# Multi-Label Classification on Tree- and DAG-Structured Hierarchies

**Wei Bi** WEIBI@CSE.UST.HK
**James T. Kwok** JAMESK@CSE.UST.HK
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong

## Abstract

Many real-world applications involve multi-label classification, in which the labels are organized in the form of a tree or directed acyclic graph (DAG). However, current research efforts typically ignore the label dependencies or can only exploit the dependencies in tree-structured hierarchies. In this paper, we present a novel hierarchical multi-label classification algorithm which can be used on both tree- and DAG-structured hierarchies. The key idea is to formulate the search for the optimal consistent multi-label as the finding of the best subgraph in a tree/DAG. Using a simple greedy strategy, the proposed algorithm is computationally efficient, easy to implement, does not suffer from the problem of insufficient/skewed training data in classifier training, and can be readily used on large hierarchies. Theoretical results guarantee the optimality of the obtained solution. Experiments are performed on a large number of functional genomics data sets. The proposed method consistently outperforms the state-of-the-art method on both tree- and DAG-structured hierarchies.

## 1. Introduction

Many real-world applications involve multi-label classification in which an instance can have multiple labels. For example, a document can belong to more than one categories in text categorization (Rousu et al., 2006); a gene may be associated with more than one functions in bioinformatics (Barutcuoglu & Troyanskaya, 2006); and an image may belong to multiple semantic categories in image classification (Zhang & Zhou, 2007). A recent survey on the progress of multi-label classification and its use in different applications can be found in (Silla & Freitas, 2010; Tsoumakas et al., 2010).

In general, multi-label classification algorithms can be divided into two categories: *problem transformation* and *algorithm adaptation* (Tsoumakas et al., 2010). Problem transformation methods transform a multi-label classification problem into one or more single-label classification problems, while algorithm adaptation methods extend a specific learning algorithm for multi-label classification. Examples include boosting, decision trees, ensemble methods, neural networks, support vector machines, genetic algorithms and the nearest-neighbor classifier.

In many applications such as text categorization (Rousu et al., 2006) and functional genomics (Barutcuoglu & Troyanskaya, 2006), the labels are often organized in a tree-structured hierarchy. An instance is associated with a certain label only if it is also associated with the label's parent in the hierarchy. However, most existing multi-label classification algorithms do not take the label structure into consideration. Instead, the labels are simply treated separately, leading to the need to train a large number of classifiers (one for each label). Moreover, as some labels (such as those at the lower levels of the hierarchy) may have very few positive examples, the training data become highly skewed, which can be problematic to many classifiers. Besides, the inconsistent labellings between child and parent causes difficulty in interpretation. Finally, the prediction performance is impaired as structural dependencies among labels are not utilized in the learning process.

Recently, progress has been made on multi-label classification on tree-structured hierarchies. A simple remedy is to allow a classifier for a particular node to predict positive only if the classifier of its parent also predicts positive (Barutcuoglu & Troyanskaya, 2006). A better approach is to construct a training set for each node such that it only consists of samples belonging to its parent (Cesa-Bianchi et al., 2006). Alternatively, large margin methods for structured out-

Appearing in *Proceedings of the 28$^{th}$ International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

put prediction can also be used on tree-structured hierarchies (Rousu et al., 2006). A recent, state-of-the-art approach is based on adapting the decision trees (Blockeel et al., 2006).

Besides trees, more general label hierarchies can be modeled as a directed acyclic graph (DAG), in which a node can have multiple parents. An important example of a DAG-structured hierarchy is the Gene Ontology (GO), which contains valuable gene information of functional class taxonomies. However, as surveyed in (Silla & Freitas, 2010), most current research efforts are only concerned with the easier tree-structured hierarchies. The very few exceptions include (Barutcuoglu & Troyanskaya, 2006), which first starts with independently trained classifiers for each node, and then uses a Bayesian network to enforce consistency with the hierarchy constraints. However, this still needs a large number of independently trained classifiers. Moreover, training of the Bayesian network is computationally expensive. Also, it requires a fairly large number of training examples, which would eliminate the prediction of most biologically specific GO terms (Jin et al., 2008). The state-of-the-art algorithm for multi-label classification on DAG-structured hierarchies is CLUS-HMC (Vens et al., 2008), which is an extension of (Blockeel et al., 2006). Again, it is an adaptation of the decision trees, and so belongs to the category of algorithm adaptation methods. Hence, it cannot benefit from advances in the developments of other classification algorithms.

In this paper, we propose a problem transformation approach for tree- and DAG-structured hierarchies. This is thus more flexible and can be used with any learner. It first uses the kernel dependency estimation (KDE) approach (Weston et al., 2003) to reduce the possibly large number of labels to a manageable number of single-label learning problems. The core issue is then on how to preserve the hierarchy information among the labels. Inspired by a signal processing algorithm called CSSA (Condensing Sort and Select Algorithm) (Baraniuk & Jones, 1994), which aims to find an optimal approximation subtree in a tree, we propose our generalized CSSA that finds an optimal subgraph in a DAG. This is then used to construct a multi-label that is consistent with respect to the tree- or DAG-structured hierarchy.

The rest of this paper is organized as follows. Section 2 reviews the use of KDE in multi-label classification. Section 3 then describes the proposed algorithm. Experiments are presented in Section 4, and the last section gives some concluding remarks. For the lack of space, proofs will be deferred to the full version.

## 2. Kernel Dependency Estimation

In a multi-label classification problem, we are given the training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, where $\mathbf{x}_i$ is in some input space $\mathcal{X}$, $\mathbf{y}_i \in \{0,1\}^d$ is the output vector, and $d$ is the number of labels. Note that each $\mathbf{y}_i$ can have more than one nonzero entries. In many real-world applications, $d$ can easily be in the thousands. For example, in automatic keyword tagging for content retrieval, $d$ is the number of possible tags. Hence, it is computationally inefficient to train one classifier for each label.

This problem can be alleviated by first reducing the large number of classifiers to a manageable amount using the kernel dependency estimation (KDE) framework (Weston et al., 2003). In general, KDE can be used to learn the dependencies between two classes of objects. When using it for multi-label classification, the KDE algorithm consists of three steps:

1. Projection: Each label vector $\mathbf{y}_i$ is projected to a low-dimensional vector $\mathbf{z}_i \in \mathbb{R}^m$.
2. Learning: Using $\{(\mathbf{x}_i, \mathbf{z}_i)\}_{i=1}^n$, learn a mapping from $\mathcal{X}$ to each projected dimension $j = 1, \ldots, m$. Since the projected directions are orthogonal, these $m$ models can be trained independently.
3. Prediction: For a test sample $\mathbf{x}$, first obtain the prediction $\hat{\mathbf{z}} = [\hat{z}_1, \ldots, \hat{z}_m]^T$ from the $m$ learned models. This is then mapped back (decoded) to $\hat{\mathbf{y}} \in \mathbb{R}^d$ in the original label space and followed by further rounding to $\{0,1\}^d$.

The projection step can be implemented by various standard techniques. For example, Hsu *et al.* (2009) used compressed sensing (CS), while Tai & Lin (2010) used principal components analysis (PCA). In this paper, we will use PCA, which has been shown to outperform CS in both accuracy and efficiency (Tai & Lin, 2010). Specifically, PCA is performed on the label vectors $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$. The $m$ leading eigenvectors are used to form the projection matrix $\mathbf{P} \in \mathbb{R}^{m \times d}$, and the projected label vector is obtained as $\mathbf{z} = \mathbf{P}\mathbf{y}$. On prediction, $\hat{\mathbf{y}}$ can be simply obtained as $\mathbf{P}^T\hat{\mathbf{z}}$ (Tai & Lin, 2010). Note that this approach to multi-label classification is a problem transformation method. It is thus more flexible than algorithm adaptation methods and any regressor can be used in the learning step.

Another advantage of KDE is that all $m$ learners in the projected space can learn from the whole training set. In contrast, many hierarchical multi-label classification algorithms have to keep splitting the training data along the path from the root to the leaf. Thus, labels that are near the bottom of the hierarchy may not have sufficient data for training.

However, a major limitation of the current KDE-based methods is that the label dependencies are not considered during prediction. Hence, the obtained labellings between parent and child are inconsistent with the hierarchy structure.

## 3. Proposed Method

In general, the label hierarchy can be modeled as a DAG, in which each node corresponds to a label and the directed edges represent label dependencies. For the simpler case where the DAG is a tree $\mathcal{T}$, the labels are consistent if they satisfy the following $\mathcal{T}$-property:

**Definition 3.1** ($\mathcal{T}$-property)**.** *If a node is labeled positive, its parent must also be labeled positive.*

In a general DAG, each node may have multiple parents. As pointed out in (Vens et al., 2008), there are two interpretations for label consistency:

**Definition 3.2** (AND-$\mathcal{G}$ property)**.** *If a node is labeled positive, all its parents must also be labeled positive.*

**Definition 3.3** (OR-$\mathcal{G}$ property)**.** *If a node is labeled positive, one of its parents must also be labeled positive.*

On many data sets (such as the Gene Ontology), the AND-$\mathcal{G}$ property is more relevant. The following sections describe how the projection and prediction steps in KDE are to be modified to cater for the label hierarchy information.

### 3.1. Projection with Hierarchy Information

In the projection step, Tai & Lin (2010) perform PCA directly on the label vectors. This implicitly assumes that the labels are unrelated. Because of the flexibility of KDE, we can preserve the dependencies in a label hierarchy by instead performing kernel PCA on an appropriate label kernel. In this paper, we will use the feature map commonly used in structured output prediction (Tsochantaridis et al., 2005). Though originally used on trees, it can also be used for encoding structure information in DAGs. Specifically, let $N$ be the number of nodes in the tree/DAG. The feature vector for node $i$ is $l(i) = [l_1, \ldots, l_N]^T \in \{0,1\}^N$, where $l_k = 1$ if $k$ is an ancestor of $i$ or $k = i$; and 0 otherwise. The label kernel evaluation for two nodes $i$ and $j$ is then $l(i)^T l(j)$.

### 3.2. Prediction on Tree Hierarchies

Suppose that it is known that the test sample $\mathbf{x}$ has $L$ labels. If the labels are unstructured, one can simply pick the $L$ largest entries in $\hat{\mathbf{y}}$ (in step 3 of KDE) as its labels. Equivalently, this can be formulated as the optimization problem:

$$\max_{\psi} \sum_{i=1}^{d} \hat{y}_i \psi_i \text{ s.t. } \sum_{i=1}^{d} \psi_i = L,$$

where $\psi = [\psi_1, \ldots, \psi_d]^T$ and $\psi_i \in \{0,1\}$. When the label hierarchy is a tree $\mathcal{T}$, a natural extension is to constrain $\psi$ such that the $\mathcal{T}$-property in Definition 3.1 is observed. In the sequel, such a $\psi$ is called *$\mathcal{T}$-nonincreasing*[1]. The optimization problem is then

$$\begin{aligned} \max_{\psi} \quad & \sum_{i \in \mathcal{T}} \hat{y}_i \psi_i \qquad (1) \\ \text{s.t.} \quad & \psi_i \in \{0,1\}, \ \ \forall i \in \mathcal{T}, \\ & \sum_{i \in \mathcal{T}} \psi_i = L. \\ & \psi \text{ is } \mathcal{T}\text{-nonincreasing}, \end{aligned}$$

#### 3.2.1. The CSSA Algorithm

We first consider the following optimization problem that originates in signal processing (Baraniuk & Jones, 1994):[2]

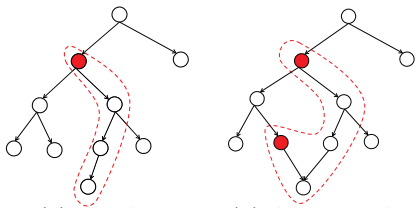
$$\begin{aligned} \max_{\psi} \quad & \sum_{i \in \mathcal{T}} w_i \psi_i, \qquad (2) \\ \text{s.t.} \quad & \psi_i \geq 0, \ \ \forall i \in \mathcal{T}, \\ & \psi_0 = 1, \ \ \sum_{i \in \mathcal{T}} \psi_i \leq L, \\ & \psi \text{ is } \mathcal{T}\text{-nonincreasing}, \qquad (3) \end{aligned}$$

where $w_i \in \mathbb{R}$, and the root of $\mathcal{T}$ is indexed 0. If the constraint (3) is replaced by the bound constraint $0 \leq \psi_i \leq 1$, (2) reduces to the fractional knapsack problem and can be easily solved by greedy algorithm. Interestingly, Baraniuk & Jones (1994) showed that (2) can still be efficiently solved by using greedy algorithm.

Its key idea is to ensure that $\{w_i\}$ is nonincreasing by *condensing* the non-monotonic tree segments to *supernodes*. A supernode $S$ is formed by merging a node (or a supernode) with its parent (Figure 1(a)). It is assigned a supernode value (SNV) which is the average of the $w_i$ values over all its constituent nodes. The resultant *Condensing Sort and Select Algorithm* (CSSA) (Algorithm 1) is iterative. In each iteration, an unassigned supernode $S^*$ with the largest SNV is selected. If assigning $\psi(S^*)$ to 1 does not violate the $\mathcal{T}$-property (i.e., $\psi(\text{pa}(S^*)) = 1$), the assignment will be made permanent; otherwise, $S^*$ is condensed with its parent. The process is repeated until $\sum_{i \in \mathcal{T}} \psi_i = L$.

[1] It is called *$\mathcal{T}$-nonincreasing* because the $\psi$ values are nonincreasing as one traverses from the root to the leaves.

[2] In (Baraniuk & Jones, 1994), it is stated that $w_i \geq 0$. However, this condition is indeed not required in the proof.



(a) CSSA. (b) AND-CSSA.

*Figure 1.* Examples of a supernode. Roots of the supernode are labeled in red.

**Algorithm 1** The CSSA algorithm. Here, pa($i$) is the parent of (super)node $i$, and $n(S)$ is the number of nodes in $S$.

Initialize $\psi_0 \leftarrow 1$; $\Gamma \leftarrow 1$.
Initialize all other nodes as supernodes with $\psi_i \leftarrow 0$ and sort them according to the SNV's.
**while** $\Gamma < L$ **do**
  Pick the unassigned supernode $S^*$ with the largest SNV.
  **if** $\psi(\text{pa}(S^*)) = 1$ **then**
    $\psi(S^*) \leftarrow \min\{1, (L-\Gamma)/n(S^*)\}$.
    $\Gamma \leftarrow \Gamma + n(S^*)$.
  **else**
    Condense $S^*$ and pa($S^*$) as a new supernode.
  **end if**
**end while**

The CSSA algorithm has been successfully used in wavelet approximation (Baraniuk, 1999) and more recently in model-based compressed sensing (Baraniuk et al., 2010). Computationally, it is very efficient. The most expensive step is in the initial sorting of the $N$ SNV's (where $N$ is the number of nodes in $\mathcal{T}$). Hence, its time complexity is only $O(N \log N)$.

#### 3.2.2. CSSA for Tree-Hierarchies

On setting $w_i$ to $y_i$, the two optimization problems (1) and (2) are almost identical, except that the $\psi$ solution in (2) may not be binary. However, it can be easily seen that this $\psi$ solution is *essentially binary* (Baraniuk & Jones, 1994), i.e., all the nodes with nonzero $\psi_i$ values have $\psi_i = 1$, except possibly for those that are assigned in the last iteration. In our multi-label classification context, these nodes (that are assigned in the last iteration) are considered by the algorithm as equally likely to be labeled positive. However, because of the insufficient "quota" left, each of these just receive a fractional amount. In general, the handling of these fractional labels may depend on the application. In our experiments, since we are interested in obtaining the recall-precision curves, we can treat these fractional labels as ones, and allow the total number of labels for this instance to be slightly larger than $L$. The whole algorithm is shown in Algorithm 2.

**Algorithm 2** Multi-label classification on hierarchies.

{(Projection)} Perform KPCA using the label kernel in Section 3.1, and obtain projection matrix $\mathbf{P}$.
**for** $j = 1, \ldots, m$ **do** {(Learning)}
  Train the $j$th regressor using $\{(\mathbf{x}_i, (\mathbf{P}\mathbf{y}_i)_j)\}_{i=1}^N$.
**end for**
{(Prediction)} Obtain $\mathbf{z} \in \mathbb{R}^m$ from the $m$ learned regressors, and decode as $\hat{\mathbf{y}} = \mathbf{P}^T\mathbf{z}$.
Obtain the final labels by feeding $\hat{\mathbf{y}}$ to the CSSA (for tree) or CSSAG (for DAG) algorithm.

### 3.3. Prediction on DAG Hierarchies

In this section, we consider the more challenging situation where the label hierarchy is a DAG. Analogous to (2), the optimization problem is now:

$$\begin{aligned} \max_{\psi} \quad & \sum_{i \in \mathcal{G}} w_i \psi_i, \\ \text{s.t.} \quad & \psi_i \geq 0, \ \ \forall i \in \mathcal{G}, \\ & \psi_0 = 1, \ \ \sum_{i \in \mathcal{G}} \psi_i \leq L, \\ & \psi \text{ is } \mathcal{G}\text{-nonincreasing}, \end{aligned} \tag{4}$$

where $\psi$ is said to be AND-$\mathcal{G}$ (resp. OR-$\mathcal{G}$) nonincreasing if it satisfies the AND-$\mathcal{G}$ (resp. OR-$\mathcal{G}$) property.

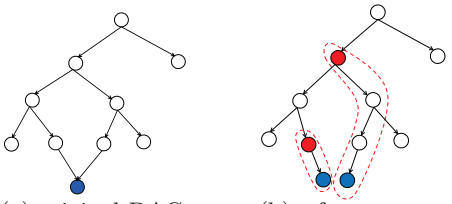
#### 3.3.1. DAG with AND-$\mathcal{G}$ Property

In a DAG, a node may have multiple parents. Subsequently, a supernode may also have multiple roots (Figure 1(b)), and the CSSA does not work. In this section, we extend it to CSSAG (CSSA for Graphs), which is shown in Algorithm 3. Interestingly, the only change required is on the condensation step. When the current labeling is not consistent with the DAG (steps 8 and 9), we merge the supernode $S^*$ with the unassigned parent that has the smallest SNV. Intuitively, when $S^*$ is not consistent, its SNV is large while the SNVs of its parents are smaller. Moreover, the whole path from the root to $S^*$ may also have a low total SNV. By merging $S^*$ with the parent with the smallest SNV, the new supernode will have the smallest possible SNV. Since step 3 always selects the supernode with the largest SNV, existing supernodes that may be more promising than the newly merged supernode will be able to be considered first.

The following theorem justifies its optimality. More-

**Algorithm 3** CSSAG for DAG with AND-$\mathcal{G}$ property.

1: Initialization (same as Algorithm 1).
2: **while** $\Gamma < L$ **do**
3: Pick the unassigned supernode $S^*$ with the largest SNV.
4: **if** $\psi(\tilde{S}) = 1$ for all the parents $\tilde{S}$ of $S^*$ **then**
5: $\psi(S^*) \leftarrow \min\{1, (L - \Gamma)/n(S^*)$.
6: $\Gamma \leftarrow \Gamma + n(S^*)$.
7: **else**
8: Among all the unassigned parents of $S^*$, pick $\tilde{S}$ with the smallest SNV.
9: Condense $S^*$ and $\tilde{S}$ as a new supernode.
10: **end if**
11: **end while**



(a) original DAG. (b) after conversion.

*Figure 2.* Replicating a node in the DAG satisfying the OR-$\mathcal{G}$ property.

over, obviously, its time complexity is still $O(N \log N)$.

**Theorem 1.** *The $\psi$ obtained by Algorithm 3 is an optimal solution of (4) satisfying the AND-$\mathcal{G}$ property.*

#### 3.3.2. DAG with OR-$\mathcal{G}$ Property

In this section, we consider the case where the DAG hierarchy satisfies the OR-$\mathcal{G}$ property. For a node to be labeled positive, only one of its parents needs to be labeled positive. The key idea is to preserve the label consistencies by converting part of the DAG to a tree. Specifically, on condensing a supernode $S^*$ with $n_{pa}$ parent supernodes (step 12 of Algorithm 4), since all its parents are equally desirable because of the OR-$\mathcal{G}$ property, we replicate $S^*$ $n_{pa}$ times and merge each replicate with one of its parents (Figure 2). Subsequently, the $\mathcal{G}$-nonincreasing property degenerates to the $\mathcal{T}$-nonincreasing property, and the original CSSA (for trees) is almost ready to be used. However, note that when a supernode $S^*$ is selected and assigned $\psi$ value (step 4 of Algorithm 4), replicates of nodes in $S^*$ may still be present in other supernodes $\{S'\}$. Since nodes in $S^*$ should never be considered again after the $\psi$ assignment, supernodes containing these replicates have to be removed, while the unassigned nodes in these supernodes are created as new supernodes for future consideration (step 9). The complete algorithm is shown in Algorithm 4. Again, its time complexity is still $O(N \log N)$.

**Algorithm 4** CSSAG for DAG with OR-$\mathcal{G}$ property.

1: Initialization (same as Algorithm 1).
2: **while** $\Gamma < L$ **do**
3: Pick the unassigned supernode $S^*$ with the largest SNV.
4: **if** $\psi(\tilde{S}) = 1$ for one of the parents $\tilde{S}$ of $S^*$ **then**
5: $\psi(S^*) \leftarrow \min\{1, (L - \Gamma)/n(S^*)$.
6: $\Gamma \leftarrow \Gamma + n(S^*)$.
7: Find all the unassigned supernodes $\{S'\}$ such that $S' \cap S^* \neq \emptyset$.
8: **for** each $S'$ in that list **do**
9: Delete $S'$, and form a new supernode for each node in $S' \setminus S^*$.
10: **end for**
11: **else**
12: **for** each unassigned parent $\tilde{S}$ of $S^*$ **do**
13: Condense $S^*$ and $\tilde{S}$ as a new supernode.
14: **end for**
15: **end if**
16: **end while**

**Theorem 2.** *The $\psi$ obtained by Algorithm 4 is an optimal solution of (4) satisfying the OR-$\mathcal{G}$ property.*

## 4. Experiments

### 4.1. Setup

In this section, we perform experiments on a number of functional genomics data sets, using the same setup as in (Vens et al., 2008). We use 12 yeast data sets from (Clare, 2003), with each data set describing a different aspect of the genes in the yeast genome. Each data set has two versions of output labels. The first version contains tree-structured labels, and are annotated from MIPS's FunCat[3]. The resultant tree hierarchy has a depth of 5, and each example has an average of 8.8 labels. The second version has DAG-structured labels, and are annotated from the Gene Ontology (GO)[4]. The resultant DAG has a depth of 14, and each example has 35 labels on average.

As in (Clare, 2003; Vens et al., 2008), two-thirds of each data set are used for training and the remaining one third is for testing. Out of the training set, two-thirds are used for the actual training and one third is for validation. As some of the data sets contain missing features, we impute the missing values with

[3] http://mips.gsf.de/projects/funcat
[4] http://www.geneontology.org

*Table 1.* The yeast data sets used in the experiments.

| data set | #training samples | #validation samples | #test samples | #attributes | #classes FunCat | #classes GO |
|---|---|---|---|---|---|---|
| seq | 1692 | 876 | 1332 | 478 | 500 | 4134 |
| pheno | 653 | 352 | 581 | 69 | 456 | 3128 |
| struc | 1659 | 859 | 1306 | 19629 | 500 | 4133 |
| hom | 1661 | 876 | 1309 | 47035 | 500 | 4127 |
| cellcycle | 1625 | 848 | 1278 | 77 | 500 | 4126 |
| church | 1627 | 844 | 1278 | 27 | 500 | 4126 |
| derisi | 1605 | 842 | 1272 | 63 | 500 | 4120 |
| eisen | 1055 | 528 | 835 | 79 | 462 | 3758 |
| gasch1 | 1631 | 846 | 1281 | 173 | 500 | 4126 |
| gasch2 | 1635 | 849 | 1288 | 52 | 500 | 4132 |
| spo | 1597 | 837 | 1263 | 80 | 500 | 4120 |
| expr | 1636 | 849 | 1288 | 551 | 500 | 4132 |

the mean of the non-missing values for that feature. A summary of the data sets is shown in Table 1.

In the following, we will focus on comparing methods that can produce predictions consistent with the given label hierarchy:

1. The proposed method: We project the labels to a 50-dimensional space, and use ridge regression (with its ridge parameter selected by the validation data) in the learning step.

2. CLUS-HMC (Vens et al., 2008)[5]: Extensive experiments in (Blockeel et al., 2006; Vens et al., 2008) has shown that CLUS-HMC is state-of-the-art. It has outperformed methods such as CLUS-SC, which learns a separate tree for each label, and CLUS-HSC, which learns trees hierarchically.

3. Hierarchical SVM (H-SVM) (Cesa-Bianchi et al., 2006) on the tree-structured hierarchies (it cannot be used on DAG-structured hierarchies): This is a hierarchical version of the SVM, in which the SVM in each node $i$ is trained only on those examples that the parent of $i$ also predicts one.

### 4.2. Performance Measures

For binary classification problems, precision and recall are defined as

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

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. In a multi-label classification problem, let $\text{TP}_i/\text{FP}_i/\text{FN}_i$ be the number of true positives / false positives / false negatives for label $i$. The precision and recall are then defined as (Vens et al., 2008):

$$\text{Prec} = \frac{\sum_i \text{TP}_i}{\sum_i \text{TP}_i + \sum_i \text{FP}_i}, \text{Rec} = \frac{\sum_i \text{TP}_i}{\sum_i \text{TP}_i + \sum_i \text{FN}_i}.$$

Typically, multi-label methods rely on a threshold to decide how many labels are to be predicted for each sample. However, the setting of this threshold depends heavily on the application. As in (Vens et al., 2008), we avoid this controversy by evaluating the methods using the precision-recall (PR) curve, which is obtained by varying the threshold of the multi-label learner over the whole range[6]. For easy comparison among the different models, this whole PR curve can also be summarized by a single performance score called *Area Under the PR Curve* (AUPRC). The larger the AUPRC, the better the model.

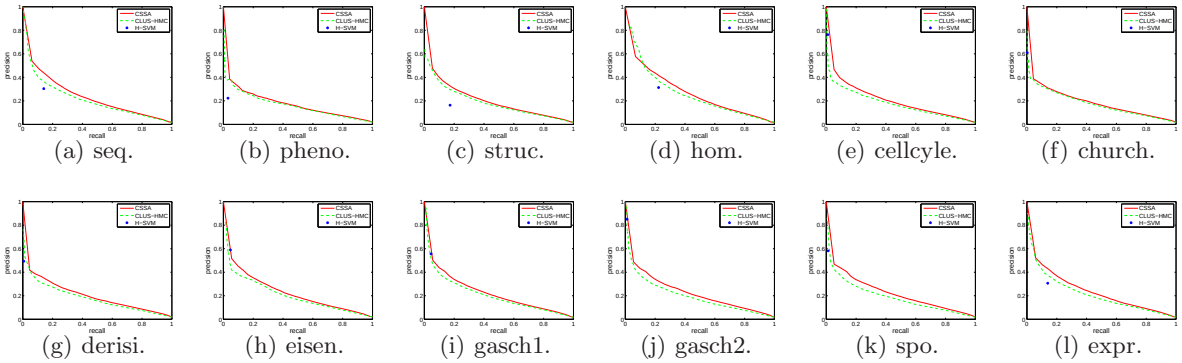
### 4.3. Results on Tree-Structured Labels

Figure 3 shows the PR curves for the various methods. Overall, our method is better than HMC-CLUS. The only exception is at the low-recall regime on the hom data set, where our precision is slightly lower. Moreover, in terms of the AUPRC values, CSSA consistently yields higher AUPRC than CLUS-HMC on all the data sets (Table 2).

The precision and recall values for the H-SVM do not change much with different settings of the regularization parameter, and so we can only show one point for it in each PR plot. Moreover, its recall value is often smaller than 0.2. In many applications of hierarchical multi-label classification, the number of negative

[5] http://www.cs.kuleuven.be/~dtai/clus

[6] For example, for the proposed method, we vary its $L$ parameter from 1 to $N$.




(a) seq. (b) pheno. (c) struc. (d) hom. (e) cellcyle. (f) church.

(g) derisi. (h) eisen. (i) gasch1. (j) gasch2. (k) spo. (l) expr.

*Figure 3.* PR curves for the FunCat data sets.

*Table 2.* AUPRC values for the FunCat data sets.

| data set | CSSA | CLUS-HMC |
|---|---|---|
| seq | **0.226** | 0.218 |
| pheno | **0.167** | 0.166 |
| struc | **0.194** | 0.189 |
| hom | **0.257** | 0.254 |
| cellcycle | **0.196** | 0.180 |
| church | **0.179** | 0.178 |
| derisi | **0.194** | 0.183 |
| eisen | **0.220** | 0.212 |
| gasch1 | **0.216** | 0.212 |
| gasch2 | **0.218** | 0.203 |
| spo | **0.216** | 0.195 |
| expr | **0.228** | 0.218 |

*Table 3.* AUPRC values for the GO data sets.

| data set | CSSAG | CLUS-HMC |
|---|---|---|
| seq | **0.478** | 0.469 |
| pheno | **0.426** | 0.425 |
| struc | **0.455** | 0.446 |
| hom | **0.493** | 0.481 |
| cellcycle | **0.454** | 0.443 |
| church | **0.442** | 0.436 |
| derisi | **0.442** | 0.440 |
| eisen | **0.479** | 0.454 |
| gasch1 | **0.468** | 0.453 |
| gasch2 | **0.454** | 0.449 |
| spo | **0.442** | 0.440 |
| expr | **0.473** | 0.453 |

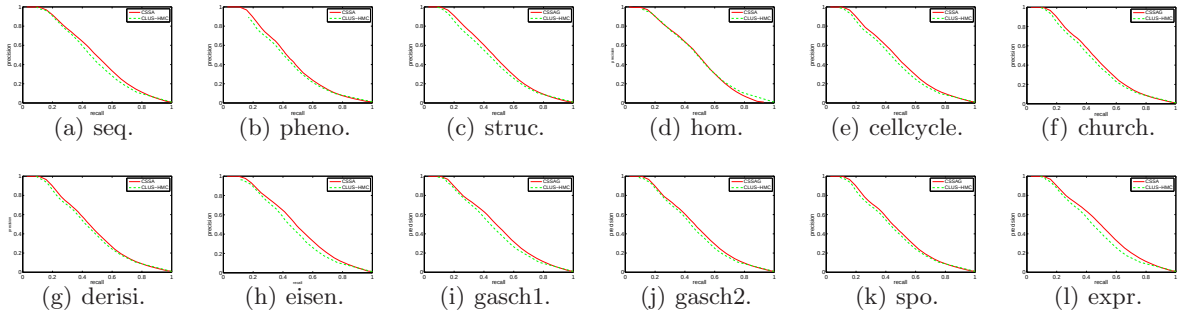
instances for each label far exceeds that of positive instances. Moreover, typically, we are more interested in testing instances with true positive labels. Hence, the small recall value of H-SVM indicates that it may not be of much interest in practical applications.

### 4.4. Results on DAG-Structured Labels

In this section, we report results on the experiments with DAG-structured labels annotated from GO. Figure 4 compares the PR curves of the proposed method with CLUS-HMC. As can be seen, except at very high recalls, the PR curve of the proposed method is consistently better than that of CLUS-HMC on every data set. The AUPRC values shown in Table 3 also confirm the superiority of our method.

## 5. Conclusion

In this paper, we proposed a novel hierarchical multi-label classification algorithm which is applicable on both tree- and DAG-structured hierarchies. There are two main novelties. First, we adapt and integrate an efficient greedy algorithm (CSSA), that is originally used for subtree optimization in signal processing, into the kernel dependency estimation framework for multi-label classification on tree-structured hierarchies. Second, we further extend CSSA to the CSSAG algorithm, which can then be used for DAG-structured hierarchies satisfying either the AND-$\mathcal{G}$ or the OR-$\mathcal{G}$ property. The proposed algorithm is computationally efficient, easy to implement, does not suffer from the problem of insufficient/skewed training data in classifier training, and can be readily used on large hierarchies with thousands of labels (or more). Moreover, it belongs to the problem transformation approach and so can be used with any regressor in the underlying learning process. Theoretical analysis shows that both versions of the CSSAG algorithm generate optimal solutions consistent with the given DAG. Experimental results on a large number of functional genomics data sets show that the proposed method consistently outperforms the state-of-the-art method of CLUS-HMC on



(a) seq. (b) pheno. (c) struc. (d) hom. (e) cellcycle. (f) church.

(g) derisi. (h) eisen. (i) gasch1. (j) gasch2. (k) spo. (l) expr.

*Figure 4.* PR curves for the GO data sets.

both tree- and DAG-structured hierarchies.

## Acknowledgments

This research was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region (Grant 615209).

## References

Baraniuk, R. Optimal tree approximation with wavelets. In *Wavelet Applications in Signal and Image Processing VII*, pp. 196–207, 1999.

Baraniuk, R.G., Cevher, V., Duarte, M.F., and Hegde, C. Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001, 2010.

Baraniuk, Richard G. and Jones, Douglas L. A signal-dependent time-frequency representation: Fast algorithm for optimal kernel design. *IEEE Transactions on Signal Processing*, 42(1):134–146, January 1994.

Barutcuoglu, Z. and Troyanskaya, O.G. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7), 2006.

Blockeel, H., Schietgat, L., Struyf, J., Dzeroski, S., and Clare, A. Decision trees for hierarchical multilabel classification: A case study in functional genomics. In *PKDD*, Berlin, Germany, 2006.

Cesa-Bianchi, N., Gentile, C., and Zaniboni, L. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7:31–54, 2006.

Clare, A. *Machine Learning and Data Mining for Yeast Functional Genomics*. PhD thesis, University of Wales, 2003.

Hsu, D., Kakade, S.M., Langford, J., and Zhang, T. Multi-label prediction via compressed sensing. In *NIPS 22*, pp. 772–780. MIT Press, 2009.

Jin, B., Muller, B., Zhai, C., and Lu, X. Multi-label literature classification based on the gene ontology graph. *BMC Bioinformatics*, 9, 2008.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7:1601–1626, 2006.

Silla, C.N. and Freitas, A.A. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2010.

Tai, F. and Lin, H.T. Multi-label classification with principle label space transformation. In *Proceedings of the 2nd International Workshop on Learning from Multi-Label Data*, Haifa, Israel, 2010.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, December 2005.

Tsoumakas, G., Katakis, I., and Vlahavas, I. Mining multi-label data. In Maimon, O. and Rokach, L. (eds.), *Data Mining and Knowledge Discovery Handbook*. Springer, 2nd edition, 2010.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B., and Vapnik, V. Kernel dependency estimation. In *NIPS 15*, 2003.

Zhang, M.-L. and Zhou, Z.-H. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7), 2007.