A study on the selection of local training sets for hierarchical classification tasks

Jean Metz[1], **Alex A. Freitas**[2], **Maria Carolina Monard**[1], **Everton Alvares Cherman**[1]

[1]Institute of Mathematics and Computer Science, University of São Paulo (USP)
Av. Trabalhador São Carlense 400, Centro, CEP: 13566-590, São Carlos – SP – Brasil

[2]School of Computing, University of Kent
CT2 7NF, Canterbury – Kent – U.K.

{metzz,mcmonard,echerman}@icmc.usp.br, A.A.Freitas@Kent.ac.uk

Abstract. *In hierarchical classification tasks using the local approach, an important decision concerns the selection of training examples to build the local classifiers. To this end, several policies, which take into account the class taxonomy information, have been proposed. However, a study of a comprehensive comparison concerning the performance of these policies is still lacking. This paper presents a comprehensive empirical evaluation of eight different policies using 13 datasets. The results have shown that three of these policies outperformed the other five policies with statistically significant differences.*

1. Introduction

Hierarchical classification is the task of categorizing examples into a set of classes consistent with a predefined taxonomy organized in a hierarchical fashion. In other words, the classes in the taxonomy relate to each other by means of generalization and specialization relationships. Nowadays, a growing number of applications are specially suited for the task of classifying objects into hierarchical structures [Silla Jr and Freitas 2011].

Hierarchical classification methods can be categorized according to several different criteria, such as data structure and the use of hierarchical information during the training and prediction phase. Regarding the data structure, the class hierarchy is usually represented by a tree or a direct acyclic graph (DAG) [Vens et al. 2008], where the main difference is related to the number of parent classes per node. When a tree structure is used, each node has only one parent node, with the obvious exception of the root node that has no parent. On the other hand, whenever DAG is used, each non-root node is allowed to have more than one parent. A direct consequence of this fact is the possibility of reaching a certain node going through various paths in the class hierarchy. This increases the complexity of the methods that use the DAG data structure. This paper focuses on tree-based class hierarchy.

Considering the use of hierarchical information, the methods can be divided into two types [Kiritchenko et al. 2006]: *global* (or big-bang) and *local* (or top-down level-based). A learning approach is called global if it builds only one classifier to discriminate all categories in a hierarchy, somehow respecting the relationships between classes in different levels of the taxonomy, and it is called local if it builds separate classifiers associated to some of the nodes in the hierarchy, called local classifiers. During the classification stage, a local approach usually proceeds in a top-down fashion, also known as

Pachinko Machine strategy, first predicting the most relevant category(ies) of the top level and then recursively predicting its child classes. This top-down prediction strategy stops when a stop criterion is met or a leaf node is reached [Sun et al. 2003].

Furthermore, according to [Freitas and de Carvalho 2007], there are two distinct broad types of prediction strategy for hierarchical classification methods: *Mandatory leaf node prediction* (MLN), where the most specific class assigned to an instance must be a leaf class, and *Non-mandatory leaf node prediction* (NMLN), where the most specific class assigned to an instance may be either an internal or a leaf class in the class taxonomy.

A popular strategy for hierarchical classification using local methods builds a local binary classifier for each node except the root. We call this method *Hierarchical Binary Relevance* (HBR) in accordance with the *Binary Relevance* method for multilabel classification introduced in [Tsoumakas et al. 2010]. It is worth noting that a hierarchical classification task is in essence a multilabel task, since each example is possibly associated with more than one class at a time, one for each level of the taxonomy. However, in the hierarchical context, the multiple labels assigned to each instance obey a specific rule respecting the hierarchical organization of classes. In this paper, we focus on this local HBR approach, which has the advantage of having a standard flat classification algorithm which can be used to produce the local classifiers, avoiding the need for designing a complex global hierarchical classification algorithm.

In the HBR approach, the main question is how to select the local training sets used by the learning algorithm running at each node of the class hierarchy. Using a straightforward strategy, for each class in the taxonomy, the examples of the entire training set are labeled as positive if the instance is annotated with the respective class and negative otherwise, resulting in a new binary training set of examples which is used to build the local classifier. This naive approach solves the problem of selecting local training sets, but it does not consider the hierarchical information in the class taxonomy and, consequently, may result in suboptimal training sets.

Nonetheless, there are several other methods that can be applied to refine this selection process, considering local information and paying attention to the topology of the classification scheme. Although these methods have been proposed in the literature, to the best of our knowledge, there is no work presenting a comprehensive empirical comparison of training set selection methods across several different application domains. In this work, we experimentally analyzed the results obtained by 8 different training set selection policies in 13 hierarchical datasets. Six of these policies are based on mathematics set operations and 2 are based on distance between the examples. Results show that the *set operation based* policies are more efficient and effective than the *distance based* policies. Moreover, three of the *set operation based* policies present better results.

The remainder of this work is organized as follows: Section 2 presents fundamental concepts of the hierarchical classification tasks, as well as the selection of local training set policies and the evaluation measures. Moreover, Section 2 also discusses related work. The experimental scenarios and analyses are presented in Section 3. Finally, in Section 4 the conclusions and the future work directions are given.

2. Hierarchical classification

As stated before, a tree data structure or a DAG can be used to represent the class hierarchy. In this work, we use methods based on the tree data structure and assume the existence of the root class, which is the ancestor of all the other categories in the hierarchy. Thus, the class taxonomy is defined as a partially order finite set $\tau = (L, \prec)$ that enumerates all class concepts (labels) in the application domain. $L = \{y_1, y_2, ..., y_q\}$ represents the finite set of class labels and the operator $\prec$ represents the *is-a* relationship among the labels in L. Thus, whenever an instance belongs to a class y_j it also belongs to all ancestor nodes of that class (*true path rule*). In this way, the assigned labels would be clearly consistent with the class taxonomy structure. Following [Fagni and Sebastiani 2007], other operators to describe hierarchical relationships are defined in Table 1.

Table 1. Hierarchical relationships.

Operator	Description
$\uparrow y_j$	The parent of class y_j
$\downarrow y_j$	The set of child classes of the class y_j
$\Uparrow y_j$	The set of ancestor classes of the class y_j
$\Downarrow y_j$	The set of descendant classes of the class y_j
$\leftrightarrow y_j$	The set of sibling classes of the class y_j

The HBR is a popular local approach for hierarchical classification. In order to build each local classifier in the HBR method, it is necessary to select the local training set related to each class y_j, namely Tr_{y_j}. Therefore, HBR builds $|L| - 1$ local binary classifiers considering the respective training set, *i.e.*, one classifier for each class y_j in the taxonomy except the root node — Equation (1).

$$HBR = \{C_{y_j}(\mathbf{x}, y_j) \rightarrow \hat{y}_j \in \{0, 1\} |\ y_j \in (L, \prec)\ \wedge\ y_j \neq root\} \tag{1}$$

Each of these local binary classifiers is able to predict whether a given example is positive ($\hat{y}_j = 1$) or negative ($\hat{y}_j = 0$) for the respective class. Using a top-down prediction strategy, whenever an example is predicted as positive, it is recursively presented to the local child classifiers to perform the classification. This process continues until a stop criterion is met or a leaf node is reached. The final hierarchical classification for a given example is the aggregation of the labels positively predicted by all the independent binary classifiers that respect the *true path rule*. It is worth noting that this method is naturally multilabel in the sense that each example may be classified in different branches of the class taxonomy. This aspect may be avoided by only assigning to the given example the most likely class at each level, forcing only the children of the assigned class to classify the example at the next level. In this work, we apply this strategy to constrain each example to be assigned just one class at each level of the class hierarchy. Next we present the local training set selection policies studied in this paper.

2.1. Local training set selection policies

Supervised learning is a two-stage process where known examples (labeled examples) of a domain, named global training set Tr, are used by the learning algorithm to build (train) a classifier, which is used during the second stage to make predictions of examples whose classifications are unknown (unlabeled examples).

During the HBR training stage, for each class y_j in the taxonomy a local classifier C_{y_j} is built by a learning algorithm using the positive set of examples $Tr^+_{y_j}$ (those that

represent the node y_j) and the negative set of examples $Tr^-_{y_j}$ (those that represent instances outside the node y_j). Several policies have been proposed to select these sets of examples from Tr in order to build each local classifier C_{y_j}, described in Table 2.

Six of theses policies are defined based on mathematical operations over sets of examples considering the class taxonomy and the most specific class assigned to each example in Tr. Furthermore, the operator $*$, which is applied to a set of class labels, represents all examples *annotated* with one of the specified labels as its most specific class. The other two policies (*B-Global* and *B-Local*) are based on the distance between examples.

Table 2. Policies for local training sets selection.

Policy	$Tr^+_{y_j}$	$Tr^-_{y_j}$	Reference
All (*L-Inclusive*)	$*(y_j \cup \Downarrow y_j)$	$Tr - Tr^+_{y_j}$	[Mladenić and Grobelnik 1998]
Hierarchical training set (*S-Inclusive* §)	$*(y_j \cup \Downarrow y_j)$	$*(\leftrightarrow y_j \cup \Downarrow (\leftrightarrow y_j))$	[Ceci and Malerba 2003]
Proper training set (*S-Exclusive* §)	$*(y_j)$	$*(\leftrightarrow y_j)$	[Ceci and Malerba 2003]
Inclusive	$*(y_j \cup \Downarrow y_j)$	$Tr - (Tr^+_{y_j} \cup *(\Uparrow y_j))$	[Eisner et al. 2005]
Exclusive	$*(y_j)$	$Tr - Tr^+_{y_j}$	[Eisner et al. 2005]
L-Exclusive	$*(y_j)$	$Tr - *(y_j \cup \Downarrow y_j)$	[Eisner et al. 2005]
B-Global	$*(y_j \cup \Downarrow y_j)$	$\underset{E^- \notin Tr^+_{y_j}}{\overset{\eta}{\operatorname{argmin}}} \sigma(\zeta_{y_j}, E^-)$	[Fagni and Sebastiani 2007]
B-Local	$*(y_j \cup \Downarrow y_j)$	$\underset{E^- \notin Tr^+_{y_j}}{\overset{\eta}{\operatorname{argmin}}} \sum_{E^+ \in \chi^{\eta}_{E^-}} \sigma(E^+, E^-)$ $\chi^k_{E^-} = \underset{E^+ \in Tr^+(y_j)}{\overset{k}{\operatorname{argmin}}} \sigma(E^-, E^+)$	[Fagni and Sebastiani 2007]

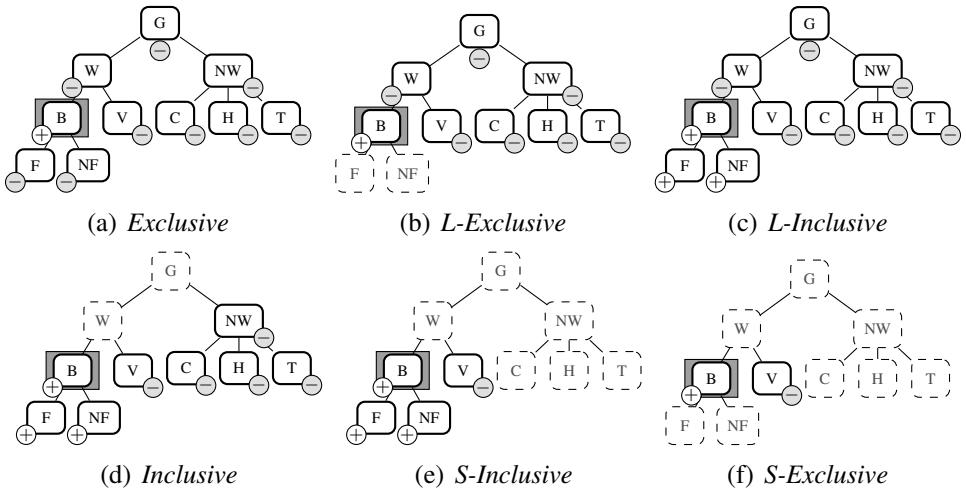
§ The *S* in these policies means siblings.

The rationale behind the *All* policy [Mladenić and Grobelnik 1998] is the use of hierarchical information to select the local training set Tr_{y_j}. Therefore, examples explicitly annotated with y_j as well as any of the descendant classes of y_j are included in $Tr^+_{y_j}$, since by definition $\forall y_k \in (\Downarrow y_j), y_k \prec y_j$. All examples not included in the positive set are used as negative training examples. In other words, $Tr^-_{y_j} = Tr - Tr^+_{y_j}$. This policy was later used in [Eisner et al. 2005] and was called *Less-Inclusive*. From now on, the *L-Inclusive* term is adopted to refer to this policy.

The concept of *exclusive* and *inclusive* classifiers, presented in [Eisner et al. 2005], can be used to clarify the definitions of some policies. According to the authors, a hierarchical classifier is considered *exclusive* if for a given instance which is to be assigned to a class y_j, only the local classifier C_{y_j} predicts positive, while every other local classifier in the hierarchy predicts negative, including the predecessor classifiers of C_{y_j}. In other words, the C_{y_j} classifier excludes (negates) every instance except the ones explicitly labeled with class y_j, *i.e.*, *excludes* all examples in the set $\neg(*y_j)$. On the other hand, the classifier is *inclusive* whenever an example belongs to $*y_j$ and the classifier C_{y_j}, as well as all ancestor classifiers predict positive for this example. Thus, it considers (includes) the prediction of the ancestor classifiers and not only C_{y_j}.

All the 6 *set operator* based policies work in a similar way, first selecting the examples in $Tr^+_{y_j}$ based on the most specific class ($*y_j$), and applying afterwards a mathematical set operation over Tr and $Tr^+_{y_j}$ to define the $Tr^-_{y_j}$ set — Table 2. The main difference among the 6 policies lies on the operation used and whether it is applied over all examples not in $Tr^+_{y_j}$ or a sub-set of it. Figure 1 illustrates the use of these policies to select the local training set of class B, namely Tr_B, where $L =$

$\{G, W, B, F, NF, V, Z, NW, C, H, T\}$. The gray square indicates the target class for which the training set is selected. The other class nodes participate with different roles for each policy, determining whether an example is positive ($\oplus$), negative ($\ominus$) or ignored (nodes with dotted lines).



(a) *Exclusive* (b) *L-Exclusive* (c) *L-Inclusive*

(d) *Inclusive* (e) *S-Inclusive* (f) *S-Exclusive*

Figure 1. Illustration of the 6 set operation based policies.

According to the *S-Inclusive* policy, $Tr^-_{y_j}$ is chosen among the training examples that are not positive for C_{y_j} and may be assumed to be mostly correlated to y_j, *i.e.*, it consists of all training examples labeled with classes that are siblings of the positive class or descendants of those siblings. The basic idea is that $Tr^-_{y_j}$ includes the "quasi-positive" examples of y_j [Schapire et al. 1998]. In other words, the examples not in $Tr^+_{y_j}$ that are closest to the boundary between the positive and the negative region of y_j, and are thus the most informative negative examples that can be used to build the local classifier. This is beneficial also from the standpoint of efficiency, since fewer training examples are involved. A variant of the *S-Inclusive* policy is called *Siblings-Exclusive* (*S-Exclusive*), which restricts the selection of examples to the ones explicitly labeled with class y_j and their siblings, ignoring descendant classes.

The *distance* based policies (*B-Global* and *B-Local*) [Fagni and Sebastiani 2007] also make use of the "quasi-positive" notion, more precisely, the "query-zoning" selection strategy introduced in [Singhal et al. 1997]. In the vector space model, a query zone can be envisioned as a volume of the vector space around a query object. In the context of classification, a query zone for a new example can be simulated by considering a set of training examples that have some similarity to the new example. Moreover, this concept can be interpreted as a boundary region on the input space between positive and non-positive examples. Therefore, it can be used to select the non-positive examples that lie on the query-zone of the $Tr^+_{y_j}$ examples. The basic idea is that more informative examples that lie on the borderline between positive and negative classes might increase the classifier's prediction power with a thinner adjustment on the hyperplanes between classes.

In both *distance* based policies, the $Tr^+_{y_j}$ examples are defined as the set of examples with the most specific class label in the set $\{y_j \cup (\Downarrow y_j)\}$, *i.e.*, the examples explicitly labeled with one of the classes in the sub-tree rooted at y_j. The $Tr^-_{y_j}$ examples are selected

from the non-positive example set, depending on the policy. To this end, the *B-Global* policy finds the centroid of $Tr^{+}_{y_j}$ and calculates the distances between the centroid and each non-positive example. After that, it selects the η closest non-positive examples and includes them in $Tr^{-}_{y_j}$. The *B-Local* policy uses a similar approach, but tries to identify the "quasi-positive" examples within a more complex boundary region. In order to find this boundary, *B-Local* uses not only one centroid but a small subset of k positive examples to identify the closest non-positive examples while considering the overall shape of the positive example space. Putting it in a different perspective, the *B-Local* policy calculates for each non-positive example the sum of the distances to the k closest positive examples, and selects the η non-positive examples that minimizes the calculated distance.

Although some papers report the impact of these policies on the prediction performance of the HBR method, summarized in Table 3, to the best of our knowledge there is no study showing conclusive experimental results comparing all these policies on different domains and/or several datasets. Note that each of the projects mentioned in Table 3 carried out experiments with at most 4 training set selection policies and 2 datasets, whilst in this paper we perform experiments with 8 policies and 13 datasets.

Table 3. Policies considered in related work (▲ better, and ▼ worse).

Reference	Policy							
	Exclusive	*L-Exclusive*	*L-Inclusive*	*Inclusive*	*S-Inclusive*	*S-Exclusive*	*B-Global*	*B-Local*
[Ceci and Malerba 2003]					▲	▼		
[Eisner et al. 2005]	▼	▼	▲	▲				
[Fagni and Sebastiani 2007]			▲		▲		▼	▼

2.2. Hierarchical classifier evaluation

Conventional flat classification measures are not suitable for hierarchical classification tasks, since they cannot handle different types of misclassification errors present in the hierarchical scenario. Intuitively, a sibling or a parent node misclassification is less harmful then a more distant node misclassification. For that reason, specialized measures consistent with conventional non-hierarchical measures, as well as with the class taxonomy, allowing for less severe misclassification errors, such as generalization and specialization errors, are needed. Furthermore, flat classification measures cannot handle a partially correct prediction, which is essential to evaluate hierarchical classifiers performance. In this work, the hierarchical version of the standard precision and recall measures proposed in [Kiritchenko et al. 2006] and defined by Equations (2) and (3), respectively, are used, where TP refers to *true positive*, FP to *false positive* and FN to *false negative*. It is important to mention that these measures are actually a micro-average of the traditional precision (Pr_j) and recall (Re_j) obtained for each label in the hierarchy. The use of averaged measures contributes to a more precise overall evaluation of the experimental results, since local evaluation would produce as many precision and recall values as the number of classes in the taxonomy, making the analysis impractical when there are too many classes in the taxonomy.

Another measure commonly used combines the precision and recall values into one single value called *f-Measure* — Equation (4). In our experimental evaluation, we used $\beta = 1$, giving precision and recall equal weights.

$$hPr^{\mu} = \frac{\sum_{j=1}^{|L|} |TP_{y_j}|}{\sum_{j=1}^{|L|}(|TP_{y_j}| + |FP_{y_j}|)} \quad (2) \qquad hRe^{\mu} = \frac{\sum_{j=1}^{|L|} |TP_{y_j}|}{\sum_{j=1}^{|L|}(|TP_{y_j}| + |FN_{y_j}|)} \quad (3)$$

$$F_{\beta} = \frac{(\beta^2 + 1) \times hPr^{\mu} \times hRe^{\mu}}{\beta^2 \times hPr^{\mu} + hRe^{\mu}};\ where \beta \in [0, \infty) \quad (4)$$

3. Experimental Analysis

The main purpose of this experimental analysis is to assess the impact of the 8 different policies to select the local training sets on the performance of the HBR method.

3.1. Datasets and Protocol

The experiments were carried out with 13 datasets from three different domains. Table 4 describes the datasets, where #E, #F and $|L|$ are the number of examples, number of features and number of class labels, respectively. Column $|L|$ *per level* shows the number of labels in each level of the class taxonomy, except for the root. The last column (*Label Cardin.*), shows the average number of class labels per example.

Table 4. Datasets description.

Dataset	Domain	MLN	#E	#F	$\|L\|$	$\|L\|$ *per level*	*Label Cardin.*
HGlass	Glass Identification	yes	214	9	9	2, 5, 2	2.682
Fabien	Music Genre Classification	yes	4188	40	20	2, 9, 9	2.645
Marsyas	Music Genre Classification	yes	4188	30	20	2, 9, 9	2.645
ThomasRH	Music Genre Classification	yes	4188	60	20	2, 9, 9	2.645
ThomasSSD	Music Genre Classification	yes	4188	168	20	2, 9, 9	2.645
G-Pfam †	Protein Function	no	7053	75	192	12, 52, 79, 49	2.841
G-Prints †	Protein Function	no	5404	283	179	8, 46, 76, 49	3.009
G-Prosite †	Protein Function	no	6246	129	187	9, 50, 79, 49	2.951
G-Interpro †	Protein Function	no	7444	450	198	12, 54, 82, 50	2.823
E-Pfam ‡	Protein Function	no	13987	708	333	6, 41, 96, 190	3.667
E-Prints ‡	Protein Function	no	14025	382	351	6, 45, 92, 208	3.699
E-Prosite ‡	Protein Function	no	14041	585	324	6, 42, 89, 187	3.690
E-Interpro ‡	Protein Function	no	14027	1216	330	6, 41, 96, 187	3.660

†: Classes are functions of GPCR proteins; ‡: Classes are functions of enzymes

The HBR method with top-down prediction strategy implemented in HARPIA [1], a framework for hierarchical classification, was used. This framework makes extensive use of the Weka[2] java library for machine learning. The experiments were carried out using two different flat classification algorithms to build the local binary classifiers: J48, a decision tree learning algorithm and Naïve Bayes (NB), both implemented in the Weka library [Witten et al. 2011].

The 8 selection policies were used to construct the training sets to build the local classifiers. As the *distance* based policies are parametric, for a fair comparison with the *set operator* based policies, we set up the number of negative examples (η) to be equal to the number of positive examples. Moreover, 5 different values for k (1, 3, 5, 7 and 9) were used with the *B-Local* policy.

All the reported results were obtained using 5×2-fold cross validation with paired folds, *i.e.*, the same training and testing partitions were used to assess the performance of

[1]http://labic.icmc.usp.br/software-and-application-tools

[2]http://www.cs.waikato.ac.nz/ml/weka

all different configurations of the HBR method. In order to analyze whether there is a difference among the policies impact on the predictive accuracy, we run the Friedman's test[3] with the null hypothesis that the performance of the algorithms using any of the policies are comparable considering all the results. When the null hypothesis is rejected by the Friedman's test, at 95% of confidence level, we proceed with the Nemenyi's post-hoc test to detect which differences among the methods are significant. According to this test, the performance of two methods is significantly different if the corresponding average ranks differ by at least the critical difference (CD). The results and discussion are presented next.

3.2. Evaluation

The evaluation considers (a) whether the degree of *inclusion* affects the overall performance of the algorithm when using a *set operation* based policy; (b) analyzing the effect of the k parameter for the *B-Local* policy; and (c) the overall comparison of all policies.

Figure 2 shows the results obtained with J48 as a base classifier using *set operation* based policies, where policies are indicated on the X-axis and the average of the $F_1^{\mu 4}$ values are shown on the Y-axis. For the sake of visualization, only 6 datasets are shown in this figure, since the results for some datasets were similar. More specifically, there is no significant difference using *E-Pfam*, *E-Prints*, *E-Prosite* and *E-Interpro* datasets. The same happened for other pairs of datasets, such as *G-Pfam* and *G-Prosite*; *G-Interpro* and *G-Prints*; *Marsyas* and *ThomasSSD*; and finally *Fabien* and *ThomasRH*. Due to space restriction, as the results using NB as the base classifier follow the same pattern, the corresponding figure is omitted[5].

Figure 2 clearly shows the tendency of improvement on the overall performance of the HBR method using more *inclusive* policies, as for all cases the *exclusive* policies were outperformed by the *inclusive* ones. In fact, when using *exclusive policies* with datasets where all examples are annotated with only leaf classes, the local training sets selected to build the the internal classifiers will be empty, resulting in very poor classifiers that reject (predict negative) any new example. These "empty" classifiers have a strong influence on the overall degradation of the HBR global performance. Even when the datasets have examples explicitly annotated with internal classes, there may exist some classes with no examples, also contributing to the degradation of the final model.

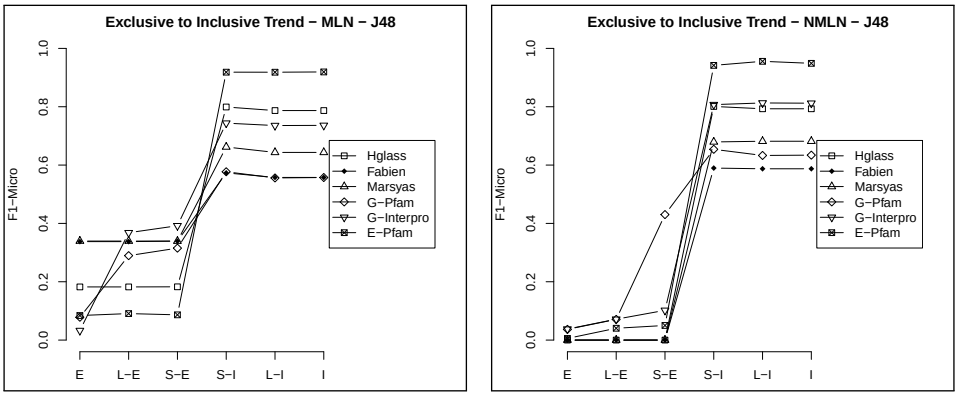
When the prediction type is set to MLN (Figure 2(a)), *i.e.*, mandatory leaf node prediction, even when examples are classified as negative by all the classifiers at a certain level, they are inspected by all the immediate lower level classifiers in the branch where the confidence on the negative prediction is lower ("less negative"), *i.e.*, the classification continues through the branch being mostly "positive". This means the final classification is incorrect with a high probability, or even random. On the other hand, if the prediction type is NMLN, *i.e.*, non-mandatory leaf node prediction, whenever an example is predicted as negative by all classifiers at a certain level, it is not inspected by deeper classifiers.

As the first level local classifiers might be "empty" for all *exclusive* policies, they would predict any presented examples as negative, resulting in cumulative errors and

[3]See [Demsar 2006] for a thorough discussion regarding statistical tests in machine learning research.

[4]F-Measure using $\beta = 1$ and the micro version of the precision (hPr^{μ}) and recall (hRe^{μ}).

[5]All tabulated results can be found in http://www.icmc.usp.br/ metzz/archives/harpia/

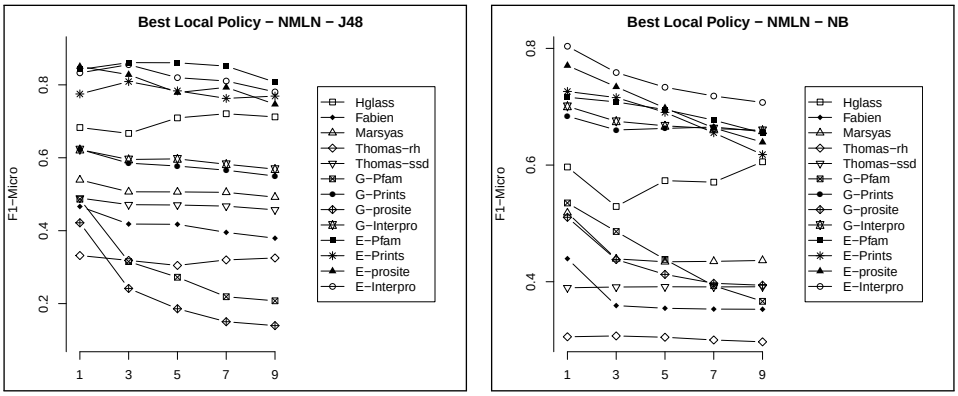


(a) (b)

Figure 2. Set operator based policies. *Exclusive* (E), *L-Exclusive* (L-E), *S-Exclusive* (S-E), *S-Inclusive* (S-I), *L-Inclusive* (L-I) and *Inclusive* (I).

degrading the HBR global performance. This explains Figure 2(b), where the use of the policies *Exclusive*, *L-Exclusive* and *S-Exclusive* with the datasets *Fabien* and *Marsyas* resulted in 0.00 F_1^μ. Similar results were obtained with *G-Pfam*, *G-Interpro* and *E-Pfam* datasets, showing F_1^μ values close to zero. In contrast, the more *inclusive* policies, which prevent the creation of classifiers with empty training sets, show better results.

As the *exclusive* policies performed clearly worse than the *inclusive* ones, from now on we focus our analysis on the *inclusive* and *distance* based policies. Figure 3 shows the effect of the variation on the k parameter for the *B-Local* policy using the NMLN prediction strategy, where the different k values are indicated on the X-axis and the average of the F_1^μ values are shown on the Y-axis. It can be observed that, in most cases, increasing the value of k results in smaller F_1^μ values. The only exception is for the *HGlass* dataset. In some cases, the effect of increasing k shows slight degradation or even no degradation at all, as for *ThomasRH* and *ThomasSSD* datasets in Figure 3(a). In other cases, there is a considerable deterioration of F_1^μ, such as with *G-Pfam* and *G-Prosite* datasets. Similar behavior was observed for the MLN prediction strategy. As the best F_1^μ values of the *B-Local* policy were obtained for $k = 1$, we proceed our analysis of the *B-Local* policy only considering $k = 1$.



(a) (b)

Figure 3. The influence of the k parameter value on the *B-Local* policy.

The results for the *inclusive* and *distance* based policies are presented in Table 5, which is divided into 4 quarters, each one corresponding to a combination of one base

classifier (J48 or NB) and one prediction type (MLN or NMLN). For each quarter, the average of the F_1^μ measure obtained by the HBR method using the different policies to generate the local training sets is shown. The numbers in brackets correspond to the standard deviation. Moreover, the last line (labeled as # wins) shows the number of datasets for which the respective policy achieved the greatest F_1^μ value. For the sake of visualization, the best values for each dataset in each quarter are shown in bold.

Table 5. Average F_1^μ values using NB and J48 with MLN and NMLN prediction types.

	J48									
Dataset	*B-Local* $k-1$	*B-Global*	*S-Inclusive*	*L-Inclusive*	*Inclusive*	*B-Local* $k-1$	*B-Global*	*S-Inclusive*	*L-Inclusive*	*Inclusive*
	MLN					NMLN				
HGlass	0.68(0.09)	0.68(0.07)	**0.80(0.02)**	0.79(0.02)	0.79(0.02)	0.68(0.09)	0.68(0.07)	**0.80(0.03)**	0.79(0.02)	0.79(0.02)
Fabien	0.47(0.03)	0.38(0.03)	**0.57(0.01)**	0.56(0.01)	0.56(0.01)	0.47(0.03)	0.37(0.03)	**0.59(0.01)**	**0.59(0.01)**	**0.59(0.01)**
Marsyas	0.53(0.02)	0.47(0.02)	**0.66(0.01)**	0.64(0.01)	0.64(0.01)	0.54(0.02)	0.47(0.02)	**0.68(0.01)**	**0.68(0.01)**	**0.68(0.01)**
ThomasRH	0.34(0.03)	0.30(0.03)	**0.53(0.01)**	0.52(0.01)	0.52(0.01)	0.33(0.03)	0.29(0.03)	**0.55(0.01)**	**0.55(0.01)**	**0.55(0.01)**
ThomasSSD	0.49(0.02)	0.42(0.03)	**0.66(0.01)**	0.65(0.01)	0.65(0.01)	0.49(0.02)	0.42(0.03)	0.68(0.01)	**0.69(0.01)**	**0.69(0.01)**
G-Pfam	0.44(0.04)	0.33(0.05)	**0.58(0.01)**	0.56(0.01)	0.56(0.01)	0.49(0.04)	0.38(0.05)	**0.65(0.00)**	0.63(0.01)	0.63(0.01)
G-Prints	0.59(0.05)	0.56(0.05)	**0.73(0.01)**	0.72(0.01)	0.72(0.01)	0.62(0.05)	0.58(0.04)	0.80(0.00)	**0.81(0.01)**	**0.81(0.00)**
G-Prosite	0.39(0.03)	0.22(0.05)	**0.56(0.01)**	0.54(0.01)	0.54(0.01)	0.42(0.02)	0.25(0.05)	**0.64(0.01)**	0.61(0.01)	0.61(0.00)
G-Interpro	0.60(0.07)	0.58(0.04)	**0.74(0.01)**	**0.74(0.00)**	**0.74(0.00)**	0.62(0.07)	0.60(0.04)	**0.81(0.00)**	**0.81(0.00)**	**0.81(0.00)**
E-Pfam	0.80(0.06)	0.57(0.10)	**0.92(0.00)**	**0.92(0.00)**	**0.92(0.00)**	0.84(0.06)	0.63(0.09)	0.94(0.00)	**0.96(0.00)**	0.95(0.00)
E-Prints	0.74(0.04)	0.51(0.03)	**0.90(0.00)**	**0.90(0.00)**	**0.90(0.00)**	0.78(0.04)	0.56(0.03)	0.92(0.00)	**0.94(0.00)**	0.93(0.00)
E-Prosite	0.82(0.06)	0.58(0.14)	**0.93(0.00)**	0.92(0.00)	**0.93(0.00)**	0.85(0.06)	0.63(0.14)	0.94(0.00)	**0.96(0.00)**	0.95(0.00)
E-Interpro	0.79(0.06)	0.72(0.03)	**0.93(0.00)**	0.92(0.00)	**0.93(0.00)**	0.83(0.05)	0.77(0.03)	0.95(0.00)	**0.96(0.00)**	**0.96(0.00)**
# wins	0	0	13	3	5	0	0	7	10	7
	NB									
HGlass	0.62(0.08)	0.63(0.08)	0.68(0.04)	**0.75(0.03)**	**0.75(0.03)**	0.60(0.08)	0.61(0.07)	0.68(0.04)	**0.75(0.03)**	**0.75(0.03)**
Fabien	0.44(0.04)	0.37(0.01)	0.51(0.01)	**0.52(0.01)**	**0.52(0.01)**	0.44(0.04)	0.37(0.01)	0.51(0.00)	**0.52(0.01)**	**0.52(0.01)**
Marsyas	0.53(0.03)	0.48(0.01)	**0.65(0.00)**	**0.65(0.00)**	**0.65(0.00)**	0.52(0.03)	0.47(0.01)	**0.66(0.00)**	**0.66(0.00)**	**0.66(0.00)**
ThomasRH	0.35(0.01)	0.34(0.00)	**0.51(0.00)**	**0.51(0.00)**	**0.51(0.00)**	0.31(0.00)	0.29(0.01)	**0.51(0.00)**	**0.51(0.00)**	**0.51(0.00)**
ThomasSSD	0.45(0.01)	0.42(0.01)	**0.60(0.01)**	0.57(0.01)	0.57(0.01)	0.39(0.01)	0.40(0.01)	**0.60(0.01)**	0.57(0.01)	0.57(0.01)
G-Pfam	0.53(0.01)	0.46(0.01)	**0.57(0.00)**	**0.57(0.00)**	**0.57(0.00)**	0.54(0.01)	0.48(0.01)	**0.64(0.00)**	**0.64(0.00)**	**0.64(0.00)**
G-Prints	**0.66(0.01)**	0.63(0.01)	**0.66(0.00)**	**0.66(0.00)**	**0.66(0.00)**	0.68(0.01)	0.63(0.01)	**0.72(0.01)**	**0.72(0.01)**	**0.72(0.01)**
G-Prosite	0.50(0.01)	0.41(0.01)	**0.55(0.01)**	**0.55(0.01)**	**0.55(0.01)**	0.51(0.01)	0.42(0.01)	0.61(0.00)	0.61(0.00)	**0.62(0.00)**
G-Interpro	**0.68(0.01)**	0.67(0.01)	0.66(0.01)	0.66(0.01)	0.66(0.01)	0.70(0.01)	0.68(0.01)	0.72(0.00)	**0.73(0.00)**	**0.73(0.00)**
E-Pfam	0.68(0.03)	0.55(0.03)	0.80(0.00)	0.80(0.00)	**0.81(0.00)**	0.72(0.02)	0.59(0.03)	**0.82(0.00)**	0.79(0.01)	0.79(0.00)
E-Prints	0.70(0.02)	0.58(0.03)	0.83(0.00)	**0.84(0.00)**	**0.84(0.00)**	0.73(0.02)	0.61(0.03)	**0.85(0.00)**	0.84(0.00)	0.84(0.00)
E-Prosite	0.75(0.04)	0.65(0.02)	**0.85(0.00)**	**0.85(0.00)**	**0.85(0.00)**	0.77(0.04)	0.67(0.02)	**0.87(0.00)**	0.86(0.00)	0.86(0.00)
E-Interpro	0.77(0.02)	0.77(0.01)	**0.80(0.00)**	**0.80(0.00)**	**0.80(0.00)**	0.80(0.02)	0.81(0.02)	**0.83(0.01)**	0.81(0.00)	0.81(0.00)
# wins	2	0	8	10	11	0	0	9	7	8

An interesting aspect to be evaluated is whether the MLN prediction is better than the NMLN prediction. Considering the performance for each dataset, NMLN has outperformed MLN in more than 80% of the experiments, while for the other portion, both prediction types performed equally. Summing up, NMLN would be a preferable prediction type.

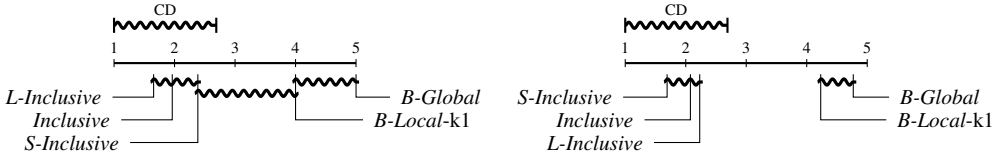
Regarding the policies and considering the number of times that the highest F_1^μ value was obtained (Table 5), it is easily observed that the *S-Inclusive* policy outperforms the others, and that the *set operation* based policies outperform the *distance* based policies. These numbers, for the 52 experiments, are summarized in Table 6.

Table 6. Comparison of Policies.

B-Local $k-1$	*B-Global*	*S-Inclusive*	*L-Inclusive*	*Inclusive*
2	0	37	30	31

In order to analyze whether the difference among the policies impact on the HBR performance, we run the Friedman's hypothesis test with the null hypothesis that the different policies do not affect the final performance of the HBR method. As the hypothesis

was rejected, we run the post-hoc test Nemenyi, that shows significant differences between methods whenever their average ranks differ by at least the critical value (CD). The results of the Nemenyi's test can be represented in a simple diagram (Figure 4), where the main line is the X-axis on which the average ranks of the methods are plotted (lower average rank is better). The secondary lines beneath the main axis connect groups that are not significantly different. Moreover, the critical difference (CD) is shown just above the main line. These diagrams ensure the observations aforementioned, that the *set operation* based policies are better than the *distance* based policies. Notwithstanding, they also show that there is no statistically significant difference among *S-Inclusive*, *L-Inclusive* and *Inclusive* policies. Thus, a good criterion to choose one among these policies could be the number of examples included in the negative local training set, since the smaller the set the faster the training stage is. Taking this into account, *S-Inclusive* policy should be chosen, since it always selects less examples to use in the local training sets than the other two policies (see Figure 1).



(a) J48 base classifier. (b) NB base classifier.

Figure 4. Visualization of the Nemenyi post-hoc test. Groups of methods that are not significantly different (at $p < 0.05$) are connected.

4. Conclusions

This paper presented a discussion on 8 different policies to select local training sets for the *Hierarchical Binary Relevance* method, a popular hierarchical classification method. The policies were evaluated using two different base classifiers in 13 datasets from three different domains. Results show that the *inclusive set operation* based policies are better than the *exclusive* and the *distance* based ones. Moreover, hypothesis tests show that using NB as the base classifier, all *inclusive set operation* policies are significantly better (at $p < 0.05$) than the others. Using J48 as the base classifier, although the *S-Inclusive* policy shows better results than the *Exclusive* and the *distance* based policies, the difference is not statistically significantly better. Regarding *L-Inclusive* and *Inclusive* policies, both were significantly better than distance based policies.

As future work, we intend to extend the experimental evaluation of the three best policies, considering not only datasets from other domains but also a more detailed analysis, as well as a greater number of evaluation measures.

Acknowldgements: This research was supported by the Brazilian Research Councils FAPESP and CAPES. We would like to thank Mr. Carlos Silla Jr for providing the datasets with protein descriptors used in this paper.

References

Ceci, M. and Malerba, D. (2003). Hierarchical classification of HTML documents with WebClass II. In *Proceedings of the 25th European conference on IR research*, ECIR'03, pages 57–72, Berlin, Heidelberg. Springer-Verlag.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Eisner, R., Poulin, B., Szafron, D., Lu, P., and Greiner, R. (2005). Improving protein function prediction using the hierarchical structure of the gene ontology. In *In Proc. IEEE CIBCB*, pages 1–10.

Fagni, T. and Sebastiani, F. (2007). On the selection of negative examples for hierarchical text categorization. In *Proceedings of The 3rd Language Technology Conference*, pages 24–28.

Freitas, A. and de Carvalho, A. C. (2007). A tutorial on hierarchical classification with applications in bioinformatics. In Taniar, D., editor, *Research and Trends in Data Mining Technologies and Applications*, chapter 7, pages 175–208. IGI Global.

Kiritchenko, S., Matwin, S., Nock, R., and Famili, A. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In Lamontagne, L. and Marchand, M., editors, *Advances in Artificial Intelligence*, volume 4013 of *Lecture Notes in Computer Science*, pages 395–406. Springer Berlin / Heidelberg.

Mladenić, D. and Grobelnik, M. (1998). Feature selection for classification based on text hierarchy. In *Text and the Web, Conference on Automated Learning and Discovery CONALD-98*, pages 1–6.

Schapire, R. E., Singer, Y., and Singhal, A. (1998). Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 215–223, New York, NY, USA. ACM.

Silla Jr, C. and Freitas, A. (2011). A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 1:1–42.

Singhal, A., Mitra, M., and Buckley, C. (1997). Learning routing queries in a query zone. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '97, pages 25–32, New York, NY, USA. ACM.

Sun, A., Lim, E., and Ng, W. (2003). Performance measurement framework for hierarchical text classification. *Journal of the American Society for Information Science and Technology*, 54:1014–1028.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010). Mining multi-label data. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, pages 667–685. Springer US.

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

Witten, I. H., Frank, E., Holmes, G., and Hall, M. (2011). *Data Mining: Practical machine learning tools and techniques*, volume 1. Morgan Kaufmann Publishers Inc., San Francisco, California, USA, 3rd edition.