trees for hierarchical multi-label classification

earning · November 2008

7-3 · Source: DBLP

Decision Trees for Hierarchical Multi-label Classification

Celine Vens[1], Jan Struyf[1],
Leander Schietgat[1], Sašo Džeroski[2], and Hendrik Blockeel[1]

[1]Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
{Celine.Vens,Jan.Struyf,Leander.Schietgat,Hendrik.Blockeel}@cs.kuleuven.be
[2]Department of Knowledge Technologies, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si

Abstract. Hierarchical multi-label classification (HMC) is a variant of classification where instances may belong to multiple classes at the same time and these classes are organized in a hierarchy. This article presents several approaches to the induction of decision trees for HMC, as well as an empirical study of their use in functional genomics. We compare learning a single HMC tree (which makes predictions for all classes together) to two approaches that learn a set of regular classification trees (one for each class). The first approach defines an independent single-label classification task for each class (SC). Obviously, the hierarchy introduces dependencies between the classes. While they are ignored by the first approach, they are exploited by the second approach, named hierarchical single-label classification (HSC). Depending on the application at hand, the hierarchy of classes can be such that each class has at most one parent (tree structure) or such that classes may have multiple parents (DAG structure). The latter case has not been considered before and we show how the HMC and HSC approaches can be modified to support this setting. We compare the three approaches on 24 yeast data sets using as classification schemes MIPS's FunCat (tree structure) and the Gene Ontology (DAG structure). We show that HMC trees outperform HSC and SC trees along three dimensions: predictive accuracy, model size, and induction time. We conclude that HMC trees should definitely be considered in HMC tasks where interpretable models are desired.

1 Introduction

Classification refers to the task of learning from a set of classified instances a model that can predict the class of previously unseen instances. Hierarchical multi-label classification (HMC) differs from normal classification in two ways: (1) a single example may belong to multiple classes simultaneously; and (2) the classes are organized in a hierarchy: an example that belongs to some class automatically belongs to all its superclasses (we call this the *hierarchy constraint*).

```
1 METABOLISM
1.1 amino acid metabolism
1.1.3 assimilation of ammonia, metabolism of the glutamate group
1.1.3.1 metabolism of glutamine
1.1.3.1.1 biosynthesis of glutamine
1.1.3.1.2 degradation of glutamine
...
1.2 nitrogen, sulfur, and selenium metabolism
...
2 ENERGY
2.1 glycolysis and gluconeogenesis
...
```

Fig. 1. A small part of the hierarchical FunCat classification scheme [24].

Examples of this kind of problems are found in several domains, including text classification [29], functional genomics [3], and object recognition [31]. In functional genomics, which is the application on which we focus, an important problem is predicting the functions of genes. Biologists have a set of possible functions that genes may have, and these functions are organized in a hierarchy (see Fig. 1 for an example). It is known that a single gene may have multiple functions. In order to understand the interactions between different genes, it is important to obtain an interpretable model.

Several methods can be distinguished that handle HMC tasks. A first approach transforms an HMC task into a separate binary classification task for each class in the hierarchy and applies an existing classification algorithm. We refer to it as the SC (single-label classification) approach. This technique has several disadvantages. First, it is *inefficient*, because the learner has to be run $|C|$ times, with $|C|$ the number of classes, which can be hundreds or thousands in some applications. Second, it often results in *learning from strongly skewed class distributions*: in typical HMC applications classes at lower levels of the hierarchy often have very small frequencies, while (because of the hierarchy constraint) the frequency of classes at higher levels tends to be very high. Many learners have problems with strongly skewed class distributions [37]. Third, from the knowledge discovery point of view, the learned models *identify features relevant for one class*, rather than identifying features with high overall relevance. Finally, *the hierarchy constraint is not taken into account*, i.e. it is not automatically imposed that an instance belonging to a class should belong to all its superclasses.

A second approach is to adapt the SC method, so that this last issue is dealt with. Some authors have proposed to hierarchically combine the class-wise models in the prediction stage, so that a classifier constructed for a class c can only predict positive if the classifier for the parent class of c has predicted positive [3, 9]. In addition, one can also take the hierarchy constraint into account during training by restricting the training set for the classifier for class c to those instances belonging to the parent class of c [9]. This approach is called the HSC (hierarchical single-label classification) approach throughout the text.

A third approach is to develop learners that learn a single multi-label model that predicts all the classes of an example at once [11, 7]. Next to taking the

hierarchy constraint into account, this approach is also able to identify features that are relevant to all classes. We call this the HMC approach.

Given our target application of functional genomics, we focus on decision tree methods, because of their interpretability. In Blockeel et al. [7], we presented an empirical study on the use of decision trees for HMC tasks. We presented an HMC decision tree learner, and showed that it can outperform the SC approach on all fronts: predictive performance, model size, and induction time.

In this article, we further investigate the suitability of decision trees for HMC tasks, by extending the analysis along several dimensions. The most important contributions of this work are the following:

- We consider three decision tree approaches towards HMC tasks: (1) learning a separate binary decision tree for each class label (SC), (2) learning and applying such single-label decision trees in a hierarchical way (HSC), and (3) learning one tree that predicts all classes at once (HMC). The HSC approach has not been considered before in the context of decision trees.
- We consider more complex class hierarchies. In particular, the hierarchies are no longer constrained to trees, but can be directed acyclic graphs (DAGs). To our knowledge, this setting has not been thoroughly studied before. We show how the decision tree approaches can be modified to support class hierarchies with a DAG structure.
- The approaches are compared by performing an extensive experimental evaluation on 24 data sets from yeast functional genomics, using as classification schemes MIPS's FunCat [24] (tree structure) and the Gene Ontology [2] (DAG structure). The latter results in datasets with (on average) 4000 class labels, which underlines the scalability of the approaches to large class hierarchies.
- When dealing with the highly skewed class distributions that are characteristic for the HMC setting, precision-recall curves are the most suitable evaluation tool [13]. We propose several ways to perform a precision-recall based analysis in domains with multiple (hierarchically organized) class labels and discuss the difference in their behavior.

The text is organized as follows. We start by discussing previous work in Section 2. Section 3 presents the three decision tree methods for HMC in detail. In Section 4, we extend the algorithms towards DAG structured class hierarchies. In Section 5, we propose the precision-recall based performance measures, used for the empirical study described in Section 6. Finally, we conclude in Section 7.

2 Related Work

Much work in hierarchical multi-label classification (HMC) has been motivated by text classification. Rousu et al. [29] present the state of the art in this domain, which consists mostly of Bayesian and kernel-based classifiers.

Koller and Sahami [22] consider a hierarchical text classification problem setting where each text document belongs to exactly one class at the bottom level

of a topic hierarchy. For each topic in an internal node of the hierarchy, a Bayesian classifier is learned that distinguishes between the possible subtopics, using only those training instances that belong to the parent topic. Test documents are then classified by filtering them through the hierarchy, predicting one topic at each level, until the documents reach the bottom level, thereby ensuring the hierarchy constraint. Errors made at higher levels of the hierarchy are unrecoverable at the lower levels. The procedure is similar to the HSC approach. Nevertheless, as only one path in the hierarchy is predicted, the method is not strictly multi-label. Another difference with HSC is that the node classifiers are not binary.

In the work of Cesa-Bianchi et al. [9], every data instance is labeled with a set of class labels, which may belong to more than one path in the hierarchy. Instances can also be tagged with labels belonging to a path that does not end on a leaf. At each node of the (tree-shaped) taxonomy a binary linear threshold classifier is built, using as training instances only those instances belonging to the node's parent class. This is thus an HSC method. The parameters of the classifier are trained incrementally: at each timestamp, an example is presented to the current set of classifiers, the predicted labels are compared to the real labels, and the classifiers' parameters are updated. In that process, a classifier can only predict positive if its parent classifier has predicted positive, ensuring that the hierarchy constraint is satisfied.

Barutcuoglu et al. [3] recently presented a two-step approach where support vector machines (SVMs) are learned for each class separately, and then combined using a Bayesian network model so that the predictions are consistent with the hierarchy constraint.

Rousu et al. [29] presented a more direct approach that does not require a second step to make sure that the hierarchy constraint is satisfied. Their approach is based on a large margin method for structured output prediction [34, 35]. Such work defines a joint feature map $\Psi(x, y)$ over the input space X and the output space Y. In the context of HMC, the output space Y is the set of all possible subtrees of the class hierarchy. Next, it applies SVM based techniques to learn the weights w of the discriminant function $F(x, y) = \langle w, \Psi(x, y)\rangle$, with $\langle \cdot, \cdot \rangle$ the dot product. The discriminant function is then used to classify a (new) instance x as $\text{argmax}_{y \in Y} F(x, y)$. There are two main challenges that must be tackled when applying this approach to a structured output prediction problem: (a) defining Ψ, and (b) finding an efficient way to compute the argmax function (the range of this function is Y, which is of size exponential in the number of classes). Rousu et al. [29] describe a suitable Ψ and propose an efficient method based on dynamic programming to compute the argmax.

From the point of view of knowledge discovery, it is sometimes useful to obtain more interpretable models, such as decision trees, which is the kind of approach we study here.

Clare and King [12] presented a decision tree method for multi-label classification in the context of functional genomics. In their approach, a tree predicts not a single class but a vector of Boolean class variables. They propose a simple adaptation of C4.5 to learn such trees: where C4.5 normally uses class entropy

for choosing the best split, their version uses the sum of entropies of the class variables. Clare [11] extended the method to predict classes on several levels of the hierarchy, assigning a larger cost to misclassifications higher up in the hierarchy, and presented an evaluation on the twelve data sets that we also use here.

Blockeel et al. [7] proposed the idea of using predictive clustering trees [5, 4] for HMC tasks. As mentioned in the introduction, this work [7] presents the first thorough empirical comparison between an HMC and SC decision tree method in the context of tree shaped class hierarchies.

Geurts et al. [21] recently presented a decision tree based approach related to predictive clustering trees. They start from a different definition of variance and then kernelize this variance function. The result is a decision tree induction system that can be applied to structured output prediction using a method similar to the large margin methods mentioned above [35, 34]. Therefore, this system could also be used for HMC after defining a suitable kernel. To this end, an approach similar to that of Rousu et al. [29] could be used.

3 Decision Tree Approaches for HMC

We start this section by defining the HMC task more formally (Section 3.1). Next, we present the framework of predictive clustering trees (Section 3.2), which will be used to instantiate three decision tree algorithms for HMC tasks: an HMC algorithm (Section 3.3), an SC algorithm (Section 3.4), and an HSC algorithm (Section 3.5). Section 3.6 compares the three algorithms at a conceptual level. In this section, we assume that the class hierarchy has a tree structure. Section 4 will discuss extensions towards hierarchies structured as a DAG.

3.1 Formal Task Description

We define the task of hierarchical multi-label classification as follows:

Given:

- an instance space X,
- a class hierarchy $(C, \leq_h)$, where C is a set of classes and $\leq_h$ is a partial order (structured as a rooted tree for now) representing the superclass relationship (for all $c_1, c_2 \in C$: $c_1 \leq_h c_2$ if and only if c_1 is a superclass of c_2),
- a set T of examples (x_i, S_i) with $x_i \in X$ and $S_i \subseteq C$ such that $c \in S_i \Rightarrow \forall c' \leq_h c : c' \in S_i$, and
- a quality criterion q (which typically rewards models with high predictive accuracy and low complexity).

Find: a function $f : X \rightarrow 2^C$ (where 2^C is the power set of C) such that f maximizes q and $c \in f(x) \Rightarrow \forall c' \leq_h c : c' \in f(x)$. We call this last condition the hierarchy constraint.

In this article, the function f is represented with decision trees.

Table 1. The top-down induction algorithm for PCTs. I denotes the current training instances, t an attribute-value test, $\mathcal{P}$ the partition induced by t on I, and h the heuristic value of t. The superscript * indicates the current best test and its corresponding partition and heuristic. The functions Var, Prototype, and Acceptable are described in the text.

procedure PCT(I) **returns** tree
1: $(t^*, \mathcal{P}^*) = \text{BestTest}(I)$
2: **if** $t^* \neq none$
3: **for each** $I_k \in \mathcal{P}^*$
4: $tree_k = \text{PCT}(I_k)$
5: **return** node(t^*, $\bigcup_k \{tree_k\}$)
6: **else**
7: **return** leaf(Prototype(I))

procedure BestTest(I)
1: $(t^*, h^*, \mathcal{P}^*) = (none, 0, \emptyset)$
2: **for each** possible test t
3: $\mathcal{P}$ = partition induced by t on I
4: $h = \text{Var}(I) - \sum_{I_k \in \mathcal{P}} \frac{|I_k|}{|I|} \text{Var}(I_k)$
5: **if** $(h > h^*) \wedge \text{Acceptable}(t, \mathcal{P})$
6: $(t^*, h^*, \mathcal{P}^*) = (t, h, \mathcal{P})$
7: **return** $(t^*, \mathcal{P}^*)$

3.2 Predictive Clustering Trees

The decision tree methods that we present in the next sections are set in the predictive clustering tree (PCT) framework [5]. This framework views a decision tree as a hierarchy of clusters: the top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. PCTs are constructed so that each split maximally reduces intra-cluster variance. They can be applied to both clustering and prediction tasks, and have clustering trees and (multi-objective) classification and regression trees as special cases.

PCTs [5] can be constructed with a standard "top-down induction of decision trees" (TDIDT) algorithm, similar to CART [8] or C4.5 [27]. The algorithm (Table 1) takes as input a set of training instances I. The main loop (Table 1, BestTest) searches for the best acceptable attribute-value test that can be put in a node. If such a test t^* can be found then the algorithm creates a new internal node labeled t^* and calls itself recursively to construct a subtree for each subset (cluster) in the partition $\mathcal{P}^*$ induced by t^* on the training instances. To select the best test, the algorithm scores the tests by the reduction in variance (which is to be defined further) they induce on the instances (Line 4 of BestTest). Maximizing variance reduction maximizes cluster homogeneity and improves predictive performance. If no acceptable test can be found, that is, if no test significantly reduces variance, then the algorithm creates a leaf and labels it with a representative case, or prototype, of the given instances.

The above description is not very different from that of standard decision tree learners. The main difference is that PCTs treat the variance and prototype functions as parameters, and these parameters are instantiated based on the learning task at hand. To construct a regression tree, for example, the variance function returns the variance of the given instances' target values, and the prototype is the average of their target values. By appropriately defining the variance and prototype functions, PCTs have been used for clustering [5, 33], multi-objective classification and regression [5, 6, 32, 14], and time series data

analysis [16]. Section 3.3 shows how PCTs can, in a similar way, be applied to hierarchical multi-label classification.

The PCT framework is implemented in the Inductive Logic Programming system TILDE [5] and in the CLUS system. We will use the CLUS implementation. More information about CLUS can be found at `http://www.cs.kuleuven.be/~dtai/clus`.

3.3 Clus-HMC: An HMC Decision Tree Learner

To apply PCTs to the task of hierarchical multi-label classification, the variance and prototype parameters are instantiated as follows [7, 4].

First, the example labels are represented as vectors with Boolean components; the i'th component of the vector is 1 if the example belongs to class c_i and 0 otherwise. It is easily checked that the arithmetic mean of a set of such vectors contains as i'th component the proportion of examples of the set belonging to class c_i. We define the variance of a set of examples as the average squared distance between each example's label v_i and the set's mean label $\overline{v}$, i.e.,

$$Var(S) = \frac{\sum_i d(v_i, \overline{v})^2}{|S|}.$$

In the HMC context, it makes sense to consider similarity on higher levels of the hierarchy more important than similarity on lower levels. To that aim, we use a weighted Euclidean distance

$$d(v_1, v_2) = \sqrt{\sum_i w(c_i) \cdot (v_{1,i} - v_{2,i})^2},$$

where $v_{k,i}$ is the i'th component of the class vector v_k of an instance x_k, and the class weights $w(c)$ decrease with the depth of the class in the hierarchy (e.g., $w(c) = w_0^{\text{depth}(c)}$, with $0 < w_0 < 1$). Consider for example the class hierarchy shown in Fig. 2, and two examples (x_1, S_1) and (x_2, S_2) with $S_1 = \{1, 2, 2.2\}$ and $S_2 = \{2\}$. Using a vector representation with consecutive components representing membership of class 1, 2, 2.1, 2.2 and 3, in that order,

$$d([1, 1, 0, 1, 0], [0, 1, 0, 0, 0]) = \sqrt{w_0 + w_0^2}.$$

The heuristic for choosing the best test for a node of the tree is then maximization of the variance reduction as discussed in Section 3.2, with the above definition of variance. Note that this essentially corresponds to converting the example labels to 0/1 vectors and then using the same variance definition as is used when applying PCTs to multi-objective regression [5, 6], but with appropriate weights. In the single-label case, this heuristic is in turn identical to the heuristic used in regression tree learners such as CART [8], and equivalent to the Gini index used by CART in classification tree mode.

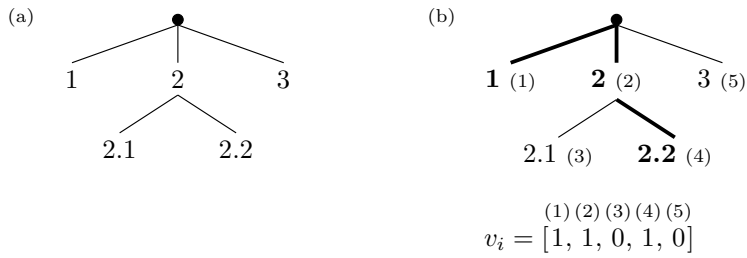

Fig. 2. (a) A small hierarchy. Class label names reflect the position in the hierarchy, e.g., '2.1' is a subclass of '2'. (b) The set of classes {1,2,2.2}, indicated in bold in the hierarchy, and represented as a vector.

A classification tree stores in a leaf the majority class for that leaf; this class will be the tree's prediction for examples arriving in the leaf. But in our case, since an example may have multiple classes, the notion of "majority class" does not apply in a straightforward manner. Instead, the mean $\bar{v}$ of the vectors of the examples in that leaf is stored; in other words, the prototype function returns $\bar{v}$. Fig. 3a shows a simple HMC tree for the hierarchy of Fig. 2.

Recall that $\bar{v}_i$ is the proportion of examples in the leaf belonging to class c_i, which can be interpreted as the probability that an example arriving in the leaf has class c_i. If $\bar{v}_i$ is above some threshold t_i, the example is predicted to belong to class c_i. To ensure that the predictions fulfill the hierarchy constraint (whenever a class is predicted its superclasses are also predicted), it suffices to choose $t_i \leq t_j$ whenever $c_i \leq_h c_j$.

Exactly how the thresholds should be chosen is a question that we do not address here. Depending on the context, a user may want to set the thresholds such that the resulting classifier has maximal predictive accuracy, high precision at the cost of lower recall or vice versa, maximal F1-score (which reflects a particular trade-off between precision and recall), minimal expected misclassification cost (where different types of mistakes may be assigned different costs), maximal interpretability or plausibility of the resulting model, etc. Instead of committing to a particular rule for choosing the threshold, we will study the performance of the predictive models using threshold-independent measures. More precisely, we will use precision-recall curves (as will be clear in Section 5).

Finally, the function Acceptable in Table 1 verifies for a given test that the number of instances in each subset of the corresponding partition $\mathcal{P}$ is at least *mincases* (a parameter) and that the variance reduction is significant according to a statistical F-test. We call the resulting algorithm Clus-HMC.

3.4 Clus-SC: Learning a Separate Tree for Each Class

The second approach that we consider builds a separate tree for each class in the hierarchy (Fig. 3b). Each of these trees is a single-label binary classification tree. Assume that the tree learner takes as input a set of examples labeled positive

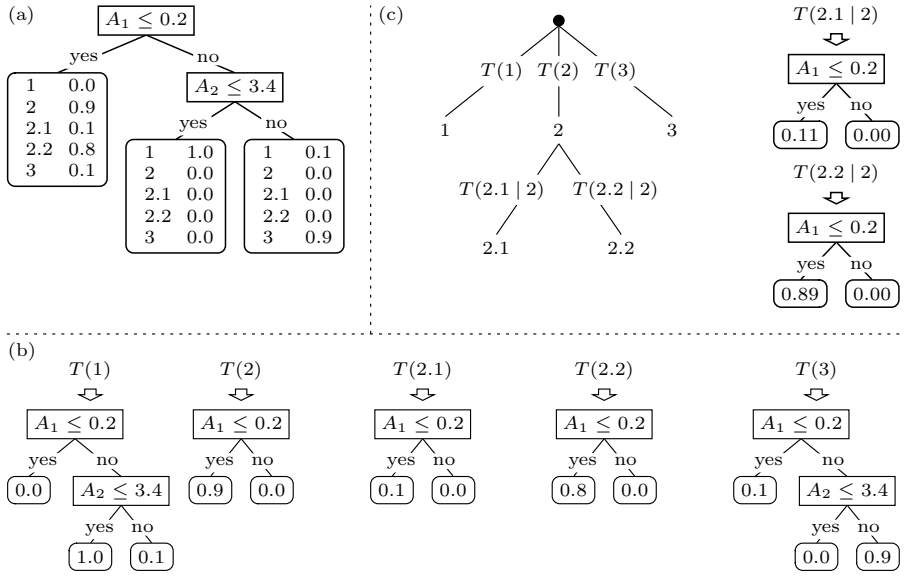

Fig. 3. (a) HMC: one tree predicting, in each leaf, the probability for each class in the hierarchy. (b) SC: a separate tree $T(c_i)$ for each class c_i. (c) HSC: a separate tree for each hierarchy edge. The left part of (c) shows how the HSC trees are organized in the class hierarchy. The right part shows $T(2.1 \mid 2)$ and $T(2.2 \mid 2)$; trees $T(1)$, $T(2)$, and $T(3)$ are identical to those of SC. Note that the leaves of $T(2.1 \mid 2)$ and $T(2.2 \mid 2)$ predict conditional probabilities.

or negative. To construct the tree for class c with such a learner, we label the class c examples positive and all the other examples negative. The resulting tree predicts the probability that a new instance belongs to c. We refer to this method as single-label classification (SC).

In order to classify a new instance, SC thresholds the predictions of the different single-label trees, similar to Clus-HMC. Note, however, that this does not guarantee that the hierarchy constraint holds, even if the thresholds are chosen such that $t_i \leq t_j$ whenever $c_i \leq_h c_j$. Indeed, the structure of the SC trees can be different from that of their parent class's SC tree[1], and therefore, the tree built for, e.g., class 2.1 may very well predict a higher probability than the tree built for class 2 for a given instance. In practice, post-processing techniques can be applied to ensure that a class probability does not exceed its parent class probability. This problem does not occur with Clus-HMC; Clus-HMC always predicts smaller probabilities for specific classes than for more general classes.

The class-wise trees can be constructed with any classification tree induction algorithm. Note that Clus-HMC reduces to a single-label binary classification

[1] Fig. 3 was chosen to show that the different approaches (HMC/SC/HSC) are able to express the same concept; the SC trees all have the same structure and are subtrees of the Clus-HMC tree. In general, this is not the case.

tree learner when applied to such data; its class vector then reduces to a single component and its heuristic reduces to CART's Gini index [8], as pointed out in Section 3.3. We can therefore use the same induction algorithm (CLUS-HMC) for both the HMC and SC approaches. This makes the results easier to interpret. It has been confirmed [7] that on binary classification tasks, CLUS-HMC performs comparably to state of the art decision tree learners. We call the SC approach with CLUS-HMC as decision tree learner CLUS-SC.

3.5 Clus-HSC: Learning a Separate Tree for Each Hierarchy Edge

Building a separate decision tree for each class has several disadvantages, such as the possibility of violating the hierarchy constraint. In order to deal with this issue, the CLUS-SC algorithm can be adapted as follows (Fig. 3c).

For a non top-level class c, it holds that an instance can only belong to c if it belongs to c's parent $par(c)$. An alternative approach to learning a tree that directly predicts c, is therefore to learn a tree that predicts c given that the instance belongs to $par(c)$. Learning such a tree requires fewer training instances: only the instances belonging to $par(c)$ are relevant. The subset of these instances that also belong to c become the positive instances and the other instances (those belonging to $par(c)$ but not to c) the negative instances. The resulting tree predicts the conditional probability $P(c|par(c))$. W.r.t. the top-level classes, the approach is identical to CLUS-SC, i.e., all training instances are used.

To make predictions for a new instance, we use the product rule $P(c) = P(c \mid par(c)) \cdot P(par(c))$ (for non top-level classes). This rule applies the trees recursively, starting from the tree for a top-level class. For example, to compute the probability that the instance belongs to class 2.2, we first use the tree for class 2 to predict $P(2)$ and next the tree for class 2.2 to predict $P(2.2 \mid 2)$. The resulting probability is then $P(2.2) = P(2.2 \mid 2) \cdot P(2)$. Again, these probabilities are thresholded to obtain the predicted set of classes. As with CLUS-HMC, to ensure that this set fulfills the hierarchy constraint, it suffices to choose a threshold $t_i \leq t_j$ whenever $c_i \leq_h c_j$. We call the resulting algorithm CLUS-HSC (hierarchical single-label classification).

3.6 Comparison

To conclude this section, we compare the three proposed approaches (HMC, SC, and HSC) at a conceptual level, according to the properties mentioned in the introduction: the efficiency of learning the models, how skewed class distributions are dealt with, whether the hierarchy constraint is obeyed, and whether global or local features are identified. Other comparison measures, such as predictive performance and model size, will be investigated in depth in the experiments section. Table 2 gives an overview.

Concerning efficiency, Blockeel et al. [7] have shown that CLUS-HMC is more efficient than CLUS-SC. The CLUS-HSC algorithm is expected to be more efficient than CLUS-SC, since smaller training sets are used for constructing the

Table 2. Comparing the three decision tree approaches at a conceptual level.

	CLUS-HMC	CLUS-HSC	CLUS-SC
efficiency	+	+	-
dealing with imbalanced class distributions	?	+/-	-
obeying the hierarchy constraint	+	+	-
identifying global features	+	-	-

trees. Experimental evaluation will have to demonstrate how CLUS-HMC and CLUS-HSC relate.

As mentioned in the introduction, CLUS-SC has to deal with highly skewed class distributions for many of the trees it builds. CLUS-HSC reduces the training data for each class by discarding negative examples that do not belong to the parent class. In most cases, this yields a more balanced class distribution, although there is a small probability that the distribution becomes even more skewed[2]. On average we expect CLUS-HSC to suffer less from imbalanced class distributions than CLUS-SC. For CLUS-HMC, which learns all classes at once, it is difficult to estimate the effect of individual imbalanced class distributions.

As explained before, both CLUS-HMC and CLUS-HSC obey the hierarchy constraint if appropriate threshold values are chosen for each class (e.g., if all thresholds are the same), while CLUS-SC does not.

Finally, whereas the models found by CLUS-HSC and CLUS-SC will contain features relevant for predicting one particular class, CLUS-HMC will identify features with high overall relevance.

4 Hierarchies Structured as DAGs

Until now, we have assumed that the class hierarchy is structured as a rooted tree. In this section, we discuss the issues that arise when dealing with more general hierarchies that are structured as directed acyclic graphs (DAGs). Such a class structure occurs when a given class can have more than one parent class in the hierarchy. An example of such a hierarchy is the Gene Ontology [2], a biological classification hierarchy for genes. In general, a classification scheme structured as a DAG can have two interpretations: if an instance belongs to a class c, then it either belongs also to all superclasses of c, or it belongs also to at least one superclass of c. We focus on the first case, which corresponds to the

[2] Suppose we have 200 examples, of which 100 belong to class 1 and 20 to class 1.1; then when learning class 1.1 from the whole set, 10% of the training examples are positive, while when learning from examples of class 1 only, 20% are positive. So, the problem becomes better balanced. If, on the other hand, among the 100 class 1 examples, 90 belong to 1.1, then the original distribution has $90/200 = 45\%$ positives, whereas when learning from class 1 examples only we have 90% positives: a more skewed dataset. Generally, the problem will become more balanced for classes c where $\frac{N_c}{N} + \frac{N_c}{N_{par(c)}} < 1$ (N denotes the number of examples and $par(c)$ the parent class of c).

"multiple inheritance" interpretation, where a given class inherits the properties (classes) of all its parents. This interpretation is correct for the Gene Ontology.

In the following sections, we discuss the issues that arise when dealing with a DAG type class hierarchy, and discuss the modifications that are required to the algorithms discussed in the previous section to be able to deal with such hierarchies. Obviously, CLUS-SC requires no changes because this method ignores the hierarchical structure of the classes.

4.1 Adaptations to Clus-HMC

CLUS-HMC computes the variance based on the weighted Euclidean distance between class vectors, where a class c's weight $w(c)$ depends on the depth of c in the class hierarchy (e.g., $w(c) = w_0^{\text{depth}(c)}$). When the classes are structured as a DAG, however, the depth of a class is no longer unique: a class may have several depths, depending on the path followed from a top-level class to the given class (see for instance class c_6 in Fig. 4a). As a result, the class weights are no longer properly defined. We therefore propose the following approach. Observe that $w(c) = w_0^{\text{depth}(c)}$ can be rewritten as the recurrence relation $w(c) = w_0 \cdot w(par(c))$, with $par(c)$ the parent class of c, and the weights of the top-level classes equal to w_0. This recurrence relation naturally generalizes to hierarchies where classes may have multiple parents by replacing $w(par(c))$ by an aggregation function computed over the weights of c's parents. Depending on the aggregation function used (sum, min, max, average), we obtain the following approaches:

- $w(c) = w_0 \sum_j w(par_j(c))$ is equivalent to flattening the DAG into a tree (by copying the subtrees that have multiple parents) and then using $w(c) = w_0^{\text{depth}(c)}$. The more paths in the DAG lead to a class, the more important this class is considered by this method. A drawback is that there is no guarantee that $w(c) < w(par_j(c))$. For example, in Fig. 4a, the weight of class c_6 is larger than the weights of both its parents.
- $w(c) = w_0 \cdot \min_j w(par_j(c))$ has the advantage that it guarantees $\forall c, j : w(c) < w(par_j(c))$. A drawback is that it assigns a small weight to a class that has multiple parents and that appears both close to the top-level and deep in the hierarchy.
- $w(c) = w_0 \cdot \max_j w(par_j(c))$ guarantees a high weight for classes that appear close to the top-level of the hierarchy. It does not satisfy $w(c) < w(par_j(c))$, but still yields smaller weights than $w(c) = w_0 \sum_j w(par_j(c))$.
- $w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$ can be considered a compromise in between the "min" and "max" approaches.

We compare the above weighting schemes in the experimental evaluation. Note that all the weighting schemes become equivalent for tree shaped hierarchies.

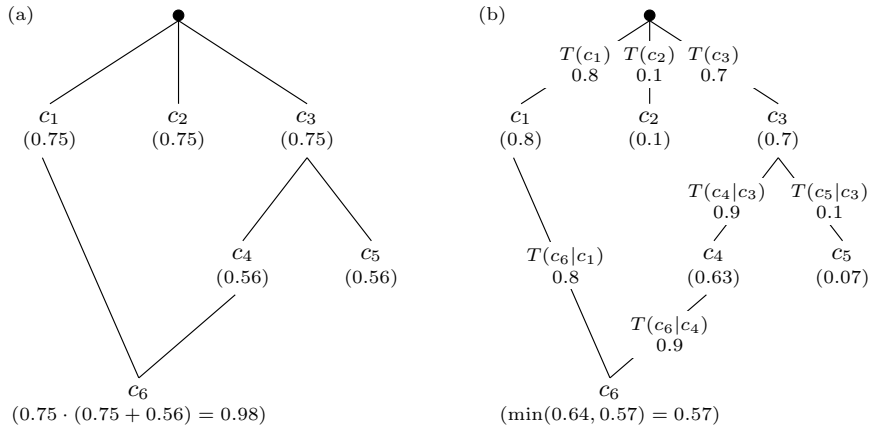

Fig. 4. (a) A class hierarchy structured as a DAG. The class-wise weights computed for Clus-HMC with the weighting scheme $w(c) = w_0 \sum_j w(par_j(c))$ and $w_0 = 0.75$ are indicated below each class. (b) The trees constructed by Clus-HSC. Assume that these trees predict, for a given test instance, the conditional probabilities indicated below each tree. Clus-HSC then predicts the probability of a given class c with the combining rule $P(c) = \min_j P(c \mid par_j(c)) \cdot P(par_j(c))$ (indicated below each class).

4.2 Adaptations to Clus-HSC

Recall that Clus-HSC builds models that predict $P(c \mid par(c))$. This approach can be trivially extended to DAG structured hierarchies by creating one model for each combination of a parent class with one of its children (or equivalently, one model for each hierarchy edge) predicting $P(c \mid par_j(c))$ (Fig. 4b). To make a prediction, the product rule $P(c) = P(c \mid par_j(c)) \cdot P(par_j(c))$ can be applied for each parent class $par_j(c)$, and will yield a valid estimate of $P(c)$ based on that parent. In order to obtain an estimate of $P(c)$ based on all parent classes, we aggregate over the parent-wise estimates.

Recall that Clus-HSC fulfills the hierarchy constraint in the context of tree structured class hierarchies. We want to preserve this property in the case of DAGs. To that aim, we use as aggregate function the minimum of the parent-wise estimates, i.e., $P(c) = \min_j P(c \mid par_j(c)) \cdot P(par_j(c))$. Clus-HSC applies this rule in a top-down fashion (starting with the top-level classes) to compute predicted probabilities for all classes in the hierarchy. Fig. 4b illustrates this process.

Instead of building one tree for each hierarchy edge, one could consider building a tree for each hierarchy node and using as training set for such a tree the instances labeled with all parent classes. This would yield trees predicting $P(c \mid \bigwedge par_j(c))$. While this approach builds fewer trees, it has two important disadvantages. First, the number of training instances per tree can become very small (only the instances that belong to all parent classes are used). Second, the predicted class probabilities are now given by the rule $P(c) =$

$P(c \mid \bigwedge par_j(c)) \cdot P(\bigwedge par_j(c))$, and it is unclear how the last term of this rule can be estimated for a test example. CLUS-HSC therefore relies on the approach outlined above with one model per hierarchy edge.

5 Predictive Performance Measures

After having proposed three decision tree methods for HMC tasks with DAG structured class hierarchies, our next step is to compare their predictive performance, model size, and induction times. Before proceeding, we discuss how to evaluate the predictive performance of the classifiers.

5.1 Hierarchical Loss

Cesa-Bianchi et al. [9] have defined a hierarchical loss function that considers mistakes made at higher levels in the class hierarchy more important than mistakes made at lower levels. The hierarchical loss function for an instance x_k is defined as follows:

$$l_H(x_k) = \sum_i [v_{k,i} \neq y_{k,i} \; and \; \forall c_j \leq_h c_i : v_{k,j} = y_{k,j}],$$

where i iterates over all class labels, v represents the predicted class vector, and y the real class vector. In the work of Cesa-Bianchi et al., the class hierarchy is structured as a tree, and thus, it penalizes the first mistake along the path from the root to a node. In the case of a DAG, the loss function can be generalized in two different ways. One can penalize a mistake if *all* ancestor nodes are predicted correctly (in this case, the above definition carries over), or one can penalize a mistake if there *exists* a correctly predicted path from the root to the node.

In the rest of the article, we do not consider this evaluation function, since it requires thresholded predictions, and we are interested in evaluating our methods regardless of any threshold.

5.2 Precision-Recall Based Evaluation

As argued before, we wish to evaluate our predictive models independently from the threshold, as different contexts may require different threshold settings. Generally, in the binary case, two types of evaluation are suitable for this: ROC analysis and analysis of precision-recall curves (PR curves). While ROC analysis is probably better known in the machine learning community, in our case PR analysis is more suitable. We will explain why this is so in a moment, first we define PR curves.

Precision and recall are traditionally defined for a binary classification task with positive and negative classes. Precision is the proportion of positive predictions that are correct, and recall is the proportion of positive examples that are correctly predicted positive. That is,

$$Prec = \frac{TP}{TP + FP}, \quad \text{and} \quad Rec = \frac{TP}{TP + FN},$$

with TP the number of true positives (correctly predicted positive examples), FP the number of false positives (positive predictions that are incorrect), and FN the number of false negatives (positive examples that are incorrectly predicted negative). Note that these measures ignore the number of correctly predicted negative examples.

A precision-recall curve (PR curve) plots the precision of a model as a function of its recall. Assume the model predicts the probability that a new instance is positive, and that we threshold this probability with a threshold t to obtain the predicted class. A given threshold corresponds to a single point in PR space, and by varying the threshold we obtain a PR curve: while decreasing t from 1.0 to 0.0, an increasing number of instances is predicted positive, causing the recall to increase whereas precision may increase or decrease (with normally a tendency to decrease).

Although a PR curve helps in understanding the predictive behavior of the model, a single performance score is more useful to compare models. A score often used to this end is the area between the PR curve and the recall axis, the so-called "area under the PR curve" (AUPRC). The closer the AUPRC is to 1.0, the better the model is.

The reason why we believe PR curves to be a more suitable evaluation measure in this context is the following. In HMC datasets, it is often the case that individual classes have few positive instances. For example, in functional genomics, typically only a few genes have a particular function. This implies that for most classes, the number of negative instances by far exceeds the number of positive instances. We are more interested in recognizing the positive instances (that an instance has a given label), rather than correctly predicting the negative ones (that an instance does not have a particular label). Although ROC curves are better known, we believe that they are less suited for this task, exactly because they reward a learner if it correctly predicts negative instances (giving rise to a low false positive rate). This can present an overly optimistic view of the algorithm's performance. This effect has been convincingly demonstrated and studied by Davis and Goadrich [13], and we refer to them for further details.

A final point to note is that PR curves can be constructed for each individual class in a multi-label classification task by taking as positives the examples belonging to the class and as negatives the other examples. How to combine the class-wise performances in order to quantify the overall performance, is less straightforward. The following two paragraphs discuss two approaches, each of which are meaningful.

Area Under the Average PR Curve. A first approach to obtain an overall performance score is to construct an overall PR curve by transforming the multi-label problem into a binary problem as follows [39, 36, 7]. Consider a binary classifier that takes as input an (instance,class) couple and predicts whether that instance belongs to that class or not. Precision is then the proportion of positively predicted couples that are positive and recall is the proportion of positive couples that are correctly predicted positive. A rank classifier (which

predicts how likely it is that the instance belongs to the class) can be turned into such a binary classifier by choosing a threshold, and by varying this threshold a PR curve is obtained. We will evaluate our predictive model in exactly the same way as such a rank classifier.

For a given threshold value, this yields one point $(\overline{Prec}, \overline{Rec})$ in PR space, which can be computed as follows:

$$\overline{Prec} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i}, \quad \text{and} \quad \overline{Rec} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i},$$

where i ranges over all classes. (This corresponds to micro-averaging the precision and recall.) In terms of the original problem definition, $\overline{Prec}$ corresponds to the proportion of predicted labels that are correct and $\overline{Rec}$ to the proportion of labels in the data that are correctly predicted.

By varying the threshold, we obtain an average PR curve. We denote the area under this curve with AU($\overline{\text{PRC}}$).

Average Area Under the PR Curves. A second approach is to take the (weighted) average of the areas under the individual (per class) PR curves, computed as follows:

$$\overline{AUPRC}_{w_1,\ldots,w_{|C|}} = \sum_i w_i \cdot AUPRC_i$$

The most obvious instantiation of this approach is to set all weights to $1/|C|$, with C the set of classes. In the results, we denote this measure with $\overline{\text{AUPRC}}$. A second instantiation is to weigh the contribution of a class with its frequency, that is, $w_i = v_i / \sum_j v_j$, with v_i c_i's frequency in the data. The rationale behind this is that for some applications more frequent classes may be more important. We denote the latter measure with $\overline{\text{AUPRC}}_{\text{w}}$.

A corresponding PR curve that has precisely $\overline{AUPRC}_{w_1,\ldots,w_{|C|}}$ as area can be drawn by taking, for each value on the recall axis, the (weighted) pointwise average of the class-wise precision values. Note that the interpretation of this curve is different from that of the micro-averaged PR curve defined in the previous section. For example, each point on this curve may correspond to a different threshold for each class. Section 6.3 presents examples of both types of curves and provides more insight in the difference in interpretation.

6 Experiments in Yeast Functional Genomics

In earlier work [7], Clus-HMC has been compared experimentally to Clus-SC on a tree-shaped hierarchy, showing that Clus-HMC has advantages with respect to accuracy, model size, and computational efficiency (time needed for learning and applying the model). Here we report on a broader and more detailed experimental study, which differs from the previous one in the following ways:

Table 3. Data set properties: number of instances $|D|$, number of attributes $|A|$.

	Data set	$\|D\|$	$\|A\|$		Data set	$\|D\|$	$\|A\|$
D_1	Sequence [11] (seq)	3932	478	D_7	DeRisi et al. [15] (derisi)	3733	63
D_2	Phenotype [11] (pheno)	1592	69	D_8	Eisen et al. [17] (eisen)	2425	79
D_3	Secondary structure [11] (struc)	3851	19628	D_9	Gasch et al. [20] (gasch1)	3773	173
D_4	Homology search [11] (hom)	3867	47034	D_{10}	Gasch et al. [19] (gasch2)	3788	52
D_5	Spellman et al. [30] (cellcycle)	3766	77	D_{11}	Chu et al. [10] (spo)	3711	80
D_6	Roth et al. [28] (church)	3764	27	D_{12}	All microarray [11] (expr)	3788	551

- In the earlier work it was assumed that it is sensible to use greater weights in CLUS-HMC's distance measure for classes higher up in the hierarchy. This was not validated experimentally, however. Here we experiment with different weighting schemes.
- Besides CLUS-HMC and CLUS-SC, CLUS-HSC is included in the comparison. One could argue that CLUS-HSC is a better baseline learner to compare to than CLUS-SC, because the general principle of applying single class learners hierarchically has been proposed before, though not in the decision tree context.
- Only tree-shaped hierarchies were considered previously. We have described how the method can be extended to DAG-shaped hierarchies, but it is not obvious that results on tree-shaped hierarchies will carry over towards DAG-shaped hierarchies.
- For multi-label classification, it is not obvious how to measure the overall performance of a predictive system, averaged out over all classes. In our previous work, precision-recall curves were constructed, using a natural definition of precision and recall over all classes together. Here we suggest a number of alternative measures. It turns out that different measures may give a quite different view of the relative performance of the methods.

Before presenting the results (Section 6.3), we first discuss the data sets used in our evaluation (Section 6.1) and the applied methodology (Section 6.2).

6.1 Data Sets

Saccharomyces cerevisiae (baker's or brewer's yeast) is one of biology's classic model organisms, and has been the subject of intensive study for years.

We use 12 yeast data sets from Clare [11] (Table 3), but with new and updated class labels. The different data sets describe different aspects of the genes in the yeast genome. They include five types of bioinformatic data: sequence statistics, phenotype, secondary structure, homology, and expression. The different sources of data highlight different aspects of gene function. Below, we describe each data set in turn.

D_1 (seq) records sequence statistics that depend on the amino acid sequence of the protein for which the gene codes. These include amino acid ratios, sequence

Table 4. Properties of the two classification schemes. $|C|$ is the average number of classes actually used in the data sets (out of the total number of classes defined by the scheme). $|S|$ is the average number of labels per example, with between parentheses the average number counting only the most specific classes of an example.

	FunCat	GO
Scheme version	2.1 (2007/01/09)	1.2 (2007/04/11)
Yeast annotations	2007/03/16	2007/04/07
Total classes	1362	22960
Data set average $\|C\|$	492 (6 levels)	3997 (14 levels)
Data set average $\|S\|$	8.8 (3.2 most spec.)	35.0 (5.0 most spec.)

length, molecular weight and hydrophobicity. Some of the properties were calculated using ProtParam [18], some were taken from MIPS [24] (e.g., the gene's chromosome number), and some were simply calculated directly. D_1's attributes are mostly real valued, although some (like chromosome number or strand) are discrete.

D_2 (pheno) contains phenotype data, which represents the growth or lack of growth of knock-out mutants that are missing the gene in question. The gene is removed or disabled and the resulting organism is grown with a variety of media to determine what the modified organism might be sensitive or resistant to. This data was taken from EUROFAN, MIPS and TRIPLES [25, 24, 23]. The attributes are discrete, and the data set is sparse, since not all knock-outs have been grown under all conditions.

D_3 (struc) stores features computed from the secondary structure of the yeast proteins. The secondary structure is not known for all yeast genes; however, it can be predicted from the protein sequence with reasonable accuracy. The program Prof [26] was used to this end. Due to the relational nature of secondary structure data, Clare performed a preprocessing step of relational frequent pattern mining; D_3 includes the constructed patterns as binary attributes.

D_4 (hom) includes for each yeast gene, information from other, homologous genes. Homology is usually determined by sequence similarity. PSI-BLAST [1] was used to compare yeast genes both with other yeast genes, and with all genes indexed in SwissProt 39. This provided for each yeast gene, a list of homologous genes. For each of these, various properties were extracted (keywords, sequence length, names of databases they are listed in, ...). Clare preprocessed this data in a similar way as the secondary structure data, to produce binary attributes.

$D_5, \ldots, D_{12}$. The use of microarrays to record the expression of genes is popular in biology and bioinformatics. Microarray chips provide the means to test the expression levels of genes across an entire genome in a single experiment. Many expression data sets exist for yeast, and several of these were used. Attributes for these data sets are real valued, representing fold changes in expression levels.

We construct two versions of each data set. The input attributes are identical in both versions, but the classes are taken from two different classification schemes. In the first version, they are from FunCat (`http://mips.gsf.de/`

projects/funcat), a scheme for classifying the functions of gene products, developed by MIPS [24]. FunCat is a tree-structured class hierarchy; a small part is shown in Fig. 1. In the second version of the data sets, the genes are annotated with terms from the Gene Ontology (GO) [2] (http://www.geneontology.org), which forms a directed acyclic graph instead of a tree: each term can have multiple parents (we use GO's "is-a" relationship between terms)[3]. Table 4 compares the properties of FunCat and GO. Note that GO has an order of magnitude more classes than FunCat for our data sets. The 24 resulting datasets can be found on the following webpage http://www.cs.kuleuven.be/~dtai/clus/hmcdatasets.html.

6.2 Method

Clare [11] presents models trained on 2/3 of each data set and tested on the remaining 1/3. In our experiments we use exactly the same training and test sets.

The stopping criterion (i.e., the function Acceptable in Table 1) was implemented as follows. The minimal number of examples a leaf has to cover was set to 5 for all algorithms. The F-test that is used to check the significance of the variance reduction takes a significance level parameter s, which was optimized as follows: for each out of 6 available values for s, Clus-HMC was run on 2/3 of the training set and its PR curve for the remaining 1/3 validation set was constructed. The s parameter yielding the largest area under this average validation PR curve was then used to train the model on the complete training set. This optimization was performed independently for each evaluation measure (discussed in Section 5) and each weighting scheme (Section 4.1). The PR curves or AUPRC values that are reported are obtained by testing the resulting model on the test set. The results for Clus-SC and Clus-HSC were obtained in the same way as for Clus-HMC, but with a separate run for each class (including separate optimization of s for each class).

We compare the AUPRC of the different methods, using the approaches discussed in Section 5. For GO, which consists of three separate hierarchies, we left out the three classes representing these hierarchies' roots, since they occur for all examples. PR curves were constructed with proper (non-linear) interpolation between points, as described by Davis and Goadrich [13]. The non-linearity comes from the fact that precision is non-linear in the number of true positives and false positives.

To estimate significance of the AUPRC comparison, we use the (two-sided) Wilcoxon signed rank test [38], which is a non-parametric alternative to the paired Student's t-test that does not make any assumption about the distribution of the measurements. In the results, we report the p-value of the test and the corresponding rank sums[4].

[3] The GO versions of the datasets may contain slightly fewer examples, since not all genes in the original datasets are annotated with GO terms.

[4] The Wilcoxon test compares two methods by ranking the pairwise differences in their performances by absolute value. Then it calculates the sums for the ranks

The experiments were run on a cluster of AMD Opteron processors (1.8 - 2.4GHz, >2GB RAM) running Linux.

6.3 Results

In the experiments, we are dealing with many dimensions: we have 12 different descriptions of gene aspects and 2 class hierarchies, resulting in 24 datasets with several hundreds of classes each, on which we want to compare 3 algorithms. Moreover, we consider 3 precision-recall evaluation measures, and for Clus-HMC we proposed 4 weighting schemes. In order to deal with this complex structure, we proceed as follows.

We start by evaluating the different weighting schemes used in the Clus-HMC algorithm. Then we compare the predictive performance of the three algorithms Clus-HMC, Clus-HSC, and Clus-SC. Next, we study the relation between the three evaluation measures AU($\overline{\text{PRC}}$), $\overline{\text{AUPRC}}$, and $\overline{\text{AUPRC}}_{\text{w}}$. Afterwards, we give some example PR curves for specific datasets. Finally, we compare the model size and induction times of the three algorithms.

Comparison of Weighting Schemes. First, we investigate different instantiations for the weights in the weighted Euclidean distance metric used in the heuristic of Clus-HMC. We have arbitrarily set w_0 to 0.75. The precise questions that we want to answer are:

1. Is it useful to use weights in Clus-HMC's heuristic? In other words, is there a difference between using weights that decrease with the hierarchy depth and setting all weights to 1.0?
2. If yes, which of the weighting schemes for combining the weights of multiple parents (Section 4.1) yields the best results for data sets with DAG structured class labels?

Tables 5 (upper part) and 6 show the average AUPRC values, and the Wilcoxon test outcomes for FunCat. As can be seen from the tables, using weights has slight advantages over not using weights. Therefore, for FunCat only results using weights will be reported in the rest of the paper. Recall that FunCat is a tree hierarchy, and thus, the second question does not apply.

For GO, the results are less clear. Table 5 (lower part) shows the average AUPRC values and Fig. 5 visualizes the Wilcoxon outcomes. W.r.t. $\overline{\text{AUPRC}}_{\text{w}}$, there are no differences between the methods. For AU($\overline{\text{PRC}}$), $w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$ performs slightly better than all other methods (although not significant), while for $\overline{\text{AUPRC}}$, $w(c) = w_0 \cdot \max_j w(par_j(c))$ performs better. For the rest of the experiments, we decided to use the former because it also performs well for $\overline{\text{AUPRC}}$ and because the averaging may make the scheme more robust than the scheme that takes the parents' weights maximum.

corresponding to positive and negative differences. The smaller of these two rank sums is compared to a table of all possible distributions of ranks to calculate p.

Table 5. Weighting schemes for FunCat and GO: AU($\overline{\text{PRC}}$), $\overline{\text{AUPRC}}$, and $\overline{\text{AUPRC}}_{\text{w}}$ averaged over all data sets. (90% confidence intervals are indicated after the '±' sign.)

FunCat	AU($\overline{\text{PRC}}$)	$\overline{\text{AUPRC}}$	$\overline{\text{AUPRC}}_{\text{w}}$
1.0	0.191 ± 0.012	0.042 ± 0.006	0.162 ± 0.015
$w_0 \cdot w(par_j(c))$	0.194 ± 0.013	0.045 ± 0.009	0.164 ± 0.017
GO	AU($\overline{\text{PRC}}$)	$\overline{\text{AUPRC}}$	$\overline{\text{AUPRC}}_{\text{w}}$
1.0	0.364 ± 0.011	0.028 ± 0.005	0.342 ± 0.015
$w_0 \sum_j w(par_j(c))$	0.364 ± 0.010	0.028 ± 0.005	0.342 ± 0.015
$w_0 \cdot \min_j w(par_j(c))$	0.365 ± 0.009	0.027 ± 0.005	0.342 ± 0.014
$w_0 \cdot \text{avg}_j w(par_j(c))$	0.365 ± 0.009	0.028 ± 0.005	0.342 ± 0.013
$w_0 \cdot \max_j w(par_j(c))$	0.364 ± 0.009	0.028 ± 0.005	0.342 ± 0.013

Table 6. Weighting schemes for FunCat: comparing $w(c) = w_0 \cdot w(par(c))$ to $w(c) = 1.0$. A '$\oplus$' means that $w(c) = w_0 \cdot w(par(c))$ performs better than $w(c) = 1.0$ according to the Wilcoxon signed rank test. The table indicates the rank sums and corresponding p-values computed by the test.

	$w(c) = w_0 \cdot w(par(c))$	
FunCat	Score	p
AU($\overline{\text{PRC}}$)	$\oplus$ 51/15	0.12
$\overline{\text{AUPRC}}$	$\oplus$ 61/17	0.09
$\overline{\text{AUPRC}}_{\text{w}}$	$\oplus$ 53/25	0.30

Conclusion. Blockeel et al. [7] assumed that it is advisable to use weights in the calculation of Clus-HMC's distance measure, giving greater weights to classes appearing higher in the hierarchy. However, it turns out that using weights is only slightly better than not using weights. For GO, averaging the weights of the parent nodes seems the best option. Recall that for tree shaped hierarchies, the DAG weighting schemes all become identical to the tree weighting scheme. As a result, we can use the same weighting method ($w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$) for both the GO and FunCat experiments.

Precision-recall based comparison of Clus-HMC/SC/HSC. Clus-HMC has been shown to outperform Clus-SC before [7]. This study was performed on datasets with tree structured class hierarchies. Here we investigate how Clus-HSC performs, compared to Clus-HMC and Clus-SC, and whether the results carry over to DAG structured class hierarchies.

Tables 7 and 8 show AUPRC values for the three algorithms for FunCat and GO, respectively. Summarizing Wilcoxon outcomes comparing Clus-HMC to Clus-SC and Clus-HSC are shown in Table 9. We see that Clus-HMC performs better than Clus-SC and Clus-HSC, both for FunCat and GO, and for all evaluation measures.

Table 10 compares Clus-HSC to Clus-SC. Clus-HSC performs better than Clus-SC on GO, w.r.t. all evaluation measures. On FunCat, Clus-HSC is bet-

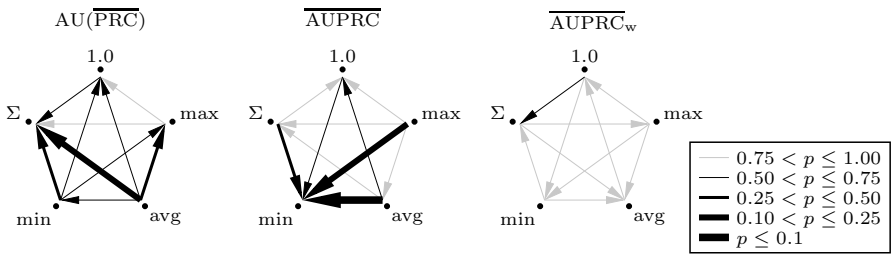

Fig. 5. Weighting schemes for GO: $w(c) = 1.0$, $w(c) = w_0 \sum_j w(par_j(c))$, $w(c) = w_0 \cdot \min_j w(par_j(c))$, $w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$, $w(c) = w_0 \cdot \max_j w(par_j(c))$. An arrow from scheme A to B indicates that A is better than B. The line width of the arrow indicates the significance of the difference according to the Wilcoxon signed rank test.

Table 7. Predictive performance (AUPRC) of the different algorithms for FunCat.

	$\text{AU}(\overline{\text{PRC}})$			$\overline{\text{AUPRC}}$			$\overline{\text{AUPRC}}_w$		
Data set	HMC	HSC	SC	HMC	HSC	SC	HMC	HSC	SC
seq	0.211	0.091	0.095	0.053	0.043	0.042	0.183	0.151	0.154
pheno	0.160	0.152	0.149	0.030	0.031	0.031	0.124	0.125	0.127
struc	0.181	0.118	0.114	0.041	0.039	0.040	0.161	0.152	0.152
hom	0.254	0.155	0.153	0.089	0.067	0.076	0.240	0.205	0.205
cellcycle	0.172	0.111	0.106	0.034	0.036	0.038	0.142	0.146	0.146
church	0.170	0.131	0.128	0.029	0.029	0.031	0.129	0.127	0.128
derisi	0.175	0.094	0.089	0.033	0.029	0.028	0.137	0.125	0.122
eisen	0.204	0.127	0.132	0.052	0.052	0.055	0.183	0.169	0.173
gasch1	0.205	0.106	0.104	0.049	0.047	0.047	0.176	0.154	0.153
gasch2	0.195	0.121	0.119	0.039	0.042	0.037	0.156	0.148	0.147
spo	0.186	0.103	0.098	0.035	0.038	0.034	0.153	0.139	0.139
expr	0.210	0.127	0.123	0.052	0.054	0.050	0.179	0.167	0.167
Average:	0.194	0.120	0.118	0.045	0.042	0.042	0.164	0.151	0.151

ter than Clus-SC w.r.t. $\text{AU}(\overline{\text{PRC}})$. According to the two other evaluation measures, Clus-SC performs better, but the difference is not significant.

Conclusion. The result that Clus-HMC performs better than Clus-SC carries over to DAG structured class hierarchies. Moreover, Clus-HMC also outperforms Clus-HSC in both settings. Clus-HSC in turn outperforms Clus-SC on GO. For FunCat, the results depend on the evaluation measure, and the differences are not significant.

Relation between the different AUPRC measures. In Section 5, we have proposed several ways of combining class-wise PR-curves into a single PR-curve. It turns out that these methods are quite different with respect to what they measure.

Table 8. Predictive performance (AUPRC) of the different algorithms for GO.

	AU($\overline{\text{PRC}}$)			$\overline{\text{AUPRC}}$			$\overline{\text{AUPRC}}_{\text{w}}$		
Data set	HMC	HSC	SC	HMC	HSC	SC	HMC	HSC	SC
seq	0.386	0.282	0.197	0.036	0.035	0.035	0.373	0.283	0.279
pheno	0.337	0.416	0.316	0.021	0.019	0.021	0.299	0.239	0.238
struc	0.358	0.353	0.228	0.025	0.026	0.026	0.328	0.266	0.262
hom	0.401	0.353	0.252	0.051	0.053	0.052	0.389	0.317	0.313
cellcycle	0.357	0.371	0.252	0.021	0.024	0.020	0.335	0.275	0.267
church	0.348	0.397	0.289	0.018	0.016	0.017	0.316	0.248	0.247
derisi	0.355	0.349	0.218	0.019	0.017	0.017	0.321	0.248	0.246
eisen	0.380	0.365	0.270	0.036	0.035	0.031	0.362	0.303	0.294
gasch1	0.371	0.351	0.239	0.030	0.028	0.026	0.353	0.290	0.282
gasch2	0.365	0.378	0.267	0.024	0.026	0.023	0.347	0.282	0.278
spo	0.352	0.371	0.213	0.026	0.020	0.020	0.324	0.254	0.254
expr	0.368	0.351	0.249	0.029	0.028	0.028	0.353	0.286	0.284
Average:	0.365	0.361	0.249	0.028	0.027	0.026	0.342	0.274	0.270

Table 9. CLUS-HMC compared to CLUS-SC and CLUS-HSC. A '$\oplus$' ('$\ominus$') means that CLUS-HMC performs better (worse) than the given method according to the Wilcoxon signed rank test. The table indicates the rank sums and corresponding p-values. Differences significant at the 0.01 level are indicated in bold.

	HMC vs. SC		HMC vs. HSC	
FunCat	Score	p	Score	p
AU($\overline{\text{PRC}}$)	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
$\overline{\text{AUPRC}}$	$\oplus$ 51/27	$3.8 \cdot 10^{-1}$	$\oplus$ 43/35	$7.9 \cdot 10^{-1}$
$\overline{\text{AUPRC}}_{\text{w}}$	$\oplus$ **73/5**	$\mathbf{4.9 \cdot 10^{-3}}$	$\oplus$ **74/4**	$\mathbf{3.4 \cdot 10^{-3}}$
GO	Score	p	Score	p
AU($\overline{\text{PRC}}$)	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ 43/35	$7.9 \cdot 10^{-1}$
$\overline{\text{AUPRC}}$	$\oplus$ 68/10	$2.1 \cdot 10^{-2}$	$\oplus$ 55/23	$2.3 \cdot 10^{-1}$
$\overline{\text{AUPRC}}_{\text{w}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$

This difference is best explained by looking at the behavior of these curves for a default model, that is, a degenerate decision tree that consists of precisely one leaf (one CLUS-HMC leaf, or equivalently, a set of single leaf CLUS-SC trees). The class-wise predicted probabilities of 'default' are constant (the same for each test instance) and equal to the proportion of training instances in the corresponding class, i.e., the class frequency (Fig. 6a).

PR curves of a default classifier. The PR-curve of this default predictor for a single class c_i is as follows: if the overall frequency f_i of the class is above t, then the predictor predicts positive for all instances, so we get a recall of 1 and a precision of f_i; otherwise it predicts negative for all instances, giving a recall of 0 and an undefined precision. This leads to one point in the PR-diagram at $(1, f_i)$. To obtain a PR-curve, observe that randomly discarding a fraction of

Table 10. Clus-HSC compared to Clus-SC. A '$\oplus$' ('$\ominus$') means that Clus-HSC performs better (worse) than Clus-SC according to the Wilcoxon signed rank test.

	HSC vs. SC	
FunCat	Score	p
AU($\overline{\text{PRC}}$)	$\oplus$ 62/16	$7.7 \cdot 10^{-2}$
$\overline{\text{AUPRC}}$	$\ominus$ 37/41	$9.1 \cdot 10^{-1}$
$\overline{\text{AUPRC}}_{\text{w}}$	$\ominus$ 22/56	$2.0 \cdot 10^{-1}$

	HSC vs. SC	
GO	Score	p
AU($\overline{\text{PRC}}$)	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
$\overline{\text{AUPRC}}$	$\oplus$ 49/29	$4.7 \cdot 10^{-1}$
$\overline{\text{AUPRC}}_{\text{w}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$

the predictions results in the same precision, but a smaller recall. The PR-curve thus becomes the horizontal line (r, f_i) with $0 < r \leq 1$ (Fig. 6b) [13].

Consequently, the average PR-curve constructed using the $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ methods is also a horizontal line, at height $\frac{1}{|C|}\sum_i f_i$ or $\sum_i w_i f_i$, respectively. The former is shown in Fig. 6c.

The average PR-curve for AU($\overline{\text{PRC}}$) is quite different, though. This curve is constructed from predictions for all classes together. For a threshold t, all instances are assigned exactly the classes S with frequency above t, i.e., $S = \{c_i \mid f_i \geq t\}$. While decreasing the classification threshold t from 1.0 to 0.0, S grows from the empty set to the set of all classes C. At the same time, the average precision drops from the frequency of the most frequent class to the average of the class frequencies. Correct interpolation between the points [13] leads to curves such as the one shown in Fig. 6d.

Interpretation of different average default curves. Now consider the model 'allclasses' (Fig. 6a), that predicts each class with probability 1.0. This model's classwise PR curves are shown in Fig. 6b and are identical to those of 'default'. As a consequence, also the average PR curve combined with $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ is identical to those of 'default' (Fig. 6c). Since the set $S = C$ for all values of t for 'allclasses', its average PR curve for AU($\overline{\text{PRC}}$) is a horizontal line with precision equal to the average of the class frequencies (Fig. 6d), just as for $\overline{\text{AUPRC}}$. These results show that it is more difficult to outperform 'default' with AU($\overline{\text{PRC}}$) than with $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$: in the latter cases, the model is better than default if it is better than always predicting all classes. Another way of stating this is that AU($\overline{\text{PRC}}$) rewards a predictor for exploiting information about the frequencies of the different classes. The $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ methods, on the other hand, average the performance of individual classes, i.e., they ignore the predictor's ability to learn the class frequencies.

Comparison of Clus-HMC/SC/HSC *to default.* Table 11 compares Clus-HMC, SC, and HSC to the default model. W.r.t. $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$, all models perform better than 'default', and this is true for all 24 data sets. This means that on average, for each individual class, the models perform better than always predicting the class. Interestingly, if we consider AU($\overline{\text{PRC}}$), then Clus-SC, and also Clus-HSC on FunCat, perform worse than 'default'. W.r.t. this evaluation measure, these methods produce overly complex models and may overfit the training data.

(a)

default		allclasses	
c_i	v_i	c_i	v_i
c_1	0.9	c_1	1.0
c_2	0.5	c_2	1.0
c_3	0.1	c_3	1.0

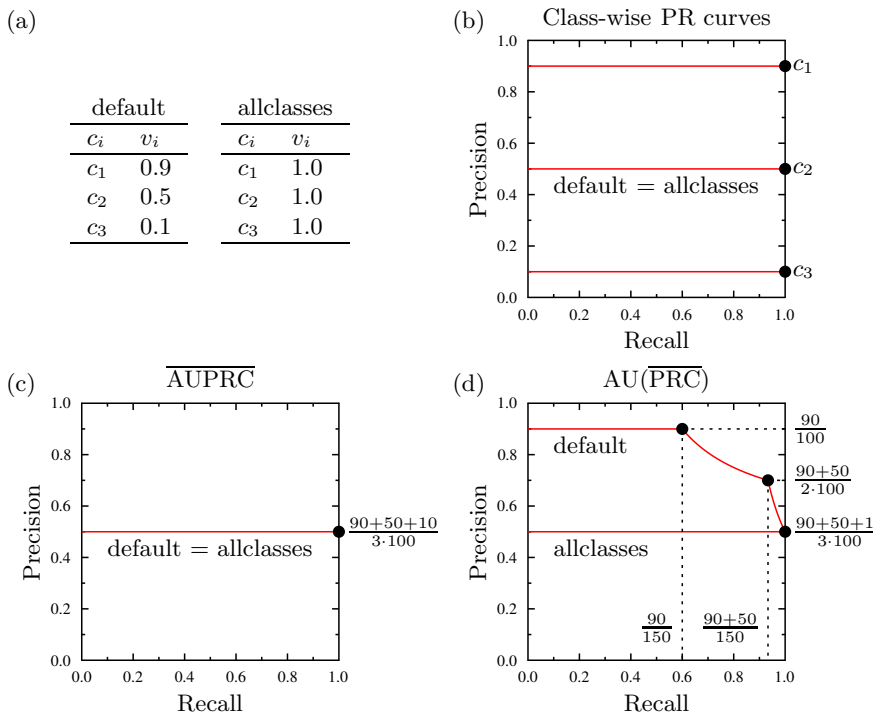

Fig. 6. Example for a dataset with 100 instances. (a) Two degenerate decision tree models: 'default' and 'allclasses'. The 'default' model's predicted set of classes depends on the classification threshold t, while the 'allclasses' model predicts all classes independent of t. (b) Class-wise PR curves (identical for 'default' and 'allclasses'). (c) Average PR curve corresponding to $\overline{\text{AUPRC}}$. (d) Average PR curves corresponding to AU($\overline{\text{PRC}}$) (the non-linear curves connecting the points are obtained by means of proper PR interpolation [13]).

The overfitting can be quantified by subtracting the AUPRC obtained on the test set from that on the training set. Table 12 shows these differences, which are indeed highest for the Clus-SC and Clus-HSC methods.

Conclusion. We have shown that the three proposed ways of averaging classwise PR curves are indeed different. The AU($\overline{\text{PRC}}$) evaluation measure looks at the performance of the model in a mix of classes, whereas the $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ measures evaluate the performance of individual classes independently. W.r.t. the AU($\overline{\text{PRC}}$) measure, Clus-SC, and also Clus-HSC on FunCat, were shown to overfit the training data.

Example PR curves for specific datasets. Fig. 7 and 8 show averaged PR curves for two data sets. These curves illustrate the above conclusions. We see

Table 11. Clus-SC, Clus-HSC, and Clus-HMC compared to the default model. A '$\oplus$' ('$\ominus$') means that the given method performs better (worse) than default according to the Wilcoxon signed rank test.

	SC vs. DEF		HSC vs. DEF		HMC vs. DEF	
FunCat	Score	p	Score	p	Score	p
AU($\overline{\text{PRC}}$)	$\ominus$ **1/77**	$\mathbf{9.8 \cdot 10^{-4}}$	$\ominus$ **2/76**	$\mathbf{1.5 \cdot 10^{-3}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
$\overline{\text{AUPRC}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
$\overline{\text{AUPRC}}_{\text{w}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
GO	Score	p	Score	p	Score	p
AU($\overline{\text{PRC}}$)	$\ominus$ **0/78**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ 68/10	$2.1 \cdot 10^{-2}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
$\overline{\text{AUPRC}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$
$\overline{\text{AUPRC}}_{\text{w}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$	$\oplus$ **78/0**	$\mathbf{4.9 \cdot 10^{-4}}$

Table 12. Difference between AUPRC obtained on the training set and AUPRC obtained on the test set. A higher (lower) difference indicates more (less) overfitting.

	FunCat			GO		
	HMC	HSC	SC	HMC	HSC	SC
AU($\overline{\text{PRC}}$)	0.034	0.435	0.464	0.027	0.293	0.402
$\overline{\text{AUPRC}}$	0.075	0.248	0.267	0.045	0.218	0.190
$\overline{\text{AUPRC}}_{\text{w}}$	0.109	0.375	0.389	0.061	0.317	0.308

that, for both data sets, Clus-HMC performs best, especially for the AU($\overline{\text{PRC}}$) or $\overline{\text{AUPRC}}_{\text{w}}$ evaluation measures. If we consider AU($\overline{\text{PRC}}$), then overfitting can be detected for Clus-SC and Clus-HSC.

Fig. 9 shows a number of class-wise PR curves for the dataset 'hom'. We have chosen the four classes for which Clus-HMC (parameter s optimized for AU($\overline{\text{PRC}}$)) yields the largest AUPRC on the validation set, compared to the default AUPRC for that class. Since not all of these classes occurred in the test set, we have chosen the four best classes that occur in at least 5% of the test examples. We see that for FunCat, Clus-HMC performs better on these classes, while for GO, the results are less clear. Indeed, if we look at Table 8, we see that the three algorithms perform similarly for this data set if all classes are considered equally important (corresponding to the $\overline{\text{AUPRC}}$ evaluation method). However, the other evaluation methods (which do take into account class frequencies) show a higher gain for Clus-HMC, which indicates that the latter performs better on the more frequent classes. Plotting the difference in AUPRC against class frequency (Fig. 10) confirms this result.

Fig. 11 shows a part of the tree learned by Clus-HMC for the dataset 'hom' for GO. The tree contains in total 51 leaves, each predicting a probability for each of the 3997 classes. In the figure, we only show classes for which this probability exceeds 85% and which are most specific. The homology features are based on a sequence similarity search for each gene in yeast against all the genes in a large database called SwissProt. The root test, for instance, tests whether there exists a SwissProt protein A that has a high similarity (e-value lower than 1.0e-

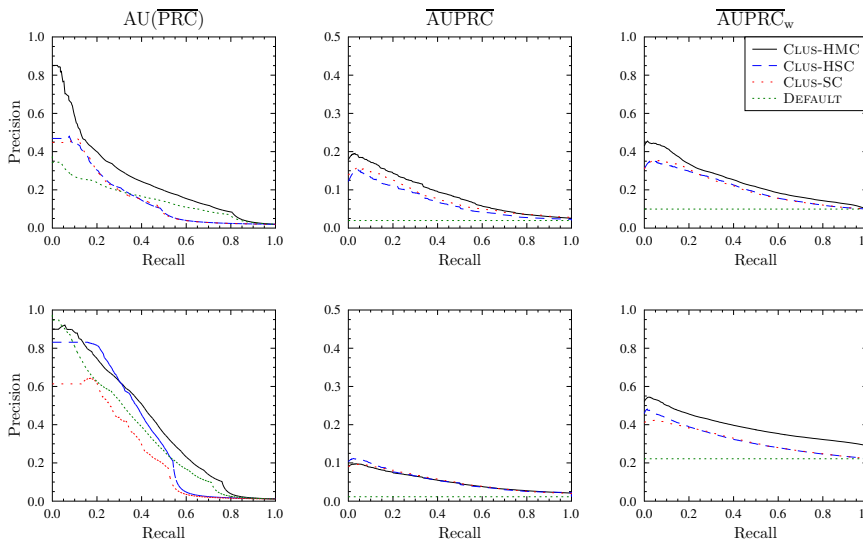

Fig. 7. PR curves averaged over all classes according to the 3 evaluation measures for FunCat (top) and GO (bottom) for the data set 'hom'.

8) with the gene under consideration, has "inner_membrane" listed as one of its keywords and has references to a database called "prints".

Comparison of Clus-HMC/SC/HSC's Tree Size and Induction Time. We conclude this experimental evaluation by comparing the model size and computational efficiency of the three algorithms.

Tables 13 and 14 present the tree sizes obtained with the different methods. We measure tree size as the number of leaves. The tables include three numbers for Clus-HMC: one number for each of the evaluation measures. Recall that Clus-HMC uses the evaluation measure to tune its F-test parameter s. Different evaluation measures may yield different optimal s values and therefore different trees. SC and HSC trees, on the other hand, predict only one class, so there is no need to average PR-curves; the tree induction algorithm tunes its F-test parameter to maximize its class's AUPRC.

Averaged over the evaluation measures and datasets, the HMC trees contain 60.9 (FunCat) and 53.6 (GO) leaves. The SC trees, on the other hand, are smaller because they each model only one class. They include on average 15.9 (FunCat) and 7.6 (GO) leaves. Nevertheless, the total size of all SC trees is on average a factor 311.2 (FunCat) and 1049.8 (GO) larger than the corresponding HMC tree. This difference is bigger for GO than for FunCat because GO has an order of magnitude more classes (Table 4) and therefore also an order of magnitude more SC trees. Comparing HMC to HSC yields similar conclusions.

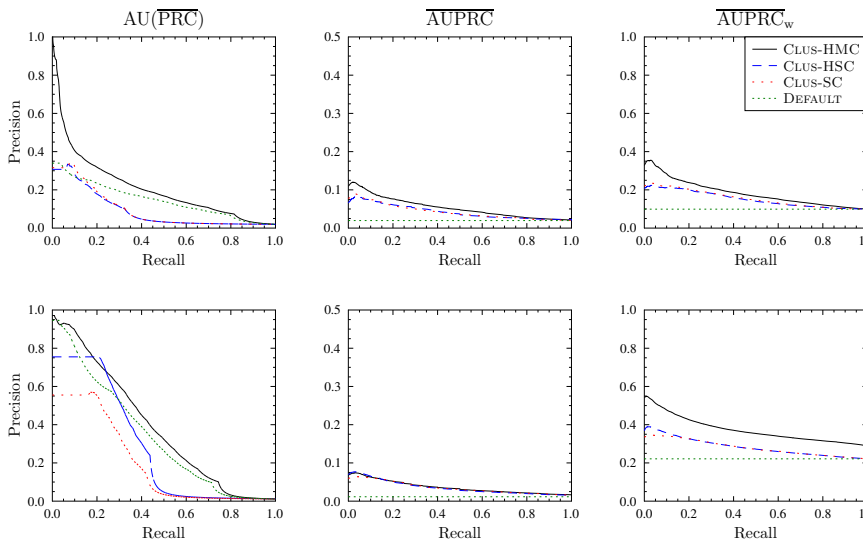

Fig. 8. PR curves averaged over all classes according to the 3 evaluation measures for FunCat (top) and GO (bottom) for the data set 'seq'.

Compared to Blockeel et al. [7], we observe a similar average tree size for the single-label SC trees. However, the HMC trees, which had on average 12.5 leaves for the reported AU($\overline{\text{PRC}}$) measure in [7], now contain on average 19.8 leaves. This can be explained by the fact that the HMC trees now have to predict more classes: the FunCat classification scheme has grown from 250 to 1362 classes meanwhile.

Observe that the HSC trees are smaller than the SC trees (a factor 2.2 on FunCat and 2.8 on GO). We see two reasons for this. First, HSC trees encode less knowledge than SC ones because they are conditioned on their parent class. That is, if a given feature subset is relevant to all classes in a sub-lattice of hierarchy, then Clus-SC must include this subset in each tree of the sub-lattice, while Clus-HSC only needs them in the trees for the sub-lattice's most general border. Second, HSC trees use fewer training examples than SC trees, and tree size typically grows with training set size.

We also measure the total induction time for all methods. This is the time for building the actual trees; it does not include the time for loading the data and tuning the F-test parameter. Clus-HMC requires on average 3.3 (FunCat) and 24.4 (GO) minutes to build a tree. Clus-SC is a factor 58.6 (FunCat) and 129.0 (GO) slower than Clus-HMC. Clus-HSC is a factor 10.2 (FunCat) and 5.1 (GO) faster than Clus-SC, but still a factor 6.3 (FunCat) and 55.9 (GO) slower than Clus-HMC.

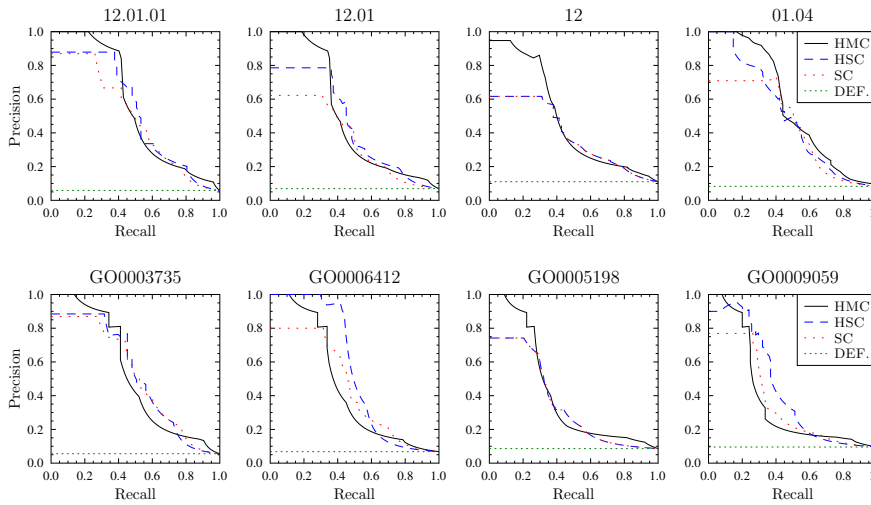

Fig. 9. Example class-wise PR curves for FunCat (top) and GO (bottom) for the data set 'hom'.

Conclusion. Whereas the size of the individual trees learned by Clus-HSC and Clus-SC is smaller than the size of the trees output by Clus-HMC, the total model size of the latter is much smaller than the total size of the models output by the single-label tree learners. As was expected, the Clus-HSC models are smaller than the Clus-SC models. Also w.r.t. efficiency, Clus-HMC outperforms the other methods.

7 Conclusions

In hierarchical multi-label classification, the task is to assign a set of class labels to examples, where the class labels are organized in a hierarchy: an example can only belong to a class if it also belongs to the class's superclasses. An important application area is functional genomics, where the goal is to predict the functions of gene products.

In this article, we have compared three decision tree algorithms on the task of hierarchical multi-label classification: (1) an algorithm that learns a single tree that predicts all classes at once (Clus-HMC), (2) an algorithm that learns a separate decision tree for each class (Clus-SC), and (3) an algorithm that learns and applies such single-label decision trees in a hierarchical way (Clus-HSC). The three algorithms are instantiations of the predictive clustering tree framework [5] and are designed for problems where the class hierarchy is either structured as a tree or as a directed acyclic graph (DAG). To our knowledge, the latter setting has not been studied before, although it occurs in real life applications. For instance, the Gene Ontology (GO), a widely used classification

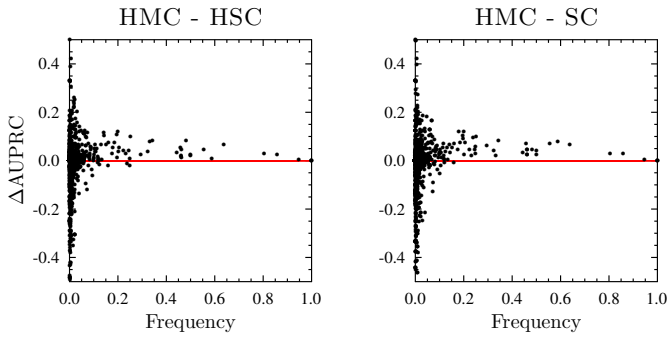

Fig. 10. Difference in AUPRC versus class frequency for GO for the data set 'hom'.

Table 13. Tree size (number of tree leaves) for FunCat.

	Clus-HMC			Clus-SC		Clus-HSC	
Data set	AU($\overline{\text{PRC}}$)	$\overline{\text{AUPRC}}$	$\overline{\text{AUPRC}}_w$	Total	Average	Total	Average
seq	14	168	168	10443	20.9	4923	9.9
pheno	8	8	8	1238	2.7	777	1.7
struc	12	125	56	8657	17.3	3917	7.8
hom	75	190	75	9137	18.3	4289	8.6
cellcycle	24	61	61	9671	19.4	4037	8.1
church	17	17	17	4186	8.4	2221	4.5
derisi	4	68	68	7807	15.6	3520	7.1
eisen	29	55	55	6311	13.7	2995	6.5
gasch1	10	96	96	10447	20.9	4761	9.5
gasch2	26	101	101	7850	15.7	3756	7.5
spo	6	43	43	8527	17.1	3623	7.3
expr	12	161	116	10262	20.6	4711	9.4
Average:	19.8	91.1	72.0	7878	15.9	3628	7.3

scheme for genes, is structured as a DAG. The DAG structure poses a number of complications to the algorithms, e.g., the depth of a class in the hierarchy is no longer unique.

We have evaluated the algorithms on 24 datasets from functional genomics. The predictive performance was measured as area under the PR curve. For a single-label classification task this measure is well-defined, but for a multi-label problem the definition needs to be extended and there are several ways to do so. We propose three ways to construct a PR curve for the multi-label case: micro-averaging precision and recall for varying thresholds, taking the point-wise average of class-wise precision values for each recall value, and weighing the contribution of each class in this average by the class's frequency.

The most important results of our empirical evaluation are as follows. First, Clus-HMC has a better predictive performance than Clus-SC and Clus-HSC,

$eval(A, S), S <$ 1e-8, $keyword(A, inner\_membrane), dbref(A, prints)$
yes: $eval(B, S), S <$ 1e-8, $class(B, lagomorpha), keyword(B, transmembrane)$
| +yes: $eval(C, S), S <$ 1e-8, $molwt(C, M), M < 74079, M > 53922, dbref(C, mim)$
| | +yes: $eval(D, S), S <$ 1e-8, $class(D, streptococcaceae), db\_ref(D, hssp)$
| | | +yes: GO0042626s,GO0044464,GO0008150 [10 ex.]
| | | +no: $eval(E, S), S <$ 1e-8, $class(E, percomorpha)$
| | | +yes: GO0005215,GO0044464,GO0006810 [15 ex.]
| | | +no: GO0003674,GO0005575,GO0008150 [69 ex.]
| | +no: $eval(F, S), 4.5\text{e-}2 < S < 1.1, molwt(F, N), N > 109335), class(F, bscg)$
| | +yes: $eval(G, S), S <$ 1e-8, $class(G, hydrozoa)$
| | | +yes: GO0015662,GO0044464,GO0008150 [6 ex.]
| | | +no: GO0003674,GO0044464,GO0008150 [12 ex.]
| | +no: GO0005215,GO0044444,GO0008150 [24 ex.]
| +no: GO0003674,GO0005575,GO0008150 [85 ex.]
+--no: ...

Fig. 11. Part of the CLUS-HMC (optimized for AU($\overline{\text{PRC}}$)) tree that was learned for the 'hom' dataset for GO. Only the most specific classes for which the predicted probability exceeds 85% are shown.

both for tree and DAG structured class hierarchies, and for all evaluation measures. Whereas this result was already shown for CLUS-HMC and CLUS-SC by Blockeel et al. [7] in a limited setting, it was unknown where CLUS-HSC would fit in. Somewhat unexpectedly, learning a single-label tree for each class separately, where one only focuses on the examples belonging to the parent class, results in lower predictive performance than learning one single model for all classes. That is, CLUS-HMC outperforms CLUS-HSC. CLUS-HSC in turn outperforms CLUS-SC for DAGs; for trees the performances are similar.

Second, we have compared the precision-recall behavior of the algorithms to that of a default model. Using micro-averaged PR curves, we have observed that CLUS-SC performs consistently (for 23 out of 24 datasets) worse than default, indicating that it builds overly complex models that overfit the training data. Interestingly, the other precision-recall averaging methods are not able to detect this overfitting.

Third, the size of the HMC tree is much smaller (2 to 3 orders of magnitude) than the total size of the models output by CLUS-HSC and CLUS-SC. As was expected, the CLUS-HSC models are smaller than the CLUS-SC models (a factor 2 to 3).

Fourth, we find that learning a single HMC tree is also much faster than learning many regular trees. Whereas CLUS-HMC has been shown to be more efficient than CLUS-SC before [7], it turns out to be also more efficient than CLUS-HSC. Obviously, a single HMC tree is also much more efficient to apply than 4000 (for GO) separate trees.

Given the positive results for HMC decision trees on predictive performance, model size, and efficiency, we can conclude that their use should definitely be considered in HMC tasks where interpretable models are desired.

Table 14. Tree size (number of tree leaves) for GO.

	Clus-HMC			Clus-SC		Clus-HSC	
Data set	AU($\overline{\text{PRC}}$)	$\overline{\text{AUPRC}}$	$\overline{\text{AUPRC}}_{\text{w}}$	Total	Average	Total	Average
seq	15	206	108	38969	9.4	21703	3.7
pheno	6	6	6	6213	2.0	5691	1.3
struc	14	76	76	36356	8.8	19147	3.3
hom	51	135	135	35270	8.5	19804	3.4
cellcycle	21	63	43	36260	8.8	19085	3.3
church	7	21	21	16049	3.9	12368	2.1
derisi	10	38	10	31175	7.6	16693	2.9
eisen	37	68	68	24844	7.0	14384	2.9
gasch1	30	129	30	37838	9.2	20070	3.4
gasch2	27	62	62	34204	8.3	18546	3.2
spo	14	60	60	35400	8.6	15552	2.7
expr	35	145	35	38313	9.3	20812	3.6
Average:	22.2	84.1	54.5	30908	7.6	16988	3.0

Acknowledgments

Celine Vens is supported by the EU FET IST project "Inductive Querying", contract number FP6-516169 and the Research Fund K.U.Leuven. Jan Struyf and Hendrik Blockeel are postdoctoral fellows of the Research Foundation - Flanders (FWO-Vlaanderen). Leander Schietgat is supported by a PhD grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

The authors would like to thank Amanda Clare for providing us with the datasets and Kurt De Grave for carefully reading the text and providing many useful suggestions. This research was conducted utilizing high performance computational resources provided by K.U.Leuven, `http://ludit.kuleuven.be/hpc`.

References

1. S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucl. Acids Res.*, 25:3389–3402, 1997.
2. M. Ashburner et al. Gene ontology: Tool for the unification of biology. The Gene Ontology Consortium. *Nature Genet.*, 25(1):25–29, 2000.
3. Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.
4. H. Blockeel, M. Bruynooghe, S. Džeroski, J. Ramon, and J. Struyf. Hierarchical multi-classification. In *Proc. of the ACM SIGKDD 2002 Workshop on Multi-Relational Data Mining (MRDM 2002)*, pages 21–35, 2002.
5. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proc. of the 15th Int'l Conf. on Machine Learning*, pages 55–63, 1998.
6. H. Blockeel, S. Džeroski, and J. Grbović. Simultaneous prediction of multiple chemical parameters of river water quality with Tilde. In *Proc. of the 3rd European Conf. on Principles of Data Mining and Knowledge Discovery*, pages 32–40, 1999.

7. H. Blockeel, L. Schietgat, J. Struyf, S. Džeroski, and A. Clare. Decision trees for hierarchical multilabel classification: A case study in functional genomics. In *Proc. of the 10th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 18–29, 2006.
8. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
9. N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Incremental algorithms for hierarchical classification. *J. Mach. Learn. Res.*, 7:31–54, 2006.
10. S. Chu, J. DeRisi, M. Eisen, J. Mulholland, D. Botstein, P. Brown, and I. Herskowitz. The transcriptional program of sporulation in budding yeast. *Science*, 282:699–705, 1998.
11. A. Clare. *Machine learning and data mining for yeast functional genomics*. PhD thesis, University of Wales, Aberystwyth, 2003.
12. A. Clare and R.D. King. Knowledge discovery in multi-label phenotype data. In *5th European Conf. on Principles of Data Mining and Knowledge Discovery*, pages 42–53, 2001.
13. J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *Proc. of the 23rd Int'l Conf. on Machine Learning*, pages 233–240, 2006.
14. D. Demšar, S. Džeroski, T. Larsen, J. Struyf, J. Axelsen, M. Bruus Pedersen, and P. Henning Krogh. Using multi-objective classification to model communities of soil microarthropods. *Ecological Modelling*, 191(1):131–143, 2006.
15. J. DeRisi, V. Iyer, and P. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.
16. S. Džeroski, I. Slavkov, V. Gjorgjioski, and J. Struyf. Analysis of time series data with predictive clustering trees. In *Proc. of the 5th Int'l Workshop on Knowledge Discovery in Inductive Databases*, pages 47–58, 2006.
17. M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Nat. Acad. Sci. USA*, 95:14863–14868, 1998.
18. Expasy. ProtParam. `http://www.expasy.org/tools/protparam.html`.
19. A. Gasch, M. Huang, S. Metzner, D. Botstein, S. Elledge, and P. Brown. Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p. *Mol. Biol. Cell*, 12(10):2987–3000, 2001.
20. A. Gasch, P. Spellman, C. Kao, O. Carmel-Harel, M. Eisen, G. Storz, D. Botstein, and P. Brown. Genomic expression program in the response of yeast cells to environmental changes. *Mol. Biol. Cell*, 11:4241–4257, 2000.
21. P. Geurts, L. Wehenkel, and F. d'Alché-Buc. Kernelizing the output of tree-based methods. In *Proc. of the 23th Int'l Conf. on Machine Learning*, pages 345–352, 2006.
22. D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proc. of the 14th Int'l Conf. on Machine Learning*, pages 170–178, 1997.
23. A. Kumar, K.H. Cheung, P. Ross-Macdonald, P.S.R. Coelho, P. Miller, and M. Snyder. TRIPLES: A database of gene function in S. cerevisiae. *Nucleic Acids Res.*, 28:81–84, 2000.
24. H.W. Mewes, K. Heumann, A. Kaps, K. Mayer, F. Pfeiffer, S. Stocker, and D. Frishman. MIPS: A database for protein sequences and complete genomes. *Nucl. Acids Res.*, 27:44–48, 1999.
25. S. Oliver. A network approach to the systematic analysis of yeast gene function. *Trends in Genetics*, 12(7):241–242, 1996.
26. M. Ouali and R.D. King. Cascaded multiple classifiers for secondary structure prediction. *Protein Science*, 9(6):1162–76, 2000.

27. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
28. F. Roth, J. Hughes, P. Estep, and G. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, 16:939–945, 1998.
29. J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *J. Mach. Learn. Res.*, 7:1601–1626, 2006.
30. P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9:3273–3297, 1998.
31. B. Stenger, A. Thayananthan, P. Torr, and R. Cipolla. Estimating 3D hand pose using hierarchical multi-label classification. *Image and Vision Computing*, 5(12):1885–1894, 2007.
32. J. Struyf and S. Džeroski. Constraint based induction of multi-objective regression trees. In *Knowledge Discovery in Inductive Databases, 4th Int'l Workshop, KDID'05, Revised, Selected and Invited Papers*, pages 222–233, 2006.
33. J. Struyf and S. Džeroski. Clustering trees with instance level constraints. In *Proc. of the 18th European Conf. on Machine Learning*, pages 359–370, 2007.
34. B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems 16*, 2003.
35. I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, 2005.
36. G. Tsoumakas and I. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Proc. of the 18th European Conf. on Machine Learning*, pages 406–417, 2007.
37. G.M. Weiss and F.J. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Art. Intell. Res.*, 19:315–354, 2003.
38. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.
39. Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, 1999.