Dependency Network Methods for Hierarchical Multi-label Classification of Gene Functions

Fabio Fabris, Alex A. Freitas
School of Computing, University of Kent.
Canterbury, Kent, CT2 7NF, UK
Email: {ff79,A.A.Freitas}@kent.ac.uk

***Abstract*—Hierarchical Multi-label Classification (HMC) is a challenging real-world problem that naturally emerges in several areas. This work proposes two new algorithms using a Probabilistic Graphical Model based on Dependency Networks (DN) to solve the HMC problem of classifying gene functions into pre-established class hierarchies. DNs are especially attractive for their capability of using traditional, "out-of-the-shelf", classification algorithms to model the relationship among classes and for their ability to cope with cyclic dependencies, resulting in greater flexibility with respect to Bayesian Networks. We tested our two algorithms: the first is a stand-alone Hierarchical Dependency Network (HDN) algorithm, and the second is a hybrid between the HDN and the Predictive Clustering Tree (PCT) algorithm, a well-known classifier for HMC. Based on our experiments, the hybrid classifier, using SVMs as base classifiers, obtained higher predictive accuracy than both the standard PCT algorithm and the HDN algorithm, considering 22 bioinformatics datasets and two out of three predictive accuracy measures specific for hierarchical classification ($AU(\overline{PRC})$ and $\overline{AUPRC}_w$).**

I. Introduction

This work focuses on the Hierarchical Multi-label Classification (HMC) problem, a type of supervised learning task that naturally emerges in several real-world problems. The HMC problem consists of learning a classification model given a pre-defined class taxonomy [16] and instances annotated with classes from that taxonomy [4]. There are notoriously successful applications of this technique: in document classification, where it is intuitive to annotate instances in topics that are organized hierarchically; in image classification, where classes are commonly organized into trees; and in bioinformatics, the focus of this work, where functions of genes and proteins are commonly organized into Directed Acyclic Graphs (DAGs) or trees.

Although using traditional (*flat*) classifiers for hierarchical classification is possible, the literature in the area consistently shows that algorithms specifically designed for HMC outperform the naive approach of treating the problem as a *flat* multi-label classification problem [16], justifying the effort of developing new algorithms for hierarchical classification.

In this work we shall adopt the definition of class taxonomy of [16], which is in turn based on the definition of [20]:

A class taxonomy is a relationship defined over a partially ordered set $(C, \prec)$ where C enumerates all classes under consideration and $\prec$ represents the "Is-A" relationship. Intuitively, the relationship must be rooted, asymmetric, anti-reflexive and transitive, formally defined as:

Rooted condition: $\exists\ root \in C \mid \forall\ c_i \in C, (c_i \neq root) \rightarrow (c_i \prec root)$,
Asymmetric condition: $\forall\ c_i, c_j \in C, (c_i \prec c_j) \rightarrow (c_j \not\prec c_i)$,
Ant-reflexive condition: $\forall\ c_i \in C, (c_i \not\prec c_i)$,
Transitive condition: $\forall\ c_i, c_j, c_k \in C, (c_i \prec c_j), (c_j \prec c_k) \rightarrow (c_i \prec c_k)$.

Note that there are some class hierarchies that are not described as "Is-A" relationships but satisfy the previous definition, therefore they are still on the scope of this work. The "Part-Of" relationship of the Gene Ontology (GO) [11] is an example of such relationship.

Generally speaking, the previously defined class hierarchy may be represented graphically as a Directed Acyclic Graph (DAG), although trees are sufficient in some domains. Whether the class hierarchy is a tree or a DAG affects the difficulty of the HMC problem, tree-structured hierarchies are usually more easily tractable by algorithms than DAG-structured hierarchies.

There are two other variants of the HMC problem that affect its difficulty: the first is whether an instance may be assigned class labels from the root only up to a non-leaf node (Non-Mandatory Leaf Node Prediction). This possibility increases the difficulty of the classification task because the classification model must support the possibility of not classifying an instance further down in its class hierarchy.

The second variant is whether or not an instance may have multiple classes in different paths of the class hierarchy (Multiple Paths of Labels). This variation also increases the difficulty of the classification task.

In this work we deal with the most complex type of hierarchical classification problem, namely, DAG-structured class hierarchies, Non-Mandatory Leaf Prediction and Multiple Paths of Labels.

Turning to bioinformatics, nowadays, in the post-genomic era, the cost of extracting genomic and proteomic data from organisms has decreased many-fold, to the extent that researchers now have free access to vast public collections of biological data. Generally speaking, these datasets comprise the sequence information (amino-acids, base-pairs, and other secondary information extracted from gene or protein sequences) and possibly a classification of the biological processes that gene or protein is involved in a curated hierarchical ontology. An example of such ontology is the Gene Ontology [11] and an example of a protein database is the Universal Protein Resource [12].

To make full use of this ever-increasing flow of complex

information to help us understand several biological processes, data mining techniques are required more than ever to assist biologists. Inferences made by data-mining algorithms can be used as a starting point to select the most promising "wet-lab" experiments to be performed, which are much more time-consuming and expensive than their computational counterparts.

In this work, we propose two new algorithms, a stand-alone Hierarchical Dependency Network algorithm (HDN) and a hybrid Hierarchical Dependency Network / Predictive Clustering Tree (HDN-PCT) algorithm to classify gene functions using class labels organized hierarchically. We evaluate our new algorithms in 22 hierarchical classification datasets to test whether it has better predictive performance than the stand-alone PCT algorithm.

This work is organized as follows: in Section II we present related works related to DNs and hierarchical classification. In Section III we introduce our first contribution: the HDN classifier, followed in Section IV by the second contribution, the hybrid HDN-PCT algorithm. In Section V we show the results of our experimental evaluation. Lastly, in Section VI we conclude our work.

II. Background and Related Work on Dependency Networks (DNs)

DNs are a relatively under-explored type of Probabilistic Graphical Model (PGM) compared to other PGMs. DNs were first described in [9]. Like in Bayesian Networks (BN), each node in a DN must encode a probability distribution function conditioned on the values of its parents. In BNs, these probability distributions are usually encoded by simple probability tables inferred from the training data.

DNs differ from BNs by allowing more generic probability distribution functions and cycles in the graphical representation. However, consistent inference in DNs has prohibitive running time [9]. In fact, consistent DNs are equivalent to another type of PGM, namely, Markov Networks, sharing the same time complexity for inference (inference in consistent DNs and Markov Networks are both NP-complete). To overcome this computational limitation, Heckerman et al. [9] proposed the use of general Dependency Networks (which we call simply DNs from now on). These networks are bounded approximations for consistent DNs that have efficient structure learning (by using feature selection techniques) and inference algorithms (by using Gibbs sampling).

DNs were used in several non-traditional classification problems: Toutanova and Klein [18] used DNs for Part-of-Speech tagging. The authors claimed that the DN's ability of representing cyclic dependencies contributed to the good performance of their algorithm in relation to the state of the art. The work of Tian et al. [17] explored DNs in the field of link-based classification of documents, which can be defined as the propagation of belief among linked objects, reporting that the use of DNs increased the predictive performance of their algorithm compared to a baseline approach. Neville and Jensen [13] have used DNs in the task of collective classification, which may be defined as the collective prediction of correlated instances into a set of classes, and reported gains in predictive performance.

Regarding more traditional classification problems, Gámez et al. [5] have used a DN in the task of multi-class classification. The authors used a DN with a Naive Bayes classifier to learn the conditional class probabilities and a χ^2 statistical test to estimate the Markov blanket of each variable x_i (the smallest set of random variables that make x_i independent from other variables) of each node. In a subsequent work [6], the authors enhanced their approach by using a specialized DN to predict each class; additionally they proposed a mechanism to improve the time-complexity and confidence on the determination of the Markov blanket of each class label. It is important to note that although Gamez et al. use DNs for classification, they do not use Gibbs sampling for inference; this is not necessary since they are in the (*flat*) multi-class, single-label scenario and do not need to consider relations between multiple classes occurring at the same time. The authors also reported that the flexibility of DNs was responsible for improving the performance of their algorithm in relation to baseline approaches.

More recently, Guo and Gu [8] proposed a method for *flat* multi-label classification using DNs, in their case Gibbs sampling was necessary to model the relations between predictive labels. The authors reported superior performance compared to several baseline multi-label classifiers.

As far as we know there is no work exploring DNs in the *Hierarchical* Multi-label Classification setting, like proposed in this paper.

III. The New Hierarchical Dependency Network Method (HDN)

A DN is a type of Probabilistic Graphical Model that has not been receiving as much attention in the data-mining field as other types of PGMs such as BNs (and its variants) and Markov networks. As discussed in Section II, there are some works exploring the use of DNs in binary and *flat* multi-label classification [6, 8]; however, as far as we know, there are no attempts of using DNs in hierarchical classification.

Like in BNs, variables in DNs are represented as nodes in a graph. However, DNs differ from BNs by the fact that, in a DN, edges exist between nodes if and only if they are in each other's Markov blankets, and the graph may contain cycles. The Markov blanket of a random variable y_k is the smallest set containing the variables that make y_k independent from all other variables. In other words, given a node in a DN, if we know the actual values of the nodes in its Markov blanket, our beliefs about that node will not change if we know the value of some other node outside the Markov blanket.

In BNs, the Markov blanket of a node is the set containing its children, parents and parents of the children, that is, the Markov blanket of a node is not explicitly represented by the graphical structure [9]. Instead, edges represent (potentially) causal relationships between variables. For this reason, some relationships are not explicitly represented in the graph structure of BNs; therefore people must be trained to interpret the graphical representation correctly [9]. Figure 1 exemplifies the representation of the same relationships in a BN and in a DN. An untrained person would look at the BN in the left and wonder why there is no edge from "Maximum Speed" to "Tire Condition", since knowing the speed of the car clearly affects

our beliefs about the condition of the tires. This confusion is avoided in DNs by explicitly representing the relationships between nodes that are directly correlated.

DNs for classification can be thought of as a collection of probability functions specialized in predicting whether or not an instance belongs to a class given its predictive attributes and the predictions of the classes in its Markov blanket. More formally, given the set of binary random variables $\mathbf{c} = \{c_1, \ldots, c_n\}$ representing the classes, and the vector $\mathbf{x} \in \mathbb{R}^m$, representing the predictive features, a DN is a directed graph $G = (V, E)$, with a probability distribution $P(c_i|pa_i, \mathbf{x})$ associated with each node $v_i \in V$, where pa_i is the set of parents of v_i, representing the Markov blanket of c_i; and E is the set of directed edges.

In the hierarchical classification task, each node in a DN encodes probabilities functions in the form $P(c_i|\mathbf{x}, c_{-i})$, where c_i is a class in the class hierarchy, $\mathbf{x}$ is the predictive feature vector and c_{-i} is the set of predictions of the classes in the Markov blanket of c_i. DNs are specially attractive for hierarchical classification for two reasons. First, traditional *flat* classifiers can be trained independently to determine both the structure of the network (by using feature selection or a classifier that selects a subset of features) and the estimation of $P(c_i|\mathbf{x}, c_{-i})$ for each class c_i, we call those classifiers "base classifiers". Second, if we relax the requirements for exact inference, there is a simple, yet effective, algorithm, Gibbs sampling, that is capable of producing approximate inference in reasonable running times. This is particularly important for Hierarchical Classification because of the large number (thousands) of classes in real-world problems, such as gene function prediction.

We assume that the set of classes containing the siblings and the parents of the children of the class c_i in the class hierarchy is a good candidate for c_{-i}, part of the Markov blanket of c_i, because they encode important relationships about the classes that are not deterministic. Initial experimentation also included the classification of the parents and the children of the class c_i as part of the Markov blanket of c_i, however, it was observed that the classifiers did not cope well with such deterministic parent-child relationships, defined by the "Is-A" class hierarchy. That is, if an instance is labeled with any of the child classes of c_i, by definition of the "Is-A" hierarchy (see Introduction), the instance must be labeled with class c_i as well. Similarly, if the instance is not labeled with any of the parent classes of c_i, by definition, it cannot have the class label c_i. It turned out that these deterministic features misled the classifiers, biasing them to over-fit their predictions to these deterministic features.

This kind of over-fitting occurred because, although deterministic parent-child relationships are very useful for achieving a good classification accuracy in the training set (where both the parent class(es) and the child class(es) of a given current class c_i are known); such parent-child relationships are less useful in the test set, where the parent class(es) and child class(es) of c_i are unknown and have to be estimated by Gibbs samping (described later).

The use of the siblings and parents of the children of class c_i as extended features exploits background knowledge (structure of the class hierarchy) defined by expert biologists and avoids the computationally expensive search for the Markov blanket of each node. However, note that the conditional probability $P(c_i|\mathbf{x}, c_{-i})$ assumes that every class in c_{-i} and every predictive feature in $\mathbf{x}$ is in the Markov blanket of c_i. However, there may be noisy predictive features or classes in c_{-i} that are not good predictors for c_i. This may misguide a classifier for c_i and clutter the PGM. For this reason, we reduce the size of the Markov blanket of c_i by applying a feature selection filter on $\mathbf{x}$ and c_{-i} (treating them as a single, extended predictive vector), generating a new filtered extended predictive feature vector $\mathbf{x}^+$. This is detailed next.

1) Training Phase: The training phase of our HDN is very similar to the training phase of binary classifiers, with the exception that we extend the predictive feature vector $\mathbf{x}$ of each instance to contain the class labels of the current class node's candidate Markov blanket. That is, for each class label c_i, we first use a simple feature selection method to produce the filtered extended predictive feature vector $\mathbf{x}^+$, as described earlier. More precisely, we use a statistical F-test [10] to select relevant features for each class c_i. We select the top n_feats features in the ranking of features produced by the F-test, where n_feats is a parameter of the F-test feature selection method. Next, we train a binary classifier $\mathcal{F}_i$ with the feature vector $\mathbf{x}^+$, which contain both selected features and selected actual class labels of the nodes in the Markov blanket of class c_i, considering the given class taxonomy. We expect that the classifiers use the extended information to learn dependencies between class labels and predictive features. We could use a different set of classes in our Markov blanket, e.g., using a statistical test to find a relationship between classes that is not explicitly encoded in the class hierarchy. We leave this possibility open for future work. In this work we have experimented with classification models capable of outputting probabilistic classification for all class labels of the instances, namely, Gaussian Naive Bayes (GBN), Naive Bayes (NB), SVM (with RBF kernel) and C4.5 decision tree. We have tried both NB and GNB classifiers because the NB classifier requires a discretization of the data set (as the majority of features are continuous) that may affect the predictive results, and GBN avoids the need for such discretization. Hence, it is worth trying both these versions of Naive Bayes.

When training the classifier for each class node c_i, we use as positive classes all instances that have the class label c_i in their classification paths (all paths in the class taxonomy graph from the $root$ node to the most specific classifications of the instances) and as negative classes all the other instances. Furthermore, we only train a classification algorithm for a particular class node if there are at least min_inst instances in the least represented class (i.e., positive or negative class), in order to mitigate the problem of overfitting. If a classifier is not trained for a class node, we output the *a-priori* class probability distribution for that class node, regardless of the filtered extended predictive feature vector $\mathbf{x}^+$.

2) Gibbs Sampling for Approximate Inference: Once we trained our binary classifiers to estimate the probabilities $P(c_i|\mathbf{x}, c_{-i})$, – which are estimated by using the filtered extended feature vector $\mathbf{x}^+$, i.e. computing $P(c_i|\mathbf{x}^+)$ – we already have a fully-parameterized HDN where each class node has edges connecting it to selected features and selected predictive classes. The next step is to use our HDN algorithm to pre-

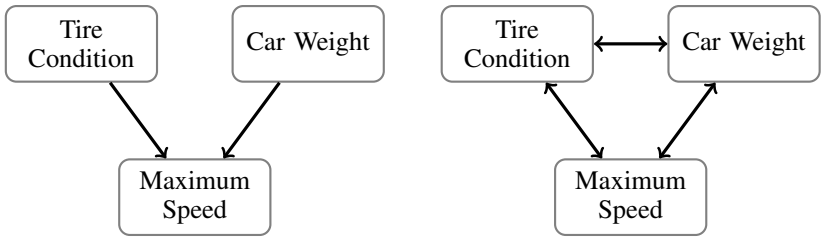

Fig. 1. Representing the same probabilistic relations with BNs (left) and DNs (right)

dict a class vector $\mathbf{c}$ given the predictive features $\mathbf{x}^+$. In other words, we wish to calculate the maximum a-posteriori (MAP) for our predictive feature vector: $\mathbf{c}^* = \arg\max_{\mathbf{c}} P(\mathbf{c}|\mathbf{x}^+)$.

Because solving this problem exactly is NP-hard [8], we use the Gibbs sampling algorithm adapted to our problem (presented in Algorithm 1) to do inference in our DN. The Gibbs sampling algorithm [7] is a Metropolis-Hastings algorithm that is both very suitable for inference in DNs and has a simple and efficient implementation [8]. After some burn-in iterations, this algorithm converges to a stationary distribution that approximates the underlying probability distribution.

Algorithm 1 Hierarchical classification with Gibbs sampling algorithm.

1: **procedure** MODIFIED GIBBS SAMPLING(Instance $\mathbf{x}$, Number of total iterations it, Number of burn-in iterations $burn$)
2: Initialize class label vector of $\mathbf{x}$ randomly using the a-priori class distribution.
3: **for** $k \in \{1..it\}$ **do**
4: **for all** $c_i \in$ the Class Taxonomy **do**
5: $c_{-i} \leftarrow$ ExtendedFeatures(c_i)
6: $\mathbf{x}^+ \leftarrow$ ApplyFeatSel$_i(\mathbf{x}, c_{-i})$
7: $P(c_i|c_{-i}, \mathbf{x}) \leftarrow \mathcal{F}_i(\mathbf{x}^+)$
8: $u \leftarrow$ Random value draw from a uniform distribution in $[0, 1]$.
9: **if** $u < P(c_i|c_{-i}, \mathbf{x})$ **then** $c_i \leftarrow 1$ **else** $c_i \leftarrow 0$ **end if**
10: **if** $it > burn$ **then**
11: $P(c_i|\mathbf{x}) \leftarrow \frac{P(c_i|\mathbf{x})\times(it-1)+P(c_i|c_{-i},\mathbf{x})}{it}$
12: **end if**
13: **end for**
14: **end for**
15: **return** the marginal probabilities $P(c_i|\mathbf{x})$
16: **end procedure**

The algorithm begins by randomly initializing the class vector of the instance whose classes will be predicted (line 2). Next, the algorithm visits the nodes of the graph structure using some ordering (line 4). We have explored three ordering possibilities: bottom-up (BU) (start by visiting the leaves of the hierarchy, recursing to its parents, in a breadth-first manner), top-down (TD) (start by visiting the $root$ node and recursing to its children in a breadth-first manner) and random order.

In line 5, the function ExtendedFeatures(c_i) returns the class labels that are candidate to be in the Markov blanket of c_i (i.e., the siblings of c_i and the parents of the children of c_i in the class hierarchy). In line 6, the function ApplyFeatSel$_i(\mathbf{x}, c_{-i})$ filters the predictive features and candidate class labels using the previously mentioned statistical F-test for the i-th class, returning the filtered extended vector $\mathbf{x}^+$. Next, the algorithm retrieves the probabilistic classification of the current node c_i given the current class labels of its Markov blanket and its predictive feature vector (line 7). Finally, the algorithm employs a stochastic rule to update the current classification of the input instance (line 9). There are many ways to estimate the MAP of the predictive classes; we have used the typical way of computing the marginal probability for each class variable and making the final prediction [8].

Our Gibbs sampling algorithm was slightly modified to return the mean probabilities associated with each class (line 11). The more common approach would be to return the most common label for each class [15] (in our case the positive or negative label). We employ an iterative approach to calculate the mean probabilities after the burn-in phase, so there is no need to store all class-wise probabilities estimations.

As we will compare our methods to a probabilistic classifier (the Clus-HMC classifier), we modified the Gibbs algorithm in order to obtain a probabilistic decision for each class at the end of the sampling procedure.

We call the previously described training procedure and inference algorithm as the Hierarchical Dependency Network (HDN) algorithm.

IV. THE HYBRID HDN-PCT ALGORITHM

Besides investigating the HDN algorithm by itself, we also exploit the power of the PCT framework. An algorithm in this framework builds a decision tree by finding a predictive feature that splits the set of instances in two clusters, maximizing the similarity of classes within each cluster and the dissimilarity of the classes across the two clusters. In order for the split to be accepted, the class distribution of the instances across the two clusters must be statistically different according to the F-test. The algorithm recurses in each cluster that it forms and eventually stops if it finds no statistically significant split or the size of a cluster falls bellow a pre-established threshold.

In the prediction phase, to classify an instance $\mathbf{x}$, a PCT algorithm first identifies the cluster associated with that instance and then assigns, to instance $\mathbf{x}$, classes whose value in the mean probability vector of that cluster is greater than a probability threshold. The threshold is varied when computing a Precision-Recall curve, as explained later.

We shall use the most well-known implementation of PCTs for hierarchical classification, the Clus-HMC algorithm [19]. The Clus-HMC obtained good predictive accuracy in relation to other PCT algorithms.

We apply our HDN algorithm in each cluster produced by the Clus-HMC algorithm (that we shall call simply PCT from now on), and we name the combination of both algorithms as the HDN-PCT algorithm. This combination introduces another parameter, min_inst_HDN, the minimum number of instances required in each cluster to train our dependency network. If the number of instances in a cluster is below that minimum, the classification of instances associated with that cluster is performed by the PCT algorithm.

In addition to the min_inst_HDN parameter, we adopt the following strategy when using our HDN-PCT algorithm: after the training phase we analyse which classifiers had better overall classification accuracy on the validation set (which is different from the test set used to evaluate the classifier's predictive accuracy) in comparison with clusters formed by the baseline PCT algorithm. If the classification accuracy of the HDN for a PCT cluster is higher than the classification accuracy of PCT for that cluster, we analyse each class c_i in that cluster and calculate the mean loss L_i, defined as:

$$L_i \equiv \frac{\sum_j ((\mathbf{1}_{i,j}) - p_{i,j})^2}{n_i}, \quad (1)$$

where $\mathbf{1}_{i,j}$ is an indicator function defined as:

$$\mathbf{1}_{i,j} \equiv \begin{cases} 1 & \text{if instance } j \text{ has } c_i \text{ as a true class,} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

p_{i_j} is the estimated probability of the j-th instance belonging to the i-th class and n_i is the number of instances of the i-th class.

Considering this loss function, the predictions of the HDN-PCT algorithm and the PCT algorithm in the validation set are compared to the real classification. Then, we discard the binary classifiers with greater loss than the loss achieved by the stand-alone PCT algorithm for a particular class and use the output of the PCT classifier instead for predicting that class.

Notice that it is not possible to calculate ranking-based quality measures for the individual classifiers (for instance ROCAUC or PRAUC) in a given cluster formed by the PCT algorithm, because the probabilities assigned to all instances are always the same, which yields an arbitrary instance ranking in respect to the probability of an instance belonging to a given class. This justifies the use of the loss function defined by Equation (1).

V. Experimental Evaluation

A. Datasets

To measure the predictive accuracy of the HDN and the HDN-PCT algorithms, we used 22 of the 24 datasets made available by [19]. 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 base classification algorithms that we used for the HDN, specially the SVM classifier. The datasets are pre-divided by their creators into training, validation and testing sets; we maintain the same division in our work. We did not employ the more tradition cross-validation procedure because of the relatively large number of instances and to maintain the consistence with the works that use the same datasets.

TABLE I. Number of instances, predictive features and classes in the datasets used in this work.

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

These datasets contain features extracted from the genes of the widely used model organism *Saccharomyces cerevisiae* (yeast). The first part of the name of the dataset indicates the type of predictive feature used. There are two types of features: 1) statistics extracted from the amino acid sequences (seq features) and 2) several types of microarray expression data (all the other features). The second part of the name indicates the type of class hierarchy used: "FUN" for the tree-structured hierarchy in the FunCat scheme [14] and "GO" for the DAG-structured Gene Ontology [11]. Table I presents statistics of the used datasets.

B. Predictive Accuracy Estimation

As a baseline algorithm, we use the Clus-HMC version of the Predictive Clustering Tree (PCT) method for hierarchical classification, discussed earlier. Our algorithms (HDN and HDN-PCT) and the baseline output the probabilities of an instance belonging to each class, instead of a crisp classification. Hence, to estimate the predictive performance, we transform the class probabilities into crisp classifications by predicting classes whose probability is greater than a certain threshold - that is, a given class is assigned to the current test instance if and only if the output probability for that class is greater than the used threshold.

In order to avoid the subjective choice of a threshold, in binary classification, it is common to calculate the Area Under the Precision Recall Curve (AUPRC). In the case of highly skewed class distributions (as is typically the case of hierarchical classification problems), the AUPRC is more appropriate than the more popular Area Under the Receiver Operating Characteristic Curve (AUROC) because the AUROC measure over-rewards classifying negative instances as negative [2], not recognizing that correctly classifying a rare positive instance in the highly negatively-skewed scenario should be more important than correctly classifying the more common negative instances.

Precision (P) and Recall (R) are common measures used in binary classification problems, defined as: $P \equiv \frac{TP}{TP+FP}$, and $R \equiv \frac{TP}{TP+FN}$.

Where TP (True Positives) is the number of correctly positively classified instances, FP (False Positives) is the number of instances classified as positive that are not positive, FN (False Negatives) is the number of positive instances that were not classified as positive and "$\equiv$" means "equal by definition".

These two measures represent conflicting optimizing goals. E.g., one can obtain a high precision (but a low recall) by classifying as positive only instances that have a high probability of belonging to the positive class; conversely, one can obtain a high recall (and a low precision) by classifying all instances as positive.

Given a binary classifier with probabilistic outputs, it is possible to construct a PR curve (a plot of the classifier's precision as a function of its recall) by thresholding the output (class probability) of the classifier using values in the interval $[0, 1]$. Each threshold is associated with a value of precision and recall, corresponding to a point in the PR space. To obtain a single performance measure from the curve, we calculate its area using a trapezoidal approximation [1]. A classifier that perfectly ranks all negative instances before the positive ones – where the ranking is by increasing order of probability of being positive – would have a $AUPRC$ of 1.0.

There is no obvious way to adapt this measure to the hierarchical classification scenario, therefore we follow the suggestions of [19] and use three alternatives: the Area Under the Average Precision-Recall Curve ($AU(\overline{PRC})$), the average Area Under the Precision Recall Curve ($\overline{AUPRC}$) and the weighted Average Area Under the Precision Recall Curve ($\overline{AUPRC}_w$).

To calculate the $AU(\overline{PRC})$, we use the hierarchical versions of precision and recall for a fixed threshold, defined as:

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

Where P_j is the set of predicted classes of the j-th instance and T_j is the set of true classes of the j-th instance.

To calculate the $\overline{AUPRC}$ measure we simply average all the class-wise $AUPRC$ performances. Similarly, to calculate the $\overline{AUPRC}_w$, we calculate the $AUPRC$ of each class independently and combine the individual performance measures by calculating an average over all classes weighted by the number of instances that belongs to 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. Training Procedure

The training procedure of the HDN algorithm consists in inducing a classifier for each node of the class hierarchy that has more than min_inst instances in the least represented class (positive or negative). For our experiments, min_inst was fixed in the value 30 for all datasets and evaluation measures. The selected ordering for visiting classes was the "random" order. The number of Gibbs sampling iterations, it, was set to 40. We perform feature selection for each induced classifier by using the 30 highest scored features, according to a F-Test. Finally, we experimented using the Support Vector Machine (SVM), C4.5, Gaussian Naive Bayes and Naive Bayes classifiers, and decided to use the SVM classifier, because it obtained higher classification accuracy. These parameters were determined by exploratory experimentation, observing the classification accuracy in the validation set (without accessing the test set).

Considering the HDN-PCT classifier, in addition to the previously mentioned parameters, the algorithm requires that we first construct a PCT that will be used as our default classifier. The PCT divides the dataset into clusters that will be further analysed by the Hierarchical Dependency Network method in our HDN-PCT algorithm.

TABLE II. CHOSEN PARAMETER VALUES (2ND COLUMN), SET OF EXPLORED PARAMETER VALUES (3RD COLUMN). TD AND BU MEAN, RESPECTIVELY, TOP-DOWN AND BOTTOM-UP

Minimum number of instances in the least represented class to train a classifier (min_inst)	30	10, 20, 30, 50, 70, 80
Order to visit class nodes	random	random, TD, BU
Gibbs Sampling Iterations (it)	40	40, 50, 60, 70, 80, 90
Number of Selected Features (n_feats)	30	5, 15, 25, 30, 35, 45, 55, 65, 75
Base Classifier	SVM	SVM, C4.5, GNB, NB
Minimum number of instances in a PCT cluster to train the HDN (min_inst_HDN)	30	10, 20, 30, 50, 80

The PCT algorithm requires the determination of the parameter s, that regulates how significant a split must be in order to be accepted by the F-test. Following [19], we have used the best s value that they reported (on the validation set) for each dataset and evaluation measure. We train the final PCT for each dataset and each $AUPRC$ evaluation measure by joining the training and validation sets. We use each of the resulting models as both our baseline hierarchical classification model and the default model for the HDN-PCT algorithm.

The hybrid HDN-PCT classifier requires one additional parameter in relation to the HDN classifier: the parameter min_inst_HDN, which is minimum number of instances necessary in a cluster in order for a HDN to be trained. Similarly to the min_inst parameter of the HDN classifier, this value was set to 30 across all datasets and evaluation measures.

The range of values that we considered in our exploratory experimentation and the chosen value for each parameter, which was the parameter value leading to highest accuracy on the validation set (without accessing the test set) is given in Table II. These parameter values were used for both HDN and the hybrid HDN-PCT.

D. Results

In Table III we present the predictive performances of the algorithms in the testing set considering the three AUPRC measures. Underlined numbers represent the largest (and best) value of a given a measure.

To compare the performance of the proposed algorithms against the baseline PCT algorithm, we use the Wilcoxon Signed-Rank statistical test with the Holm's step down procedure [3], to correct for multiple comparisons.

Considering the measure $AU(\overline{PRC})$, the HDN-PCT classifier is statistically significantly better in relation to the PCT baseline ($p = 0.011$), no significant difference was detect among the HDN-PCT algorithm and the HDN, nor among the HDN algorithm and the baseline PCT.

Concerning the measure $\overline{AUPRC}_w$, we conclude that the HDN-PCT algorithm is statistically better than the PCT algorithm ($p = 0.024$). No other significant difference was

TABLE III. TESTING PHASE RESULTS. UNDERLINED NUMBERS REPRESENT THE LARGEST VALUES FOR A GIVEN MEASURE.

Dataset	$AU(\overline{PRC})$ PCT	$AU(\overline{PRC})$ HDN-PCT	$AU(\overline{PRC})$ HDN	$\overline{AUPRC}_w$ PCT	$\overline{AUPRC}_w$ HDN-PCT	$\overline{AUPRC}_w$ HDN	$\overline{AUPRC}$ PCT	$\overline{AUPRC}$ HDN-PCT	$\overline{AUPRC}$ HDN
seq_GO	0.47611	0.47828	0.45157	0.35456	0.35459	0.36335	0.02331	0.02331	0.02312
cellcycle_GO	0.44426	0.44445	0.42362	0.32008	0.32079	0.33223	0.01681	0.01685	0.01800
church_GO	0.44343	0.44321	0.41092	0.30945	0.30955	0.29989	0.01514	0.01515	0.01411
derisi_GO	0.44890	0.44836	0.41325	0.31541	0.31389	0.30500	0.01649	0.01648	0.01506
eisen_GO	0.46124	0.45986	0.45411	0.34562	0.34587	0.36056	0.02610	0.02609	0.02376
gasch1_GO	0.45404	0.45493	0.43925	0.33811	0.34162	0.34784	0.02134	0.02133	0.02016
gasch2_GO	0.45105	0.45010	0.42770	0.33021	0.33030	0.33807	0.01821	0.01823	0.01847
spo_GO	0.44972	0.44856	0.41296	0.31779	0.31788	0.31266	0.01857	0.01858	0.01586
expr_GO	0.44963	0.45017	0.43961	0.33707	0.33984	0.34805	0.01948	0.01949	0.02016
struc_GO	0.45294	0.46285	0.40959	0.32088	0.32082	0.30795	0.01996	0.01996	0.01488
hom_GO	0.48655	0.48653	0.43445	0.36659	0.36638	0.33579	0.03267	0.03266	0.02063
seq_FUN	0.21498	0.23060	0.25613	0.16853	0.16853	0.20543	0.04180	0.04180	0.04678
cellcycle_FUN	0.17402	0.18057	0.21427	0.13628	0.14013	0.17475	0.02924	0.02957	0.03816
church_FUN	0.17309	0.17088	0.16877	0.11832	0.12275	0.12748	0.02394	0.02460	0.02570
derisi_FUN	0.18019	0.18585	0.18169	0.13167	0.13191	0.13908	0.02874	0.02873	0.02923
eisen_FUN	0.20113	0.20879	0.25067	0.14971	0.15109	0.19911	0.03704	0.03718	0.04747
gasch1_FUN	0.20615	0.22667	0.23661	0.16359	0.16386	0.19359	0.03752	0.03756	0.04435
gasch2_FUN	0.19256	0.19607	0.22100	0.14129	0.14130	0.17515	0.02947	0.02948	0.03976
spo_FUN	0.19043	0.19723	0.19010	0.14680	0.14942	0.15171	0.03110	0.03141	0.03117
expr_FUN	0.20891	0.22479	0.23556	0.16011	0.16079	0.19751	0.03681	0.03693	0.04525
struc_FUN	0.18430	0.19350	0.18025	0.15500	0.15491	0.13783	0.03106	0.03106	0.02713
hom_FUN	0.25841	0.25890	0.20708	0.22165	0.22183	0.17484	0.05930	0.05930	0.03826

detected according ot this measure, nor according to the $AU(\overline{PRC})$ measure.

VI. CONCLUSION AND FUTURE WORK

We proposed a new and effective method for combining the power of the PCT decision tree algorithm, which has been shown to be an algorithm with good predictive performance that produces interpretable models for hierarchical classification of gene functions, with the flexibility of the HDN classifier, which uses "out-of-the-shelf" classification algorithms to refine the classifications made by the PCT algorithm, improving the overall predictive performance.

We concluded from the experiments that the proposed hybrid HDN-PCT algorithm obtained statistically significantly higher predictive accuracies (p-value < 0.025) than the baseline PCT algorithm according to two out of three measures, namely $\overline{AU(PRC)}$ and $\overline{AUPRC}_w$; and there was no significant difference according to the third measure. Such predictive performance improvement is relevant to the area of hierarchical classification of gene functions, since there is a large number of genes, from many organisms, whose functions are still unknown.

As future work we plan to use a more sophisticated feature selection algorithm to select the Markov blanket of each class node. Also, to minimize over-fitting, we will investigate features selection techniques for the base classifiers, specially wrapper techniques that should select a more effective set of features to reduce the over-fitting problem. Another possibility is to use other algorithms as base classifiers and even use a mixture of classifiers in the same dependency network.

Another important future aspects of this research are to investigate the best way to visualize the generated classification models, choose base classifiers that allow for easy interpretability, maintaining good predictive performance and evaluate the run-time of different types of base algorithms and contrast them with the PCT algorithm.

REFERENCES

[1] Kendrick Boyd, Kevin H. Eng, and C. David Page. Area under the precision-recall curve: Point estimates and confidence intervals. In *Machine Learning and Knowledge Discovery in Databases*, volume 8190 of *Lecture Notes in Computer Science*, pages 451–466. Springer Berlin Heidelberg, 2013.

[2] Jesse Davis and Mark Goadrich. The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 233–240, 2006.

[3] Janez Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[4] Fabio Fabris and Alex A Freitas. An Efficient Algorithm for Hierarchical Classification of Protein and Gene Functions. In *Proceedings of DEXA14 Workshops*, Munich, 2014.

[5] J A Gámez, J L Mateo, and J M Puerta. Dependency networks based classifiers: learning models by using independence tests. In *3rd European Workshop on Probabilistic Graphical Models (PGM06)*, pages 115–122, 2006.

[6] J A Gámez, J L Mateo, T D Nielsen, and J M Puerta. Robust Classification using Mixtures of Dependency Networks. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models*, pages 129–136, 2008.

[7] Stuart Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(6):721–741, Nov 1984.

[8] Yuhong Guo and Suicheng Gu. Multi-label classification using conditional dependency networks. In *Proceedings of the 22nd inte. joint conf. on Artificial Intelligence*, volume 2, pages 1300–1305. AAAI Press, 2011.

[9] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency Networks for Inference, Collaborative Filtering, and Data Visualization. *The Journal of Machine Learning*

Research, 1:49–75, 2001.

[10] Richard G Lomax and Debbie L Hahs-Vaughn. *Statistical concepts: a second course*. Routledge, 2013.

[11] The Gene Ontology Consortium. The Gene Ontology (GO) database and informatics resource. *Nucleic acids research*, 32:D258–61, 2004.

[12] The Uniprot Consortium. The Universal Protein Resource (UniProt) in 2010. *Nucleic acids research*, 38: D142–8, January 2010.

[13] Jennifer Neville and David Jensen. Relational Dependency Networks. *Journal of Machine Learning Research*, 8:653–692, 2007.

[14] Andreas Ruepp, Alfred Zollner, Dieter Maier, Kaj Albermann, Jean Hani, et al. The funcat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic acids research*, 32(18): 5539–5545, 2004.

[15] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, and Lise Getoor. Collective Classification in Network Data. *AI magazine*, pages 1–24, 2008.

[16] Carlos N Jr Silla and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 44(1-2):31–72, 2011.

[17] Yonghong Tian, Qiang Yang, Tiejun Huang, Charles X. Ling, and Wen Gao. Learning Contextual Dependency Network Models for Link-Based Classification. *IEEE Transactions on Knowledge and Data Engineering*, 18 (11):1482–1496, 2006.

[18] Kristina Toutanova and Dan Klein. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proc. of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, volume 1 of *NAACL '03*, pages 173–180, Edmonton, Canada, 2003. Association for Computational Linguistics.

[19] Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, August 2008.

[20] Feihong Wu, Jun Zhang, and Vasant Honavar. Learning Classifiers Using Hierarchically Structured Class Taxonomies. *Abstraction, Reformulation and Approximation*, pages 313–320, 2005.