Hierarchical Boosting for Gene Function Prediction

Noor Alaydie, Chandan K. Reddy, Farshad Fotouhi
Department of Computer Science, Wayne State University,
Detroit, MI 48202, USA
Email: {alaydie,fotouhi}@wayne.edu, reddy@cs.wayne.edu

Functional classification of genes using diverse bio-molecular data obtained from high-throughput technologies is a fundamental problem in bioinformatics and functional genomics. Genes are organized and classified according to a hierarchical classification scheme and each gene will participate in multiple activities. Flat classifiers, that work on non-hierarchical classification problems independently, do not take into account the hierarchical structure of the functional class taxonomies. Therefore, they are not able to utilize the information inherent in the class hierarchy. Moreover, independent classifiers, where each classifier predicts the gene membership to a particular class, may lead to an inconsistent set of predictions for a hierarchically structured classification scheme. In this paper, we propose HML-Boosting algorithm for the problem of hierarchical multi-label classification in the context of gene function prediction. HML-Boosting exploits the hierarchical dependencies among the classes. Extensive experiments on four bio-molecular datasets using two approaches for class-membership inconsistency correction during the testing phase, the top-down approach and the bottom-up approach, show that HML-Boosting algorithm outperforms flat classifiers using different evaluation metrics. In addition, we carry out a detailed comparison of the two approaches for class-membership inconsistency correction during the testing phase.

1. INTRODUCTION

The functional classification of genes is an important and challenging problem in the field of functional genomics. First, there are many functional classes which may be related to each other in a tree or a graph structure (Funcat or Gene Ontology(GO)). Usually, the goal is to discover the specific unknown functions for a gene rather than general functions. Second, a gene may have multiple class labels. Biologically, a gene may be involved in more than one biological activity. Therefore, there is a need for a prediction algorithm that is able to identify all the possible functions of a particular gene.

In the context of gene function prediction, with the availability of data from different biological sources, assigning biological functions to genes is a challenging task in functional genomics. This is often achieved by automated prediction process that interacts with laboratory experiments [2]. Several approaches to apply machine learning techniques to predict gene functions from a predefined set of functions have been proposed in the literature [2–4]. Predictions with the highest confidence are taken to the lab for testing [5]. In gene function prediction setting, a gene may be associated with multiple biological functions.

Hierarchical multi-label classification problem is an extension of the binary classification problem where an instance can be associated with multiple classes that are related through a hierarchical categorization scheme. In other words, when an instance is labeled with a certain class, it should also be labeled with all of its superclasses. Multi-label classification problems arise in several application domains such as text classification, image classification, and gene function prediction [1]. In hierarchical multi-label classification, the training set consists of instances, each of which is associated with a set of labels that are organized according to a predefined hierarchy (For example, GO or FunCat in the functional genomics setting). The goal is to predict the label sets of unseen instances by analyzing the training instances with known label sets.

Several types of data sources can be used for the prediction task. The recent advances in high-throughput technologies, such as the DNA microarray data, has enabled measuring the expression level of thousands of genes simultaneously. Protein-protein interaction (PPI) is another important data source. In this paper, various kinds of datasets will be used in evaluating the task of hierarchical gene function prediction.

1.1. Hierarchical Taxonomies of Gene Functions

Gene functional classes such as MIPS's (The Munich Information Center for Protein Sequences) FunCat (Functional Catalogue tree structure), are controlled vocabularies which are structured hierarchically as a rooted tree with general functions appear at the top levels of the hierarchy and more specific functions appear at the lower levels. As an example, Figure 1 [6] shows a small portion of the FunCat taxonomy. In this Figure, *Metabolism, Energy and Cell Cycle and DNA Processing* are considered as general functional classes, while *conjugational DNA transfer* is a more specific class.

```
01 METABOLISM
01.01 amino acid metabolism
...
02 ENERGY
02.01 glycolysis and gluconeogenesis
02.01.01 glycolysis methylglyoxal bypass
...
10 CELL CYCLE AND DNA PROCESSING
10.01 DNA processing
10.01.01 cellular DNA uptake
10.01.01.01 bacterial competence
10.01.01.03 conjugational DNA transfer
...
```

Fig. 1. A small sample of FunCat, the hierarchical classification scheme.

Gene Ontology (GO) [7] is another hierarchical gene classification scheme, where the gene function categories are organized according to a directed acyclic graph (DAG) [8]. In the context of FunCat, where the existence of a hierarchical structure of classes is expressed as a tree, each class is subdivided into more specific classes, and these, in turn, are subdivided again and again until the most specific functions are reached [9]. According to the "true path rule" [8, 10] that governs the annotation of both GO and FunCat taxonomies, annotating a gene to a given class is automatically transferred to all of its ancestors to maintain the hierarchy constraint. In this paper, we focus on FunCat as the hierarchical scheme used for gene function prediction.

Table 1. An example of a synthetically generated dataset. Each example belongs to one or more of the hierarchically structured classes.

Examples	Classes							
	A	B	C	D	E	F	G	H
e1	1	1	1	0	0	1	0	1
e2	1	0	1	0	0	0	0	0
e3	1	1	0	1	1	0	0	0
e4	1	1	1	1	0	0	0	0
e5	1	0	1	0	0	0	0	0
e6	1	1	1	0	1	0	0	0
e7	1	1	0	0	0	0	0	0
e8	1	0	0	1	0	0	0	0
e9	1	1	0	1	1	1	1	0
e10	1	0	1	1	0	0	0	0

1.2. Hierarchical Classification vs. Flat Classification

Table 1 shows a set of classes in the columns and a set of examples in the rows. An example is considered to be associated with a certain class if the corresponding value is 1. In this example, we assume that the classes are related in a hierarchical manner as shown in Figure 2(a). In this case, class A is considered to be the most general class which all the examples belong to. Going down the tree, more specific functions are reached. Hence, less number of examples are labeled with those specific functions. In other words, for a non-root node in the tree, the set of examples that belongs to this node is a subset of the set of examples that belongs to the parent of that node. Hierarchical classification integrates the hierarchical information for the purpose of labeling unseen examples. Furthermore, the hierarchical information helps in training and testing the classifiers by identifying the relevant positive and negative examples

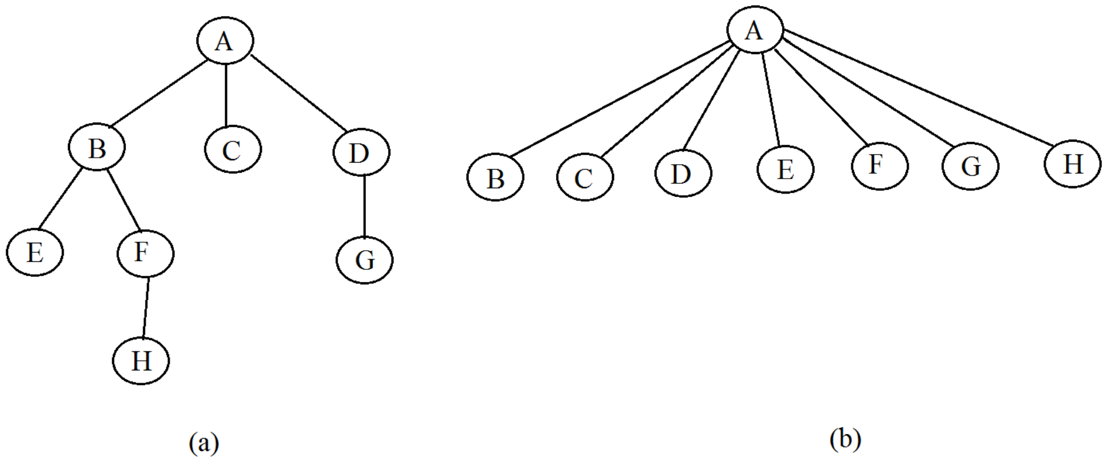


Fig. 2. (a) A tree structure representation for the classes in Table 1, (b) A flat representation for the same set of classes.

for each class, according to its location in the hierarchy, which will be discussed later in the next section. However, the flat classification does not capture such relationship between classes.

As shown in Figure 2(b), flat classifiers work independently of each other. Although class A is shown to be the root class for all of the other classes, it is considered as "*any class*". Since the flat classifiers do not capture the dependencies among the classes, all the classifiers are considered to be at the same level. The set of positive (and negative) examples used to train each classifier does not follow any constraint. As a result, an example may be classified to belong to a given class while it is not classified as belonging to the parent of that class. Such violation should be avoided when the domain knowledge about the examples is available. One of the primary goals of this work is to compare the performance of hierarchical classification against flat classification.

In this paper, the hierarchical structure of gene functions in FunCat taxonomy is integrated into the multi-label classification problem. As a result, the performance of the proposed method that incorporates both the hierarchical and multi-label concepts is significantly better compared to the standard classification approach. In this work, we develop the HML-Boosting algorithm and apply it to bio-molecular data for predicting gene functions. Boosting algorithms are well-known machine learning algorithms with rigorous theoretical properties and are usually competitive to any other classification methods [11]. HML-Boosting algorithm relies on the hierarchical information and utilizes the hierarchy to improve the prediction accuracy. To test the HML-Boosting framework in the functional genomics context, each classifier is evaluated in a leave-one-out scheme in which FunCat annotations of the test genes are hidden. We describe the evaluation metrics that we used and show extensive experiments using HML-Boosting algorithm on four bio-molecular datasets using two approaches for class-membership inconsistency correction during the testing phase, the top-down approach and the bottom-up approach. Furthermore, HML-Boosting is compared with flat classifiers and the results are analyzed. *Our results demonstrate that utilizing the hierarchical scheme in the prediction process yields improved performance over flat classifiers, when both are applied on the same datasets.*

The rest of the paper is organized as follows: In section 2, the formal definition of hierarchical multi-label classification is given and related work in this area is discussed. In section 3, the proposed HML-Boosting is described. In section 4, extensive experimental evaluation on four bio-molecular datasets is reported. Finally, section 5 concludes the paper with future directions.

2. Related Work

Several methods have been proposed to handle the hierarchical multi-label classification task for gene/protein function prediction application

[2–5, 8, 10, 12]. The majority of hierarchical multi-label classification works have been motivated by the text classification problem. Many proposed methods in that domain use either Bayesian or kernel-based classifiers [3].

Different approaches have been proposed to tackle the hierarchical multi-label classification problem. Generally, these approaches can be grouped into two groups: the local classifier approach and the global classifier approach. Methods that adopted the local classifier approach use local information to train the classifiers. Based on whether the hierarchy constraint is taken into account or not, this approach can be further sub-grouped into two main approaches. In one approach, a separate binary classier is generated for each class in the hierarchy. The learning for each classifier is performed on a strongly skewed class distribution [4]. This is because few instances belong to classes at lower levels of the hierarchy, while more instances are positive examples of classes at higher levels of the hierarchy as shown in Table 1. Moreover, this approach does not take the hierarchy constraint into consideration and hence, can produce a *hierarchically inconsistent set of predictions*. In other words, a class may predict a test instance to be positive while its parent class (or any of its ancestor classes) predicts it as negative.

To overcome the hierarchy constraint problem of the first approach, the second approach mainly focuses on taking the class hierarchy constraint into account. More specifically, the separate class-wise models are hierarchically combined in the prediction stage, so that a classifier generated for a class c, will predict positive only if the classifier for the parent class of c is also predicted to be positive [4, 2]. In essence, there are three main approaches for using the local information to build the classifiers, a local classifier per every node, a local classifier per parent node and a local classifier per level [13]. For example, in the work of Barutcuoglu et al. [2], a Bayesian framework is developed for correcting class-membership inconsistency for the separate class-wise models approach. Their method starts by training independent Support Vector Machine (SVM) classifiers for each class with no regard to the hierarchy. Bayesian combination of the output of the base binary classifiers is applied next. Thus, the hierarchical structure of the classes is captured through the use of a Bayesian network.

On the other hand, in the global classifier approach, a single global model is developed from the training set that takes the class hierarchy as a whole into account during a single run of the algorithm. The single global classifier is able to predict all the classes of an example at once. CLUS-HMC algorithm, proposed by Ven et al. [4], is an example of the global classifier approach. Their algorithm is based on the predictive clustering trees, in which a single tree is trained to make prediction for all classes at once. In order to do that, the classification output is transformed into a vector with boolean components corresponding to the possible classes. Weighted Euclidean distance is used to calculate the degree of similarity between the training examples in the classification tree.

For the testing phase, there are two main strategies for class predictions, namely, the top-down approach and the bottom-up approach. Most of the existing methods use a top-down class prediction strategy in the testing phase [2, 3, 13]. Valentini [16] followed a bottom-up approach in correcting the class-membership inconsistency in the testing phase. Basically, after evaluating all the classifier nodes' outputs, a "consensus" probability is computed in a bottom-up fashion to obtain consistent predictions.

Our proposed approach falls into the local classifier approach. More specifically, for each parent node in the class hierarchy a multi-class multi-label classifier is built using AdaBoost.MH. Moreover, we evaluated the performance of the HML-Boosting algorithm using the two different policies for the testing phase, the top-down class prediction approach and the bottom-up class prediction approach.

The hierarchical multi-label classification problem can be defined as follows [4]: Let $\mathcal{G}$ be the example space, and let $\mathcal{C}$ be the set of classes. The hierarchical relationships among classes in $\mathcal{C}$ are defined as follows:
Given $c_1, c_2 \in \mathcal{C}, c_1$ is the ancestor of c_2, denoted by $(\uparrow c_2) = c_1$, if and only if c_1 is a superclass of c_2. Let $\mathcal{T} = < g_1, \mathcal{S}_1 >, ..., < g_n, \mathcal{S}_n >$ where $g_i \in \mathcal{G}$ and $\mathcal{S}_i \subseteq \mathcal{C}$ such that $c_i \in \mathcal{S}_i \Rightarrow c_i' \in S_i, \forall(\uparrow c_i) = c_i'$.
Having q as the quality criterion for evaluating the model based on the prediction accuracy, the objec-

tive function is defined as follows:
A function $\mathfrak{f} : \mathcal{T} \to 2^c$. Here, 2^c is the power set of $\mathcal{C}$ such that q is maximized, and $c \in \mathfrak{f}(g) \Rightarrow c' \in \mathfrak{f}, \forall(\uparrow c) = c'$. In this paper, the function $\mathfrak{f}$ is learned using the proposed HML-Boosting algorithm.

3. HML-Boosting Algorithm

HML-Boosting is a hierarchical multi-label gene functional classification algorithm that is inspired by TREEBOOST.MH [11], a multi-label hierarchical text classification algorithm. HML-Boosting uses ADABOOST.MH as its base step and recurs over the class tree structure. HML-Boosting exploits the hierarchical taxonomy of the classes to improve prediction performance.

3.1. ADABOOST.MH

ADABOOST.MH [14] is a popular multi-class variant of the Adaboost algorithm and works well in practice [15]. It handles multi-class and multi-label problems. ADABOOST.MH (see Algorithm 3.1) works by transferring a multi-class problem into a binary classification problem by replicating positive instances for the given class labels [15]. More details of the ADABOOST.MH algorithm are given in [14]. The input to the algorithm is a training set $\mathcal{T} = <g_1, \mathcal{S}_1>, ..., <g_n, \mathcal{S}_n>$, where $\mathcal{S}_i \subseteq \mathcal{C}$ is the set of classes to which gene/gene-product g_i belongs to.

ADABOOST.MH maintains a distribution D, that is updated in each iteration. The initial distribution D_1 is uniform. At each iteration, s, a weak learner is built to form a weak hypothesis $\hat{\phi}_s$. The weak hypothesis is generally a decision stump. Next, all the weights $D_s(g_i, c_j)$ are updated to $D_{s+1}(g_i, c_j)$ using the following rule:

$$D_{s+1}(g_i, c_j) = \frac{D_s(g_i, c_j)\exp(-\phi(g_i, c_j)\hat{\phi}_s(g_i, c_j))}{Z_s} \quad (1)$$

Here, the target function $\phi(g_i, c_j)$ is defined as follows:

$$\phi(g_i, c_j) = \begin{cases} 1, & g_i \in c_j \\ -1, & \text{Otherwise} \end{cases}$$

and

$$Z_s = \sum_{i=1}^{n}\sum_{j=1}^{m} D_s(g_i, c_j)\exp(-\phi(g_i, c_j)\hat{\phi}_s(g_i, c_j)) \quad (2)$$

is a normalization factor chosen to make $\sum_{i=1}^{n}\sum_{j=1}^{m} D_{s+1}(g_i, c_j) = 1$.

The sign of the weak hypothesis is used to decide upon the prediction made by the weak learner, while the absolute value can be interpreted as the strength of the belief. In other words, if $\phi_s(g_i, c_j) > 0$, then g_i is predicted to belong to c_j, while if $\phi_s(g_i, c_j) < 0$, then g_i is predicted as a negative example of c_j. After finishing all the iterations, the final hypothesis is generated by summing up all of the weak hypothesis, $\hat{\phi}(g, c) = \sum_{s=1}^{S} \hat{\phi}_s(g, c)$ for $c \in C = c_1, ..., c_m$.

Algorithm 3.1 *ADABOOST.MH*

```
Input:
A training set $\mathcal{T} = <g_1, \mathcal{S}_1>, ..., <g_n, \mathcal{S}_n>$ where $\mathcal{S}_i \subseteq \mathcal{C} = \{c_1, ..., c_m\}$ for all $i = 1, ..., n$.
Output: A final hypothesis $\hat{\phi}(g, c) = \sum_{s=1}^{S} \hat{\phi}_s(g, c)$ for $c \in C = c_1, ..., c_m$.
Algorithm:
Set $D_1(g_i, c_j) = \frac{1}{mn}$ for all $i = 1, ..., n$ and for all $j = 1, ..., m$
for $s = 1, ..., S$ do
  Pass distribution $D_s(g_i, c_j)$ to the weak learner
  Get the weak hypothesis $\hat{\phi}_s$ from the weak learner
  Set $D_{s+1}(g_i, c_j) = \frac{D_s(g_i, c_j)\exp(-\phi(g_i, c_j)\hat{\phi}_s(g_i, c_j))}{Z_s}$
  Where $Z_s = \sum_{i=1}^{n}\sum_{j=1}^{m} D_s(g_i, c_j)\exp(-\phi(g_i, c_j)\hat{\phi}_s(g_i, c_j))$
end for
```

3.2. HML-Boosting

We focus our discussion on FunCat taxonomy. According to the True Path Rule (TPR), that governs both FunCat and GO taxonomies annotations, the TPR indicates a two-way asymmetric flow of information. A gene g that is annotated with a particular functional class, c_j, is also annotated with the parent class and with all of the ancestor classes of c_j in a recursive way [16]. In this scenario, gene g is called as "bubbled-up" positive example in the sense that it has been bubbled up to c_j from somewhere down below [11]. On the other hand, if a gene is not annotated with a class c_j, that gene cannot be annotated with any of the descendant classes of c_j. In that case, gene g is called an "own" positive example of c_j [11].

At each classifier, we need to carefully choose the set of positive and negative examples. We followed the same approach discussed in Ref. 11, in which the positive training examples of a non-leaf category, c_j, is a superset of the union of sets of positive training examples of all of its descendent (leaf) classes [11].

HML-Boosting algorithm is explained in Algorithm 3.2. Each non-root class, c_j, has a binary classifier, $\hat{\phi}_j$, that is associated with it. The classifier should act as a "filter" to prevent unsuitable examples from spreading out to the lower levels in the hierarchy. Hence, only the test genes that a classifier $\hat{\phi}_j$ decides as belonging to c_j are passed to all the binary classifiers corresponding to the children classes of c_j. While the genes that classifier $\hat{\phi}_j$ marks as not belonging to c_j are "blocked" and no further analysis is carried out on them.

Algorithm 3.2 *HML – Boosting*

```
**Input:** A pair $< C, L >$ where $C$ is a tree-structured set of classes and $L$ is the total number of classes of $C$.
**Output:** For each non-leaf class $c_t \in C$, a final hypothesis $\hat{\phi}(g,c) = \sum_{s=1}^{S} \hat{\phi}_s(g,c)$ for $c \in children(c_t)$.
**Algorithm:**
for $i = 1, ..., L$ do
  if class $i$ is a leaf class then
    Do nothing
  else
    Let $children(i) = c_{1i}, ..., c_{ki}$ be the $k$ children classes of $i$
    Run ADABOOST.MH on $children(i)$
  end if
end for
```

Relying on the hierarchical topology of the classification scheme, the training of each of the binary classifiers, $\hat{\phi}_j$, is performed locally using *the siblings policy*. During classification, the classifier $\hat{\phi}_j$ at class c_j will only be presented with examples that are positive at the parent class of c_j. Hence, the reached examples at $\hat{\phi}_j$ are positive examples to c_j and/or to the siblings of c_j. In other words, the training for classifier $\hat{\phi}_j$ is performed by feeding as negative training examples, the positive examples at the parent of c_j that are not positive examples at c_j.

It should be noted that the selected negative training examples at $\hat{\phi}_j$ are the most informative negative examples for training [11]. Moreover, since as we go down in the hierarchy, fewer training examples are survived, this will be reflected in the efficiency of the algorithm, for both training and classification time.

HML-Boosting converts the hierarchical multi-label classification problem to multiple flat multi-label classification problems, in which one classifier is built for every internal node in the tree [11]. HML-Boosting iteratively calls ADABOOST.MH to generate a multi-label flat classifier for the children of every internal node. In other words, a binary classifier, $\hat{\phi}$, for each non-root class $c_j \in C$ is generated so that the hierarchical classification can be performed. The algorithm first checks whether the reached class is a leaf node or not. The classifiers are built for internal nodes only and hence, no work needs to be done if a leaf function class has been reached.

If an internal class, c_j, has been reached, we identify the set of children classes of c_j. Next, ADABOOST.MH is called for a multi-label flat classification for the children classes of c_j. Note that the negative examples of a class c_j are the set of positive examples at the parent class but are not positive examples at class c_j. Next, for each class $c_q \in children(c_j)$, where $children(c_j)$ refers to the set of children of class c_j, HML-Boosting is called iteratively on the children of a particular class. HML-Boosting results in a tree of binary classifiers, one for each non-root node, where each one consisting of the combination of decision stumps.

We evaluated the performance of the HML-Boosting algorithm using two different policies for the testing phase, the top-down class prediction approach and the bottom-up class prediction approach. In the top-down class prediction approach, the negative predictions are propagated from top to bottom along the hierarchy. In other words, for an unseen example, the positive predictions of the higher-levels (most generic) classes are propagated from top to bottom nodes along the hierarchy.

On the other hand, in the bottom-up class prediction strategy, the propagation of positive decisions are carried out from bottom to top nodes along the hierarchy. In other words, positive decisions of descendant nodes contribute to the decision of their ancestors. More specifically, the prediction of a node/class is dependent on the local prediction of that node/class and on the prediction of the descendant nodes of that node according to the following [16]:

$$P(x) = \frac{P_{local}(x) + \sum P_{child}(x)}{1 + |children(x)|} \tag{3}$$

where $P_{local}(x)$ is the local prediction at the node x and $\sum P_{child}(x)$ is the summation of the prediction of the descendant nodes of x.

4. Experiments

For our experiments, we used the functional classification of yeast genes at genome-wide level, where the classes are structured according to FunCat taxonomy. Each dataset provides a different description of a specific gene aspect. For each dataset, we report the evaluation of the performance of the flat ADABOOST.MH multi-label classifiers and the HML-Boosting algorithm using precision, recall and F_1 evaluation metrics that are described below.

The flat multi-label classifier does not incorporate the hierarchical structure of the classes as discussed earlier. Each base classifier in the flat multi-label classification framework, is separately trained to identify the set of genes belonging to a specific functional class without considering the hierarchical relationship between the classes.

4.1. Datasets

We chose to demonstrate the performance of our algorithm for the prediction of gene functions in yeast using four bio-molecular datasets that were used in Ref. 16. Valentini [16] pre-processed the datasets so that for each dataset, only genes that are annotated with FunCat taxonomy are selected. To make this paper self-contained, we briefly explain the data collection process and the pre-processing steps performed on the data. Uninformed features that have the same value for all of the examples are removed. Class “99” in FunCat corresponds to an “unclassified protein”. Therefore, genes that are annotated only with that class are excluded. Finally, in order to have a good size of positive training examples for each class, selection has been performed to classes with at least 20 positive examples. Dataset characteristics are summarized in Table 2.

The gene expression dataset, Gene-Expr, is obtained by merging the results of two studies, gene expression measures relative to 77 conditions [17] and transcriptional responses of yeast to environmental stress measured on 173 conditions [18]. For each gene product in the protein-protein interaction dataset, PPI-BG, a binary vector is generated that implies the presence or absence of protein-protein interaction. Protein-protein interaction data have been downloaded from BioGRID database [19, 16].

In Pfam-1 dataset, a binary vector is generated for every gene product that reflects the presence or absence of 4950 protein domains obtained from Pfam (Protein families) database [20, 16]. For PPI-VM dataset, Von Mering experiments produced protein-protein data from yeast two-hybrid assay, mass spectrometry of purified complexes, correlated mRNA expression and genetic interactions [21].

4.2. Evaluation metrics

We have used F_1 measure to jointly consider the contribution of both precision (P) and recall (R). Precision and Recall are defined as follows:

$$P = \frac{TP}{TP + FP} \tag{4}$$

$$R = \frac{TP}{TP + FN} \tag{5}$$

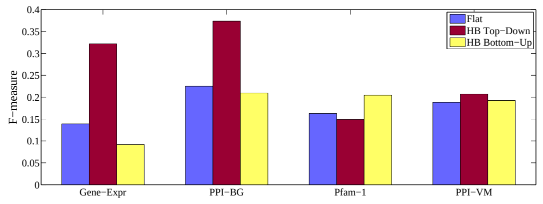
While F_1 measure is defined as follows:

$$F_1 = \frac{2PR}{P + R} = \frac{2TP}{2TP + FP + FN} \tag{6}$$

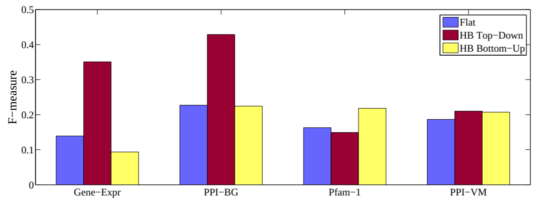
where TP stands for True Positive, TN for True Negative, FP for False Positive and FN for False Negative. When TP=FP=FN=0, we made F_1 measure to equal to 1 as the classifier has correctly classified all the examples as negative examples [11]. The comparison between HML-Boosting algorithm and the Flat method is based on the “per-class” F_1 measure that is obtained by averaging the F-measure for all the classes in the FunCat hierarchy for each dataset. In other words, an overall F_1 measure is obtained by computing F_1 measure for each class separately and then averaging F_1 measure across all the classes.

Table 2. The characteristics of four bio-molecular datasets used in our experiments

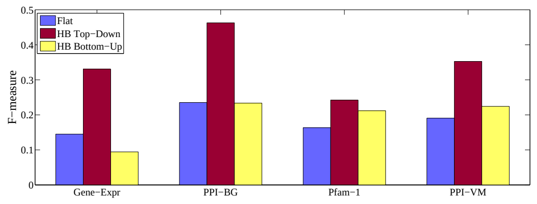
Dataset	Description	Samples	Features	classes
Gene-Expr	Gene expression data	4532	250	230
PPI-BG	PPI data from BioGRID	4531	5367	232
Pfam-1	Protein domain binary data	3529	4950	211
PPI-VM	PPI data from Von Mering experiments	2338	2559	177



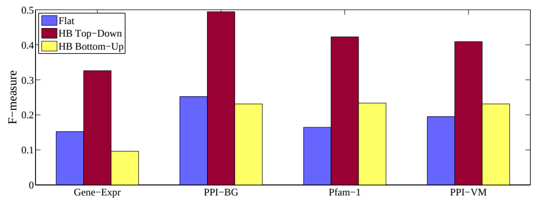
(a) Boosting Iterations = 5



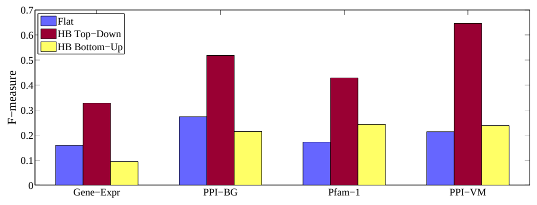
(b) Boosting Iterations = 10



(c) Boosting Iterations = 20



(d) Boosting Iterations = 50



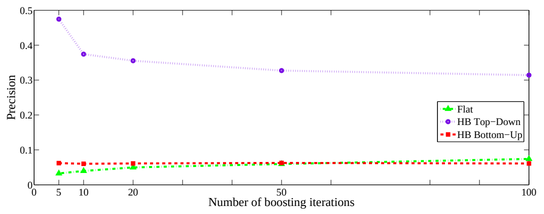
(e) Boosting Iterations = 100

Fig. 3. Overall per-class F_1 measure comparison between flat method, HML-Boosting top-down and HML-Boosting bottom-up.

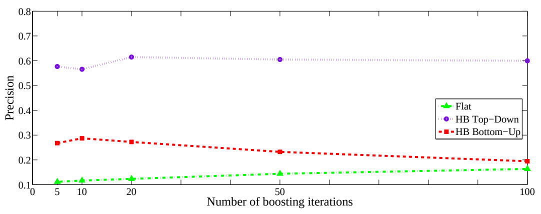
4.3. Results and Discussion

Consistent with most of the other works in the literature, in the hierarchical multi-label classification setting, precision, recall and F_1 measure are used as appropriate evaluation metrics for comparing the performance. The performance of the flat method, the HML-Boosting algorithm using the top-down and the HML-Boosting algorithm using the bottom-up class prediction strategies were compared using per-class F_1 measure; then, we analyzed the performance of the HML-Boosting at each level of FunCat hierarchy. We also studied the influence of changing the number of boosting iterations on the performance.

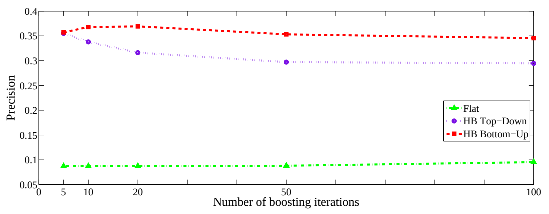
Figure 3 shows the F_1 measure for each dataset using the HML-Boosting with the top-down class prediction strategy during the testing phase, the HML-Boosting with the bottom-up class prediction strategy during the testing phase and the flat classification methods with different boosting iterations. In our experimental setting, five values of boosting iterations: $5, 10, 20, 50$ and 100 have been examined. Each barplot group refers to the results of applying the flat method and the HML-Boosting algorithm (with the top-down and the bottom-up strategies) on a particular dataset. The difference between the HML-boosting and the flat classification is significant in most of the cases. We noticed that as the number of boosting iterations (s) is increased, HML-Boosting significantly outperforms the flat classification method on all the datasets. In other words,



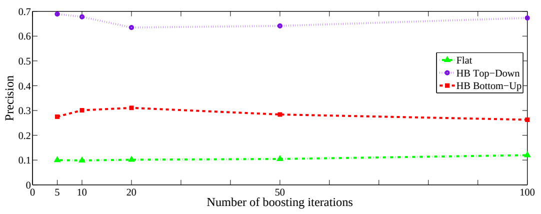
(a) Gene-Expr



(b) PPI-BG



(c) Pfam-1



(d) PPI-VM

Fig. 4. The overall Precision for each dataset using the flat method, HML-Boosting top-down and HML-Boosting bottom-up algorithms.

the improvements become much higher as we increase the number of boosting iterations. Additionally, in most of the cases, the HML-Boosting with the top-down class prediction strategy during the testing phase outperforms its counterpart, the HML-Boosting with the bottom-up class prediction strategy.

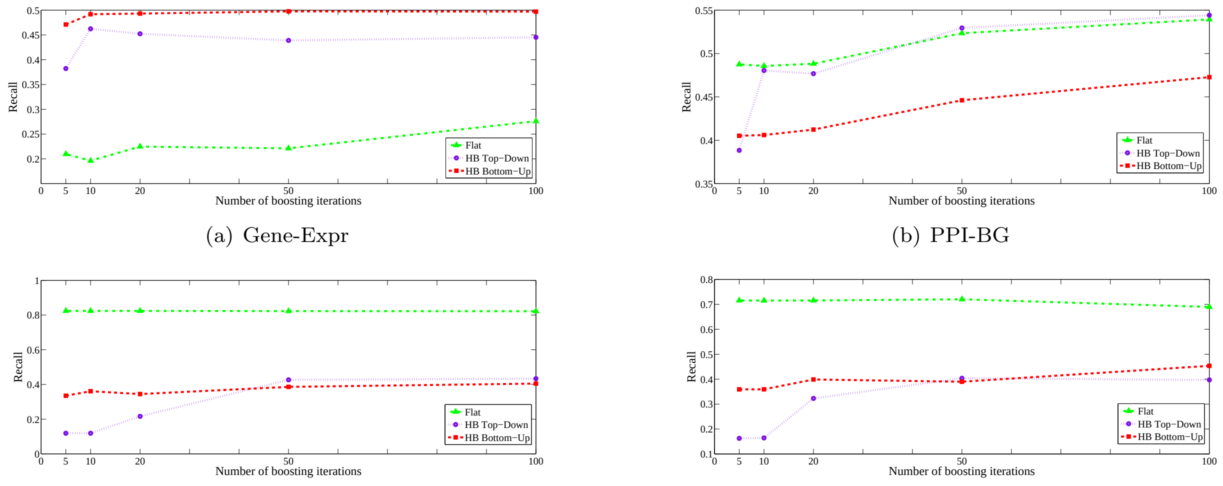
Figure 4 and Figure 5 show the overall "per-class" precision and the overall "per-class" recall, as a function of boosting iterations, for each dataset using the HML-Boosting with the top-down class prediction strategy, the HML-Boosting with the bottom-up class prediction strategy and the flat method. We observed that the HML-Boosting with the top-down class prediction strategy tends to have the best results with respect to precision and F_1 measure while the flat method tends to achieve better results with respect to recall. In fact, both of the class-membership inconsistency correction have similar behavior in terms of the recall.

To get more insights into the performance of the HML-Boosting algorithm with the different testing strategies, we performed a level-wise analysis of the precision, recall, F_1 measure and accuracy on the four datasets. In measuring the level-wise performance, level 1 reflects the root nodes while all other classes are at depth i, where $2 \leq i \leq 5$. We show the results for the top four levels in the hierarchy. Moreover, we show the performance of the HML-Boosting algorithm when the number of boosting iterations is varied. In general, the results improve as we increase the number of iterations. Tables 3, 4, 5 and 6 show the results of per-level evaluation for Gene-Expr, PPI-BG, Pfam-1 and PPI-VM datasets, respectively. The most significant measures for each level are highlighted.

It should be noted that the accuracy of the HML-Boosting using the top-down class prediction strategy in the testing phase reduces as we go down in the tree, from the higher levels to the lower levels. The main reason for this performance reduction is related to the testing mechanism followed by the algorithm. A classifier will be able to test any example only if that example was predicted to be a positive example by the parent of the corresponding class. Therefore, if a classifier at a particular class misclassifies a positive example, this example will be considered as a negative example by all the descendant classes of that class. Hence, the accuracy of the lower levels will be affected by the behavior of the classifiers at the higher levels. For the same reason, less number of examples will be propagated to the classifiers in the lower levels.

5. Conclusion and Future Work

The functional classification of genes is an important and challenging problem in the area of functional genomics. In this paper, we developed HML-Boosting



(a) Gene-Expr

(b) PPI-BG

(c) Pfam-1

(d) PPI-VM

Fig. 5. The overall recall for each dataset using the flat method, HML-Boosting top-down and HML-Boosting bottom-up algorithms.

Table 3. Per-level precision, recall, F_1 measure and accuracy for Gene-Expr dataset using HML-Boosting with top-down class prediction and HML-Boosting with bottom-up class prediction algorithms for the different choice of boosting iterations.

Level	Measurement	HML-Boosting top-down					HML-Boosting bottom-up				
		5 iter	10 iter	20 iter	50 iter	100 iter	5 iter	10 iter	20 iter	50 iter	100 iter
Level 1	Precision	**0.5405**	0.4776	0.4895	0.4719	0.4425	0.1573	0.1620	0.1662	0.1673	**0.1679**
	Recall	0.0527	0.1020	0.1245	0.1606	**0.1809**	0.8337	0.8889	0.9058	0.9290	**0.9351**
	F1	0.0960	0.1681	0.1986	0.2396	**0.2568**	0.2646	0.2740	0.2808	0.2836	**0.2847**
	Accuracy	**0.8627**	0.8602	0.8608	0.8589	0.8550	**0.3479**	0.3371	0.3471	0.3393	0.3388
Level 2	Precision	**0.4385**	0.3448	0.3524	0.3308	0.3050	0.0814	0.0809	**0.0815**	0.0805	0.0793
	Recall	0.5238	0.4904	0.5345	**0.5529**	0.5485	0.6322	0.6723	0.6963	0.7260	**0.7333**
	F1	**0.4774**	0.4049	0.4248	0.4139	0.3921	0.1442	0.1444	0.1460	**0.1449**	0.1431
	Accuracy	**0.8370**	0.8151	0.8150	0.8095	0.8025	**0.6512**	0.6298	0.6214	0.6019	0.5918
Level 3	Precision	**0.3780**	0.2908	0.2739	0.2743	0.2557	**0.0446**	0.0436	0.0440	0.0425	0.0424
	Recall	**0.7619**	0.6547	0.6165	0.6291	0.6402	0.5629	0.5972	0.6141	0.6216	**0.6283**
	F1	**0.5053**	0.4028	0.3793	0.3820	0.3654	**0.0827**	0.0812	0.0821	0.0795	0.0794
	Accuracy	0.7981	**0.8299**	0.8058	0.8152	0.8215	**0.7336**	0.7118	0.7071	0.6931	0.6894
Level 4	Precision	0.18990	**0.2083**	0.2000	0.1905	0.1929	0.0309	0.0308	**0.0311**	0.0304	0.0309
	Recall	0.7143	0.7576	**0.7750**	0.6496	0.7154	0.6192	0.6277	0.6433	0.6449	**0.6651**
	F1	0.3000	**0.3268**	0.3179	0.2946	0.3039	0.0588	0.0588	**0.0594**	0.0580	0.0591
	Accuracy	**0.6889**	0.6460	0.6518	0.6353	0.6365	**0.6487**	0.6434	0.6387	0.6283	0.6246

algorithm and applied it to the gene function prediction problem. HML-Boosting algorithm leverages the hierarchical structure of the classes. Hence, the classifiers that are built tend to be more efficient and effective compared to flat classification methods. The integration of the classes hierarchical structure improves the performance of the prediction. In addition, HML-Boosting is compared with flat classification and the results of experiments on four biomolecular datasets showed that HML-Boosting significantly outperforms flat classification. In addition, the performance of the HML-Boosting algorithm using the top-down and the bottom-up class prediction strategies is evaluated.

For future work, we plan to generalize the proposed approach to general graph structures. We will adapt the HML-Boosting method for the GO taxonomy. Since GO is represented as a DAG, more efforts need to be done to handle the complex relationship between the classes. We also plan to develop a mechanism that enables the children of any node to correct the classification done by the parent node.

Table 4. Per-level precision, recall, F_1 measure and accuracy for PPI-BG dataset using HML-Boosting with top-down class prediction and HML-Boosting with bottom-up class prediction algorithms for the different choice of boosting iterations.

Level	Measurement	HML-Boosting top-down					HML-Boosting bottom-up				
		5 iter	10 iter	20 iter	50 iter	100 iter	5 iter	10 iter	20 iter	50 iter	100 iter
Level 1	Precision	0.7127	0.7177	0.7319	0.7447	**0.7526**	0.1572	0.1606	0.1637	**0.1681**	0.1675
	Recall	0.0708	0.0747	0.0848	0.1086	**0.1405**	0.7285	0.7102	0.7362	0.7780	**0.8182**
	F1	0.1288	0.1354	0.1520	0.1895	**0.2368**	0.2586	0.2619	0.2678	0.2764	**0.2780**
	Accuracy	0.8667	0.8671	0.8683	0.8707	**0.8739**	0.4121	**0.4366**	0.4335	0.4268	0.4019
Level 2	Precision	0.5911	0.5914	0.6337	**0.6487**	0.6287	0.0955	0.0966	**0.0983**	0.0959	0.0926
	Recall	0.8304	0.8506	**0.8600**	0.8252	0.8348	0.5758	0.5797	0.6060	0.6488	**0.6804**
	F1	0.6906	0.6977	0.7297	0.7264	**0.7447**	0.1638	0.1657	**0.1692**	0.1671	0.1630
	Accuracy	0.8223	0.8210	0.8404	0.8481	**0.8574**	0.7268	**0.7286**	0.7235	0.6995	0.6754
Level 3	Precision	0.5625	0.5413	0.6077	0.6143	**0.6722**	0.0531	0.0541	**0.0556**	0.0554	0.0538
	Recall	0.7246	0.7170	0.7645	0.7798	**0.7871**	0.5410	0.5501	0.5771	0.6254	**0.6559**
	F1	0.6333	0.6169	0.6771	0.6872	**0.6990**	0.0967	0.0985	0.1014	**0.1018**	0.0994
	Accuracy	0.8667	0.8520	0.8661	**0.8712**	0.8711	0.7879	**0.7886**	0.7852	0.7682	0.7504
Level 4	Precision	0.6078	0.5139	0.6133	**0.6226**	0.6163	0.0353	0.0326	0.0349	**0.0358**	**0.0358**
	Recall	0.6940	0.7115	0.7500	0.8075	**0.8328**	0.6352	0.6584	0.6793	0.7002	**0.7235**
	F1	0.6481	0.5968	0.6748	0.7031	**0.7084**	0.0669	0.0621	0.0664	0.0681	**0.0683**
	Accuracy	**0.7918**	0.7331	0.7837	0.7770	0.7789	**0.6846**	0.6456	0.6598	0.6587	0.6482

Table 5. Per-level precision, recall, F_1 measure and accuracy for Pfam-1 dataset using HML-Boosting with top-down class prediction and HML-Boosting with bottom-up class prediction algorithms for the different choice of boosting iterations.

Level	Measurement	HML-Boosting top-down					HML-Boosting bottom-up				
		5 iter	10 iter	20 iter	50 iter	100 iter	5 iter	10 iter	20 iter	50 iter	100 iter
Level 1	Precision	0.9065	0.9065	**0.9128**	0.8817	0.8842	0.1584	**0.1585**	0.1584	0.1529	0.1541
	Recall	0.0433	0.0433	0.0468	0.0564	**0.0578**	0.7018	0.7001	0.6994	0.7312	**0.7417**
	F1	0.0827	0.0827	0.0890	0.1060	**0.1085**	0.2584	**0.2585**	0.2583	0.2529	0.2552
	Accuracy	**0.9240**	0.8515	0.8520	0.8530	0.8532	0.3678	**0.3695**	**0.3695**	0.3217	0.3205
Level 2	Precision	0.8521	0.8898	**0.8994**	0.8398	0.8178	0.0977	0.0979	0.0984	0.0981	**0.0993**
	Recall	0.8345	0.7793	0.8512	0.8782	**0.9154**	0.5764	0.5767	0.5804	0.6098	**0.6221**
	F1	0.8432	0.8309	**0.8746**	0.8586	0.8638	0.1670	0.1674	0.1683	0.1690	**0.1713**
	Accuracy	0.8515	0.9223	**0.9341**	0.9231	0.9230	0.7000	**0.7007**	**0.7007**	0.6870	0.6860
Level 3	Precision	0.7500	0.7857	**0.8000**	0.7500	0.7143	0.0544	0.0546	0.0551	**0.0570**	0.0561
	Recall	0.8438	**0.8750**	0.8421	0.8298	0.8224	0.5209	0.5235	0.5286	0.5497	**0.5726**
	F1	0.7941	**0.8280**	0.8205	0.7879	0.7645	0.0985	0.0990	0.0999	**0.1032**	0.1022
	Accuracy	0.9288	0.9340	**0.9351**	0.9186	0.9136	0.7662	0.7664	**0.7665**	0.7659	0.7534
Level 4	Precision	0.7714	**0.8636**	**0.8636**	0.7736	0.7368	0.0363	0.0349	0.0373	**0.0382**	0.0366
	Recall	**0.7297**	0.5135	0.5135	0.6949	0.7000	0.6079	0.6221	0.6239	0.6390	**0.6513**
	F1	**0.7500**	0.6441	0.6441	0.7321	0.7179	0.0685	0.0660	0.0704	**0.0720**	0.0692
	Accuracy	**0.8571**	0.8108	0.8108	0.8113	0.8146	0.6448	0.6219	**0.6463**	0.6462	0.6238

More specifically, we are interested in minimizing the misclassified instances that propagate down the tree.

Acknowledgment

The authors would like to thank Giorgio Valentini for providing the pre-processed data used in this paper.

References

1. D. Aleksovski, D. Kocev and S. Dzeroski. Evaluation of Distance Measures for Hierarchical Multi-Label Classification in Functional Genomics. ECML/PKDD 2009 Workshop on Learning from Multi-Label Data, Bled, Slovenia, 2009.
2. Z. Barutcuoglu, R. Schapire, and O. Troyanskaya. Hierarchical multi-label prediction of gene function. Bioinformatics 2006; 22(7):830–836.
3. J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-Based Learning of Hierarchical Multilabel Classification Models. The Journal of Machine Learning Research 2006; 7:1601–1626.
4. C. Vens, J. Struyf, L. Schietgat, S. Dzeroski, and H. Blockeel. Decision trees for hierarchical multi-label classification. Machine Learning 2008; 73:185–214.

Table 6. Per-level precision, recall, F_1 measure and accuracy for PPI-VM dataset using HML-Boosting with top-down class prediction and HML-Boosting with bottom-up class prediction algorithms for the different choice of boosting iterations.

Level	Measurement	HML-Boosting top-down					HML-Boosting bottom-up				
		5 iteration	10 iter	20 iter	50 iter	100 iter	5 iter	10 iter	20 iter	50 iter	100 iter
Level 1	Precision	0.7355	0.7266	0.7654	0.7641	**0.7661**	0.1716	0.1650	0.1672	0.1700	**0.1801**
	Recall	0.0445	0.0465	0.0619	0.0744	**0.0834**	0.7393	0.6603	0.6723	0.6943	**0.8357**
	F1	0.0838	0.0873	0.1146	0.1356	**0.1505**	0.2786	0.2640	0.2678	0.2731	**0.2963**
	Accuracy	0.8440	0.8440	0.8463	0.8476	**0.8487**	0.3850	0.4087	**0.4094**	0.4064	0.3626
Level 2	Precision	0.5859	0.7094	**0.7170**	0.7026	0.7000	0.1025	**0.1069**	0.1050	0.1066	0.1014
	Recall	0.6818	0.7217	**0.7651**	0.7446	0.7624	0.5588	0.5270	0.5527	**0.5750**	0.6517
	F1	0.6303	0.7155	**0.7403**	0.7230	0.7299	0.1732	0.1777	0.1764	**0.1799**	0.1755
	Accuracy	0.8151	0.8759	**0.8818**	0.8754	0.8781	0.6979	**0.7238**	0.7078	0.7031	0.6533
Level 3	Precision	0.6180	0.6598	**0.6797**	0.5989	0.6244	0.0596	0.0629	0.0621	**0.0638**	0.0611
	Recall	0.8088	0.8421	0.8365	0.8195	**0.8477**	0.5250	0.5063	0.5288	0.5569	**0.6045**
	F1	0.7006	0.7399	**0.7500**	0.6921	0.7191	0.1070	0.1119	0.1111	**0.1146**	0.1110
	Accuracy	**0.8618**	0.8427	0.8528	0.8170	0.8339	0.7277	**0.7503**	0.7369	0.7324	0.6990
Level 4	Precision	0.5238	0.7059	**0.8158**	0.6935	0.7356	0.0385	**0.0421**	0.0409	0.0414	0.0415
	Recall	0.6875	0.7059	0.8611	0.8431	**0.8767**	0.6839	0.6667	0.6868	0.7011	**0.7385**
	F1	0.5946	0.7059	**0.8378**	0.7611	0.8000	0.0730	**0.0793**	0.0772	0.0782	0.0786
	Accuracy	0.7619	0.8182	**0.8537**	0.8029	0.8025	0.5147	**0.5676**	0.5413	0.5385	0.5165

5. L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Dzeroski. Predicting gene function in S. cerevisiae and A. thaliana using hierarchical multi-label decision trees. European Conference on Computational Biology. 22–26, 2008. Cagliari, Italy.
6. H.W. Mewes, K. Heumann, A. Kaps, K. Mayer, F. Pfeiffer, S. Stocker, and D. Frishman. MIPS: a database for genomes and protein sequences. Nucleic Acids Research 1999; 27(1):44–48.
7. The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. Nat Genet 2000; 25(1):25–29.
8. S. Mostafavi and Q. Morris. Using the gene ontology hierarchy when predicting gene function. In Conference on Uncertainty in Artificial Intelligence (UAI). 22–26, 2009. Montreal, Canada.
9. G. Tsoumakas and I. Katakis. Multi-label classification: An overview. International Journal of Data Warehousing and Mining 2007; 3(3):1–13.
10. G. Valentini and M. Re. Weighted True Path Rule: a multilabel hierarchical algorithm for gene function prediction. MLD-ECML 2009: 1st International Workshop on learning from Multi-Label Data. 133–146, 2009. Bled, Slovenia.
11. A. Esuli, T. Fagni, and F. Sebastiani. Boosting multi-label hierarchical text categorization. Information Retrieval 2008; 11:287–313.
12. A. Sokolov and A. Ben-Hur. Hierarchical classification of gene ontology terms using the GOstruct method. JBCB 2010; 8(2):357–376.
13. C. N. Silla Jr. and Alex A. Freitas. A survey of hierarchical classification across different application domains. Data Mining and Knowledge Discovery. Data Mining and Knowledge Discovery. *to appear.*
14. R. Schapire and Y. Singer. BOOSTEXTER: Aboosting-based system for text categorization. Machine Learning 2000; 39(2/3):135–168.
15. G. Jun and J. Ghosh. Multi-class Boosting with Class Hierarchies. MCS 2009; 32–41.
16. G. Valentini. True Path Rule hierarchical ensembles for genome-wide gene function prediction. IEEE ACM Transactions on Computational Biology and Bioinformatics. 2010. in press.
17. P. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomices cerevisiae by microarray hybridization. Mol. Biol. Cell 1998; 9:3273–3297.
18. A.P. Gasch, P.T. Spellman, C.M. Kao, O. Carmel-Harel, M.B. Eisen, G. Storz, D. Botstein, and P.O. Brown. Genomic expression programs in the response of yeast cells to environmental changes. Mol. Biol. Cell 2000; 11:4241–4257.
19. C. Stark, B. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. Nucleic Acids Research 2006; 34:D535–D539.
20. M. Deng, T. Chen, and F. Sun. An integrated probabilistic model for functional prediction of proteins. in Proc 7th Int Conf Comp Mol Biol. 95–103, 2003.
21. C. Von Mering, R. Krause, B. Snel, M. Cornell, S. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. Nature 2002; 417:399–403.