\mathbf{y}'_{i,des(4)}$) of the descendants of class 4. To obtain these predictions we will have used the binary classifiers $\mathbf{w}_7$, $\mathbf{w}_8$ and $\mathbf{w}_9$ that have been learned previously according to the bottom-up procedure of our hierarchical learner.

For ease of understanding, we shall assume that the functions fp and fn are always 1. In this situation, we use l_Δ instead of $l_\mathcal{T}$ as our loss function. The cost of each example to learn binary model $\mathbf{w}_4$ (Eq. 8 and Eq. 9) will be computed by means of:

$$c_{ij} = \begin{cases} 1 + \left|\mathbf{y}_{i,des(j)}\right| - \left|\mathbf{y}_{i,des(j)} \ominus \mathbf{y}'_{i,des(j)}\right| & \text{if } y_{ij} = +1, \\ 1 + \left|\mathbf{y}'_{i,des(j)}\right| & \text{if } y_{ij} = -1. \end{cases} \quad (10)$$

Table 1
Data set to illustrate how to compute c_{i4} costs in order to optimize $\mathbf{w}_4$ in the hierarchical classification task shown in Figure 1: $\mathcal{S}_4^+ = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ and $\mathcal{S}_4^- = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$

	$\{y_{i4}, \mathbf{y}_{i,des(4)}\}$				$\{y'_{i4}, \mathbf{y}'_{i,des(4)}\}$			
$\mathcal{S}_4$	y_{i4}	y_{i7}	y_{i8}	y_{i9}	y_{i4}	y_{i7}	y_{i8}	y_{i9}
$\mathbf{x}_1$	1	1	1	1	?	1	-1	1
$\mathbf{x}_2$	1	-1	-1	-1	?	1	1	-1
$\mathbf{x}_3$	1	-1	-1	1	?	1	1	1
$\mathbf{x}_4$	-1	-1	-1	-1	?	-1	-1	-1
$\mathbf{x}_5$	-1	-1	-1	-1	?	1	1	-1
$\mathbf{x}_6$	-1	-1	-1	-1	?	1	-1	-1

Using this expression, we can now easily calculate the cost for every example in $\mathcal{S}_4$:

$$\begin{aligned}
c_{14} &= 1 + |\{7,8,9\}| - |\{7,8,9\} \ominus \{7,9\}| = 3,\\
c_{24} &= 1 + |\{\emptyset\}| - |\{\emptyset\} \ominus \{7,8\}| = -1,\\
c_{34} &= 1 + |\{9\}| - |\{9\} \ominus \{7,8,9\}| = 0,\\
c_{44} &= 1 + |\{\emptyset\}| = 1,\\
c_{54} &= 1 + |\{7,8\}| = 3,\\
c_{64} &= 1 + |\{7\}| = 2.
\end{aligned}$$

The importance of each example is now very different. For instance, $\mathbf{x}_1$ and $\mathbf{x}_5$ are the most relevant examples to learn $\mathbf{w}_4$: if this model does not classify both correctly, the loss in that subtree will increase more than if it fails the other examples. As was discussed before, some positive examples ($\mathbf{x}_2$) can have a negative cost, meaning that it is preferable to classify them incorrectly because the loss in its descendant classes is greater; or cost 0 ($\mathbf{x}_3$): it makes no difference whether the classification of these examples is right or wrong. Costs for negative examples are always positive, see Eq 10. In the next section, we describe how to deal with all these different situations in order to implement individual binary classifiers.

3.4 A straightforward implementation of our approach using binary SVMs

Algorithm 1 describes our bottom-up learning procedure. It constructs individual models from leaves to the root. In each iteration, binary classifiers of the deepest nodes not built yet are computed. We observe that an efficient implementation of this algorithm can obtain these models in parallel. However,

Algorithm 1 Learning hierarchical classifier

```
1: function LEARNING($\mathcal{S}, \mathcal{T}$) : $\{\mathbf{w}_1, \dots, \mathbf{w}_r\}$
2: $CurrentLevel$ = LEAVES($\mathcal{T}$)
3: while $CurrentLevel \neq \emptyset$ do
4:     compute $\mathcal{S}_j,\ \forall j \in CurrentLevel$
5:     compute $c_{ij},\ \forall j \in CurrentLevel,\ \forall (\mathbf{x}_i, y_{ij}) \in \mathcal{S}_j$
6:     learn $\mathbf{w}_j,\ \forall j \in CurrentLevel$
7:     compute $\{y'_{ij}, \mathbf{y}'_{i,des(j)}\},\ \forall (\mathbf{x}_i, y_{ij}) \in \mathcal{S},\ \forall j \in CurrentLevel$
8:     $CurrentLevel$ = PREVIOUSLEVEL($\mathcal{T}, CurrentLevel$)
9: end while
10: return $\{\mathbf{w}_1, \dots, \mathbf{w}_r\}$
11: end function
```

the degree of parallelism of our algorithm is less than in the case of flat SVM or H-SVM , in which all binary classifiers can be computed in parallel.

Before learning each classifier $\mathbf{w}_j$, the algorithm constructs the corresponding data set, $\mathcal{S}_j$. The most important part of this process is to calculate the cost of every example of $\mathcal{S}_j$ using the expressions discussed previously. After building each individual classifier, the algorithm obtains the predicted labels for all the examples in our original data set, $\mathcal{S}$. Notice that the algorithm not only computes y'_{ij}, it also recalculates the predicted labels of all descendant classes of j ($\mathbf{y}'_{i,des(j)}$) to take into account the fact that, in the prediction process, the subset of models already learned $\{\mathbf{w}_j, \mathbf{w}_k : \forall k \in des(j)\}$ will be applied using the top-down evaluation procedure.

To learn each individual classifier, practitioners can employ their favorite binary learner whenever it is able to assign a different cost to each example. Users can also choose the most suitable hierarchical loss function for their application, but this must be a particular case of the general loss function $l_{\mathcal{T}}$ presented in Eq. 2. If the selected loss function is l_{Δ} (Eq. 3), most binary learners can be used, as the cost of every example will be an integer value. In that case, so as to reflect the different relevance of the examples in $\mathcal{S}_j$, the algorithm can repeat each example as many times as the absolute value of its cost. If this value is negative, then it must use the absolute value of that cost and invert its original class, y_{ij}. Finally, if an example has cost 0, it must be removed from $\mathcal{S}_j$.

In the experimental results reported in the next section, we have used a weighted SVMs [19] as our binary learner. This kind of SVM is based on assigning a different weight or cost to each example that is proportional to the importance of correctly classifying that example. Formally, our method solves the following kind of optimization problems [1] :

[1] For ease of reading, we omit bias b_j; however, bias can be easily included by adding an additional feature of constant value to each $\mathbf{x}_i$

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\langle \mathbf{w}_j, \mathbf{w}_j\rangle + C \sum_{\mathbf{x}_i \in \mathcal{S}_j} |c_{ij}| \cdot \xi_{ij}, \\ \text{s.t.} \quad & \bar{y}_{ij}\langle \mathbf{w}_j, \mathbf{x}_i\rangle \geq 1 - \xi_{ij}, \\ & \xi_{ij} \geq 0, \qquad\qquad \forall i : (\mathbf{x}_i, y_{ij}) \in \mathcal{S}_j, \end{aligned} \tag{11}$$

where factor C controls the amount of regularization, c_{ij} is calculated using Eq. 8 and Eq. 9 (to optimize $l_{\mathcal{T}}$) or Eq 10 (l_Δ), and $\bar{y}_{ij} = y_{ij} \cdot \text{sign}(c_{ij})$. We use the absolute value of c_{ij} and change the original class when this value is negative. Let us remark that the number of constrains of this optimization problem is the same as of the traditional binary SVM. In fact, if we set all c_{ij} as 1, as happens when our method optimizing l_Δ learns classifiers attached to leaf nodes, both problems are equal.

The dual problem can be derived by standard Lagrangian techniques:

$$\begin{aligned} \max_{\alpha} \quad & -\frac{1}{2} \sum_{\mathbf{x}_i \in \mathcal{S}_j} \sum_{\mathbf{x}_k \in \mathcal{S}_j} \alpha_i \alpha_k \bar{y}_{ij} \bar{y}_{kj} \langle \mathbf{x}_i, \mathbf{x}_k \rangle + \sum_{\mathbf{x}_i \in \mathcal{S}_j} \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq |c_{ij}| C, \quad \forall i : (\mathbf{x}_i, y_{ij}) \in \mathcal{S}_j. \end{aligned} \tag{12}$$

Therefore, the only difference with respect to binary SVM is in the upper bound of the box constraint.

Theorem 1 *At the solution $\mathbf{w}_j^*$, ξ^* of the optimization problem in Eq 11 on the training data set $\mathcal{S}_j$, using Eq. 8 and Eq. 9 to calculate costs $c_{ij}, \forall(\mathbf{x}_i, y_{ij}) \in \mathcal{S}_j$, the value of $\sum_{\mathbf{x}_i \in \mathcal{S}_j} |c_{ij}| \cdot \xi_{ij}$ is an upper bound of*

$$\sum_{\mathbf{x}_i \in \mathcal{S}_j} l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) - l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y^B_{ij}, \mathbf{y}^B_{i,des(j)}\}),$$

in which $\{y'_{ij}, \mathbf{y}'_{i,des(j)}\}$ are the consistent predictions using $\{\mathbf{w}_j^*, \mathbf{w}_k : \forall k \in des(j)\}$ and considering that $\{y^B_{ij}, \mathbf{y}^B_{i,des(j)}\}$ are obtained applying the subset of models $\{\mathbf{w}^B_j, \mathbf{w}_k : \forall k \in des(j)\}$, in which $\mathbf{w}^B_j$ is the best or ideal model (which may be impossible to obtain) for class j, in the sense that it always makes the prediction that causes less loss in the subtree $\mathcal{T}_j$.

PROOF. In other learning tasks, the second term of the above expression is always 0 (the ideal model has no loss), but here we must consider the loss caused by descendant models. In other words, we want to prove that $\sum_{\mathbf{x}_i \in \mathcal{S}_j} |c_{ij}| \cdot \xi_{ij}$ is an upper bound of the loss caused by $\mathbf{w}_j$ when its descendant models are already learned. We only have to demonstrate that the following holds for each example

$$|c_{ij}| \cdot \xi_{ij} \geq l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) - l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y^B_{ij}, \mathbf{y}^B_{i,des(j)}\}).$$

Following the definition in Eq 7, for negative examples we have that:

$$l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y^B_{ij}, \mathbf{y}^B_{i,des(j)}\}) = \min(0, fp(j) + \sum_{k \in \mathbf{y}'_{i,des(j)}} fp(k)) = 0.$$

Using Eq 8,

$$\left| fp(j) + \sum_{k \in \mathbf{y}'_{i,des(j)}} fp(k) \right| \cdot \xi_{ij} \geq l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) - 0,$$

and this expression is always true because

$$l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) = \begin{cases} 0 & \text{if } 0 \leq \xi_{ij} < 1 \\ fp(j) + \sum_{k \in \mathbf{y}'_{i,des(j)}} fp(k) & \text{if } \xi_{ij} \geq 1. \end{cases}$$

In the case of positive examples, $l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y^B_{ij}, \mathbf{y}^B_{i,des(j)}\})$ is (using once again Eq 7)

$$\min\left(\sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) + \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k), fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) \right). \quad (13)$$

If the minimum is the first term ($c_{ij} > 0$ and $\bar{y}_{ij} = y_{ij} \cdot \text{sign}(c_{ij}) = +1$, see Eq 9), and now applying Eq 9,

$$\left| fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) - \sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) - \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) \right| \cdot \xi_{ij} \geq$$

$$l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) - \left(\sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) + \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) \right),$$

and this is always true, because $l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\})$ (see Eq 7) is

$$\begin{cases} \sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) + \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) & \text{if } 0 \leq \xi_{ij} < 1 \\ fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) & \text{if } \xi_{ij} \geq 1. \end{cases}$$

If the minimum in Eq 13 is the second term ($c_{ij} < 0$, $\bar{y}_{ij} = -1$), then we have to prove that

$$\left| fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) - \sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) - \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) \right| \cdot \xi_{ij} \geq$$

$$l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) - \left(fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) \right).$$

And this inequality holds because $l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\})$ is the opposite to that of the previous case (its original class is inverted in the optimization problem in Eq 11),

$$\begin{cases} fn(j) + \sum_{k \in \mathbf{y}_{i,des(j)}} fn(k) & \text{if } 0 \leq \xi_{ij} < 1 \\ \sum_{k \in \mathbf{y}_{i,des(j)} - \mathbf{y}'_{i,des(j)}} fn(k) + \sum_{k \in \mathbf{y}'_{i,des(j)} - \mathbf{y}_{i,des(j)}} fp(k) & \text{if } \xi_{ij} \geq 1. \end{cases}$$

The case when both terms are equal ($c_{ij} = 0$) is trivial because

$$l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y'_{ij}, \mathbf{y}'_{i,des(j)}\}) = l_{\mathcal{T}}(\{y_{ij}, \mathbf{y}_{i,des(j)}\}, \{y^B_{ij}, \mathbf{y}^B_{i,des(j)}\}).$$

In fact, as mentioned previously, these examples are deleted from $\mathcal{S}_j$. $\square$

The same demonstration can be employed to prove that, using Eq 10 to compute c_{ij} in the optimization problem equation (11), term $\sum_{\mathbf{x}_i \in \mathcal{S}_j} |c_{ij}| \cdot \xi_{ij}$ is an upper bound of the l_Δ loss caused by $\mathbf{w}^*_j$, since l_Δ is a particular case of the $l_{\mathcal{T}}$ loss function.

4 Experimental results

The main aim of this section is to show that the method presented here can improve the performance of the two decomposition algorithms, flat SVM and H-SVM , that are usually selected as baseline methods in papers that present new hierarchical classifiers. For that reason, we implemented, not only our proposed algorithm H-SVM$_{l_{\mathcal{T}}}$, but also another version of our approach, called SVM$_{l_{\mathcal{T}}}$, that is the counterpart method of flat SVM. Since we did not have available functions fp and fn for the data sets used, both versions optimized locally (at every subtree) the loss function l_Δ (Eq. 3). In the following we denoted them as H-SVM$_{l_\Delta}$ and SVM$_{l_\Delta}$, and the difference between them is the same that between H-SVM and flat SVM: the data sets used to learn each individual binary classifier (see Eq. 4 and Eq. 5).

Let us recall that H-SVM$_{l_\Delta}$ and H-SVM use the same examples to learn each model (applying Eq. 5 in step 4 of Algorithm 1). The only difference between them is that the former optimizes locally (at every subtree) the hierarchical loss function l_Δ and H-SVM optimizes the binary zero-one loss function. The same occurs between, SVM$_{l_\Delta}$ and flat SVM, both use the same training points (all examples in this case, Eq. 4) and the difference, again, is that our approach learns cost-sensitive classifiers. Thus, since we are using l_Δ as our target loss function, classifiers at leaf nodes are exactly the same in H-SVM$_{l_\Delta}$ and H-SVM ; then, they only differ in those classifiers attached to internal nodes. The same happens between SVM$_{l_\Delta}$ and flat SVM.

Additionally, we wanted to compare our approach with two state-of-the-art learners that use different methods. The algorithms used were H-M^3 of Rousu et al. [6] and HMSVM of Lujuan et al. [5], both of them search for a global optimum loss (in this case the loss function l_Δ).

The comparison was done with three benchmark information retrieval (IR) data sets. Thus, in addition to our objective loss function l_Δ (Eq. 3) and $l_{0/1}$ (Eq. 1), we also report the most common performance measures of IR tasks: *precision*, *recall*, and *F1*.

The implementation of H-M^3 and HMSVM were provided by the authors while the implementation of the baseline algorithms (SVM and H-SVM) and the versions in which binary classifiers are not independent (SVM$_{l_\Delta}$ and H-SVM$_{l_\Delta}$) were done modifying slightly Joachims' SVM^{perf} [20]; this SVM implementation provided us with an excellent base due to its linear complexity. In all cases, when it was needed, we set the regularization parameter $C = 1$ and we used a linear kernel. All the scores reported were estimated by means of a stratified fivefold cross validation repeated two times. Following Demsar [21], we used the Wilcoxon signed-ranks test to compare the performance of two classifiers.

We used three well-known data sets in information retrieval [2] . The documents were represented as *bag-of-words* and no word stemming or stop-word removal was performed. The first data set, REUTERS Corpus Volume 1 (RCV1) [22], is composed of 7500 documents described by 19770 features. The 'CCAT' family of categories (Corporate/Industrial news articles) was used as the label hierarchy. This hierarchy represents a tree with maximum depth 3 and with a total of 34 nodes. The tree is quite unbalanced: there are 18 nodes residing in depth 1, 14 nodes in depth 2 and one node in depth 3 (see Figure 2a). In this data set, 8% of examples have multiple partial paths; that is, these examples are classified into classes that are not in the same path from the root. The

[2] These data sets can be downloaded from the following urls: http://users.ecs.soton.ac.uk/cjs/resource_files/hierarchy_data.tar.gz http://people.csail.mit.edu/jrennie/20Newsgroups/

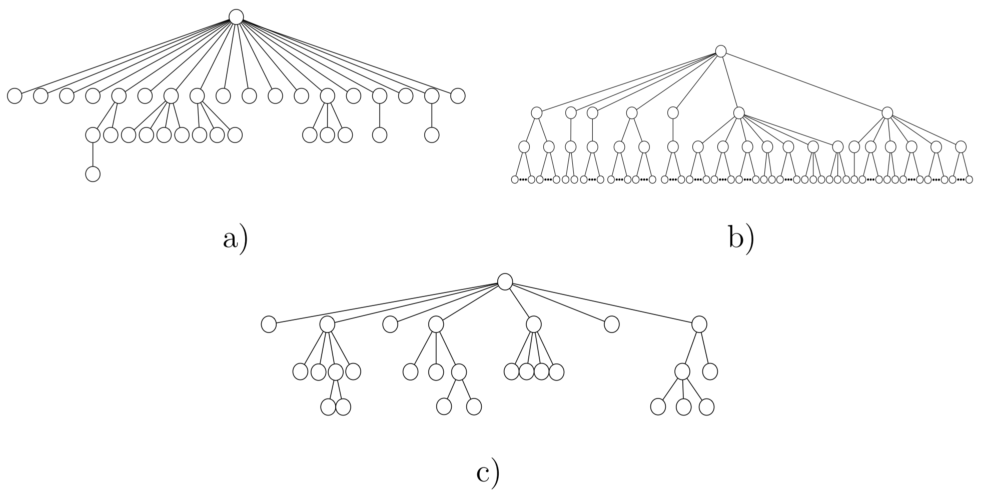

Fig. 2. Hierarchy of the three data sets: a) REUTERS Corpus Volume 1, b) World Intellectual Property Organization, and c) 20 Newsgroups

second dataset, WIPO-alpha, was published by the World Intellectual Property Organization [23], and it is the second data set used in these experiments. We used the D section of its hierarchy. This section contains 1730 documents described by 74436 features. There are 188 nodes in the tree organized as follows: 7 in depth 1, 20 in depth 2, and 160 in depth 3 (see Figure 2b). In this data set there are no examples classified into more than one path and all of them end on a leaf. The last data set, 20 Newsgroups, is a collection of 18774 documents (after removing duplicates) described by 61188 features and belonging to 20 different newsgroups in a hierarchy of 28 nodes: 7 in depth 1, 13 in depth 2, and 7 in depth 3 (see Figure 2c). No partial or multi paths are presented in this data set.

Table 2 shows the scores obtained in RCV1 using the setting described above and stands out those scores that have significant difference with respect to H-SVM$_{l_\Delta}$. The key loss is l_Δ, since this is our optimization target; on the other hand, from an IR point of view *F1* is the most relevant measure. If we refer to the baseline methods and our counterpart versions, H-SVM$_{l_\Delta}$ achieves better l_Δ and *F1* results than H-SVM , and also better than the one-vs-rest variants SVM$_{l_\Delta}$ and SVM . Moreover, the differences between SVM$_{l_\Delta}$ and SVM are significant too in l_Δ ($p < 0.07$) and *F1* ($p < 0.01$). In l_Δ and *F1*, H-SVM$_{l_\Delta}$ outperforms H-M^3 and HMSVM , even the baseline H-SVM is better than both algorithms. Looking at Precision and Recall scores, our method increases the number of false positives and decreases the number of false negatives. Finally, in loss $l_{0/1}$ the differences between our approach and baseline methods are not very clear; in this loss H-M^3 achieves the best result.

The scores in WIPO-alpha data set are shown in Table 3. Here, again, scores that have significant difference with respect to H-SVM$_{l_\Delta}$ are stand out. We can see bigger differences in this data set than in RCV1. Again, algorithm

Table 2
Prediction losses $l_{0/1}$, l_Δ, precision P, recall R and $F1$ (and standard deviations) on RCV1 data set. All scores are given as percentages, but the values of the column labeled by l_Δ are the average of Eq. 3 across the test set. The best result in each measure is highlighted in bold. When the difference between H-SVM$_{l_\Delta}$ and another algorithm is statistically significant in a Wilcoxon signed-ranks test with threshold of 0.05 then a † or a $*$ is attached to the value. The † indicates that the corresponding algorithm is significantly better than H-SVM$_{l_\Delta}$, whereas a $*$ indicates that H-SVM$_{l_\Delta}$ is significantly better than the other algorithm

	$l_{0/1}$	l_Δ	P	R	$F1$
SVM	27.18* ±1.09	0.504* ±0.027	**93.31**† ±0.81	68.94* ±1.56	79.29* ±1.17
SVM$_{l_\Delta}$	27.61* ±0.93	0.500* ±0.024	92.64† ±0.60	69.83* ±1.36	79.63* ±1.02
H-SVM	24.20 ±0.99	0.475 ±0.027	90.56† ±0.89	73.77* ±1.42	81.30* ±1.08
H-SVM$_{l_\Delta}$	24.38 ±0.91	**0.473** ±0.027	89.31 ±1.02	**75.16** ±1.54	**81.62** ±1.07
H-M^3	**23.39**† ±1.15	0.490* ±0.032	89.51 ±0.90	73.59* ±1.72	80.77* ±1.29
HMSVM	28.44* ±1.02	0.743* ±0.033	93.21† ±0.86	50.60* ±1.97	65.58* ±1.76

Table 3
Prediction losses $l_{0/1}$, l_Δ, precision P, recall R and $F1$ (and standard deviations) on WIPO-alpha data set. See column and symbol descriptions in Table 2

	$l_{0/1}$	l_Δ	P	R	$F1$
SVM	83.60* ±2.34	1.641* ±0.029	**94.99**† ±1.11	62.27* ±0.53	75.22* ±0.39
SVM$_{l_\Delta}$	82.34* ±4.01	1.621* ±0.043	94.20† ±1.22	63.41* ±1.28	75.78* ±0.79
H-SVM	74.44* ±3.31	1.553* ±0.055	92.77† ±1.22	66.38* ±1.97	77.36* ±1.11
H-SVM$_{l_\Delta}$	71.58 ±2.12	**1.512** ±0.060	90.76 ±1.12	69.27 ±1.03	**78.57** ±0.88
H-M^3	69.33† ±1.94	1.582* ±0.061	91.92† ±1.23	66.29* ±1.40	77.02* ±1.00
HMSVM	**53.83**† ±2.42	2.240* ±0.139	72.00* ±1.74	**72.00**† ±1.74	72.00* ±1.74

H-SVM$_{l_\Delta}$ outperforms the rest of algorithms in l_Δ and *F1*. Differences between SVM and SVM$_{l_\Delta}$ are significant too in l_Δ ($p < 0.02$) and *F1* ($p < 0.01$). In this data set, HMSVM obtains the best result in the loss $l_{0/1}$ and there is a big difference between H-SVM$_{l_\Delta}$ and its counterpart H-SVM , although in the one-vs-rest variants the difference is smaller.

The same behavior is presented in the scores of 20 Newsgroups dataset (Table 4). Here again, H-SVM$_{l_\Delta}$ achieves better l_Δ and *F1* results than H-SVM , and also better than the one-vs-rest variants SVM$_{l_\Delta}$ and SVM . Differences between SVM$_{l_\Delta}$ and SVM are significant too in l_Δ and *F1* ($p < 0.01$). In this data set, best results are achieved by HMSVM .

We can appreciate that the proposed algorithms perform better in the WIPO-alpha data set. The reason of that behavior is that, paying attention to the

Table 4
Prediction losses $l_{0/1}$, l_{Δ}, precision P, recall R and $F1$ (and standard deviations) on 20 Newsgroups data set. See column and symbol descriptions in Table 2

	$l_{0/1}$	l_{Δ}	P	R	$F1$
SVM	$35.64^{*} \pm 1.16$	$0.716^{*} \pm 0.029$	$89.92^{*} \pm 0.48$	$87.44^{\dagger} \pm 0.65$	$88.66^{*} \pm 0.47$
SVM$_{l_{\Delta}}$	$35.59^{*} \pm 1.22$	$0.714^{*} \pm 0.030$	$89.93^{*} \pm 0.55$	$87.51^{\dagger} \pm 0.57$	88.70 ± 0.46
H-SVM	30.82 ± 1.16	$0.696^{*} \pm 0.030$	92.02 ± 0.41	$85.71^{*} \pm 0.67$	$88.75^{*} \pm 0.51$
H-SVM$_{l_{\Delta}}$	30.78 ± 1.17	0.694 ± 0.030	92.01 ± 0.46	85.78 ± 0.65	88.79 ± 0.50
H-M^3	$34.98^{*} \pm 1.64$	0.687 ± 0.037	$\mathbf{93.11}^{\dagger} \pm 1.12$	$84.85^{*} \pm 0.90$	88.78 ± 0.59
HMSVM	$\mathbf{18.06}^{\dagger} \pm 0.69$	$\mathbf{0.582}^{\dagger} \pm 0.025$	$90.92^{*} \pm 0.38$	$\mathbf{90.89}^{\dagger} \pm 0.41$	$\mathbf{90.91}^{\dagger} \pm 0.40$

hierarchies in Figure 2, RCV1 and 20 Newsgroups data sets have a more simple hierarchy, with several nodes at depth 1. In fact, in RCV1, H-SVM$_{l_{\Delta}}$ and H-SVM only differ in seven out of 34 binary classifiers (the number of internal nodes), and only four of them have a hierarchy with more than two nodes. A similar situation is presented in 20 Newsgroup data set where they only differ in seven out of 28 binary classifiers. The hierarchy of WIPO-alpha is bigger and there are many nodes at different levels. In this case, there are many more internal nodes and most of them have several descendants. This suggests that our method can improve more the performance of H-SVM in those data sets in which the hierarchical structure is more complex.

In our opinion, these results show that we need to consider the existent dependencies between individual classifiers in order to build a useful decomposition hierarchical learning algorithm based on binary SVMs. Our method exploits this aspect, capturing some of those dependencies, thanks to every individual model is learned considering its descendant classifiers and optimizing locally a function loss appropriated to hierarchical classification tasks.

5 Conclusions

In this paper, we have presented a learning method for hierarchical classifications tasks based on the decomposition approach: our learner is composed of a set of individual binary cost-sensitive classifiers, one for each class of the hierarchy. The main novelty of our approach is that these models are not independent from one another, as usually occurs in decomposition approaches. Each individual classifier depends on the models of its descendant classes. The algorithm is based on the use of a bottom-up learning procedure that allows a generic hierarchical loss function to be considered when models are learned. In the experiments reported in Section 4, we also confirm the good scores of this approach against both, state-of-the-art algorithms like [5,6] and baseline methods.

One of the aims of this work is to prove that decomposition methods can be as useful in hierarchical classification tasks as in multiclass classification where this kind of approach is widely used in real applications. Additionally, decomposition strategies generally have the advantage of modularity. It is thus possible to have fast parallel implementations for hierarchical classifications. Moreover, each binary classification could be learned using well-known SVM implementations, and the regularization parameters or the kernels employed may be tuned. As always, the choice of the right kernel may be a crucial point in the performance of any classification task.

Acknowledgements

The authors would like to thank Juho Rousu, Thorsten Joachims and Lijuan Cai for the source code of H-M^3 , SVM_{Perf} and HMSVM , respectively.

The research reported in this paper has been supported in part under Spanish Ministerio de Ciencia e Innovación (MICINN) Grant TIN2008-06247 and by the Fundación para la Investigación Científica y Técnica (FICYT), Asturias, Spain, under Grant IB09-059-C2.

References

[1] D. Koller and M. Sahami, “Hierarchically classifying documents using very few words,” in *ICML’97: Proceedings of the International Conference on Machine Learning*, pp. 170–178, 1997.

[2] S. T. Dumais and H. Chen, “Hierarchical classification of web content,” in *SIGIR-00: Proceedings of the on Research and Development in Information Retrieval*, (New York, NY, USA), pp. 256–263, ACM Press, 2000.

[3] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni, “Incremental algorithms for hierarchical classification,” *Journal of Machine Learning Research*, vol. 7, pp. 31–54, 2006.

[4] L. Cai and T. Hofmann, “Hierarchical document categorization with support vector machines,” in *CIKM ’04: Proceedings of the* 13th *ACM International Conference on Information and Knowledge Management*, (New York, NY, USA), pp. 78–87, ACM Press, 2004.

[5] L. Cai and T. Hofmann, “Exploiting known taxonomies in learning overlapping concepts,” in *IJCAI ’07: Proceedings of the* 20^{th} *International Joint Conference on Artificial Intelligence*, pp. 708–713, 2007.

[6] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor, “Kernel-based learning of hierarchical multilabel classification models,” *Journal of Machine Learning Research*, vol. 7, pp. 1601–1626, 2006.

[7] O. Dekel, J. Keshet, and Y. Singer, “Large margin hierarchical classification,” in *Proceedings of the* 21^{st} *International Conference on Machine learning*, pp. 209–216, 2004.

[8] O. Dekel, J. Keshet, and Y. Singer, “An efficient online algorithm for hierarchical phoneme classification,” in *Proceedings of* 1^{st} *International Workshop on Machine Learning for Multimodal Interaction*, pp. 146–158, 2005.

[9] J. Sinapov and A. Stoytchev, “Detecting the functional similarities between tools using a hierarchical representation of outcomes,” in *7th IEEE International Conference on Development and Learning, 2008. ICDL 2008*, pp. 91–96, 2008.

[10] M. Ashburner, C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, *et al.*, “Gene Ontology: tool for the unification of biology,” *Nature genetics*, vol. 25, no. 1, pp. 25–29, 2000.

[11] Z. Barutcuoglu, R. Schapire, and O. Troyanskaya, “Hierarchical multi-label prediction of gene function,” *Bioinformatics*, vol. 22, no. 7, pp. 830–836, 2006.

[12] J. Bo, M. Brian, Z. Chengxiang, and L. Xinghua, “Multi-label literature classification based on the Gene Ontology graph,” *BMC Bioinformatics*, vol. 9, 2008.

[13] G. Tsoumakas and I. Katakis, “Multi-Label Classification: An Overview,” *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1–13, 2007.

[14] M.-L. Zhang and Z.-H. Zhou, “Ml-knn: A lazy learning approach to multi-label learning,” *Pattern Recognition*, vol. 40, no. 7, pp. 2038 – 2048, 2007.

[15] R. Schapire and Y. Singer, “BoosTexter: A boosting-based system for text categorization,” *Machine learning*, vol. 39, no. 2, pp. 135–168, 2000.

[16] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel, “Decision trees for hierarchical multi-label classification,” *Machine Learning*, vol. 73, no. 2, pp. 185–214, 2008.

[17] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass support vector machines,” *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415–425, 2002.

[18] C. Elkan, “The foundations of cost-sensitive learning,” in *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pp. 973–978, 2001.

[19] H. Fan and K. Ramamohanarao, "A weighting scheme based on emerging patterns for weighted support vector machines," in *Proceedings of IEEE International Conference on Granular Computing*, pp. 435– 440, 2005.

[20] T. Joachims, "Training linear svms in linear time," in *KDD '06: Proceedings of the* 12th *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (New York, NY, USA), pp. 217–226, ACM Press, 2006.

[21] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[22] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *The Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[23] WIPO, "World intellectual property organization," *http://www.wipo.int/classifications/en*, 2001.