World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:12, No:7, 2018

Meta-Learning for Hierarchical Classification and Applications in Bioinformatics

Fabio Fabris, Alex A. Freitas

***Abstract*—**Hierarchical classification is a special type of classification task where the class labels are organised into a hierarchy, with more generic class labels being ancestors of more specific ones. Meta-learning for classification-algorithm recommendation consists of recommending to the user a classification algorithm, from a pool of candidate algorithms, for a dataset, based on the past performance of the candidate algorithms in other datasets. Meta-learning is normally used in conventional, non-hierarchical classification. By contrast, this paper proposes a meta-learning approach for more challenging task of hierarchical classification, and evaluates it in a large number of bioinformatics datasets. Hierarchical classification is especially relevant for bioinformatics problems, as protein and gene functions tend to be organised into a hierarchy of class labels.

This work proposes meta-learning approach for recommending the best hierarchical classification algorithm to a hierarchical classification dataset. This work's contributions are: 1) proposing an algorithm for splitting hierarchical datasets into new datasets to increase the number of meta-instances, 2) proposing meta-features for hierarchical classification, and 3) interpreting decision-tree meta-models for hierarchical classification algorithm recommendation.

***Keywords*—**Algorithm recommendation, meta-learning, bioinformatics, hierarchical classification.

I. Introduction

THE cost of performing high-throughput biological assays has been constantly decreasing throughout the years. This increases the amount of freely available biological data and thus, the need for computational methods to help biologists extract useful knowledge from this data.

One of the types of computational methods available for biologists are *classification algorithms*. These algorithms take as input a dataset containing instances described by features and some class labels that annotate the instances, and learn a model that can be used to label previously unseen instances. Classification algorithms can be used, for instance, to predict the probable function of proteins [1] or to predict the survivability of cancer patients [2].

In this paper we focus on the problem of predicting protein functions given features that describe the protein. Protein functions are commonly organised into ontologies structured as a tree or a Directed Acyclic Graph (DAG), where each node is a class label and each edge represents a "IS-A" relationship between labels. This means that if an instance is annotated with one class label, it is implicitly annotated with all of that class label's ancestor labels. This type of problem, where class labels are organized into a hierarchy, is called hierarchical classification [3].

Fabio Fabris and Alex A. Freitas are with the School of Computing, University of Kent, Canterbury, Kent, CT2 7NF, UK (e-mail: ff79@kent.ac.uk, A.A.Freitas@kent.ac.uk).

Meta-learning for classification algorithm recommendation is the computational task of recommending to the user a classification algorithm (or a ranking of classification algorithms) from a pool of algorithms, for any new classification dataset (meta-instance), given the past performance of the algorithms in other datasets [4]. To make this recommendation, a meta-classifier is induced using meta-features describing characteristics of datasets in the meta-training set (where each meta-instance represents a dataset), and using as meta-class labels the best classification algorithm for each dataset in the meta-training set. Then, when a new dataset becomes available, the meta-classifier is used to recommend the best algorithm for that dataset. Meta-learning is useful in two main ways: 1) Since it automates the choice of the best algorithm to a new dataset, it avoids the need for running many classification algorithms on the new dataset, which is an *ad hoc* but very popular approach to choose the best classification algorithm in practice. 2) If interpretable meta-classification models are induced, they can be used to explain why a classification algorithm is recommended for a new dataset.

Meta-learning approaches can be of great use in hierarchical classification, where it is typically difficult to choose the best hierarchical classification algorithm for a new dataset. The high problem complexity and the usually very long run times associated with applying many hierarchical classification algorithms to a new dataset make the use of exploratory experiments more difficult.

In this work we have experimented with the following commonly used hierarchical classification algorithms: Predictive Clustering Tree (PCT) [5], Predictive Clustering Trees Ensemble (PCTEN) [1], and Local Hierarchical Classifiers (LHC) [6]. This work proposes meta-learning for automatically recommending a hierarchical classification algorithm. This work's contribution are: 1) an algorithm for generating new datasets from existing ones, to increase the number of meta-instances. 2) proposing meta-features for meta-learning in hierarchical classification, and also 3) we interpret decision tree-based meta-classification models for algorithm recommendation; getting some insight about which dataset properties are good predictors of the best hierarchical classification algorithm for a new dataset.

This paper is organised as follows: Section II presents background on hierarchical classification and meta-learning. Section III defines the meta-features used to describe the hierarchical classification datasets. Section IV presents the hierarchical datasets. Section V describes the algorithm proposed to split existing hierarchical classification datasets

into new datasets (using different parts of the class hierarchy). Section VI presents the experimental setup used in our meta-learning framework. Section VII presents the results of our meta-learning approach, including an analysis of the predictive accuracy of our meta-classification system and an interpretation of the meta-models generated to choose between the three above mentioned candidate hierarchical classification algorithms. Finally, Section VIII draws conclusions.

II. Background on Hierarchical Classification and Meta-Learning

A. Hierarchical Classification

In bioinformatics applications, it is common to use hierarchical classification algorithms to predict gene or protein functions. This type of algorithm is used due to the fact that gene and protein functions are usually categorized by a hierarchical scheme like the Gene Ontology [7]. In other words, in hierarchical classification problems, the instances' class labels are organized in a tree or DAG (Directed Acyclic Graph), where each node represents a class label and the edges represent generalization-specialization (or "IS-A") relationships among class labels.

There are two major types of hierarchical classification algorithms [3]: global or local. Local Hierarchical Classification (LHC) algorithms train several classification models considering only a (typically small) part of the class hierarchy. Then, in the testing phase, the predictions of the local models are combined using some strategy that takes the structure of the class hierarchy into account, so the predictions are consistent with the underlying structure of the classes. On the other hand, global hierarchical classification algorithms follow the approach of building a single specialized global classification model predicting classes in the whole class hierarchy.

One of the most popular families of global hierarchical classification algorithms is the PCT (Predictive Clustering Tree) family. PCTs are the hierarchical classification equivalent of traditional Decision Trees for 'flat' classification. PCT algorithms build a decision tree by sub-dividing the dataset into two disjoint clusters of instances that increase the similarity of classes within each cluster and the dissimilarity of the classes across the two clusters. These clusters are formed by finding a value for a predictive feature that splits the current set of instances. Then, the algorithm recursively applies this same strategy in each new cluster, eventually stopping if the cluster is not good (based on some quality measure) or its size falls bellow a pre-established threshold [8]. In the testing phase, to classify a new instance $\mathbf{x}$, the PCT algorithm first identifies the cluster of training instances associated with the testing instance and then assigns to instance $\mathbf{x}$ class probabilities. These probabilities are calculated using the class labels of the training instances in the cluster associated with instance $\mathbf{x}$.

Note that, for a given testing instance, the PCT model outputs a *class probability vector*, with one probability value for each term in the class hierarchy. These probability values are guaranteed to be consistent with the underlying class hierarchy, that is, for every class label, its class label probability is always bigger or equal than the class label probability of its children. Therefore, it is the user's responsibility to choose a threshold to get a list of 'crisp' predictions. This work avoids the necessity of choosing a subjective threshold by using, as predictive measure, the area under the precision-recall curve (AUPRC) predictive measure, which considers *all* threshold values for its calculation.

The Clus-HMC algorithm is the most popular version of the PCT algorithm [9]. There is also a corresponding ensemble version using the PCT algorithm as the 'base' hierarchical classifier, called Clus-HMC-Ens [1], which can be considered the state-of-the-art for hierarchical classification, consistently achieving high predictive performance.

LHC algorithms have the advantage of algorithmic simplicity, since they transform the original hierarchical classification problem into a set of simpler flat classification problems in the training phase, but they produce a large number of local (flat) classifiers, one for each class node or one for each parent node in the class hierarchy, depending on the approach used. In this work, LHC learns one local classifier for each class node, a simpler and more popular approach. Conversely, global hierarchical classification algorithms have the advantage of producing a single classification model, which tends to be more easily interpreted than a large number of local classifiers.

Our meta-learning experiments, described later, have the objective of predicting the predictive accuracy rank of the three previously defined hierarchical classification algorithms (LHC, PCT and PCTEN) when they are applied to our hierarchical classification datasets (the meta-instances).

These 3 algorithms were selected based first on their popularity, being commonly used in hierarchical classification problems, and second by their empirical performances in the full version of the datasets used in this paper. Arguably, adding more than 3 hierarchical classification algorithms to our study would have two drawbacks. First, it would lead to some meta-class labels to be associated with a relatively small number of meta-instances – since in general, the higher the number of meta-class-labels, the smaller the number of meta-instances per meta-class-label. Second, it would result in meta-models that are too complex for a careful manual analysis of the results, like the one being done in this paper.

We have tested 8 hierarchical classification algorithms: PCT [5], PCTEN [1], HDN-PCT [10], HDN [10], PCT-LHC [11], LHC [6], HDN-nHPC [12] and ELHNB [13]. The PCTEN and LHC algorithms were the best algorithms in terms of mean average rank across the three measures of predictive performance (variations of the AUPRC measure) used in this paper. Table I shows the average mean rank across predictive measures.

The PCT algorithm also performed well and was added due to its popularity and because its ensemble version, the PCTEN, was included: it is interesting to investigate when the PCT algorithm is recommended instead of its ensemble version (PCTEN), since ensemble versions of classification algorithms tend to have better predictive performance than their standard counterparts. In addition, the PCT algorithm has, by far, the

best interpretability potential among the 8 tested hierarchical classification algorithms. Although the PCT models were not interpreted in this paper, it is useful to investigate when the PCT model performs better, given that model interpretability is important in many application domains [14], [15].

B. Predictive Accuracy Measures

Special measures of predictive accuracy have been developed for hierarchical classification; we have used three of them: $AU(\overline{PRC})$, $\overline{AUPRC}_w$, and $\overline{AUPRC}$ [9]. These three measures are inspired by the measure $AUPRC$ (Area Under the Precision Recall Curve), created for 'flat' classification with probabilistic outputs. The $AUPRC$ measure works by creating, for each class, a PR curve (a plot of the classifier's precision as a function of its recall). This curve is created by thresholding the output (class probability) of the classifier using values in the interval $[0, 1]$. Each threshold defines a set of predictions, which in turn have a value of precision and recall. Each precision and recall pair are coordinates in the PR space, so if one connects the PR points created by varying the threshold, one can create a curve, which corresponds to how the classifier's precision varies as a function of its recall. Finally, we calculate the area under this curve using a trapezoidal approximation [16]. The bigger the area the better the classifier. A perfect classifier would have an $AUPRC$ of 1.0.

The $AU(\overline{PRC})$ is calculated by using hierarchical forms of the precision and recall measures for a pre-established threshold, defined by the following formulas:

$hP \equiv \frac{\sum_j |P_j \cap T_j|}{\sum_j |P_j|}$ and $hR \equiv \frac{\sum_j |P_j \cap T_j|}{\sum_j |T_j|}$.

Where P_j and T_j denote respectively the sets containing the predicted and true classes of the current j-th instance.

To calculate $\overline{AUPRC}$ we average all the class-wise $AUPRC$ values. Similarly, to calculate $\overline{AUPRC}_w$, we calculate the $AUPRC$ of each class and then average over all classes weighted by the number of instances in each class, that is, $\overline{AUPRC_w} \equiv \frac{\sum_i AUPRC_i \times S_i}{\sum_i S_i}$; where S_i is the number of instances in the i-th class.

C. Meta-Learning

Meta-learning for algorithm recommendation, in the classification setting, is the computational task of predicting the performance of classification algorithms (representing meta-classes) given their past performance and meta-features that describe the characteristics of datasets (meta-instances).

Broadly speaking, there are three types of meta-features: 1) dataset-derived meta-features [17], 2) landmarking meta-features [17] and 3) sampling meta-features [18]. The first type of meta-features characterise some aspect of the dataset, such as the number of instances, the number of classes, the class distribution, and so on, which can be directly extracted from the dataset, without inducing a classification model. Landmarking meta-features are defined as features characterising some classification model induced using the dataset. Typically, this classification model must be relatively computationally fast to induce and should provide insights about the classification problem at hand. The last type of meta-feature, sampling meta-features, consists of applying the classification algorithms whose performance are being predicted in a sample of the testing dataset and using the predictive accuracy results as meta-features.

Both landmarking and sampling approaches aim to extract meta-features from classification models that can be induced fast. Note, however, that landmarking approaches usually use a *fast classification algorithm* and extract meta-features from the model that was induced using the full base dataset (meta-instance). For instance, if the model is a decision tree, structural descriptors of the tree can be used as meta-features. By contrast, sampling approaches can use slower classification algorithms trained on a small sample of each base dataset (meta-instance). Usually, sampling approaches use as meta-features the predictive accuracy of the classification models on dataset samples, instead of characteristics of the model. Commonly, the set of classification algorithms used in the sampling approach is the same set as the set of algorithms that can be recommended. The principle is that the accuracy on a dataset sample is a good predictor of the accuracy on the full dataset.

There are more refined meta-features. For instance, in [19] the authors induce a decision tree at the meta-level and use the boolean value of each rule (each path from the root node to a leaf node) as a meta-feature. In this work the authors also propose the Approximate Ranking Trees (ART) ensemble method, which is an adaptation of decision tree algorithms to predict algorithms' ranks. In [20] the authors build a learning curve by plotting predictive performance against sampling successively larger samples from the datasets (meta-instances). Next they analyse the behaviour of this learning curve to recommend the best classification algorithm for a new meta-instance.

In [21] authors propose an *active testing* framework to choose the best algorithm to be applied to a new meta-instance. Their algorithm works by doing several tournament-style comparisons between the current best classifier and a promising competitor. Their objective is to minimize the number of models that must be trained to get a reasonably good algorithm recommendation.

Note that all these previous works addressed only the conventional classification task. By contrast, this work addresses the more challenging hierarchical classification task.

The meta-learning approach proposed in this work has two goals. First, we want to build meta-classification models with a high predictive accuracy, in order to provide reliable algorithm recommendations to the user. Hence, we do not employ predictive accuracy measures that combine runtime and predictive performance, such as the ones presented in [17], [20], [22], as optimizing runtime often reduces classification performance. Second, we want to discover general (and meaningful) relationships between the meta-features representing characteristics of the hierarchical classification datasets and the hierarchical classification algorithms (meta-classes) used in our experiments. These relationships could be used as explanations for the algorithm recommendations output by the meta-learning system.

TABLE I
MEAN RANK OF HIERARCHICAL CLASSIFICATION ALGORITHMS FOR EACH PREDICTIVE MEASURE USED IN THIS PAPER USING THE FULL DATASETS (THE TOP PART OF THE TABLE) AND THE AVERAGE MEAN RANK ACROSS PREDICTIVE MEASURES (THE BOTTOM PART OF THE TABLE). THE MEAN RANK IS CALCULATED FOR EACH PREDICTIVE MEASURE BY FIRST RANKING THE HIERARCHICAL CLASSIFICATION ALGORITHMS FROM BEST (RANK OF 1) TO WORST (RANK OF 8) FOR A GIVEN DATASET; NEXT, WE AVERAGE THE RANKS ACROSS DATASETS TO CALCULATE THE MEAN RANK FOR EACH ALGORITHM. THE AVERAGE MEAN RANK IS CALCULATED BY SIMPLY AVERAGING THE PREVIOUSLY DESCRIBED PER-MEASURE MEAN RANKS. THE BOLD NUMBERS HIGHLIGHT THE BEST HIERARCHICAL CLASSIFICATION ALGORITHMS ACCORDING TO THEIR AVERAGE MEAN RANK

Predictive Measure	PCT	PCTEN	HDN-PCT	HDN	PCT-LHC	LHC	HDN-nHPC	ELHNB
$AU(\overline{PRC})$	4.7	2.6	5.3	6.5	4.1	3.0	2.5	7.3
$\overline{AUPRC_w}$	4.5	3.5	5.0	5.6	4.1	3.9	3.7	5.7
$\overline{AUPRC}$	3.7	4.4	4.2	5.3	3.6	3.6	5.8	5.5
Avg. Mean Rank	4.3	**3.5**	4.8	5.8	3.9	**3.5**	4.0	6.2

To the best of our knowledge, there is only one related work on meta-learning for algorithm recommendation in hierarchical classification. This work is based on the local hierarchical classification approach, using meta-learning for recommending the best local '*flat*' classification algorithm for predicting each class in the hierarchy, using meta-features to describe each local '*flat*' classification problem [23]. More precisely, the authors use meta-features to select between the SVM or Naive Bayes algorithms for predicting each class label in the hierarchy using a C4.5 meta-learner. Note the previouly cited work is very different from our work, which performs meta-learning using meta-features that describe *hierarchical* classification datasets and recommends *hierarchical* classification algorithms.

III. DEFINITION OF THE PROPOSED META-FEATURES

The proposed meta-features for describing properties of hierarchical classification datasets are divided into three broad types: simple *multi-label dataset-derived meta-features*, *hierarchical dataset-specific meta-features*, and *meta-features extracted from the landmarking PCT classification model*.

The meta-feature type describes properties of a multi-label classification dataset [24], without referring to hierarchical classification aspects. However, since every hierarchical classification dataset is implicitly a multi-label dataset [3] (an instance is assigned class labels at multiple levels of the class hierarchy), such meta-features are still potentially useful to describe hierarchical classification datasets.

The motivation behind these meta-features is to capture high-level dataset characteristics such as the number of classes, instances and features. These meta-features are the easiest to interpret but lack preciseness since they are unaware of the class hierarchy.

The second meta-feature type is generated by inducing a PCT hierarchical classification model using the base dataset and extracting meta-features directly from the induced decision tree. These meta-features are broadly based on the meta-features proposed in [17] for standard (flat) classification.

The intuition behind these meta-features is that the characteristics of the PCT model induced using the dataset (meta-instance) will reflect characteristics of the underlying classification problem. For instance, an 'easy' meta-instance will tend to have a 'simple' hierarchical classification model, whereas a 'hard' meta-instance will tend to generate 'complex' hierarchical classification models. The main advantage of using this feature type is that it can measure the 'complexity' of the meta-instance. The main downside is that it is hard to interpret. It is not always clear what is a 'simple' or 'complex' model.

These first two types of meta-feature have been used before in *flat* classification, but they are used here in the more complex task of *hierarchical* classification.

The third meta-feature type describes the characteristics of the graph that represents the class hierarchy [3]. This is a meta-feature type proposed specifically for meta-learning in hierarchical classification, with no equivalent in meta-learning for flat classification.

The motivation of this meta-feature is to characterize the class hierarchy, so its topology can be taken into account when choosing which hierarchical classification algorithm to use. The description of the class hierarchy can provide important information about the meta-instance (dataset), as different hierarchical classification algorithms can perform better when the characteristics of the class hierarchy change.

1) Multi-label dataset-derived meta-features
   a) NumClasses: Number of class labels.
   b) LabCard (label cardinality): Average number of class labels per instance.
   c) DistLabSetSize: Number of distinct label sets that occur in at least one instance.
   d) NumFeats: Number of features.
   e) NumInsts: Number of instances.
   f) InstFeatRatio: Number of instances divided by the number of features.
2) Meta-features extracted from the landmarking PCT hierarchical classification model
   a) NumNodesPCT: Number of nodes in the decision tree.
   b) NumLeavesPCT: Number of leaves in the decision tree.
   c) MaxLevelSizePCT: Maximum number of internal nodes in a level of the decision tree, across all tree levels.
   d) MeanLevelSizePCT: Mean number of internal nodes in a level of the decision tree across all levels of the tree.
   e) LongBranchPCT: Longest path among all possible paths from the root to a leaf node of the decision tree.
   f) ShortBranchPCT: Shortest path among all possible paths from the root to a leaf node of the decision tree.
   g) MeanBranchPCT: Mean path length among all possible paths from the root to a leaf node of the decision tree.
   h) PercSelPCT: % of input features selected for inclusion

in the decision tree induced by PCT.

i) BalancednessPCT: This measures how distant the PCT tree (T) is from a balanced tree, defined as: $balancedness(T) = \frac{(unbAvgDepth(|T|) - actualAvgDepth(T))}{(unbAvgDepth(|T|) - balAvgDepth(|T|))}$. where: $unbAvgDepth(|T|)$ is the average depth of a completely unbalanced tree (worst case scenario) containing $|T|$ nodes, $balAvgDepth(|T|)$ is the average depth of the most balanced binary tree possible (best case scenario) containing $|T|$ nodes, and $actualAvgDepth(T)$ is the actual average depth of the tree T. This meta-feature has value 1 (0) if T is the most balanced (unbalanced) tree possible.

3) Hierarchical dataset-specific meta-features

a) AvgDepth: The average length of all possible paths from the root to all the leaf classes.

b) ClassImbal: Average class imbalance, defined as the average proportion of positive class instances across all class nodes in the hierarchy. Recall that a hierarchical classification problem can be viewed as a collection of binary classification problems, with restrictions defined by the class hierarchy.

c) NumLeaves: Number of leaf class labels.

d) HierType: Class hierarchy's structure type (tree or DAG).

e) AvgDegree: Average degree (# of edges) per class node.

f) MaxDegree: Maximum degree across all class nodes.

g) MaxLevelSize: Maximum number of nodes in a class level, across all levels. A class label is in the i-th level if there is a path of length "i" from the root to that class label's node. Note: for DAGs the same node may be in multiple levels.

h) MinLevelSize: Minimum number of class nodes in a level across all levels.

i) MeanLevelSize: Mean number of class nodes in a level across all levels.

j) LongBranch: Longest path among all possible paths from the root to a leaf class.

k) ShortBranch: Shortest path among all possible paths from the root to a leaf class.

l) MeanBranch: Mean path length among all possible paths from the root to a leaf class.

IV. Hierarchical Datasets Used in This Work

The first requirement to apply meta-learning for hierarchical classification algorithm recommendation is to collect a reasonable number of datasets (meta-instances) to learn associations between dataset characteristics (the meta-features) and the choice of the best hierarchical classification algorithms (meta-classes) for the datasets.

To this end, we have collected 42 hierarchical classification datasets from 3 different sources. The first source is the work of Vens [9], from where we collected 22 of the 24 available datasets. We discarded the datasets pheno_GO and pheno_FUN, since they contain many (more than 50%) missing values, and adding them would require a non-trivial adaptation of the hierarchical classifiers we used.

These datasets contain features extracted from the genes of the widely used model organism *Saccharomyces cerevisiae* (yeast). There are two types of predictive features: 1) statistics extracted from the amino acid sequences (seq features) and 2) several types of microarray expression data (all the other features). There are also two types of class hierarchies: "FUN" (the tree-structured hierarchy in the FunCat scheme [25]) and "GO" (the DAG-structured Gene Ontology [7]).

The second hierarchical classification dataset source is a work of Fabris [11], from where we collected 15 datasets containing features extracted from the proteins encoded by the genes in the Ageing Gene Database (GenAge) [26]. GenAge is a catalogue of ageing-related genes coming from several species, including human and model organisms such as *S. cerevisiae* (baker's yeast) and *M. musculus* (the house mouse).

Each species is associated with three datasets containing three broad types of features, namely numeric alignment independent features, protein motif features and protein-protein interaction features, leaving us with 15 ageing-related datasets. The hierarchical classes were created for each model organism by retrieving the over-expressed GO terms associated with the genes (instances) in each one of the 15 datasets.

The last hierarchical classification dataset source is also from a work of Fabris [27], where the authors built 5 hierarchical classification datasets containing features extracted from the proteins encoded by the genes in the *Phenotypes and Mutant Alleles* section of the *Mouse Genome Informatics* (MGI) database. The hierarchical classes of these datasets are classes of the *Mammalian Phenotype Ontology* (MPO). The datasets vary in terms of feature type, namely: numeric features, protein motifs features, Protein-Protein Interaction (PPI) features, and two types of KEGG pathway features.

Tables II and III show the main characteristics of the 42 datasets used in this work. We encourage readers interested in the details of the constructions of these datasets to refer to the original papers. The original hierarchical datasets used in this work are available at [28] and [29].

TABLE II
Number of Instances, Predictive Features and Classes in the Vens' Datasets Used in This Work. Note That Although the Table Has 11 Entries, the Total Number of Hierarchical Datasets Is 22, 11 for Each Type of Class Hierarchy (GO and FunCat)

Predictive Features	Number of FunCat Classes	Number of GO classes	Number of Instances	Number of Features
seq	476	3704	3932	478
cellcycle	476	3695	3766	77
church	476	3696	3764	27
derisi	476	3691	3733	63
eisen	447	3176	2425	79
gasch1	476	3698	3773	173
gasch2	476	3698	3788	52
spo	476	3691	3711	80
expr	476	3698	3788	551
struc	476	3703	3851	19628
hom	476	3695	3867	47034

TABLE III
NUMBER CLASSES, INSTANCES, AND PREDICTIVE FEATURES IN THE FABRIS' DATASETS

Hier.	Feature Type		Number of Classes	Number of Instances	Number of Features
Ageing GO	worm	Numeric	350	263	59
		PPI			162
		Motifs			112
	fly	Numeric	385	79	59
		PPI			105
		Motifs			55
	human	Numeric	1713	301	60
		PPI			2425
		Motifs			284
	mouse	Numeric	683	107	59
		PPI			29
		Motifs			40
	yeast	Numeric	583	762	59
		PPI			4397
		Motifs			296
Ageing MPO	KEGG		84	3886	221
	KEGGI				618
	Motifs				372
	Numeric				59
	PPI				123

V. AN ALGORITHM FOR SPLITTING HIERARCHICAL DATASETS FOR META-LEARNING

One of the main challenges of applying meta-learning to data mining tasks is collecting a reasonable number of datasets to be used as meta-instances to train the meta-classifiers. This problem is exacerbated when dealing with hierarchical classification problems, as there are substantially fewer freely available datasets for this specific task than for standard ('flat') classification tasks. Actually, there is no systematic repository of hierarchical classification datasets, unlike the case for standard classification, where there are well-known dataset repositories like the UCI one [30].

Hence, we propose an approach for creating a larger number of hierarchical classification datasets from an existing set of available hierarchical classification datasets. The created datasets preserve part of the data contained in the original datasets, i.e., they still contain real-world data, rather than being synthetic, randomly generated datasets. We have applied this approach (formalized by Algorithm 1) to divide the original 42 hierarchical classification datasets into 863 new hierarchical classification datasets. Broadly speaking, for each original dataset, Algorithm 1 divides the existing class label hierarchy into sub-hierarchies and creates a hierarchical classification dataset (a meta-instance) for each sub-hierarchy. Each instance in the full original dataset is present in a hierarchical classification dataset if that instance is annotated with at least one class label from the sub-hierarchy corresponding to that new dataset.

Algorithm 1 requires a user-defined "spanning set". A "spanning set" is a set of class labels that Algorithm 1 uses to decide how to "break down" the class hierarchy: the children of the class labels in this set define the unique classes of the new sub-hierarchies, i.e., these children will not be shared between the new sub-hierarchies. The classes in the spanning set must be chosen based on the structure of the hierarchy: a good spanning set contains class labels with a reasonable number of child class labels (so that a considerable number of sub-hierarchies can be generated) and close to the hierarchy's root node (so that each generated sub-hierarchy can have a reasonable number of instances). In this work, the class labels in the spanning set have the characteristic that they cover a reasonable number of instances (more than 10) and yet are not so shallow as to generate few datasets (meta-instances). The full list of class labels forming the spanning set will be made available when the paper is published.

Algorithm 1 works by iterating over the children of the class labels in the "spanning set" (line 5) and checking if the current child node has less than $minDesSize$ (minimum Desirable Size) instances (line 6) – a user-defined parameter. If this is the case, the instances in the current child node are added to the temporary set $toGenerate$ (line 7). Notice that if an instance belongs to multiple classes, it cannot appear multiple times in this set; this is guaranteed by the append operator (line 7). If $toGenerate$ has accumulated $minDesSize$ or more instances (line 8), a new hierarchical classification dataset containing the instances in $toGenerate$ is created (line 9) and the algorithm proceeds to iterate over the next child node of a class node in the spanning set.

If the current child class node has $minDesSize$ instances or more (i.e., the *if* test in line 6 fails), a new hierarchical classification dataset is created using that child node and its descendants (line 13). Finally, when the *for* loop ends, the algorithm checks if the current number of instances in $toGenerate$ is greater than or equal to $minAccSize$ (minimum Acceptable Size) – another user-defined parameter. If that is the case, a new hierarchical classification dataset is created (line 17); otherwise the instances are discarded, as we consider that a dataset with less than $minAccSize$ instances does not contain enough information to be used in our experiments.

Algorithm 1 Split a hierarchical classification dataset into many smaller ones

```
1: procedure SPLIT(span (The spanning set))
2:     minDesSize = 200
3:     minAccSize = 20
4:     toGenerate = {}
5:     for each child ∈ span.children do
6:         if |child.instances| < minDesSize then
7:             toGenerate.append(child.instances)
8:             if |toGenerate| ≥ minDesSize then
9:                 generateDS(toGenerate)
10:                toGenerate = {}
11:            end if
12:        else
13:            generateDS({child.instances})
14:        end if
15:    end for
16:    if |toGenerate| ≥ minAccSize then
17:        generateDS(toGenerate)
18:    end if
19: end procedure
```

In our experiments we have set the value of $MinDesSize$ to 200 and $MinAccSize$ to 20. Thus, the algorithm will first generate hierarchical datasets with at least 200 instances

when executing the *for* loop. At the end of the algorithm, it will generate at most one dataset with at least 20 instances with class labels that did not pass the $MinDesSize$ criterion, in order to avoid discarding instances unnecessarily. The meta-datasets created and used in our experiments will be freely available on the *web* when the paper is published.

VI. Experimental Setup

Our approach consists of inducing a multi-class meta-classifier using the meta-features presented in Section III and using the name of the best hierarchical classifier for a particular meta-instance (hierarchical dataset) as the meta-classes. We use 10-fold cross validation to estimate the accuracy of hierarchical classification algorithms. This multi-class meta-classifier must be capable of outputting a score that represents the likelihood of a meta-instance belonging to each of the meta-classes. The scores are used to determine the recommended algorithm ranking, i.e., for each meta-instance, the hierarchical classification algorithm with the highest score is ranked first, and so on.

In order to induce the meta-classifier, we use two multi-class classification algorithms: a Support Vector Machine (SVM) [31] and the C4.5 decision tree algorithm [32]. These algorithms have complementary characteristics: SVM models usually have high predictive accuracy, but are difficult to interpret; while the C4.5 algorithm often produces models that are associated with good interpretability but with inferior predictive accuracy when compared with SVM – although of course the issue of which algorithm is more accurate depends on the underlying dataset. Actually, in our experiments J48 performed slightly better than SVM, as reported later. We have used the J48 implementation of C4.5 from the Weka data-mining framework [33] and the SVM from LibSVM [34]. We have used the default parameters for the J48 algorithm and the *Gaussian* kernel for the SVM algorithm, using Weka's Grid Search implementation to select the best values for the parameters γ and C, using the intervals suggested in [34].

In summary, the SVM algorithm implicitly maps the original problem to a high-dimensional space using kernel functions and finds a meta-class-separation hyperplane on this space that minimizes classification error. The J48 algorithm builds a decision tree that recursively divides the data using the feature-based condition that best separates the meta-classes of the meta-instances. We call our meta-learner using the J48 algorithm the Decision Tree Meta-Ranker (DTMR) and our meta-learner using the SVM algorithm the SVM Meta-Ranker (SVMMR).

We have used the three predictive accuracy measures defined in Section II-B to define our meta-classes. Because the measures have different biases, each one leads to a different ranking for the hierarchical classifiers. Therefore we created three meta-datasets, one for each predictive accuracy measure. That is, the meta-features present in the three meta-datasets are the same, but the meta-classes are different: The meta-class of each meta-instance in each of the three meta-datasets is the hierarchical classifier with the highest predictive accuracy for the particular measure we are considering ($AU(\overline{PRC})$, $\overline{AUPRC}_w$, and $\overline{AUPRC}$). Recall that the three hierarchical classifiers used as meta-classes are PCT, PCTEN and LHC (see Section II-A).

In addition to the DTMR and SVMMR rankers, we also test two simple baselines: the first is a simple naive classification algorithm called the Prior Ranker (PR). This algorithm outputs as the predicted classifier ranking the ranking observed in the meta-training set as a whole; i.e., the hierarchical classifier with most wins across all training meta-instances is assigned rank 1, with the second and third best classifiers being assigned ranks 2 and 3. The second baseline is the Random Ranker (RR), which randomly assigns the rankings of the hierarchical classifiers to each meta-instance.

VII. Experimental Results

This section presents the meta-learning results of applying the approach for inducing our meta-classifiers (as described in Section VI) to our meta-instances generated using Algorithm 1 (described in Section V). Subsection VII-A presents the predictive accuracy results, using 10-fold cross-validation and the *Spearman's rank correlation coefficient* measure of predictive accuracy [35]. This measure is often used in meta-learning research. In Subsection VII-B we interpret the meta-models induced by J48 using the whole meta-dataset to try to extract useful information from those meta-models.

A. Meta-Learning Performance Evaluation

The measure of predictive performance we are using is the mean *Spearman's rank correlation coefficient* ($\bar{R}$) across all the datasets. $\bar{R}$ measures the agreement between the ranking of the base hierarchical classification algorithms predicted by the meta-classifier and their corresponding actual ranking, and it is defined as [35]:

$$\bar{R} = \frac{1}{J}\sum_{j=1}^{J}\left[1 - \frac{6\sum_{a=1}^{A}(pr_{j,a} - ar_{j,a})^2}{A^3 - A}\right]. \quad (1)$$

Where J is the number of meta-instances (hierarchical classification datasets), A is the number of base hierarchical classification algorithms, $pr_{j,a}$ is the predicted rank for the a-th base hierarchical classification algorithm in the j-th meta-instance, and $ar_{j,a}$ is the actual rank for the a-th base hierarchical classification algorithm in the j-th meta-instance. The multi-class meta-classification models we are using do not output meta-class ranks, however they output scores, which can be simply transformed to ranks by ordering the scores from the largest (most probable meta-class) to the smallest (least probable meta-class).

The $\bar{R}$ correlation measure lies in the interval $[-1, 1]$. $\bar{R} = 1$ means that there is a perfect agreement between the predicted and actual ranks, $\bar{R} = 0$ means that there is no correlation between the predicted and actual ranks, and $\bar{R} = -1$ means that there is a perfect disagreement between the predicted and actual ranks. This ranking correlation coefficient has been used in several works dealing with ranking-based meta-learner evaluation, e.g.: [17], [19], [36].

Table IV shows the results of applying our meta-rankers (DTMR and SVMMR), the Prior Ranker (PR), and the Random Ranker (RR) to our three meta-datasets.

TABLE IV
RANK CORRELATION COEFFICIENT RESULTS OF APPLYING THE DTMR, SVMMR, PR, AND RR META-RANKERS TO OUR META-DATASETS, ONE META-DATASET FOR EACH HIERARCHICAL CLASSIFICATION VERSION OF THE AUPRC MEASURE, USING 10-FOLD CROSS-VALIDATION

	R rank correlation			
	DTMR	SVMMR	PR	RR
$AU(\overline{PRC})$	0.600	0.546	0.386	-0.013
$\overline{AUPRC}_w$	0.446	0.411	0.246	-0.013
$\overline{AUPRC}$	0.437	0.442	0.122	-0.008

Table IV shows that the DTMR meta-ranker is superior to every other meta-ranker we tested in two out of three predictive accuracy measures. The only exception is when using the $\overline{AUPRC}$ measure, when the SVM classifier performed marginally better.

We have applied pairwise *paired t-tests* to the results of the 10 folds of the cross-validation procedure, as suggested in [37], to compare the results of DTMR against each of the other baseline approaches. The test rejected the null hypothesis of classifier equivalence in respect with PR and RR, with $\alpha = 0.05$, with all p-values $< 10^{-4}$.

When comparing DTMR and SVMR, the test has detected a statistical difference when considering the $AU(\overline{PRC})$ measure ($p = 0.03$) but did not detect statistically significant differences when considering the $\overline{AUPRC}_w$ and $\overline{AUPRC}$ measures ($p = 0.18$ and $p = 0.86$, respectively).

In addition, to confirm the usefulness of Algorithm 1, the ranking results when using the 42 original datasets (without using Algorithm 1) were, as expected, much worse: both the DTMR and SVMMR were statistically equivalent to the rather simple PR algorithm in every occasion.

B. Interpreting the Meta-Models

The meta-classification models induced by J48 to predict which hierarchical classification algorithm (meta-class) is more accurate in each dataset (meta-instance) are shown in Figs. 1, 2, and 3, for the 3 accuracy measures we have used, respectively: $AU(PRC)$, $\overline{AUPRC}_w$ and $\overline{AUPRC}$. These meta-models were induced by J48 using the whole meta-dataset, maximizing J48's potential to find interesting meta-classification rules. We interpret the J48 model instead of the SVM model because the J48 model is much easier to interpret and it has achieved at least statistically equivalent predictive performance for all three measures. In addition to showing each meta-model, we select some interesting meta-rules, interpret their meaning and, when possible, compare them with similar meta-rules found when using different accuracy measures, to reach conclusions across the 3 measures.

In Figs. 1-3, each line of the meta-model represents a decision split, i.e., a condition that must be satisfied by the meta-feature of a meta-instance in order for it to be passed to the next decision split, eventually reaching a leaf node, when a meta-classification is made. On the leaf nodes

```
InstFeatRatio ≤ 0.02: PCT (43.0/4.0)
InstFeatRatio > 0.02
| NumFeats ≤ 551
| | HierType = Tree
| | | AvgDepth ≤ 2.19: PCTEN (29.0/9.0)
| | | AvgDepth > 2.19: LHC (228.0/59.0)
| | HierType = DAG
| | | NumNodesPCT ≤ 44
| | | | LabCard ≤ 10.11
| | | | | AvgDegree ≤ 2.21: PCTEN (20.0/4.0)
| | | | | AvgDegree > 2.21: PCT (44.0/21.0)
| | | | LabCard > 10.11
| | | | | AvgDepth ≤ 6.76
| | | | | | NumInst ≤ 353
| | | | | | | MeanLevelSize ≤ 20.1
| | | | | | | | AvgDegree ≤ 3.39: PCTEN (130.0/45.0)
| | | | | | | | AvgDegree > 3.39: LHC (42.0/12.0)
| | | | | | | MeanLevelSize > 20.1: PCTEN (138.0/42.0)
| | | | | | NumInst > 353: LHC (21.0/6.0)
| | | | | AvgDepth > 6.76: LHC (32.0/9.0)
| | | NumNodesPCT > 44
| | | | NumInst ≤ 1077
| | | | | MaxDegree ≤ 13: LHC (20.0/5.0)
| | | | | MaxDegree > 13: PCTEN (12.0/2.0)
| | | | NumInst > 1077: LHC (50.0)
| NumFeats > 551: PCTEN (53.0/8.0)
```

Fig. 1 Meta-model generated to classify the meta-instances into meta-classes PCTEN, PCT, and LHC, when considering the $AU(\overline{PRC})$ measure

of the meta-model we show in general two values, the first is the total number of meta-instances classified by the leaf node, the second is the number of meta-instances misclassified by that leaf node. If there is no second value, it means there is no misclassified meta-instance in that node.

In the next section we evaluate some meta-classification rules in terms of their *precision* and *recall*. Precision is the number of correct predictions (*true positives* predictions) made by the rule divided by the number of meta-instances covered by the rule. Recall is the number of correct predictions made by the rule divided by the total number of meta-instances with the meta-class predicted by the rule.

1) Interpreting the $AU(\overline{PRC})$-Based Meta-Model: Fig. 1 shows the meta-model induced when considering the $AU(\overline{PRC})$ measure. The first line of the model encodes the meta-rule: "If the instance to feature ratio ($InstFeatRatio$) is smaller than or equal to 0.02, i.e., the base dataset has one instance for every 50 or more features, use the PCT classifier". This meta-rule has high precision (0.91), much higher than the *a priori* probability for the 'PCT' meta-class (0.13). This meta-rule's recall is also reasonably good, covering 35% (43 out of 122) of all meta-instances annotated with the 'PCT' meta-class. This meta-rule indicates that the decision tree-based PCT algorithm deals well with datasets with relatively few instances and many features, which seems due to its implicit class hierarchy-aware feature selection procedure. That is, by finding successive conditions that divide the set of meta-instances based on their different hierarchical classes well, the PCT performs feature selection by analysing each feature's predictive power across a large set of hierarchical classes, instead of analysing just one class at a time, like the LHC algorithm does.

Although the advantage of the PCT classifier over the LHC classifier is clear when the instance to feature ratio

is so small, it is not clear why the PCTEN classifier did not perform so well on the datasets with $InstFeatRatio \leq 0.02$. Upon further analysis, we have concluded that such a low $InstFeatRatio$ is correlated with a simpler classification problem, that is, hierarchical classification datasets (meta-instances) with low $InstFeatRatio$ also tend to have much lower average values for $NumClasses$, $NumInst$, $NumLeaves$, and $DistLabSetSize$. This justifies why PCTEN was not the best performing algorithm for these datasets, i.e., given the relative simplicity of these datasets, the power of the PCTEN ensemble is not needed to maximize accuracy. To show this point, Table V shows the average values of the above meta-features considering the full meta-dataset and the subset of meta-instances with $InstFeatRatio \leq 0.02$.

TABLE V
MEAN VALUE OF SOME META-FEATURES (FIRST COLUMN) IN THE WHOLE META-DATASET (SECOND COLUMN) AND IN THE LEAF NODE IN THE FIRST LINE OF THE META-MODEL SHOWN IN FIG. MOD:ALL$_m odel, WhereAllMeta - InstancesHave$INSTFEATRATIO 0.02$(ThirdColumn)$

Feature Name	Mean in meta-dataset	Mean in leaf
$InstFeatRatio$	5.63	0.01
$NumClasses$	152.27	85.95
$NumInst$	530.52	217.57
$NumLeaves$	47.08	23.95
$DistLabSetSize$	72.52	31.24

The last line of the decision tree shown in Fig. 1 contains another interesting meta-rule: 'if the instance to feature ratio ($InstFeatRatio$) is greater than 0.02, and the number of features ($NumFeats$) is greater than 551, use PCTEN'. This points to the advantage of using PCTEN when the problem is more complex (higher number of features), but with enough instances to learn from (with a not too small $InstFeatRatio$). This meta-rule also has a high precision of 0.85, much higher than the *a priori* meta-class probability of 0.40. It also has a reasonable recall of 13% of all meta-instances annotated with the 'PCTEN' meta-class.

The last meta-rule from Fig. 1 that we would like to highlight is as follows.

```
IF (InstFeatRatio > 0.02) AND (NumFeats ≤ 551)
   AND (HierType = DAG) AND (numNodesPCT > 44)
   AND (NumInst > 1077) THEN LHC (50.0)
```

Overall, this meta-rule seems to suggest that the LHC classifier is clearly recommended when the problem is moderately difficult (relatively many instances, not very many features, a DAG class hierarchy) and the number of tree nodes in the landmark PCT model (used as a meta-feature) is relatively large, suggesting that the PCT model may be overfitting to this problem. Note that this meta-rule has a precision of 100% on the meta-dataset and a reasonable recall, capturing 12% (50) of all meta-instances annotated with the 'LHC' meta-class label.

2) Interpreting the $\overline{AUPRC}_w$-Based Meta-Model: Analysing the meta-model shown in Fig. 2 we observe that the same condition that the J48 algorithm selected to predict the 'PCT' meta-class using the $AU(\overline{PRC})$ measure was selected again to predict the 'PCT' meta-class for the $\overline{AUPRC}_w$ measure. It is interesting that both models chose this condition as the root of the decision tree, highlighting the

```
InstFeatRatio ≤ 0.02: PCT (43.0/2.0)
InstFeatRatio > 0.02
| MeanLevelSizePCT ≤ 2.5
| | ClassImbal ≤ 0.60
| | | ShortBranchPCT ≤ 1: PCT (144.0/58.0)
| | | ShortBranchPCT > 1
| | | | NumLeaves ≤ 82
| | | | | NumLeaves ≤ 8: LHC (31.0/10.0)
| | | | | NumLeaves > 8
| | | | | | AvgDegree ≤ 3.60: PCT (171.0/93.0)
| | | | | | AvgDegree > 3.60: LHC (13.0/6.0)
| | | | NumLeaves > 82: LHC (15.0/3.0)
| | ClassImbal > 0.60: PCTEN (16.0/2.0)
| meanLevelSizePCT > 2.5
| | NumFeats ≤ 2425: LHC (412.0/137.0)
| | NumFeats > 2425: PCTEN (17.0/3.0)
```

Fig. 2 Meta-model generated to classify the meta-instances into meta-classes PCTEN, PCT, and LHC, when considering the $\overline{AUPRC}_w$ measure

importance of the meta-feature $InstFeatRatio$. Note also that the same threshold of 0.02 was consistently chosen for the $InstFeatRatio$ meta-feature in both Figs. 1 and 2. This meta-rule has a precision 0.95, much higher than the *a priori* meta-class probability of 0.31. In addition, this meta-rule covers the same reasonably good number of 41 meta-instances as the meta-rule for the $AU(\overline{PRC})$ measure (Section VII-B1). One difference is that, for the current meta-rule (for the $\overline{AUPRC}_w$ measure), this coverage represents a recall of only 15% of the meta-instances annotated with the 'PCT' meta-class label, rather than 35% as in the corresponding meta-rule for the $AU(\overline{PRC})$ measure. This is because the PCT algorithm performed better when considering the $AU(\overline{PRC})$ measure, being ranked first in more occasions.

For predicting the PCTEN meta-class, we highlight the following meta-rule in Fig. 2:

```
IF (InstFeatRatio > 0.02) AND (MeanLevelSizePCT > 2.5)
   AND (NumFeats > 2425) THEN PCTEN (17.0/3.0)
```

This meta-rule is broadly similar to the one we highlighted for PCTEN in the previous section. The differences are that the current meta-rule has a much larger threshold for $NumFeats$ (2425, vs. 551 for the previous meta-rule), as well as having the additional condition involving the meta-feature $MeanLevelSizePCT$, which measures the mean number of nodes across the levels of the PCT decision tree. Taken all together, this meta-rule's conditions recommend to use the PCTEN algorithm when the problem has more available data to induce a meta-model (the instances to feature ratio is not too small and the number of features is relatively high) and the landmark PCT model (decision tree) used to classify the data has more than 2.5 nodes on average, across the levels of the PCT tree. That is, when the landmark PCT model used to classify the instances is more complex and there is enough data, using an ensemble of PCT classifiers tends to be better than a single PCT classifier. This meta-rule has a high precision (0.82) compared to the *a priori* probability of the 'PCTEN' meta-class (0.22), but low recall, capturing only 7% of all meta-instances annotated with the 'PCTEN' meta-class.

Lastly, we analyse an interesting meta-rule for recommending the LHC algorithm in Fig. 2:

```
IF (InstFeatRatio > 0.02) AND (MeanLevelSizePCT > 2.5)
```

```
AND (NumFeats ≤ 2425) THEN LHC (412.0/137.0)
```

This meta-rule is similar to the meta-rule for recommending the PCTEN algorithm, only differing in the last condition, which points out that when there are not so many features (2425 or less), it is recommended using LHC instead of PCTEN. This is consistent with the meta-rule for LHC in the previous section, which also recommends using LHC when the number of features is smaller than 551, $InstFeatRatio > 0.02$ (as in the current rule), and other conditions are satisfied. This meta-rule has a very high recall, capturing 70% of the meta-instances annotated with the meta-class label 'LHC', and has a reasonable precision: 0.66, substantially higher than the *a priori* probability for the meta-class LHC, which is 0.47.

3) Interpreting the $\overline{AUPRC}$-Based Meta-Model: Analysing the meta-classification model for the $\overline{AUPRC}$ measure as shown in Fig. 3, we can drawn similar conclusions to the conclusions derived for measures $AU(\overline{PRC})$ and $\overline{AUPRC}_w$, at a high level of abstraction.

Namely, when the problem has characteristics that are commonly recognized to harm classification performance (e.g. the class distribution is very imbalanced), PCTEN is more suited to solve the problem than PCT. The following two meta-rules (in decision tree format) show this behaviour.

```
PercSelPCT ≤ 0.0004
| ClassImbal ≤ 0.60: PCT (195.0/43.0)
| ClassImbal > 0.60: PCTEN (16.0/2.0)
```

In particular, the previous meta-rule leading to the prediction of the 'PCTEN' meta-class has the precision of 0.88, much higher than the *a priori* meta-class label probability of 0.30; but a low recall, only capturing 5% of the meta-instances annotated with the 'PCTEN' meta-class.

Conversely, the previous meta-rule predicting the 'PCT' meta-class has opposite characteristics: high recall (covering 44% of all meta-instances annotated with the meta-class 'PCT') and lower precision (0.78), although that value is still relatively high, compared to the *a priori* meta-class probability (0.40).

Also, once again, the LHC classifier seems to work better when the number of features in the dataset is smaller than some threshold. The next meta-rule (which predicts the meta-class LHC) exemplify this and has very good recall (covering 92% of all meta-instances annotated with meta-class 'LHC') and reasonable precision, 0.60, compared to the *a priori* probability of the meta-class 'LHC', 0.30.

```
PercSelPCT ≤ 0.0004
| ClassImbal ≤ 0.60: PCT (195.0/43.0)
| ClassImbal > 0.60: PCTEN (16.0/2.0)
PercSelPCT > 0.0004
| NumInst ≤ 378
| | AvgDegree ≤ 1.94
| | | ClassImbal ≤ 0.19: PCT (33.0/5.0)
| | | ClassImbal > 0.19
| | | | InstFeatRatio ≤ 2.83
| | | | | NumFeats ≤ 296: PCT (14.0/2.0)
| | | | | NumFeats > 296: LHC (13.0/3.0)
| | | | InstFeatRatio > 2.83: LHC (14.0/5.0)
| | AvgDegree > 1.94
| | | MeanLevelSizePCT ≤ 2.5: PCT (150.0/75.0)
| | | MeanLevelSizePCT > 2.5: PCTEN (181.0/68.0)
| NumInst > 378
| | NumFeats ≤ 1216: LHC (235.0/93.0)
| | NumFeats > 1216: PCTEN (11.0/2.0)
```

Fig. 3 Meta-model generated to classify the meta-instances into meta-classes PCTEN, PCT, and LHC, when considering the $\overline{AUPRC}$ measure

```
IF (PercSelPCT > 0.0004) AND (NumInst > 378)
   AND (NumFeats ≤ 1216) THEN LHC (235.0/93.0)
```

Note, however, that the following meta-rules (shown in decision tree format) seem to contradict this finding:

```
PercSelPCT > 0.0004
| NumInst ≤ 378
| | AvgDegree ≤ 1.94
| | | ClassImbal > 0.19
| | | | InstFeatRatio ≤ 2.83
| | | | | NumFeats ≤ 296: PCT (14.0/2.0)
| | | | | NumFeats > 296: LHC (13.0/3.0)
```

In fact, when analysing these meta-rules' sequence of conditions in more detail, the meta-instances (datasets) that satisfy all conditions until (and including) the condition "$InstFeatRatio \leq 2.83$" have, on average, fewer (base level) features than all the meta-instances in the whole meta-dataset (344.70 vs. 640.18, respectively). So, it appears that LHC also does not work well when the number of instances is too small (less than 296 in our datasets).

VIII. Conclusions and Future Work

This work has proposed the meta-learning approach for automatically recommending the hierarchical classification algorithm for a new dataset. The three main contributions of this work are as follows. First, we have proposed meta-features (*Hierarchical dataset-specific meta-features*) for performing meta-learning in the hierarchical classification task. Second, we have proposed an algorithm for splitting a hierarchical classification dataset into many hierarchical datasets, each used as a meta-instance. We then used this algorithm to greatly increase the number of meta-instances (from 42 original meta-instances to 862 meta-instances) for our meta-learning experiments. Third, we have interpreted the induced meta-classification models, identifying for the correlations between hierarchical dataset characteristics (meta-features) and the choice of the best hierarchical classification algorithm (meta-classes) for a dataset (meta-instance).

In our experiments, the Decision Tree Meta Ranker (DTMR) was overall the best meta ranker. In addition, the DTMR method provided useful meta-knowledge about the effectiveness of three different hierarchical classification algorithms (meta-classes), as discussed in Section VII.

The meta-feature type presented in this paper, the *hierarchical dataset-specific meta-features*, was useful to predict the meta-classes, being the most selected meta-feature type in the meta-models shown in Section VII-B. The DTMR meta-learner selected a meta-feature from the proposed meta-feature type a total of 14 times, out of 30 selected meta-features.

Our approach has some limitations that will be explored in future work, namely: testing more algorithms besides J48 and SVM as meta-learners, and using more hierarchical

classification algorithms to generate the meta-classes to be predicted. We also plan to evaluate the effectiveness of each individual meta-feature in a more systematic way to identify which meta-feature has the highest predictive power using measures of feature importance for decision trees.

ACKNOWLEDGMENT

The first author is financially supported by CAPES, a Brazilian research-support agency (process number 0653/13-6).

REFERENCES

[1] 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*, vol. 11, no. 2, pp. 1–14, Jan. 2010.

[2] D. Delen, G. Walker, and A. Kadam, "Predicting breast cancer survivability: a comparison of three data mining methods," *Artificial Intelligence in Medicine*, vol. 34, no. 2, pp. 113–127, 2005.

[3] C. N. Silla Jr. and A. A. Freitas, "A Survey of Hierarchical Classification Across Different Application Domains," *Data Mining and Knowledge Discovery*, vol. 44, no. 1-2, pp. 31–72, 2011.

[4] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining*. Springer, 2008.

[5] C. Vens, L. Schietgat, J. Struyf, H. Blockeel, and D. Kocev, "Predicting Gene Function using Predictive Clustering Trees," *BMC Bioinformatics*, vol. 11, no. 2, pp. 1–25, 2010.

[6] D. Koller and M. Sahami, "Hierarchically Classifying Documents Using Very Few Words," in *Proceedings of the 14th International Conference on Machine Learning*, ser. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 170—-178.

[7] M. A. Harris, J. Clark, A. Ireland, J. Lomax *et al.*, "The Gene Ontology (GO) database and informatics resource." *Nucleic Acids Research*, vol. 32, pp. D258–61, Jan. 2004.

[8] H. Blockeel, M. Bruynooghe, S. Dzeroski, J. Ramon, and J. Struyf, "Hierarchical Multi-Classification," in *Proceedings of the ACM SIGKDD 2002 workshop on multi-relational data mining (MRDM 2002)*, 2002, pp. 21–35.

[9] C. Vens, J. Struyf, L. Schietgat, S. Dzeroski, and H. Blockeel, "Decision Trees for Hierarchical Multi-label Classification," *Machine Learning*, vol. 73, no. 2, pp. 185–214, Aug. 2008.

[10] F. Fabris and A. A. Freitas, "Dependency Network Methods for Hierarchical Multi-label Classification of Gene Functions," in *Proceedings of the 2014 IEEE International Conference on Computational Intelligence and Data Mining*, Orlando, Florida, Dec. 2014, pp. 241–248.

[11] F. Fabris, A. Freitas, and J. Tullet, "An Extensive Empirical Comparison of Probabilistic Hierarchical Classifiers in Datasets of Ageing-Related Genes," *IEEE/ACM transactions on computational biology and bioinformatics/IEEE, ACM*, pp. 1–14, dec 2015. [Online]. Available: http://europepmc.org/abstract/MED/26661786

[12] F. Fabris and A. A. Freitas, "A Novel Extended Hierarchical Dependence Network Method Based on non-Hierarchical Predictive Classes and Applications to Ageing-Related Data," in *Proceedings of the 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2015, pp. 294–301.

[13] L. d. C. Merschmann and A. A. Freitas, "An Extended Local Hierarchical Classifier for Prediction of Protein and Gene Functions," in *Data Warehousing and Knowledge Discovery*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 8057, pp. 159–171.

[14] A. A. Freitas, "Comprehensible Classification Models - a position paper," *ACM SIGKDD Explor. Newsl.*, vol. 15, no. 1, pp. 1–10, 2014.

[15] A. Vellido, J. D. Martín-Guerrero, and P. J. Lisboa, "Making machine learning models interpretable," in *In Proc. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, vol. 12, 2012, pp. 163–172.

[16] K. Boyd, K. H. Eng, and C. D. Page, "Area Under the Precision-Recall Curve: Point Estimates and Confidence Intervals," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 8190, pp. 451–466.

[17] Y. Peng, P. A. Flach, C. Soares, and P. B. Brazdil, "Improved dataset characterisation for meta-learning," ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2534, pp. 141–152.

[18] R. Leite and Pavel Brazdil, "Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Meta-Learning," in *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. IOS Press, 2010, pp. 309–314.

[19] Q. Sun and B. Pfahringer, "Pairwise meta-rules for better meta-learning-based algorithm ranking," *Machine Learning*, vol. 93, no. 1, pp. 141–161, jul 2013.

[20] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren, "Fast algorithm selection using learning curves," in *International Symposium on Intelligent Data Analysis*. Springer, 2015, pp. 298–309.

[21] R. Leite, P. Brazdil, and J. Vanschoren, "Selecting classification algorithms with active testing," in *Machine Learning and Data Mining in Pattern Recognition*, ser. Lecture Notes in Computer Science, 2012, vol. 7376, pp. 117–131.

[22] S. M. Abdulrahman and P. Brazdil, "Measures for combining accuracy and time for meta-learning," in *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection (MLAS'14)*, vol. 1201, 2014, pp. 49–50.

[23] I. Partalas, R. Babbar, E. Gaussier, and C. Amblard, "Adaptive classifier selection in large-scale hierarchical classification," in *Lecture Notes in Computer Science*, vol. 7665, no. 3, 2012, pp. 612–619.

[24] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining Multi-label Data," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds., 2010, pp. 667–685.

[25] A. Ruepp, A. Zollner, D. Maier, K. Albermann, J. Hani *et al.*, "The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes," *Nucleic Acids Research*, vol. 32, no. 18, pp. 5539–5545, 2004.

[26] R. Tacutu, T. Craig, A. Budovsky, D. Wuttke, G. Lehmann, D. Taranukha, J. Costa, V. E. Fraifeld, and J. a. P. de Magalhães, "Human Ageing Genomic Resources: integrated databases and tools for the biology and genetics of ageing." *Nucleic Acids Research*, vol. 41, no. Database issue, pp. D1027–D1033, Jan. 2013.

[27] F. Fabris and A. A. Freitas, "New KEGG pathway-based interpretable features for classifying ageing-related mouse proteins," *Bioinformatics*, vol. 32, no. 19, pp. 2988–2995, jun 2016.

[28] "HMC Software and Datasets," https://dtai.cs.kuleuven.be/clus/hmcdatasets/, accessed: 2016-09-23.

[29] "Other Bioinformatics Datasets, including ageing-related datasets with GO and FunCat classes," https://www.cs.kent.ac.uk/people/rpg/ff79/Fabris_Datasets.tar.gz, accessed: 2016-09-23.

[30] M. Lichman, "UCI machine learning repository http://archive.ics.uci.edu/ml," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[31] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: ACM, 1992, pp. 144–152.

[32] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[33] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

[34] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.

[35] T. D. Gautheir, "Detecting Trends Using Spearman's Rank Correlation Coefficient," *Environmental Forensics*, vol. 2, no. 4, pp. 359–362, 2001.

[36] P. B. Brazdil, C. Soares, and J. P. Da Costa, "Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[37] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.