Intelligent Data Analysis 18 (2014) 717–738
DOI 10.3233/IDA-140665
IOS Press

Hierarchical multi-label classification with SVMs: A case study in gene function prediction

Peerapon Vateekul[a,*], Miroslav Kubat[b] and Kanoksri Sarinnapakorn[c]
[a]*Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand*
[b]*Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL, USA*
[c]*Hydro Informatics Division, Hydro and Argo Informatics Institute, Bangkok, Thailand*

Abstract. *Hierarchical multi-label classification* is a relatively new research topic in the field of classifier induction. What distinguishes it from earlier tasks is that it allows each example to belong to two or more classes at the same time, and by assuming that the classes are mutually related by generalization/specialization operators. The paper first investigates the problem of performance evaluation in these domains. After this, it proposes a new induction system, *HR*-SVM, built around support vector machines. In our experiments, we demonstrate that this system's performance compares favorably with that earlier attempts, and then we proceed to an investigation of how *HR*-SVM's individual modules contribute to the overall system's behavior. As a testbed, we use a set of benchmark domains from the field of gene-function prediction.

Keywords: Hierarchical multi-label classification, support vector machines, gene-function prediction

AUTHOR COPY

1. Introduction

In its baseline problem statement, the field of classifier induction seeks to develop mechanisms capable of assigning an example to one (and only one) out of a set of mutually independent classes. Some recent applications, though, have generalized this scenario in two significant aspects: (1) an example is allowed to belong to two or more classes at the same time, and (2) the classes are hierarchically organized. In this paper, we refer to this task by the acronym HMC (Hierarchical Multi-label Classification).

Among the applications of HMC, perhaps the most typical are web repositories/digital libraries (e.g., Dmoz, Wikipedia,[1] Yahoo [26,27], LookSmart [10], EUROVOC [34], Reuters [24], OHSUMED [16]) and image recognition [40]. Our own work deals with *gene-function prediction*. Here, genes represent examples, each gene function is a different class, and the class-to-class relations are specified by a directed acyclic graph (DAG) such as the one in Fig. 1 (note that each node in a DAG can have more than one parent).

*Corresponding author: Peerapon Vateekul, Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, 254 Phayathai Road, Pathumwan, Bangkok 10330, Thailand. Tel.: +66 2218 6989; E-mail: peerapon.v@chula.ac.th.

[1]The data sets from Wikipedia (www.wikipedia.org) and the ODP Web directory data (www.dmoz.org) are available through the 2nd Pascal Challenge on Large Scale Hierarchical Text Classification (LSHTC2).

1088-467X/14/$27.50 © 2014 – IOS Press and the authors. All rights reserved

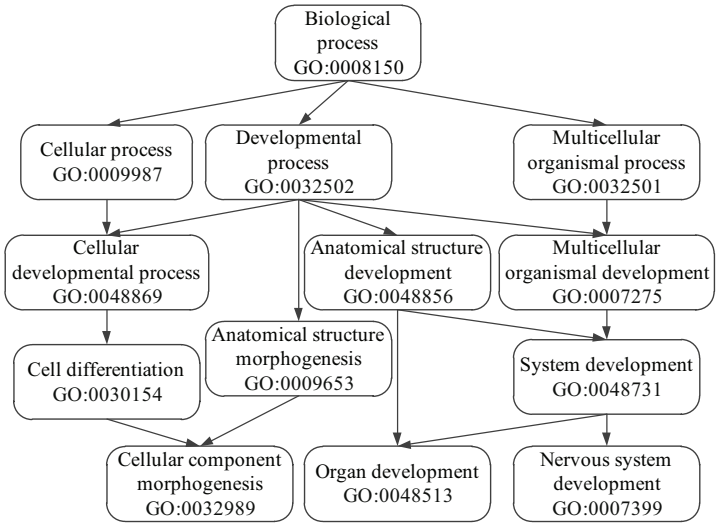

Fig. 1. An example class hierarchy of immune system processes in the field of Gene Ontology (GO).

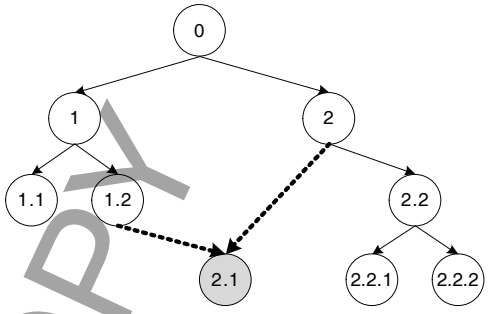

Fig. 2. An example of a DAG-structured class hierarchy.

Due to the relative novelty of the field, it is to be expected that existing induction systems for HMC domains are still far from perfect, and leave ample space for further research. In particular, we focus here on the top-down approach that begins by inducing a classifier for each class at the highest level of the DAG, and then proceeds downward, always employing the higher-level classifiers when creating the training sets for lower-level classifiers.

The performance of existing systems is limited by three problems. First, misclassifications committed at higher levels of the class hierarchy tend to get propagated downwards, making it hard to induce accurate classifiers for the lowest-level classes. Second, the decision to induce a separate binary classifier for each class often means that the training data for classifier induction tend to be imbalanced (negative examples outnumbering positive examples). Third, since different classes are characterized by different sets of attributes, it is necessary to run attribute-selection techniques separately for each of them.

To test the hypothesis that explicit treatment of these aspects will improve classification performance, we developed our own induction system. The baseline technique we used for the induction of the binary classifiers is *R-SVM* [45], our earlier version of a Support Vector Machine (SVM) [19,44,48], tailored to domains with imbalanced classes. Throughout the text, we refer to our new system by the acronym *HR*-SVM, where H stands for hierarchical). Having experimented with several benchmark domains, we observed that *HR*-SVM compared favorably with the best of the existing "competitors," including the hierarchical version of the traditional SVM (*H*-SVM)[2] [12,13] and another well-known approach, Clus-HMC [3,35,47].

In the course of the work, we have realized that HMC's performance has to be evaluated along somewhat different criteria than those used in classical machine learning. Let $\mathbf{x}$ be an example, and let C be the set of classes to which $\mathbf{x}$ belongs. A perfect classifier will label $\mathbf{x}$ with all classes from C, never suggesting any class from outside C; moreover, HMC usually requires that any $\mathbf{x}$ that has been labeled with c_i should also be labeled with all ancestors of c_i in the class hierarchy. To be able to reflect these

[2] *H*-SVM is originally called "TreeSVM" with the "siblings policy".

AUTHOR COPY

requirements in performance evaluation, we developed novel crieria based on *precision*, *recall*, and F_1 that are commonly used in information retrieval.

The rest of the paper is organized as follows. Section 2 defines the HMC task, and Section 3 briefly surveys related work. Section 4 discusses diverse aspects of HMC performance and develops novel criteria reflecting these aspects. Our own induction system, *HR*-SVM, is detailed in Section 5. Experimental data are summarized in Section 6, and experimental results are reported in Section 7.

2. Hierarchical classification: A formal problem statement

A graph consists of a set of nodes, $\mathcal{N}$, and a set of edges, $\mathcal{E}$, where an edge is an ordered pair of nodes, $(N_p, N_c) \in \mathcal{E} \subseteq \{\mathcal{N} \times \mathcal{N}\}$. In this pair, N_p is referred to as a parent, and N_c as a child. A *path*, $N_a \rightarrow N_c$, from an ancestor, N_a, to a child, N_c, is a series of edges, $\{(N_1, N_2), (N_2, N_3), \ldots, (N_{n-1}, N_n)\}$ such that $N_1 = N_a$ and $N_n = N_c$. In a *directed acyclic graph* (DAG) the existence of a path, $N_a \rightarrow N_c$, guarantees the non-existence of the opposite-direction path, $N_c \rightarrow N_a$. A node without any child is called a *leaf* node, and a node without any parent is called a *root* node.

In the task addressed by this paper, we consider a set of class labels, $\mathcal{C}$, whose mutual relations are specified by a class hierarchy, $\mathcal{H}$, that has the form of a DAG in which each node represents one and only one class. The path $C_g \rightarrow C_s$ is interpreted as meaning that C_g is a generalization of C_s (or, equivalently, that C_s is a specialization of C_g).

Let $\mathcal{X} \subset \mathcal{R}^p$ be a finite set of examples, each described by p numeric attributes. We assume that each $x_i \in \mathcal{X}$ is assigned a set of class labels, $\mathcal{L} = \{C_1, \ldots, C_l\} \subseteq \mathcal{H}$ (all classes belong to the given class hierarchy). From these data, we want to induce a classifier to carry out the mapping $\Phi : \mathcal{X} \rightarrow 2^{\mathcal{C}}$ in a way that maximizes classification performance. Moreover, an example belonging to class C_c is expected to belong also to *all* C_c's ancestor classes, C_a. This property is called "*hierarchical constraint*".

Two versions of this task exist. In the *mandatory leaf-node problem* (MLNP), only the leaf-node classes are used. By contrast, in the *non-mandatory leaf node problem* (NMLNP), an example can be labeled with any class from the given class hierarchy. Considering the class hierarchy from Fig. 2, MLNP permits an example to be labeled only with a subset of $\{C1.1, C2.1, C2.2.1, C2.2.2\}$, but NMLNP allows also the other class labels (e.g., C1 or C2.2). Our own research focuses on the more general NMLNP.

3. Related work

Surveying existing solutions to the HMC problem, [39] distinguishes three fundamental strategies: (i) flat classification, (ii) the top-down approach (local classifiers), and (iii) the "big-bang" approach (global classifiers).

3.1. Flat classification

This ignores the class hierarchy altogether, and deals only with the leaf-node classes (as if the problem were MLNP), whether by a single multi-label classifier or by a set of binary classifiers (a separate one for each leaf node). The advantage is that this makes it possible to rely on traditional machine-learning techniques such as neural networks, decision trees, or SVM, and indeed such attempts have been reported by several authors [18,33,49].

AUTHOR COPY

This approach, of course, only makes sense if the leaf-node class label is known for each example, and if the nature of the application is such that the users are willing to tolerate their inability to identify non-terminal classes. This is not the case of the domains addressed in this paper, and we will therefore not go into any further details.

3.2. Top-down approach (local classifier)

This is the most common approach in HMC induction. In the simplest case, a separate (local) classifier is induced for each node in the DAG-specified class hierarchy, starting at the top levels, and then proceeding downwards, creating in the process a whole hierarchy of classifiers. The advantage is the relative simplicity. On the other hand, the approach tends to suffer from "error propagation": misclassifications of the higher-level classes are propagated to lower levels.

Koller and Sahami [22] were perhaps the first to experiment with local classifiers, choosing Naive Bayes as the baseline learner used to induce each individual class. The authors experimented with tree-structured class hierarchies (no more than one parent for any node) that were limited to just two levels.

Fagni and Sebastiani [12,13] compared four different policies to generate the binary training data from which to induce the local classifiers: Sibling, ALL, BestGlobal, and BestLocal. They used tree-structured hierarchical versions of boosting and SVM, called TreeBoost and TreeSVM, respectively, achieving the best results by the use of the Sibling policy in which the negative training examples of the i-th node are all positive examples of its sibling nodes in the hierarchy. In our paper, we refer to TreeSVM with the Sibling policy as "*H*-SVM" because it can be regarded as a hierarchical algorithm that uses SVM.

Sun and Lim [41] applied this strategy to text classification where the class hierarchy was a plain tree structure. For each class, they induced two SVMs: a local classifier and a subtree classifier. An example is labeled as c_i by the local classifier, while the latter one decides whether or not this example should be passed to c_i's subclassifiers. Nguyen et al. [28] extended the approach to domains with DAG-structured class hierarchies by transforming the DAG hierarchy into a set of tree hierarchies. Experimental results indicated high classification performance, but also high computational costs.

Seeking to further improve the performance, Secker et al. [36] used several different induction algorithms for each node of the hierarchy: Naive Bayes, SMO, 3-NN, etc. For each node, they trained ten classifiers, and then selected the one with the best classification results. This improved classification accuracy, but also further increased computational costs.

Addressing the problem of the very high number of classes in the hierarchy, Bi and Kwok [2] applied the kernel dependency estimation (KDE) to reduce the number of the classes during the training process. In particular, they proposed an algorithm called "Condensing Sort and Selection Algorithm (CSSA)" for the tree-structured hierarchies, and then extended it to the DAG-structured hierarchies. They did not report induction time or the amount of class-number reduction.

Recently, Alaydie et al. [1] proposed a framework called HiBLADE (Hierarchical multi-label Boosting with Label Dependency), applied to tree-structured hierarchies. The baseline classifier for each class is a boosting algorithm, such as ADABOOST, where the model for each boosting iteration is updated by a method utilizing Baysian correlation.

3.3. The "big-bang" approach (global classifier)

Instead of inducing a separate binary classifier for each node, some authors prefer to induce one big (global) classifier for the entire class hierarchy. In this way, mutual interdependencies of the classes are

AUTHOR COPY

more easily taken into account, and the global classifier is sometimes smaller than the sum of the local classifiers.

Clare and King [6] developed a hierarchical extension to the decision-tree generator C4.5 [31] and applied it to functional-genomics data (they called their system HC4.5). To give higher priority to more specific classes, they introduced a mechanism to weigh, accordingly, the entropy formula.

Another attempt to apply the decision-tree paradigm to HMC domains was reported by Blockeel et al. [3] whose Clus-HMC is a hierarchical version of the earlier "predictive clustering tree" (PCT) [4]. Ven et al. [47] improved Clus-HMC so that it could be used in DAG-specified class hierarchies. An ensemble version of the algorithm, Clus-HMC-ENS, was proposed by Schietgat et al. [35]. Unfortunately, although the ensemble concept does improve classification performance, its computational costs are much higher than those of the original Clus-HMC.

Pandey et al. [29] proposed a global-approach hierarchical framework based on the k-nearest neighbor classifier (k-NN). There are many improvements in the system. First, the distance function is Lin's semantic similarity measure. Second, the prediction function of the i-th class incorporates the inter-relationship score of the i-th class to other classes in the hierarchy. Finally, the mechanism to filter insignificant class inter-relationships was suggested.

4. Performance evaluation

How to evaluate performance in HMC is not an easy question, and the research community has not reached a consensus. In this section, we attempt to improve the situation by developing evaluation criteria that we believe are sufficiently objective and robust.

4.1. Classical approach

We begin with the two-class case where each example is either positive and negative. Classical machine learning literature evaluated these classifiers by error-rate estimates obtained by the comparison of testing examples' known class labels with those recommended by the classifier. Error rate, however, is inadequate in domains where one class significantly outnumbers the other [23]. For instance, if only 1% of the examples are positive, then a classifier that labels all examples as negative will achieve 99% accuracy, and yet it is virtually useless.

For this latter case, other criteria have been used; the most popular among them are *precision* and *recall*. Let us denote by TP the number of true positives, by FN the number of false negatives, by FP the number of false positives, and by TN the number of true negatives. *Precision* and *recall* are defined as follows:

$$Pr = \frac{TP}{TP + FP} \quad Re = \frac{TP}{TP + FN} \tag{1}$$

In plain English, *precision* is the percentage of truly positive examples among those labeled as such by the classifier; *recall* is the percentage of positive examples that have been recognized as such ("recalled") by the classifier. Which of the two is more important depends on the specific needs of the concrete domain. Seeking to combine them in a single formula, [43] proposed F_β, where the user-specified parameter, $\beta \in [0, \infty)$, quantifies each component's relative importance:

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re} \tag{2}$$

AUTHOR COPY

Table 1
Macro-averaging and micro-averaging of performance criteria in domains with l classes

Average	Criteria	Equation
Macro	Precision	$Pr^M = \frac{\sum_{j=1}^{l} Pr_j}{l}$
	Recall	$Re^M = \frac{\sum_{j=1}^{l} Re_j}{l}$
	F_1	$F_1^M = \frac{\sum_{j=1}^{l} F_{1,j}}{l}$
Micro	Precision	$Pr^\mu = \frac{\sum_{j=1}^{l} TP_j}{\sum_{j=1}^{l} (TP_j + FP_j)}$
	Recall	$Re^\mu = \frac{\sum_{j=1}^{l} TP_i}{\sum_{j=1}^{l} (TP_j + FN_j)}$
	F_1	$F_1^\mu = \frac{2 \times Pr^\mu \times Re^\mu}{Pr^\mu + Re^\mu}$

Table 2
The hierarchical version of *precision*, *recall*, and F_1 for an example i

Precision	Recall	F_1
$hPr_i = \frac{\lvert\hat{P}_i \cap \hat{T}_i\rvert}{\lvert\hat{P}_i\rvert}$	$hRe_i = \frac{\lvert\hat{P}_i \cap \hat{T}_i\rvert}{\lvert\hat{T}_i\rvert}$	$hF_{1,i} = \frac{2 \times hPr_i \times hRe_i}{hPr_i + hRe_i}$

Table 3
The micro-averaging version of *precision*, *recall*, and F_1 of n examples

Criteria	Macro-equation
Precision	$hPr^\mu = \frac{\sum_{i=1}^{n} \lvert\hat{P}_i \cap \hat{T}_i\rvert}{\sum_{i=1}^{n} \lvert\hat{P}_i\rvert}$
Recall	$hRe^\mu = \frac{\sum_{i=1}^{n} \lvert\hat{P}_i \cap \hat{T}_i\rvert}{\sum_{i=1}^{n} \lvert\hat{T}_i\rvert}$
F_1	$hF_1{}^\mu = \frac{2 \times hPr^\mu \times hRe^\mu}{hPr^\mu + hRe^\mu}$

It would be easy to show that $\beta > 1$ apportions more weight to *recall* while $\beta < 1$ emphasizes *precision*. Moreover, F_β converges to *recall* if $\beta \to \infty$, and to *precision* if $\beta = 0$. If we do not want to give more weight to either of them, we use the neutral $\beta = 1$:

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{3}$$

All this, however, applies only to domains where each example is labeled with one and only one class. For multi-label domains, [51] proposed two methods to average the above metrics over multiple classes: (1) *macro-averaging*, where *precision* and *recall* are first computed separately for each class and then averaged; and (2) *micro-averaging*, where *precision* and *recall* are obtained by summing over all individual decisions. Which of the two approaches is better depends on the concrete application. Generally speaking, micro-F_1 weighs the classes by their relative frequency, whereas macro-F_1 gives equal weight to each class. The formulas are summarized in Table 1 where Pr_j, Re_j, and $F_{1,j}$ stand for *precision*, *recall*, and F_1 for the j-th class (from l classes).

4.2. Hierarchical classification

In domains with hierarchically organized classes, the above-defined classical metrics do not suffice. [7,41] surveyed several alternatives, such as distance-based, semantics-based, and hierarchy-based measures.

Among these, we prefer the metrics proposed by Kiritchenko et al. [21] as extended versions of, again, *precision*, *recall*, and F_1. In the hierarchical context, each example belongs not only to a given class, but also to all ancestors of this class in the class hierarchy. The metrics for the i-th example are summarized in Table 2, where P_i and T_i represent sets of predicted and true classes respectively, and $\hat{P}_i$ and $\hat{T}_i$ stand for P_i and T_i plus their ancestors. *Hierarchical precision* (hPr_i) is the accuracy of the *prediction* path, *hierarchical recall* (hRe_i) is the accuracy of the *true* path, and *hierarchical* F_1 ($hF_{1,i}$) is an equal-weight combination of hPr_i and hRe_i.

For illustration, referring to the hierarchy from Fig. 2, let $P_i = \{C2.2\}$ and $T_i = \{C2.2.1\}$. This yields $\hat{P}_i = \{C2, C2.2\}$ and $\hat{T}_i = \{C2, C2.2, C2.2.1\}$, so that $hPr_i = \frac{2}{2} = 1$, $hRe_i = \frac{2}{3}$, and $hF_{1,i} = 0.8$.

AUTHOR COPY

Table 4
The macro-averaging version of *precision*, *recall*, and F_1 of n examples

Criteria	Micro-equation
Precision	$hPr^M = \frac{\sum_{i=1}^{n} hPr_i}{n}$
Recall	$hRe^M = \frac{\sum_{i=1}^{n} hRe_i}{n}$
F_1	$hF_1^M = \frac{\sum_{i=1}^{n} hF_{1,i}}{n}$

Table 5
An example of hierarchical-classification results

Example	True class	Predicted class
x_1	C1.1	C1.2
x_2	C2.2.1	C1.1, C2.2.1
x_3	C1, C2.2	C2.2
x_4	C1	C1.1
x_5	C1.1	C1

Table 6
The true class matrix T (top) and the predicted class matrix P (bottom) for the examples from Table 5

	C1	C1.1	C1.2	C2	C2.1	C2.2	C2.2.1	C2.2
x_1	1	1	0	0	0	0	0	0
x_2	0	0	0	1	0	1	1	0
x_3	1	0	0	1	0	1	0	0
x_4	1	0	0	0	0	0	0	0
x_5	1	1	0	0	0	0	0	0

	C1	C1.1	C1.2	C2	C2.1	C2.2	C2.2.1	C2.2
x_1	1	0	1	0	0	0	0	0
x_2	1	1	0	1	0	1	1	0
x_3	0	0	0	1	0	1	0	0
x_4	1	1	0	0	0	0	0	0
x_5	1	0	0	0	0	0	0	0

Suppose we want to evaluate the performance measured on a data set with n examples labeled with l hierarchically organized classes. If we combine the performance of all examples by *micro-averaging* (Table 3), the value is biased towards examples with *longer* paths. To see why, consider the following data set where the true labels are as follows: $\mathcal{S} = \{(x_1, C2.2), (x_2, C1), (x_3, C2)\}$. Let the classes be organized according to the hierarchy from Fig. 2. Suppose there are two classifiers, Φ_1 and Φ_2, where $\Phi_1(\mathcal{S}) = \{(x_1, \underline{C2.2}), (x_2, C2), (x_3, C1)\}$ (correctly predicting the class of x_1), and $\Phi_2(\mathcal{S}) = \{(x_1, C1.1), (x_2, \underline{C1}), (x_3, \underline{C2})\}$ (correctly predicting classes of x_2 and x_3).[3] While it seems obvious that Φ_2 is better than Φ_1, micro-averaging seems to indicate that the performance of both classifiers is about the same, $hPr^\mu = hRe^\mu = {hF_1}^\mu = \frac{1}{2}$.

To reasonably evaluate the classifiers in the previous example, we propose to apply *macro-averaging* to merge hierarchical measures of all examples as shown in Table 4. This averaging method then evaluates the performance of Φ_2 higher than that of Φ_1. For Φ_1, $hPr^M = hRe^M = {hF_1}^M = \frac{1}{3}$, and, for Φ_2, $hPr^M = hRe^M = {hF_1}^M = \frac{2}{3}$.

4.3. A more advanced approach

To see the limitation of the above criteria, consider the domain from Table 5 and the class hierarchy from Fig. 2. For five examples, the table lists their true class labels as well as the labels assigned to them by the classifier.

This information is expressed in the matrix form in Table 6 (top) for the true classes, and in Table 6 (bottom) for the classes predicted by the classifier. Note that each row represents an example, and each column represents a class. The value of a given field is "1" if the example belongs to the class and "0" if it does not.

The hierarchical criterion from Section 4.2 evaluates the performance separately across each *row* of the matrix, and then averages the results. By contrast, the multi-label criterion from Section 4.1 evaluates

[3]The class correctly predicted by the classifier is underlined.

AUTHOR COPY

the performance separately for each *column*, and then averages the results. We want to combine the two approaches.

Incidently, such combination was encouraged by the organizers of a recent competition to develop the best system on "Large Scale Hierarchical Text Classification (LSHTC2)".[4] In their notation, the multi-label criteria are referred to as "Label-based (Macro)" ($LbMa$),[5] so that $LbPr$, $LbRe$, and LbF_1 refer to multi-label *precision*, *recall*, and F_1, respectively. Similarly, the hierarchical (Example-based) criteria are denoted by $EbPr$, $EbRe$, and EbF_1, respectively.

Whether to prefer example-based criteria or label-based ones may be a matter of some dispute, but a good classifier should satisfy both. In line with this argument, we propose a simple way to accomplish just that. Denoting the criterion by the acronym *ELb* (Example-Label-based), we define it by the following formula:

$$\mathit{ELbFunc}(Eb, Lb) = \frac{2 \times Eb \times Lb}{Eb + Lb} \tag{4}$$

where *Eb* is an example-based metric (*precision*, *recall*, or F_1) and *Lb* is a class-label-based metric. For instance, *ELbPr* is our equivalent of *precision*, *ELbRe* is our equivalent of *recall*, and *ELb*F_1 is our equivalent of F_1.

Finally, let us remark that, in some domains, the closer a class is to the root of the class hierarchy, the more important it is deemed. In the work reported here, this factor is ignored.

5. *HR-SVM*: Induction of class hierarchies

Let us now proceed to the description of *HR*-SVM, our induction system designed for the needs of multi-label domains with classes organized in DAG-specified class hierarchies. In all illustrative examples below, we will assume that the class hierarchy is the DAG from Fig. 2 that we reprint here for the reader's convenience as Fig. 3. Each example can be labeled with leaf nodes as well as with the internal nodes of the DAG. Following the top-down approach, *HR*-SVM uses a "baseline learner" to induce a local classifier for each non-root node in the class hierarchy.

Care is taken to follow the *hierarchical constraint*. For instance, if classifier $C2$ (associated with node '1' in Fig. 3) labels **x** as negative, then **x** has to be labeled as negative also by the classifiers corresponding to $C2$'s subclasses: $\{C2.1, C2.2, C2.2.1, C2.2.2\}$. Conversely, **x** is classified as $C2.2.2$ if the classifiers $\{C2, C2.2, C2.2.2\}$ all issue the positive label for **x**. *HR*-SVM also addresses the imbalanced nature of the training sets from which the "node" classifiers are induced, and it attempts to deal with the fact that errors committed by higher-level classifiers are propagated down the hierarchy.

As indicated in Fig. 4, *HR*-SVM consists of four modules; three of them for data pre-processing and the last for induction from imbalanced data.

5.1. Exclusive-Parent Training Policy (EPT)

For individual-node class induction, the first step is the generation of the corresponding (binary) training set. Several methods have been proposed in the literature so far [11–13,39]. The one we use in

[4] http://lshtc.iit.demokritos.gr/.

[5] Since this paper considers only macro-average, we will ignore "Macro (Ma)."

AUTHOR COPY

Table 7
Comparing the training sets generated by EPT and EAT

Class	The number of positive examples versus the number of all training examples	
	EPT	EAT
$C2$	100/1000	100/1000
$C2.2$	50/100	50/1000
$C2.2.2$	10/50	10/1000

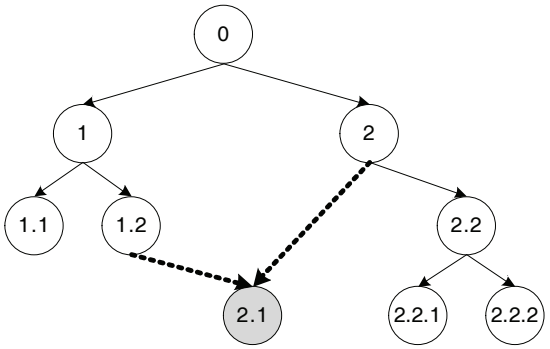

Fig. 3. A class hierarchy used to explain the *HR-SVM*'s framework.

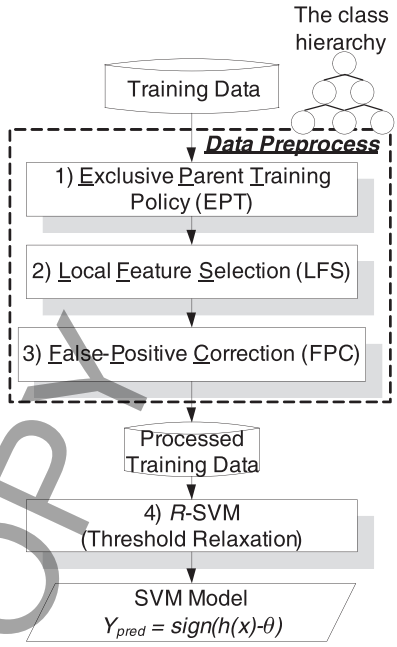

Fig. 4. *HR*-SVM's general architecture.

HR-SVM is referred to as EPT (Exclusive Parent Training Policy).[6] Let us now describe it in detail as follows.

Let Tr be the set of all training examples, let $Tr(C_i)$ denote the set of training examples used for the induction of class C_i, and let $Tr^+(C_i)$, and $Tr^-(C_i)$ denote the sets of the positive and negative training examples of C_i, respectively. $|C_i|$ denotes the number of examples representing C_i, and $\uparrow C_i$ is the set of the parents of C_i. Finally, "\" is the set exclusion operator.

HR-SVM's way of choosing the training examples for the induction of C_i is defined as follows:

$$\begin{aligned} Tr(C_i) &= Tr^+(\uparrow(C_i)) \\ Tr^-(C_i) &= Tr(C_i) \backslash Tr^+(C_i) \end{aligned} \tag{5}$$

HR-SVM thus includes in this training set all positive examples of C_i's parent class(es).

Let us illustrate the process by two examples.

- Example 1: A one-parent node (e.g., $C2.2$):
  - $Tr(C2.2) = Tr(\uparrow(C2.2)) = Tr^+(C2)$.
  - $Tr^-(C2.2) = Tr(C2.2) \backslash Tr^+(C2.2)$,
    (Note that $Tr^-(C2.2) \neq Tr^+(C2.1)$ because if this is an NMLNP problem, some examples in $C2$ may belong to neither $C2.1$ nor $C2.2$.)
- Example 2: A multiple-parents node (e.g., $C2.1$):
  - $Tr(C2.1) = Tr^+(\uparrow(C2.1)) = (Tr^+(C1.2) \bigcup Tr^+(C2))$
  - $Tr^-(C2.1) = Tr(C2.1) \backslash Tr^+(C2.1)$.

[6] EPT is similar to the "siblings policy" in [12,13] which was shown to be the best method to create training sets.

AUTHOR COPY

The advantages of EPT are best illustrated by the comparison with another policy that has been used in the past, namely EAT (Exclusive All Training Policy) [12,13][7] where each node classifier is trained using the entire training set: $Tr(C_i) = Tr$, and $Tr^-(C_i) = Tr \backslash Tr^+(C_i)$. Assuming that the root node represents $|C0| = |Tr| = 1000$ examples, we have $|C2| = 100$ examples, $|C2.2| = 50$ examples, and $|C2.2.2| = 10$ examples. The sizes of the training sets generated by EPT and EAT, respectively, are given in Table 7. The reader can see that, at the lower-level classes, EPT generates smaller and more balanced training sets than EAT. For instance, when inducing $C2.2$, EPT creates a training set of size 100, whereas EAT uses 1000 examples.

Note also that it is unnecessary to train the classifier $C2.2$ by $C1$'s examples because $C2.2$'s parent classifier, $C2$, has the responsibility to remove those $C1$'s examples.

5.2. Local Feature Selection (LFS)

In our domains, the examples are often described by thousands of attributes, which can lead not only to prohibitive induction costs, but also to performance degradation if many of the attributes are irrelevant. Importantly, the relevance of the individual attributes can vary from class to class. Many scientists have studied attribute-selection techniques (see, e.g., [15]). Choosing from these, we need one that is computationally efficient, and we need to apply it separately to each class.

In our previous work [9,46], we made a good experience with ordering the attributes by their *gain ratio*,[8] and then selecting a certain percentage of the highest ones. This means that the number of selected attributes is *fixed* and *equal* in every class. Experiments indicated that the best performance of multi-label classifiers was in our domains obtained when only the top 25% of attributes are retained. However, this strategy cannot be directly applied to the HMC domain from that preliminary experiment. Moreover, in terms of computational cost, the number of classes in the HMC domain can be much higher than in the multi-label domain, so it is more efficient to allow that the number of attributes be different for each class node.

In *HR*-SVM, we improved this (rather simplistic) approach by choosing those attributes that satisfy the following two conditions: the minimum accumulated gain ratio (or $G\%$ of total gain ratio) and the minimum number of attributes (or $P\%$ of the total number of attributes).

For illustration, suppose the user has set $G = 95\%$ and $P = 25\%$; and let the gain ratios of a given set of ten attributes be as follows: $\{0.40, 0.30, 0.10, 0.10, 0.05, 0.01, 0.01, 0.01, 0.01, 0.01\}$ (the sum of total gain ratios is 1.0). Under these conditions, the first five attributes will be chosen because their sum of gain ratios is 0.95 (which satisfies the requirement of reaching at least 95% of 1.0), and five attributes represent more than the required 25% of the total 10 attributes.

5.3. False-Positive Correction (FPC)

HR-SVM's next module seeks to correct the false positives (FP) – negative examples that are incorrectly labeled by the classifier as positive. False positives are responsible for what is known as *incorrect-path errors*.

Let us illustrate. The EPT policy ensures that classifier $C2.2$ is induced from examples belonging to $C2.2$'s parent class, $C2$, and these do not include any examples of $C1$. Suppose a testing example of $C1$

[7] [12,13], who compare various training policies, EAT is called "ALL" policy.

[8] In our case, we relied on the formula employed by Quinlan's C4.5 [32]. The fact that its implementation is available in his publicly available software facilitates the replicability of our experiments.

AUTHOR COPY

is incorrectly labeled by the $C2$-classifier as positive. This error is propagated to classifier $C2.2$, which has never been trained using $C1$'s examples, and may therefore fail. This error is then passed to $C2.2$'s subclasses, $C2.2.1$ and $C2.2.2$, which are likely to make the same mistake as $C2.2$. In this sense the FP errors from $C2$ negatively affect the performance of $C2$'s subclasses.

HR-SVM addresses this issue by our *False-Positive Correction* strategy (FPC). The idea is to add to C_i's negative training examples, $Tr^-(C_i)$, also a set of FP examples at C_i's superclass(es) (denoted in Eq. (6) by $FP(\uparrow(C_i))$), to give it a chance to learn how to correct the FP's inherited from the parents. Note that FPC cannot be applied to classifiers at the top level; the FPC process begins after the SVM models of all C_i's parents have been induced. These classifiers are then tested on their own training data, and the results are used to identify the FP examples. Finally, a subset of these examples is added to the set of negative training examples at C_i, $Tr^-(C_i)$.

$$Tr^-(C_i) = Tr(C_i) \backslash Tr^+(C_i) + \mathbf{FP}(\uparrow(\mathbf{C_i})) \tag{6}$$

For instance, the potential errors propagated from $C2.2$'s parents, $FP(\uparrow(C2.2)) = FP(C2)$, are added to $C2.2$'s training set. Note that $C2.1$ has multiple parents. The extra training examples added by FPC is determined as $FP(\uparrow(C2.1)) = FP(C1.2) \bigcup FP(2)$.

We expect that the FPC strategy can potentially decrease the number of false positives, thus improving *precision* as well as F_1.

5.4. R-SVM

The reader will recall that our system induces a separate binary classifier for each class. To induce the classifier for the i-th class, the system creates a training set where each example is labeled as positive if it belongs to the i-th class and as negative if it does not belong to the i-th class. Most of the time, each class is represented by only a small subset of the examples, which means that the corresponding training set is imbalanced. Traditional induction algorithms are in similar situations known to be biased towards the majority class, which results in many false negatives (FN).

In HMC-domains, the issue becomes even more serious due to the phenomenon called *blocking*: for instance, those of $C2.2.2$'s examples that have been misclassified by $C2$-classifier as negative will not be recognized as belonging to classes $C2.2$ and $C2.2.2$. In *HR*-SVM, we mitigate the problem by using a mechanism borrowed from our earlier work [45]. Let us briefly summarize the essence.

SVM induces a hyperplane, $h(\vec{x}) = \vec{w} \cdot \vec{x} + b = 0$, whose orientation is determined by $\vec{w}$ and offset by b. The expression can be used to calculate for each example, $\vec{x}$, its "SVM score":

$$s_i = h(\vec{x_i}) = \vec{w} \cdot \vec{x_i} + b \tag{7}$$

In domains where one class outnumbers the other, SVM's bias toward low error rate may result in high *precision*, but very low *recall* (F_1 is then low too). This is rectified by *threshold adjustment* [5,14, 17,25,30,37,42,50], a process that translates the hyperplane (by the modification of b without changing $\vec{w}$) as depicted in Fig. 5. The task is to find, in the set of all possible thresholds, Θ, the one that gives the highest value of a user-defined performance criterion, *perf* (e.g., the F_1 metric), over the set of training data mapped to SVM scores, $L = \{(s_1, C_1), \ldots, (s_n, C_n)\}$:

$$\{\theta \in \Theta | \theta = \max(\textit{perf}(L, \Theta))\} \tag{8}$$

AUTHOR COPY

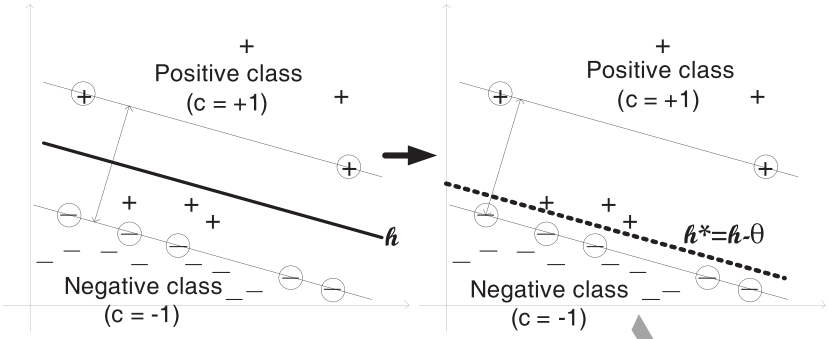

Fig. 5. SVM hyperplanes before (left) and after (right) a threshold adjustment that corrects the classification of three examples.

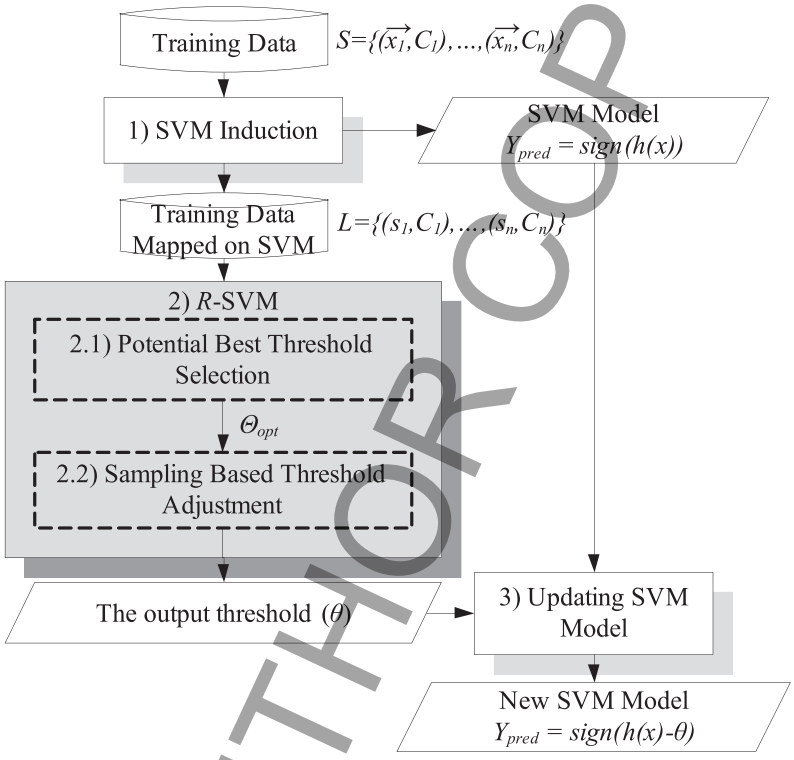

Fig. 6. The *R*-SVM based framework.

R-SVM is a threshold-adjustment algorithm shown by [45] to compare favorably with some earlier attempts, such as SVM_{F_1} [5], SVM_{CV} [5], *ScutFBR* [24], and *BetaGamma* [30,37]. The essence is outlined in Fig. 6. After inducing the initial SVM model, the first task is to identify a set of candidate thresholds, Θ. To maintain reasonable computational costs, the system orders the examples by their scores (Eq. (7)) and then defines candidate thresholds as the middle points between those pairs of neighboring examples that differ in their class labels: $\Theta_{opt} = \{\theta_1, \ldots, \theta_K\}$.

The best threshold from the candidate thresholds is found as follows. First, M auxiliary training subsets, $\{L_1, \ldots, L_M\}$, are created. Next, for each L_i, the best threshold, θ_i, from the set of candidate thresholds, Θ_{opt}, is identified. Finally, the output threshold is obtained as the average of these "winners."

AUTHOR COPY

5.5. Complexity analysis

[52] presented a complexity analysis of an SVM-based algorithm in hierarchical as well as non-hierarchical domains. Since *HR*-SVM uses a threshold-adjusted SVM (*R*-SVM) as its baseline classifier, its complexity can be derived using the same kind of analysis.

Let M be the number of classes, let N be the number of training examples, let V be the number of attributes, and let L_v be the average number of non-zero attributes. In multi-label (non-hierarchical) classification, the training time of the traditional SVM is $O(MN^c)$ (where $c \approx 1.2 \sim 1.5$ is a domain-specific constant), and its testing time is $O(ML_v)$.

The training time of *R*-SVM is given by Eq. (9), where c_1 and c_2 are constant times for Box 2.1 and Box 2.2 from Fig. 6, respectively. Since the process of threshold adjustment is applied after the model induction, the first term in the equation is the SVM induction time, and the second term is the evaluation time of the SVM model on training data.

$$\begin{aligned} \text{Training Time} &= O(MN^c) + O(ML_v) + c_1 + c_2 \\ &= O(M(N^c + L_v)) + c_1 + c_2 \end{aligned} \tag{9}$$

For the hierarchical classification system, the total complexity of the top-down approach, including *HR*-SVM, is given by Eq. (10), where h is the depth of the hierarchy, b is the number of branches at the leaf nodes, m_i is the number of classes at the i-th level, $i = \{0, \ldots, h\}$ is an index for the hierarchical level, $j = \{1, \ldots, m_i\}$ is an index for the class at the i-th level, n_{ij} is the number of local training examples, N_i is the total number of training examples at the i-th level, $N_0 \leqslant N_i$, and π_{ij} is defined as $\frac{n_{ij}}{N_i}$.

$$b \times O(N_0^c) \sum_{h-1}^{i=0} \sum_{m_i}^{j=1} \pi_{ij}^c \tag{10}$$

6. Experimental data

We experimented with several real-world databases from the field of functional genomics available from the DTAI webpage[9] [35]. The task is to predict gene functions of three organisms: *Saccharomyces cerevisiae (S. cerevisiae), Arabidopsis thaliana (A. thaliana),* and *Mus musculus (M. musculus).* In this paper, we worked with 8 data sets annotated by the functional hierarchy in Gene Ontology (GO) whose structure forms a DAG. Each data set is described by different aspects (attributes) of the genes that may originate at diverse sources. The characteristics of the experimental data sets are summarized in Table 8.

Since the data contained nominal attributes, and many values were missing, some pre-processing was necessary.

1. *Data cleaning*. Assuming that rare classes cannot be reliably induced, we ignored all classes represented by less than 1% of the total number of examples, which is 50 examples on all data sets, save the very large domain $D19$ where this minimum was set to 200 examples. We deleted examples with missing attribute values, and we also removed attributes that were never used in the "surviving" examples.

[9] http://dtai.cs.kuleuven.be/clus/hmcdatasets/.

AUTHOR COPY

Table 8
Properties of the data sets used in our experiments: The numbers of examples $|D|$, attributes $|A|$, classes $|C|$, and hierarchical levels $|H|$. "M?" indicates whether a data set includes missing values – yes (Y) or no (N)

Organism	Id	Feature description	$\|D\|$	$\|A\|$	$\|C\|$	$\|H\|$	M?
S. cerevisiae	D0	Joining all sources	3465	5931	132	7	N
A. thaliana	$D13$	Sequence statistics (seq)	11763	4451	629	6	Y
	$D14$	Affy.'s experiments (exprindiv)	10840	1252	626	6	Y
	$D15$	SCOP superfamily (scop)	9843	2004	571	6	N
	$D16$	Secondary structure (struc)	11763	14805	629	6	N
	$D17$	InterProScan (interpro)	11763	2816	629	6	N
	$D18$	All microarray (expr)	11121	72870	622	6	N
M. musculus	D19	Joining all sources	21153	18748	5620	13	Y

2. *Missing-value imputation*. In the case of nominal attributes, we replaced a missing value with the most common value. In the case of continuous attributes, we used the average (mean).
3. *Nominal-to-numerical conversion*. Some attributes were nominal, acquiring one out of m different values. We converted each of these nominal attributes to a set of m binary attributes. For instance, an attribute with three values, $\{A, B, C\}$, was replaced with the triplet of binary attributes whose possible combinations of values were limited to (0,0,1), (0,1,0), and (1,0,0).
4. *Attribute transformation*. Some attributes were constrained to very small ranges – for instance, the $D15$ were limited to the interval [3.8E-123, 5.6E-108]. In this case, we replaced the values with their logarithm (base 10).
5. *Normalizing the attribute values*. We normalized the values of numeric attributes to the interval [0,1].

Each domain provided by the DTAI website consists of 3 files that were originally intended for training, validation, and testing, respectively. To facilitate the evaluation of statistical significance of performance comparisons, we merged these three files into one, and then used 5-fold cross-validation.

7. Experiments

HR-SVM induces a hierarchical classifier by a mechanism built around the publicly available. *svm-light*[10] As for the kernel function, preliminary experiments indicated that the linear function gave better results (in terms of F_1) than the radial basis function (RBF). Perhaps this was to be expected: the training examples being described by thousands of attributes, the individual classes were easy to separate from each other.

The task of the experiments reported below is twofold. First, we want to show that our system outperforms earlier attempts. Since it is clearly not possible to compare *HR*-SVM with every single other system, we chose two systems that are regarded by the relevant literature as perhaps the most powerful: *H*-SVM and Clus-HMC.[11]

Our second goal is to find out how each of *HR*-SVM's modules (see Fig. 4) contributes to the overall performance. To this end, we added these modules one by one, obtaining the four variants listed in Table 9 where *H*-SVM is a hierarchical version of the traditional SVM [12,13], and *HR*-SVM-ALL is the complete system. The only exception was domain $D18$ where only *H*-SVM-ALL could be experimented

[10] http://svmlight.joachims.org/.
[11] The package is available at http://dtai.cs.kuleuven.be/clus/.

AUTHOR COPY

Table 9
Evolution of *HR*-SVM's framework

System	Descriptions
H-SVM	**EPT** + The baseline SVM
HR-SVM	EPT + ***R*-SVM**
HR-SVM-**FPC**	EPT + *R*-SVM + **FPC**
HR-SVM-**ALL**	EPT + *R*-SVM + FPC + **LFS**; (all modules)

Table 10
The total induction time (in seconds) of *HR*-SVM-ALL and two other systems, *H*-SVM and Clus-HMC

Data set	*HR*-SVM-ALL	*H*-SVM	Clus-HMC
D0	7.68	10.26	31.04
D13	204.68	357.75	999.27
D14	3,615.49	1,900.42	208.90
D15	13,641.90	17,408.56	159.44
D16	406.99	526.25	1,164.25
D17	141.36	100.76	146.05
D18	2,989.98	N/A	604.06
D19	17,691.80	23,334.07	3,396.86

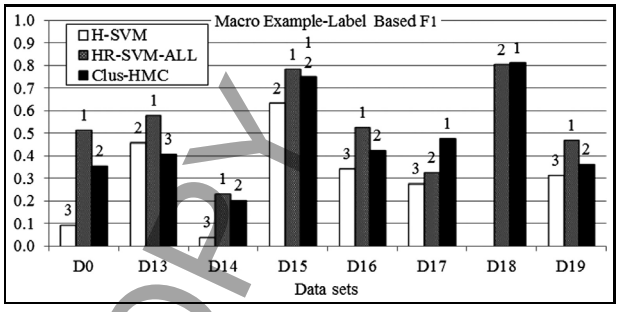

Fig. 7. Comparing the performance (along F_1) of *H*-SVM, *HR*-SVM-ALL, and Clus-HMC. The integers above the vertical bars give the ranks obtained by the ANOVA methodology followed by Bonferroni multiple comparisons [8].

with – the high number of attributes made it impossible to use SVM-based systems without attribute selection (LFS).

All results are expressed in terms of the example-label based macro-averaging (ELbMa) version of the performance criteria from Section 4.3.

7.1. Comparing our system's performance with that of its predecessors

A new technique is deemed useful if its performance compares favorably with that of other techniques. This is why we compare here *HR*-SVM-ALL with *H*-SVM (a hierarchical variant of SVM) and Clus-HMC (a global approach that relies on decision trees). Both are regarded as perhaps the best existing HMC-induction systems.

The results in terms of F_1 are summarized in Fig. 7, and the CPU times are given in Table 10. The reader can see that, in terms of F_1, *HR*-SVM-ALL outperforms *H*-SVM in all domains, the results being particularly impressive in $D0$; *H*-SVM achieved only $F_1 = 0.0929$, whereas *HR*-SVM-ALL scored $F_1 = 0.4811$ (an improvement of more than 400%). The new technique induced faster than *H*-SVM on all data sets except $D14$ and $D17$ – in these, the efficiency of the employed attribute-selection technique was impaired by the additional time needed to process the FP data sets (FPC).

When compared with Clus-HMC along F_1, *HR*-SVM-ALL was significantly better in six out of eight domains. The explanation for the unfavorable F_1-results of *HR*-SVM-ALL in $D17$ and $D18$ is that both domains are described exclusively by categorical attributes with values from $\{0, 1\}$. The input data of SVM must be numeric, while that of the decision tree can be either numeric or categorical. This means that this kind of data ($D17$ and $D18$) is more suitable for decision trees than for SVM. Note that all attributes of $D16$ attributes, too, are categorical with values of $\{0, 1\}$, but the difference between the F_1-result of *H*-SVM and Clus-HMC in $D16$ is not so conspicuous – the mechanisms proposed in *HR*-SVM-ALL can overcome the drawback of the traditional SVM on this kind of data. The induction

AUTHOR COPY

Table 11
Hierarchical systems (see also Table 9) whose performance is to be compared

Module	System comparison
1. *R*-SVM	*HR*-SVM vs. *H*-SVM
2. FPC	*HR*-SVM vs. *HR*-SVM-FPC
3. LFS	*HR*-SVM-ALL vs. *HR*-SVM-FPC

Table 12
Induction time (in seconds) of the *top-down* hierarchical classification systems from Table 9. In each column, the percentages in the parentheses give the time increase over the previous column

Data set	*H*-SVM	*HR*-SVM	*HR*-SVM-FPC	*HR*-SVM-ALL
D0	10.26	10.33 (+ <1%)	10.95 (+6%)	7.68 (−30%)
D13	357.75	358.24 (+ <1%)	393.01 (+10%)	204.68 (−48%)
D14	1,900.42	1,901.23 (+ <1%)	4,656.21 (+145%)	3,615.49 (−22%)
D15	17,408.56	17,408.77 (+ <1%)	12,002.88 (−31%)	13,641.90 (+14%)
D16	526.25	527.11 (+ <1%)	623.81 (+18%)	406.99 (−35%)
D17	100.76	101.11 (+ <1%)	123.15 (+22%)	141.36 (+15%)
D18	N/A	N/A	N/A	2,989.98
D19	23,334.07	23,336.36 (+ <1%)	40,320.39 (+73%)	17,691.80 (−56%)

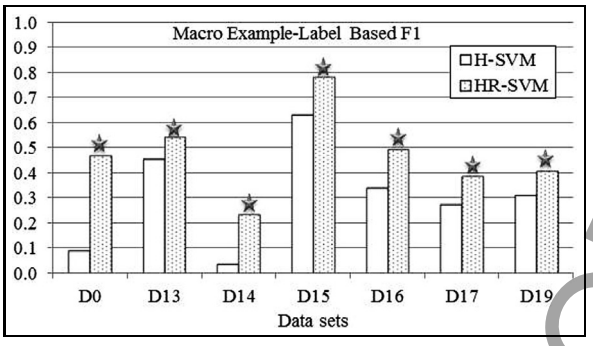

Fig. 8. Comparing *HR*-SVM with *H*-SVM in terms of F_1. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

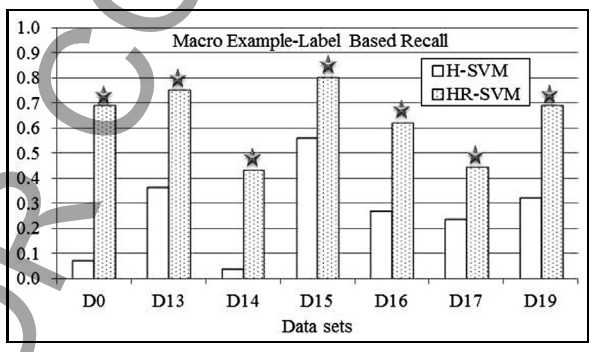

Fig. 9. Comparing *HR*-SVM with *H*-SVM in terms of *recall*. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

time is comparable, *HR*-SVM-ALL being faster than Clus-HMC in $D0$, $D13$, $D16$, and $D17$, and slower in $D14$, $D15$, $D18$, and $D19$.

7.2. Contributions of HR-SVM's modules

In the next step, we wanted to find out how much each of the modules listed in Fig. 4 contributed to *HR*-SVM's classification performance (see Table 11).

The induction times of the individual consecutive steps are summarized in Table 12. The following subsections discuss the classification-performance aspects.

7.2.1. The effect of R-SVM

In *HR*-SVM, SVM was replaced by *R*-SVM, with the intention to reduce SVM's bias to the majority class (recall that our experimental domains were dominated by false negatives). The hypothesis that *R*-SVM can help, here is tested by the comparison of *HR*-SVM's performance with that of *H*-SVM.

AUTHOR COPY

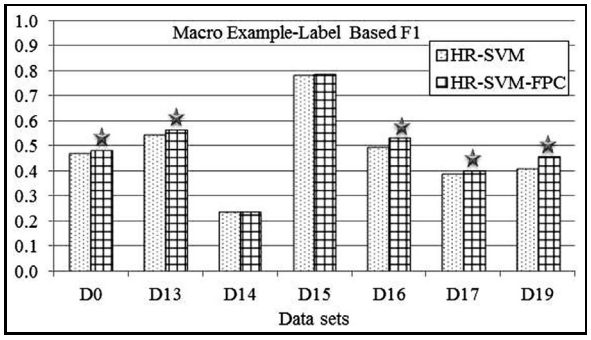

Fig. 10. Comparing *HR*-SVM-FPC with *HR*-SVM in terms of F_1. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

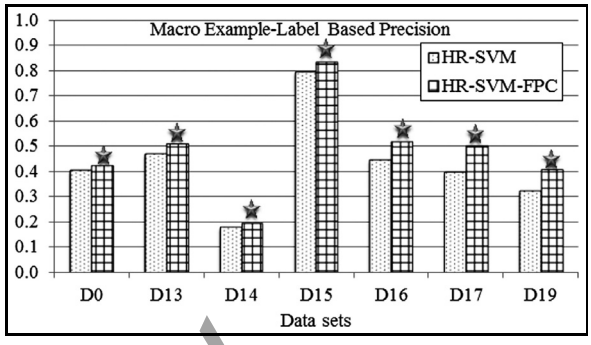

Fig. 11. Comparing *HR*-SVM-FPC with *HR*-SVM in terms of precision. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

The results in Fig. 8 show that *HR*-SVM exhibited better F_1 than *H*-SVM in all domains. In $D0$, *HR*-SVM's F_1 improvement over the baseline SVM was more than 400% (from 0.0929 to 0.4693). As for the computational costs (Table 12), both systems needed about the same time. This indicates that the additional thresholding time in *HR*-SVM is very small (only at most 1% of the SVM induction time). Especially in $D19$, our system needed only 2.29 seconds to adjust the separation hyperplane in addition to the induction time of the traditional SVM, which was 23,334.07 seconds.

HR-SVM's classification success was largely due to its significant improvement of *recall* (see Fig. 9), which was achieved at the cost of slightly reduced *precision* on some data sets, though the average of *precision* on all data sets still increased by 25%. It turns out that *R*-SVM properly adjusts the class-separation hyperplane, thus reducing the bias to the majority class, which in turn reduces the number of false negatives propagated down the class hierarchy.

7.2.2. The effect of FPC

Let us now compare *HR*-SVM-FPC with *HR*-SVM, thus finding out how the system's performance benefits from the False-Positive Correction. Recall that false positives that occur at higher levels are propagated to lower levels in the class hierarchy. The total number of false positives can be high, which means lower *precision* and F_1.

The summary of the experimental results in Fig. 10 indicates that *HR*-SVM-FPC outperforms *HR*-SVM in almost all domains. In 5 out of 7 domains, the improvement is statistically significant. As seen in Fig. 11, this is due to FPC's ability to increased *precision* (in all eight domains). This indicates that the number of FP-errors propagated throughout the system was indeed reduced.

Table 12 shows how much FPC increases the computational costs (on account of the extra false positives added to the training data). Note that, in $D15$, FPC led to reduced induction time because the SVM model here converged faster in spite of the training data being larger (a phenomenon explained by [38]).

7.2.3. The effect of LFS

The results of the comparisons between *HR*-SVM-ALL and *HR*-SVM-FPC give us an idea of the contribution of the attribute-selection technique (LFS).

Recall that LFS relies on two user-defined parameters: (i) the minimum percentage of accumulated *gain ratio* and (ii) the minimum number of attributes. In the experiments reported below, the former threshold is set to 95% of the overall gain ratio, following the general threshold used in Principal Component Analysis (PCA) [20]. The latter threshold is set to 25% of the total number of attributes as suggested in [9,46].

AUTHOR COPY

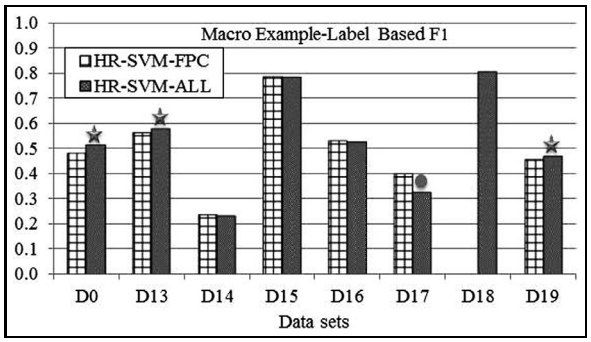

Fig. 12. Comparing *HR*-SVM-ALL with *HR*-SVM-FPC in terms of F_1. The stars and circles above some of the bars indicate significant improvements and declines according to t-tests (0.05 level).

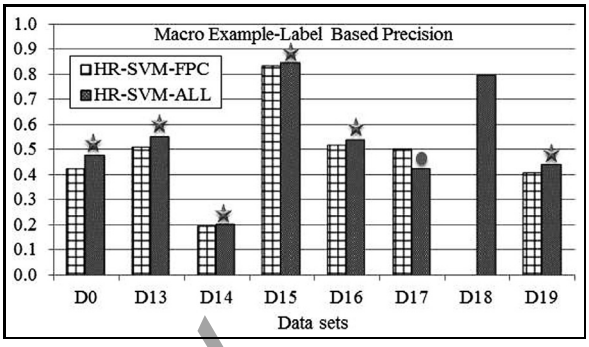

Fig. 13. Comparing *HR*-SVM-ALL with *HR*-SVM-FPC in terms of *precision*. The stars and circles above some of the bars indicate significant improvements and decline according to t-tests (0.05 level).

Comparing to *HR*-SVM-FPC in terms of F_1, Fig. 12 shows that *HR*-SVM-ALL won in $D0$, $D13$, and $D19$, but lost in $D17$. The results of the remaining data sets ($D14$, $D15$, and $D16$) are statistically indistinguishable. Figure 13 shows that the removal of less relevant attributes indeed improved *precision* on all data sets except for $D17$.

Table 12 shows that LFS reduced the training time in most domains, most remarkably in $D19$ where the time was reduced by 56% while F_1 also improved by 2%. However, *HR*-SVM-ALL's induction costs increased in $D15$ and $D17$. This is caused by SVM's *inverse convergence* [38]. LFS is especially useful in $D18$ where the number of attributes is so high that memory-resident algorithms cannot load the whole data set into the computer's memory.[12]

In conclusion, attribute selection seems beneficial, especially in our experimental data sets where examples are described by great many attributes. LFS increases F_1 by improving *precision*; it also reduces computational costs.

7.3. Performance at different hierarchical levels

Let us now try to gain more insight by comparing the classification performance of *HR*-SVM-ALL, *H*-SVM, and Clus-HMC, at different levels of the class hierarchies.

Note that the label-based metrics (*Lb*) are here better suited than the example-label based ones (*ELb*). The *ELb*-metrics evaluate the performance along the *class paths*, such as $C1 \rightarrow C1.1 \rightarrow C1.1.1$, which does not make much sense when evaluating each class-level independently.

Three factors are known to affect an induced classifier's performance: (i) the training set size, (ii) the number of positive examples, and (iii) the data distribution (e.g., the degree of imbalance). Hierarchical domains add one more factor: the "propagated error" – the accuracy of lower-level classifiers is related to that of the parent classifiers.

Table 13 summarizes the expected tendencies. Note that the accuracies of higher-level classifiers suffer from highly imbalanced training data, and the accuracies of lower-level classifiers suffer from the scarcity of positive training examples and from the propagated errors. As we go down the hierarchy, the impact of these factors increases and the performance of the lower-level classifiers declines.

[12] In the Java implementation of *HR*-SVM, the attribute value type is a "double" (8 bytes). Therefore, at least 6.5 GB of memory in the 32-bit system is needed to handle $D18$. On the other hand, Clus-HMC treats each value as an "integer" (4 bytes), thus needing less memory space (3.2 GB).

AUTHOR COPY

Table 13
Expected impact of four aspects: (i) the training-set size, (ii) the number positive examples, (iii) the degree of imbalance, and (iv) the downward-propagated errors. "$>$" indicates an increase and "$<$" indicates a decrease

Classes	#Training	#Positive	Imb. ratio	Prop. errors
Upper levels	$>$	$>$	$>$	$<$
Lower levels	$<$	$<$	$<$	$>$

Table 14
Statistics for $D0$'s hierarchical levels

Level	#Classes	AvgPositive	AvgTotal	%AvgPositive
1	23	188.39	2,834.00	*0.07*
2	18	148.94	333.56	0.52
3	13	111.92	313.46	0.40
4	7	72.57	180.00	0.53
5	3	71.33	108.33	0.69

Table 15
The number of classifiers with $F_1 = 0$ in $D0$

Level	#Classes	*H*-SVM	*HR*-SVM-ALL	Clus-HMC
1	23	14	0	4
2	18	9	0	1
3	13	5	0	2
4	7	2	0	0
5	3	2	0	0

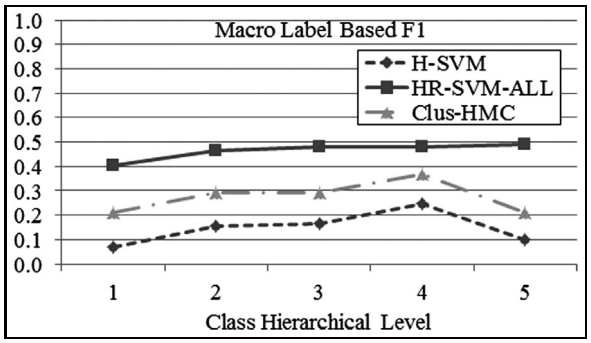

Fig. 14. Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in $D0$.

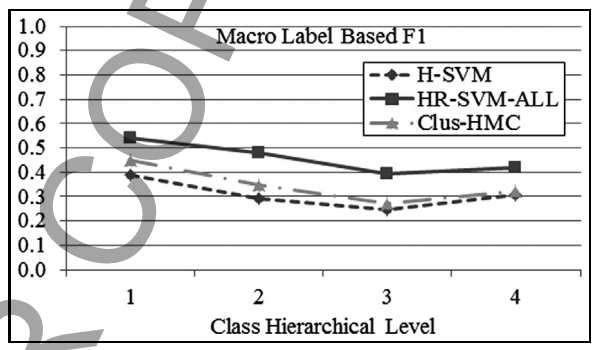

Fig. 15. Comparing F_1 of *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different class hierarchical levels in $D16$.

Let us first take a closer look at one of the domains: $D0$. Figure 14 plots the classification performance at the different levels. We can see that *HR*-SVM-ALL outperforms *H*-SVM and Clus-HMC at all levels, and *H*-SVM always lags behind the other two. As seen in Table 14, the performance at the highest level is negatively affected by the highly imbalanced class representation. The percentage of positive examples at this level is only 0.07%.

Even more insight is gained from Table 15 which gives the number of classifiers that incorrectly label *all* examples as negative (which means that $F_1 = 0$). Since the traditional SVM suffers from imbalanced class representation, most classifiers in *H*-SVM have $F_1 = 0$. However, *R*-SVM was designed in a way that improves its "reaction" to imbalanced classes, and this is why there are no classifiers with $F_1 = 0$ in the case of *HR*-SVM-ALL. On the other hand, Clus-HMC fails to recognize examples of some classes (resulting in $F_1 = 0$).

In all other domains (apart from $D17$ and $D18$), the F_1-results were quite similar to those shown in Fig. 15 for $D16$. As we proceed to lower levels, F_1 gradually decreases. This is caused by the decreasing numbers of training examples (Table 16) and by the errors propagated from higher levels (Table 17). F_1 at the fourth hierarchical level is slightly better than that at the third level because its ratio of imbalance is lower. Comparing the three hierarchical classification systems, we see that *HR*-SVM-ALL exhibits the best classification performance, whereas *H*-SVM is the worst.

In $D17$ and $D18$, Clus-HMC showed the best F_1-results at all levels due to their data characteristic as mentioned in Section 7.1.

AUTHOR COPY

Table 16
Statistics for $D16$'s hierarchical levels

Level	#Classes	AvgPositive	AvgTotal	%AvgPositive
1	11	1093.27	9548.00	0.01
2	28	391.00	2814.82	0.21
3	41	188.22	795.83	0.34
4	34	162.68	318.71	0.56

Table 17
The number of classifiers with $F_1 = 0$ in $D16$

Level	#Classes	*H*-SVM	*HR*-SVM-ALL	Clus-HMC
1	11	5	0	0
2	28	11	2	0
3	41	17	3	9
4	34	9	2	4

Based on all these results, we conclude that *HR*-SVM-ALL handles the imbalanced training sets particularly well at the upper levels of the class hierarchy where it clearly fares better than other systems. Our system also takes better care of the propagated errors at the lower levels, which (usually) prevents the creation of classifiers that fail to correctly predict any positive examples ($F_1 = 0$).

8. Conclusion

The paper presented *HR*-SVM, a new top-down induction technique for multi-label classifiers in domains with hierarchically organized classes. In designing it, we followed the common strategy that induces a separate binary classifier for each node in the hierarchy, and employs higher-level classifiers when creating the training sets for the induction of lower-level classifiers. The paper described our system, and then reported experiments illustrating its performance as well as diverse aspects of its overall behavior.

Top-down approaches often suffer from two problems: imbalanced class representation and top-down error propagation. *HR*-SVM addresses them explicitly by the following four techniques: (i) exclusive-parent training sets (EPT), (ii) attribute-selection module (LFS), (iii) a mechanism to correct false positives (FPC), and (iv) the correction of the majority-class bias. The first three can be regarded as data pre-processing techniques that help generate smaller and not-so-imbalanced training sets, "enriched" by examples misclassified by parent classifiers. The task of the last module, *R*-SVM, is to induce unbiased SVM-hyperplanes.

Importantly, we argue that, in hierarchical classification, the usual metrics for performance evaluation (*precision, recall*, and F_1) are not ideal. By way of rectification, we introduced a new measure, "example-label based macro-averaging," that uses the harmonic mean of macro-averaging performances in two dimensions: *per example* and *per class*.

We applied our system to eight real-world domains from the the field of gene-function prediction of three organisms: *S. cerevisiae, A. thaliana,* and *M. musculus*. These data are available at the DTAI website. In each domain, the data were annotated using their own functional hierarchy in Gene Ontology (GO), whose structure is a directed acyclic graph (DAG).

Experimenting with these domains, we observed that our system significantly outperformed alternative hierarchical-classification techniques such as *H*-SVM (a top-down hierarchical version of SVM) and Clus-HMC (a "global" approach based on decision trees). Especially in $D0$, the F_1-improvement of *HR*-SVM over *H*-SVM is about 400%, while the induction costs are much lower. The explanation is that the module EPT creates training sets that are less imbalanced, the module *R*-SVM reduces the number of false negatives (which resulted in better *recall*), the module FPC decreases the number of false positives (which leads to higher *precision*), and the module LFS removes irrelevant attributes (which saves a lot of induction time).

AUTHOR COPY

References

[1] N. Alaydie, C.K. Reddy and F. Fotouhi, Exploiting label dependency for hierarchical multi-label classification, in: *Proceedings of the 16th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining – Volume Part I* PAKDD'12, Berlin, Heidelberg, Springer-Verlag, (2012), 294–305.

[2] W. Bi and J. Kwok, Multi-label classification on tree- and dag-structured hierarchies, in: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, L. Getoor and T. Scheffer, eds, ICML '11, New York, NY, USA, ACM, (June 2011), 17–24.

[3] H. Blockeel, L. Schietgat, J. Struyf, S. Dzeroski and A. Clare, Decision trees for hierarchical multilabel classification: A case study in functional genomics, *Knowledge Discovery in Databases: Pkdd 2006, Proceedings* **4213** (2006), 18–29.

[4] H. Blockeel, L. De Raedt and J. Ramon, Top-down induction of clustering trees, in: *Proceedings of the 15th International Conference on Machine Learning*, J. Shavlik, ed., Morgan Kaufmann, (1998), 55–63.

[5] J. Brank, M. Grobelnik, N. Milic-Frayling and D. Mladenic, Training text classifiers with svm on very few positive examples, Technical Report, Microsoft Research, 2003.

[6] A. Clare and R.D. King, Predicting gene function in saccharomyces cerevisiae, *Bioinformatics* **19** (2003), 42–49.

[7] E.P. Costa, A.C. Lorena, Carvalho and A.A. Freitas, A review of performance evaluation measures for hierarchical classifiers, in: *2007 AAAI Workshop, Vancouver* AAAI Press, (2007).

[8] L.D. Delwiche and S.J. Slaughter, *The Little SAS Book: A Primer*, SAS Press, fourth edition, July 2008.

[9] S. Dendamrongvit, P. Vateekul and M. Kubat, Irrelevant attributes and imbalanced classes in multi-label text-categorization domains, *Intelligent Data Analysis* **15**(6) (2011), 843–859.

[10] S. Dumais and H. Chen, Hierarchical classification of web content, in: *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* ACM, (2000), 256–263.

[11] R. Eisner, B. Poulin, D. Szafron, P. Lu and R. Greiner, Improving protein function prediction using the hierarchical structure of the gene ontology, in: *Proc IEEE CIBCB* (2005).

[12] T. Fagni and F. Sebastiani, On the selection of negative examples for hierarchical text categorization, in: *Proceedings of the 3rd Language Technology Conference* (2007), 24–28.

[13] T. Fagni and F. Sebastiani, Selecting negative examples for hierarchical text classification: An experimental comparison, *Journal of the American Society for Information Science and Technology* **61**(11) (2010), 2256–2265.

[14] B. Goertzel and J. Venuto, Accurate svm text classification for highly skewed data using threshold tuning and query-expansion-based feature selection, in: *IJCNN'06: International Joint Conference on Neural Networks 2006* (2006), 1220–1225.

[15] I. Guyon and A. Elisseeff, An introduction to variable and feature selection, *J Mach Learn Res* **3** (2003), 1157–1182, Good introduction and coverage of large spetrum of feature selection related problems and state-of-the-art.

[16] W. Hersh, C. Buckley, T.J. Leone and D. Hickam, OHSUMED: An interactive retrieval evaluation and new large test collection for research, in: *SIGIR '94: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* Springer-Verlag, New York, Inc., 192–201, (1994).

[17] T. Imam, K. Ting and J. Kamruzzaman, z-SVM: An svm for improved classification of imbalanced data, *AI 2006: Advances in Artificial Intelligence* (2006), 264–273.

[18] L.J. Jensen, R. Gupta, N. Blom, D. Devos, J. Tamames, C. Kesmir, H. Nielsen, H.H. Staerfeldt, K. Rapacki, C. Workman, C.A. Andersen, S. Knudsen, A. Krogh, A. Valencia and S. Brunak, Prediction of human protein function from post-translational modifications and localization features, *J Mol Biol* **319**(5) (2002), 1257–1265.

[19] T. Joachims, Text categorization with support vector machines: learning with many relevant features, in: *The European Conference on Machine Learning (ECML'98)*, C. Nédellec and C. Rouveirol, eds, Chemnitz, DE, Springer Verlag, Heidelberg, DE, 1398, (1998), 137–142.

[20] I.T. Jolliffe, *Principal Component Analysis*, 2002.

[21] S. Kiritchenko, S. Matwin and A.F. Famili, Functional annotation of genes using hierarchical text categorization, in: *Proc of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05* (2005).

[22] D. Koller and M. Sahami, Hierarchically classifying documents using very few words, in: *Proceedings of ICML-97, 14th International Conference on Machine Learning* D. Fisher, ed., Morgan Kaufmann Publishers, San Francisco, US, (1997), 170–178.

[23] M. Kubat and S. Matwin, Addressing the curse of imbalanced training sets: One-sided selection, in: *Proceedings of the Fourteenth International Conference on Machine Learning* Morgan Kaufmann, (1997), 179–186.

[24] D.D. Lewis, Y. Yang, T.G. Rose and F. Li, RCV1: A new benchmark collection for text categorization research, *J Mach Learn Res* **5** (2004), 361–397.

[25] B. Li, L. Ma, J. Hu and K. Hirasawa, Gene classification using an improved svm classifier with soft decision boundary, in: *SICE Annual Conference, 2008* (2008), 2476–2480.

[26] A. McCallum, R. Rosenfeld, T.M. Mitchell and A.Y. Ng, Improving text classification by shrinkage in a hierarchy of

AUTHOR COPY

classes, in: *Proceedings of the Fifteenth International Conference on Machine Learning* Morgan Kaufmann Publishers Inc., (1998), 359–367.
[27] D. Mladenie, *Machine Learning on Non-homogeneous, Distributed Text Data*, PhD thesis, University of Ljubljana, Faculty of Computer and Information Science, 1998.
[28] C. Nguyen, T. Dung and T. Cao, *Text Classification for DAG-Structured Categories*, of *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg **3518** (2005), 97–114.
[29] G. Pandey, C.L. Myers and V. Kumar, Incorporating functional inter-relationships into protein function prediction algorithms, *BMC Bioinformatics* **10**(142) (2009).
[30] C.Z. Peter, P. Jansen, E. Stoica, N. Grot and D.A. Evans, Threshold calibration in clarit adaptive filtering, in: *Proceeding of Seventh Text REtrieval Conference (TREC-7)* (1998), 149–156.
[31] J.R. Quinlan, Induction of decision trees, *Machine Learning* **1**(1) (March 1986), 81–106.
[32] J.R. Quinlan, *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*, Morgan Kaufmann, 1 edition, January 1993.
[33] M. Riley, Functions of the gene products of escherichia coli, *Microbiol Mol Biol Rev* **57**(4) (1993), 862–952.
[34] K. Sarinnapakorn and M. Kubat, Combining subclassifiers in text categorization: A dst-based solution and a case study, *IEEE Transactions on Knowledge and Data Engineering* **19**(12) (2007), 1638–1651.
[35] L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev and S. Dzeroski, Predicting gene function using hierarchical multi-label decision tree ensembles, *Bmc Bioinformatics* **11** (2010).
[36] A. Secker, M.N. Davies, A.A. Freitas, J. Timmis, M. Mendao and D.R. Flower, An experimental comparison of classification algorithms for hierarchical prediction of protein function, *3rd UK Data Mining and Knowledge Discovery Symposium UKKDD 2007 Canterbury* (2007), 1318.
[37] J.G. Shanahan and N. Roma, Boosting support vector machines for text classification through parameter-free threshold relaxation, in: *CIKM '03: Proceedings of the Twelfth International Conference on Information and Knowledge Management* ACM, (2003), 247–254.
[38] S. Shwartz and N. Srebro, Svm optimization: Inverse dependence on training set size, in: *Proceedings of the 25th International Conference on Machine Learning* ACM, (2008), 928–935.
[39] C. Silla and A. Freitas, A survey of hierarchical classification across different application domains, *Data Mining and Knowledge Discovery* (2010).
[40] B. Stenger, A. Thayananthan, P.H.S. Torr and R. Cipolla, Estimating 3d hand pose using hierarchical multi-label classification, *Image Vision Comput* **25**(12) (2007), 1885–1894.
[41] A. Sun and E. Lim, Hierarchical text classification and evaluation, in: *Proceedings of the 2001 IEEE International Conference on Data Mining* ICDM '01, Washington, DC, USA, IEEE Computer Society, (2001), 521–528.
[42] A. Sun, E.-P. Lim and Y. Liu, On strategies for imbalanced text classification using svm: A comparative study, *Decision Support Systems* **48**(1) (2009), 191–201.
[43] C.J. van Rijsbergen, *Information Retrieval*, Butterworths, London, 2 edition, 1979.
[44] V.N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag New York, Inc., New York City, NY, USA, 1995.
[45] P. Vateekul, S. Dendamrongvit and M. Kubat, Improving SVM Performance in Multi-Label Domains: Threshold Adjustment, *International Journal on Artificial Intelligence Tools* **22** (2013).
[46] P. Vateekul and M. Kubat, Fast induction of multiple decision trees in text categorization from large scale, imbalanced, and multi-label data, in: *ICDMW '09: Proceedings of the 2009 IEEE International Conference on Data Mining Workshops* Miami, FL, USA, (2009), 320–325.
[47] C. Vens, J. Struyf, L. Schietgat, S. Dzeroski and H. Blockeel, Decision trees for hierarchical multi-label classification, *Machine Learning* **73**(2) (2008), 185–214.
[48] L. Wang, *Support Vector Machines: Theory and Applications*, Springer, Berlin; New York, 2005.
[49] W.R. Weinert and H.S. Lopes, Neural networks for protein classification, *Applied Bioinformatics 3* **1** (2004).
[50] L. Yan, D. Xei and Z. Du, A new method of support vector machine for class imbalance problem, in: *CSO '09: Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization* (2009), 904–907.
[51] Y. Yang, An evaluation of statistical approaches to text categorization, *Information Retrieval* **1**(1/2) (1999), 69–90.
[52] Y. Yang, J. Zhang and B. Kisiel, A scalability analysis of classifiers in text categorization, in: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval* SIGIR '03, New York, NY, USA, ACM, (2003), 96–103.

AUTHOR COPY