DECISION MAKING FOR HIERARCHICAL MULTI-LABEL CLASSIFICATION

Yuting Ye
Division of Biostatistics
University of California
Berkeley, CA, USA
yeyt@berkeley.edu

Christine Ho
Department of Statistics
University of California
Berkeley, CA, USA
christineho@berkeley.edu

Ci-Ren Jiang
Institute of Statistical Science
Academia Sinica
Taipei, Taiwan
cirenjiang@stat.sinica.edu.tw

Wayne Tai Lee
Department of Statistics
University of California, Berkeley
Berkeley, CA, USA
lwtai@stat.berkeley.edu

Haiyan Huang*
Department of Statistics
University of California
Berkeley, CA, USA
hhuang@stat.berkeley.edu

ABSTRACT

The hierarchical multi-label classification problem has drawn increasing attention over decades. Tremendous efforts have been focused on a two-stage strategy: the first stage produces binary classifiers for each class, and the second stage makes the joint decisions on all classifiers. In the second stage, however, existing methods either violate the class hierarchy or ignore the differences in the statistical distributions of the first-stage classifier scores. In this article, we propose a method that first converts the classifier scores into the newly-defined statistics mLPRs, then ranks these values while respecting the hierarchy. We demonstrate sorting mLPRs from the largest to the smallest maximizes the objective ceAUHC (conditionally expected area under the hit curve) while respects the hierarchy. It is also shown that an object positioned at the top of the ranking has a larger probability of being positive than others in the tail. Given the empirical mLPRs estimated from data, we provide a ranking method called HierRank to maximize the empirical ceAUHC under the hierarchy constraint. The performance of the resulting ranking is demonstrated on a synthetic dataset, a dataset for disease diagnosis and a dataset for document categorization. On the first dataset and the second dataset, our method outperforms other methods by at least 0.079 and 0.017 respectively in the sense of the area under the PR curve truncated at top 50% objects. On the third dataset, we bring a key insight on how to train the two-stage method.

***K*eywords** multi-label classification, hierarchy, hit curve, mLPR, ceAUHC

1 Introduction

In multi-label classification, each object is assigned to one or multiple classes (Zhang and Zhou, 2013). The related but more complicated problem of hierarchical multi-label classification (HMC) concerns the situation when the additional knowledge of the dependency relationships among classes is available and needs to be incorporated. For instance, the classes may be organized in a tree or a directed acyclic graph (DAG) to reflect a hierarchical dependency structure between the classes. Such hierarchies have to be respected when making classification decisions. HMC is an important problem in many real-world applications, and has recently attracted a large amount of attention in statistics and machine learning research. In biology and biomedicine, example applications of HMC include the disease diagnosis for a patient who may suffer from multiple closely or distantly related diseases that are organized in a DAG based on the Unified Medical Language System (UMLS), a well-known biomedical vocabulary (link); the assignment of genes to multiple (closely or distantly related) gene functional categories defined by the Gene Ontology DAG (link); the categorization of proteins along the MIPS FunCat rooted tree (link); and etc. (Alves et al., 2010; Barutcuoglu et al., 2006; Blockeel et al., 2006; Clare, 2003; Kiritchenko et al., 2005; Valentini, 2009, 2011). Outside of biology, HMC has been widely used for

*Corresponding author.

problems like the classification of text documents, music categorization, and image recognition, all of which tend to have labels that follow hierarchical structures (Rousu et al., 2006; Kiritchenko et al., 2006; Mayne and Perry, 2009).

A representative line of efforts has been made to tackle the HMC problem in two stages (Koller and Sahami, 1997; Wu et al., 2005; Holden and Freitas, 2005; Silla and Freitas, 2009; Gauch et al., 2009). In the first stage, classifiers will be trained for each class independently without considering the class hierarchy; this is equivalent to solving multiple independent classification problems. These classifiers are considered as "local" classifiers. In the second stage, given the class hierarchy as well as the classifier scores from the local classifiers, the task is to find an "optimal" decision rule. It is natural to require the final classification results to possess the following two properties: 1) meeting a pre-defined classification performance criterion, and (ii) respecting the class hierarchy. We note that a variety of classification methods can be applied in the first stage. This stage also tends to be computationally efficient since the classifiers (e.g., one for each class) can be learned in parallel. However, how to define and find an "optimal" decision rule at the second stage remains an open question.

To design an approach for the second stage, two questions are involved: what classification performance criterion to use and how to guide classification decisions by this criterion while respecting the class hierarchy. A general performance criterion is to control the number of false positives and/or false negatives. One typical approach finds class-specific cutoffs to optimize an objective such as H-loss and F-measure (Barutcuoglu et al., 2006; Triguero and Vens, 2016). This approach usually suffers from the difficulty in respecting the hierarchy. So a remedy procedure is used to adjust the initial predictions to satisfy the hierarchy constraint (Sun and Lim, 2001; Ananpiriyakul et al., 2014). However, this procedure might violate the optimality of the targeted objective. Another approach of interest is to rank all the objects, with the hope that positives should be positioned ahead of negatives. Given the resulting ranking, a universal cutoff suffices to make the final decisions. In literature, there are a few works in this line. Bi and Kwok (2011) used an algorithm that maximize the sum of top L classifier scores while respecting the hierarchy, where L is pre-defined. This algorithm can be regarded as a ranking routine if it traverses from $L = 1$, $L = 2$, ..., until $L =$ total number of objects. Yet, it lacks a comprehensive consideration into the statistical distributions of the classifier scores. Jiang et al. (2014) converted the classifier scores to the local precision rates (LPRs). They show that sorting LPRs in an descending order theoretically ensures the maximal pooled precision rate at any pooled recall rate. Nonetheless, this approach does not consider the hierarchical structure and the final results might violate the hierarchy.

In this study, we attempt to develop a ranking method that satisfies the two properties outlined above. We assess the performance of a ranking by the area under the hit curve. The hit curve is a natural surrogate of the precision-recall (PR) curve and the receiver operating characteristic (ROC) curve. It is well suited when we are most interested in the top-ranked instances. To distinguish the objects in terms of the potential of being positive, we use the conditional expected area under the hit curve (ceAUHC) given the classifier scores. The ranking is produced by maximizing ceAUHC under the hierarchy constraint. This objective function regards the objects at the top of the ranking as more important than those with low ranking. In addition, the maximization of ceAUHC forces the strong positive child nodes to help rank their parent nodes. We show that the resulting ranking method is equivalent to sorting the multivariate local precision rates (mLPR) from the largest to the least. Here mLPR is an multivariate extension of LPR proposed in Jiang et al. (2014). We demonstrate that the resulting ranking respects the hierarchy, and an object ranked at the top has a larger probability of being positive than those in the tail. However, the true mLPR values are rarely accessible in reality. Sorting the estimated mLPRs does not guarantee the hierarchical constraint. To this end, we come up with the algorithm HierRank (*Hierarchical Ranking*). We refer to the ranking obtained by sorting the estimated mLPRs as $\widehat{mLPR}$-based ranking. Finally, we provide a simple strategy to select a single cutoff on the ranking. For evaluation, we first consider a synthetic dataset, on which the $\widehat{mLPR}$-based ranking is shown to outperform other competing methods universally. Then, we study two real datasets. On the dataset for disease diagnosis, we show how the accuracy of the mLPR estimation influences the performance of the $\widehat{mLPR}$-based ranking. On the dataset for document classification, we give a practical guideline on the training procedure of the method that produces the $\widehat{mLPR}$-based ranking.

The rest of the paper is organized as follows. In Section 2, we introduce the notation and the model of our interest. In Section 3, we introduce the statistics mLPR from the metric ceAUHC that is used to generate the ranking. In Section 4, we propose the ranking algorithm HierRank. We assess the performance of the $\widehat{mLPR}$-based ranking on a synthetic dataset and two case studies in Section 5. Finally, we conclude this article in Section 6.

2 Notation and Model

2.1 Notation

There are K classes of interest, which are structured in a hierarchy $\mathcal{G}$, e.g., Figure 1 (a). In $\mathcal{G}$, denote by $pa(k)$ the set of the parent nodes of the k-th node, by $anc(k)$ the set of its ancestor nodes, and by $nbh(k)$ the set of its immediate neighborhood. For a random sample/individual, it can have multiple positive labels out of the K classes, thus associated with K nodes. Each node has a binary variable Y_k indicating if the status/label of this sample is positive for the k-th class. It also has a pre-given classifier score S_k reflecting the likelihood of $Y_k = 1$. The multi-label classification permits there are more than one nodes with positive labels or none of them is positive. We denote $\mathbf{Y} = (Y_1, \ldots, Y_K)^{\mathrm{T}}$, $\mathbf{S} = (S_1, \ldots, S_K)^{\mathrm{T}}$.

In practice, we observe M samples/individuals, thus there are $n = K \times M$ nodes/objects in total. We use $Y_k^{(m)}$ and $S_k^{(m)}$ to denote the status/label and the pre-defined classifier score for the k-th node/object of the m-th sample/individual. Denote $\mathbf{Y}^{(m)} = (Y_1^{(m)}, \ldots, Y_K^{(m)})^{\mathrm{T}}$ and $\mathbf{S}^{(m)} = (S_1^{(m)}, \ldots, S_K^{(m)})^{\mathrm{T}}$ for the m-th sample. We emphasize that we use the term sample/individual to denote the person or item to be classified, and use the node/object to denote the class.

In addition, define the ranking $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ on n objects as a permutation of $(1, 2, \ldots, n-1, n)$.

2.2 Model

The hierarchy $\mathcal{G}$ for the K classes is assumed to be a tree/DAG. It can be disconnected and consist of multiple connected components like Figure 1 (a). Throughout this article, we mainly discuss the tree structure. The extension to the DAG structure is simple and delegated to Appendix C.3. In the sequel, we introduce the model of our interest. Given the hierarchy $\mathcal{G}$, the status/label variables $\mathbf{Y}$ and the classifier scores $\mathbf{S}$ for a random individual/sample, we consider an augmented graph $\bar{\mathcal{G}}$ (Figure 1 (b)) by assuming the conditional independence stated in Assumption 1.

Assumption 1 *The scores are conditional independent given the associative class labels, i.e.,*

$$S_{k'} \perp S_k | Y_k \ \textit{ for any } \ k' \neq k.$$

The condition independence is reasonable in practice. For instance, it is satisfied when the classification training is executed class by class for the two-stage HMC methods. [More examples in literature and citations] With this assumption, we propose the following model $\mathcal{H}$ to characterize the relationship between the labels $\mathbf{Y}$ and the scores $\mathbf{S}$ of a random individual:

(i) $\mathbb{P}[S_k = s | S_1, \ldots, S_{k-1}, S_{k+1}, \ldots, S_K, Y_1, \ldots, Y_K] = \mathbb{P}[S_k = s | Y_k]$.

(ii) $\mathbb{P}[Y_k = 1 | Y_{pa(k)} = 1] \in [0, 1]$; $\mathbb{P}[Y_k = 1 | Y_{pa(k)} = 0] = 0$.

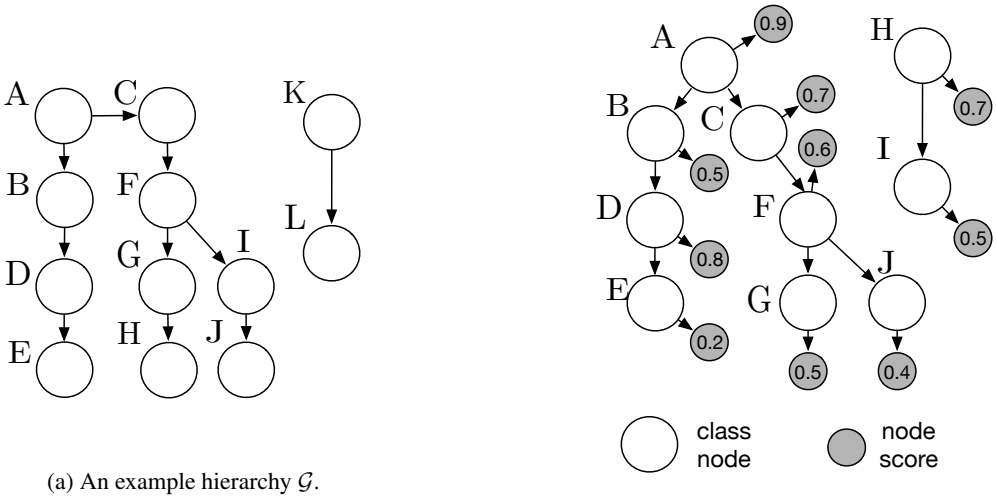
(iii) S_k has a mixture model, i.e., $\mathbb{P}[S_k = s | Y_k] = f_0^{(k)}(s)\, \mathbb{I}[Y_k = 0] + f_1^{(k)}(s)\, \mathbb{I}[Y_k = 1]$, where $f_0^{(k)}$ denotes the null distribution and $f_1^{(k)}$ denotes the alternative distribution for the k-th class.

The assumption (i) follows from the conditional independence; the assumption (ii) reflects that the labels respect the hierarchy $\mathcal{G}$, i.e, a negative node implies all of its descendants are negative as well; the assumption (iii) means that the classifier score is generated from a class-specific mixture model.

On the sample level, the M samples are independent and identically distributed, i.e., $(\mathbf{Y}^{(m)}, \mathbf{S}^{(m)}) \overset{i.i.d}{\sim} \mathcal{H}$, $m = 1, \ldots, M$. For a fixed k, scores $\{S_k^{(m)} : m = 1, \ldots, M\}$ follow the same mixture distribution. But if k and k' correspond to different nodes in $\mathcal{G}$, $S_k^{(m)}$ and $S_{k'}^{(m')}$ would follow different distributions, and so are not directly comparable. When making joint decisions across all nodes, we need to take into consideration such distinct statistical distribution properties across classes.

3 Problem Formulation and The Ideal Solution

For simplicity, we vectorize $(\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(M)})$ and $(\mathbf{S}^{(1)}, \ldots, \mathbf{S}^{(M)})$ to produce $\tilde{\mathbf{Y}} = (Y_1^{(1)}, \ldots, Y_K^{(1)}, \ldots, Y_1^{(M)}, \ldots, Y_K^{(M)})^{\mathrm{T}}$ and $\tilde{\mathbf{S}} = (S_1^{(1)}, \ldots, S_K^{(1)}, \ldots, S_1^{(M)}, \ldots, S_K^{(M)})^{\mathrm{T}}$, respectively. By representing the index i of $\tilde{\mathbf{Y}}$ and $\tilde{\mathbf{S}}$ as $i = (m-1) \cdot K + k$ where m denotes the sample id and k denotes the node id, we write $\tilde{\mathbf{Y}} = (Y_1, \ldots, Y_n)^{\mathrm{T}}$, $\tilde{\mathbf{S}} = (S_1, \ldots, S_n)^{\mathrm{T}}$ when there is no confusion. We keep using this notation throughout the rest of the paper.



(a) An example hierarchy $\mathcal{G}$.

(b) The augmented graph $\bar{\mathcal{G}}$.

Figure 1: An example hierarchical graph $\mathcal{G}$ (a) and the associative augmented graph $\bar{\mathcal{G}}$.

3.1 Motivation

Given all the n scores $\tilde{\mathbf{S}}$ and the model $\mathcal{H}$, we attempt to produce a reasonable ranking of the n nodes. We want the ranking $\boldsymbol{\pi}$ to respect the hierarchy $\mathcal{G}$. That is,

$$\pi_i < \pi_{i'} \qquad \text{if Object } i \text{ is an ancestor of Object } i'. \tag{3.1}$$

Inequalities (3.1) is a natural hierarchical constraint (with respect to $\mathcal{G}$) the ranking needs to obey. We say a ranking $\boldsymbol{\pi}$ has the **Hierarchy-Consistency** property or is a **topological** ordering for $\mathcal{G}$ if it satisfies (3.1).

Besides the Hierarchy-Consistency property, a desired ranking is expected to put all the positive objects in front of the negative objects. In practice, three graphical plots have been widely used to assess that empirical raking performance: the Precision-Recall (PR) curve plotting the precision rate (number of true discoveries divided by total number of discoveries) against the recall rate (number of true discoveries divided by total number of true objects), the Receiver Operating Characteristic (ROC) curve plotting the true positive rate (the recall rate) against the false positive rate (1 - the precision rate), the hit curve (Figure 8 of Appendix A) plotting the number of true positive discoveries against the number of discoveries. The PR curve and the ROC are obviously equivalent by definition. There is an "equivalence" between optimizing the hit curve and optimizing the ROC/PR curves. For instance, when the area under the hit curve attains the theoretical maximum, it also corresponds to the area one under the ROC/PR curve. To produce an overall evaluation based on these curves, F-measure (the geometric mean of the precision rate and the recall rate) or the area under the curve have been used as summary metrics.

Here we tend to use the hit curve and the area under it since the ROC curve and the PR curve are not sensitive or informative to the top-ranked instances, especially when the number of positive instances is tiny compared to the total size (Davis and Goadrich, 2006; Herskovic et al., 2007; Hand, 2009). The hit curve is widely used in situations where the top-ranked instances are of most interest. For example, in evaluating the performance of a web search engine, the relevance of the top-ranked pages is more important than those that appear lower in search results. For many hierarchical multi-label classification problems, like the disease diagnosis problem, we are most concerned about those diseases that are diagnosed in the very beginning, which usually correspond to general disease diagnosis. The hit curve can serve well in this situation as a graphic representation of the ranker's performance, since it would plot the results in order of decreasing relevance and the y-axis would indicate the results' relevance to the target. Another reason we prefer the hit curve lies in that it is easier to work with the number of total/true discoveries in the hit curve than the true/false discovery rates in the ROC/PR curves. More details for the hit curve can be found in Appendix A.

3.2 Conditional Expected Area under Hit Curve and Multivariate Local Precision Rate

We aim to find out a ranking $\boldsymbol{\pi}$ such that under this ranking, the area under the hit curve is maximized. Besides, we require $\boldsymbol{\pi}$ to be a topological ordering for $\mathcal{G}$. Formally, we want to solve the below optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\pi}} \quad & \text{Area under the hit curve (AUHC)} \\ s.t. \quad & \pi_i < \pi_{i'} \text{ if Object } i \text{ is an ancestor of Object } i', \end{aligned} \tag{3.2}$$

Note that the x-axis of the hit curve represents the number of calls made, thus the expression for the AUHC is equivalent to the sum of the number of true positives among the top i calls, for every i. This yields the convenient expression for the AUHC:

$$\text{AUHC} = \sum_{i=1}^{n} \sum_{j=1}^{i} \mathbb{I}\{Y_{\pi_j} = 1\} = \sum_{i=1}^{n} (n - i + 1)\, \mathbb{I}\{Y_{\pi_i} = 1\} \tag{3.3}$$

However, Equation (3.3) is comprised of indicator functions, i.e., each term is either 0 or 1. Thus it lacks the ability of distinguishing between nodes. In other words, it's a tie between all the positive (negative) nodes. To this end, we aim to distinguish the nodes in terms of their probability of being positive given the scores $\tilde{\mathbf{S}}$. We take conditional expected values of (3.3) given the node scores and arrive at

$$\text{ceAUHC} := \mathbb{E}[\text{AUC of hit curve}|\, \tilde{\mathbf{S}}] = \sum_{i=1}^{n} (n - i + 1)\, \mathbb{P}(Y_{\pi_i} = 1 | S_1, \ldots, S_n). \tag{3.4}$$

Here, we call the target metric the **conditional expected area under the hit curve (ceAUHC)**. We call $\mathbb{P}(Y_i = 1|S_1, \ldots, S_n)$ **multivariate local precision rate (mLPR)** because we note that it is a multivariate extension of the local precision rate (LPR) proposed in Jiang et al. (2014). This newly-defined statistics enjoy the desired property that the mLPR value of a node can not be smaller than those of its descendants; see Proposition 2. Besides, it can be shown that a larger mLPR indicates the associated object is more likely to be positive; see Section 3.3. Based on the two properties, we propose a population-level solution to maximizing ceAUHC under the hierarchical constraint.

Proposition 2 *If Object i is an ancestor of Object i', then $mLPR_i \geq mLPR_{i'}$.*

Proof For any i' and $i \in anc(i')$, it follows that

$$\begin{aligned} mLPR_{i'} &= \mathbb{P}(Y_{i'} = 1|S_1, \ldots, S_n) \\ &= \sum_{Y_1, \ldots Y_{i'-1}, Y_{i'+1}, \ldots, Y_n} \mathbb{P}(Y_1, \ldots, Y_{i'-1}, Y_{i'} = 1, Y_{i'+1}, \ldots, Y_n|S_1, \ldots, S_n) \\ &\overset{(a)}{=} \sum_{Y_j : j \neq i, j \neq i'} \mathbb{P}(Y_1, \ldots, Y_{i'} = 1, Y_i = 1, \ldots, Y_n|S_1, \ldots, S_n) \\ &\leq \sum_{Y_j : j \neq i} \mathbb{P}(Y_1, \ldots, Y_{i-1}, Y_i = 1, Y_{i+1}, \ldots, Y_n|S_1, \ldots, S_n) \\ &= mLPR_i, \end{aligned}$$

where Equation (a) is obtained by the condition (ii) of the model $\mathcal{H}$ (Section 2.2). ■

3.3 Sorting mLPRs In A Descending Order

Our ultimate goal is to find the ranking that maximize ceAUHC (3.4) while respecting the hierarchy, which can be mathematically written as the below optimization problem.

$$\begin{aligned} \max_{\boldsymbol{\pi}} \quad & ceAUHC \\ s.t. \quad & \pi_i \leq \pi_{i'} \text{ if Object } i \text{ is an ancestor of Object } i', \end{aligned} \tag{3.5}$$

We can generate the ranking by naively sorting any scores (here we use mLPRs) from the largest to the smallest. We call this method **naive sorting**. Proposition 2 indicates that the ranking by applying naive sorting on mLPRs satisfies the Hierarchy-Consistency property. It immediately implies that this ranking is the solution to the problem (3.5), as shown in Proposition 3. In other words, if we can get access to the population mLPRs, solving the optimization problem (3.5) boils down to converting the original scores across classes into mLPRs and sorting them in an descending order. This conclusion is reasonable because the hierarchy information has been fully incorporated into mLPRs.

Proposition 3 *The ranking obtained by naive sorting on mLPRs is a topological ordering for $\mathcal{G}$ and maximizes* (3.4)*.*

Proof Proposition 2 indicates that sorting mLPRs from the largest to the smallest can guarantee the hierarchy constraint that ancestors rank ahead of their descendants. Meanwhile, the maximum of ceAUHC (3.4) is just obtained by sorting mLPRs in this manner. ■

Furthermore, we show in Proposition 4 that a node in the top of the ranking by applying naive sorting on mLPRs is more likely to be positive than a node in the tail. In other words, given a decision rule induced by imposing a cutoff on such a ranking, the node taken as positive by this rule is more likely to be truly positive than those taken as negative. Fundamentally, it reflects that mLPRs accounts for the statistical differences across classes. Thus it is statistically reasonable to directly compare mLPRs. This is not a surprising result since the statistics mLPR can be regarded as the extension of the LPR in the scenario where there are dependencies between classes. It has been shown in Jiang et al. (2014) that the LPR is equivalent to the local true discovery rate, which is useful in comparing and ranking discoveries when classes are independent of each other (Efron, 2012). Sorting LPRs in the decreasing order guarantee the the optimal pooled precision at any pooled recall rate if there is no hierarchy constraint. More details on LPR are deferred to Appendix B.5.

Proposition 4 *Let $\boldsymbol{\pi}^{ns}$ be the ranking obtained by sorting mLPRs in an descending order. Then for any positive object i and i' with $\pi_i^{ns} < \pi_{i'}^{ns}$, we have*

$$\mathbb{P}[Y_i = 1 | \pi_i^{ns} < \pi_{i'}^{ns}] \geq \mathbb{P}[Y_{i'} = 1 | \pi_i^{ns} < \pi_{i'}^{ns}]$$

Proof For a realization $\tilde{\mathbf{s}}$ of $\tilde{\mathbf{S}}$, the resulting mLPRs gives the ranking $\boldsymbol{\pi}^{ns}$ with $\pi_i^{ns} < \pi_{i'}^{ns}$, indicating that $mLPR_i = \mathbb{P}[Y_i | \, \tilde{\mathbf{S}} = \tilde{\mathbf{s}}] \geq \mathbb{P}[Y_{i'} | \, \tilde{\mathbf{S}} = \tilde{\mathbf{s}}] = mLPR_{i'}$. Then we have

$$\begin{aligned}
\mathbb{P}[Y_i = 1, \pi_i^{ns} < \pi_{i'}^{ns}] &= \int_{\tilde{\mathbf{s}}: \pi_i^{ns} < \pi_{i'}^{ns}} \mathbb{P}[Y_i = 1, \tilde{\mathbf{S}} = \tilde{\mathbf{s}}] \\
&\geq \int_{\tilde{\mathbf{s}}: \pi_i^{ns} < \pi_{i'}^{ns}} \mathbb{P}[Y_{i'} = 1, \tilde{\mathbf{S}} = \tilde{\mathbf{s}}] \\
&= \mathbb{P}[Y_{i'} = 1, \pi_i^{ns} < \pi_{i'}^{ns}].
\end{aligned}$$

■

4 Ranking Algorithm based on estimated mLPRs

4.1 Computation of mLPRs

The sound properties of mLPRs and the naive sorting method can be guaranteed when we know the true values of mLPRs. In reality, it is hard to get this ideal solution and we have to estimate mLPRs. This problem has the same form as the smoothing problem for the hidden markov model, but it is simpler since we know the true labels Y_i's (at least for the training dataset). If we know $\mathbb{P}(Y_1, \ldots, Y_n | S_1, \ldots, S_n)$, then we can obtain the mLPRs $\mathbb{P}(Y_i | S_1, \ldots, S_n)$ by applying message passing to $\mathbb{P}(Y_1, \ldots, Y_n | S_1, \ldots, S_n)$ with respect to $\mathcal{G}$. Given the model $\mathcal{H}$ defined in Section 2, we are able to investigate $\mathbb{P}(Y_1, \ldots, Y_n | S_1, \ldots, S_n)$ in a simple manner:

$$\begin{aligned}
\mathbb{P}(Y_1, \ldots, Y_n | S_1, \ldots, S_n) &\overset{(a)}{\propto} \mathbb{P}(S_1, \ldots, S_n | Y_1, \ldots, Y_n) \cdot \mathbb{P}(Y_1, \ldots, Y_n) \\
&\overset{(b)}{=} \prod_{i=1}^{n} \mathbb{P}(S_i | Y_i) \, \mathbb{P}(Y_i | Y_{pa(i)}) \\
&\overset{(c)}{\propto} \prod_{i=1}^{n} \frac{\mathbb{P}(Y_i | S_i)}{\mathbb{P}(Y_i)} \cdot \mathbb{P}(Y_i | Y_{pa(i)}) \\
&\overset{(d)}{=} \prod_{i=1}^{n} LPR_i \cdot \frac{\mathbb{P}(Y_i | Y_{pa(i)})}{\mathbb{P}(Y_i)},
\end{aligned} \tag{4.1}$$

where (a) and (c) hold by the Bayes rule, (b) holds by using the Markov property with Assumption 1 (conditional independence). Equation (d) follows from Jiang et al. (2014) that given the scores $\tilde{\mathbf{S}}$, the associative LPR of the i-th

node is defined as

$$LPR_i = \mathbb{P}(Y_i|S_i).$$

We apply the method in Jiang et al. (2014) to estimate LPRs, and $\mathbb{P}(Y_i|Y_{pa(i)})$, $\mathbb{P}(Y_i)$ are estimated separately. Denote by $\widehat{mLPR}$ the estimator of mLPR. Given $\widehat{mLPR}$s, we consider an empirical objective function

$$\sum_{i=1}^{n}(n-i+1)\widehat{mLPR}_i, \tag{4.2}$$

which is an approximation of ceAUHC (3.4). To simplify the computation of mLPRs, we consider approximations to the mLPR in terms of the strength of the dependencies between classes. To be specific,

- We assume that Y_i is independent of $S_{i'}$ for $i' \neq i$. Then we get $mLPR_i \approx \mathbb{P}(Y_i|S_i)$, which is simply LPR_i. This type of computation is called the **independence** (short as **indpt**) approximation. Denote by $\widehat{mLPR}^{indpt}$ the estimator of mLPR with the independence approximation. This simplification has been widely used in statistics methods such as Naive Bayes and Variational Bayes [CITATIONS]. It is reasonable in practice when a weak dependence is observed between objects of the same sample.
- We assume that Y_i is only correlated with $S_{i'}$ for $i' = i$ or $i' \in nbh(i)$. Then we get $mLPR_i \approx \mathbb{P}(Y_i|S_i, S_{nbh(i)})$, which can be computed in the same fashion as Equation (4.1). This type of computation is called the **neighborhood** (short as **nbh**) approximation. Denote by $\widehat{mLPR}^{nbh}$ the estimator of mLPR with the neighborhood approximation. This simplification is a compromise between the **independence** approximation and the original mLPR computation based on the complete dependencies. The assumption of the neighborhood dependencies can be regarded as a mild augmentation of Assumption 1.
- We consider the complete dependencies of $\mathcal{G}$, and get the original $mLPR_i = \mathbb{P}(Y_i|S_1, \ldots, S_n)$. This type of computation is called the **full** version of mLPR. Denote by $\widehat{mLPR}^{full}$ the estimator of mLPR with the full version of mLPR.

4.2 Algorithms

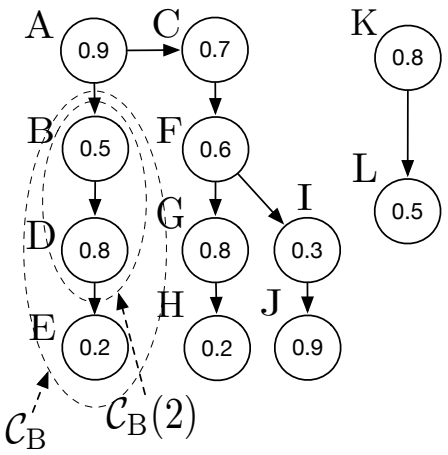
The inference and decision making based on $\widehat{mLPR}$s can be affected by the following factors:

- The departure of the estimated $\mathbb{P}(Y_i)$, $\mathbb{P}(Y_i|Y_{pa(i)})$ and LPR_i from the true values.
- The ignorance of the dependencies by using the indpt/nbh approximation instead of true mLPRs.

Using $\widehat{mLPR}$s, the ranking obtained by naive sorting might violate the Hierarchy-Consistency property. In addition, it can not guarantee maximizing (4.2). To tackle these issues, we introduce the ranking algorithm **HierRank** to solve the constraint optimization problem (3.5).

Several key concepts in HierRank are illustrated in Figure 2.

- **Single-child branch/chain**: a branch/chain of the tree, on which every node has at most one child. In Figure 2, the chains $B \to D \to E$, $G \to H$, $I \to J$ and $K \to L$ are single-child branches.
- $\boldsymbol{\mathcal{P}_1}$: a set of nodes on all single-child branches, i.e., every node in $\mathcal{P}_1$ and its descendants have at most one child. In Figure 2, nodes on the chains $B \to D \to E$, $G \to H$, $I \to J$, $K \to L$ belong to $\mathcal{P}_1$. Node A does not belong to $\mathcal{P}_1$ because it is not on a single-child branch.
- **Starting node** in $\mathcal{P}_1$: a node is contained in $\mathcal{P}_1$ but its parent(s) are excluded from $\mathcal{P}_1$. In Figure 2, Node B, G, I and K are the starting nodes in $\mathcal{P}_1$.
- $\boldsymbol{\mathcal{P}_2}$: a set of nodes with at least two children and those children are in $\mathcal{P}_1$. Any node in $\mathcal{P}_2$ is attached by the single-child branches starting from its child nodes. In Figure 2, only Node F belongs to $\mathcal{P}_2$. It is attached by the chains $G \to H$ and $I \to J$. Node A does not belong to $\mathcal{P}_2$ because its child C does not belong to $\mathcal{P}_1$.
- $\boldsymbol{\mathcal{P}_3}$: a set of nodes who are the parents/ancestors of the nodes in $\mathcal{P}_2$ and they have only one child. In Figure 2, only Node C belongs to $\mathcal{P}_3$. Node A does not belong to $\mathcal{P}_3$ since it has two child nodes.
- $\boldsymbol{\mathcal{C}_{r\to s}}$: a sub-chain that starts from Node r and ends at Node s (a sub-chain/path is unique for a tree, given the two ends). Let $|\,\mathcal{C}_{r\to s}\,|$ be the number of nodes in $\mathcal{C}_{r\to s}$. For example, in Figure 2, $\mathcal{C}_{A\to E}$ represents the chain $A \to B \to D \to E$.



$$\bar{\ell}_{\mathrm{B},2}(\mathbf{S}) = \frac{0.5 + 0.8}{2} = 0.65$$

Figure 2: Illustration of notations. The numbers inside the nodes are the associated scores.

- $\boldsymbol{\mathcal{C}_r}$: a simplified notation for $\mathcal{C}_{r\to e}$, if $r \in \mathcal{P}_1$ and Node e is a leaf node. For example, in Figure 2, $\mathcal{C}_{\mathrm{B}}$ represents the chain B $\to$ D $\to$ E.
- $\boldsymbol{\mathcal{C}_r(h)}$: a sub-chain that consists of the first h nodes of $\mathcal{C}_r$. For example, in Figure 2, $\mathcal{C}_{\mathrm{B}}(2)$ is a sub-chain of $\mathcal{C}_{\mathrm{B}}$ consisting of $\mathcal{C}_{\mathrm{B}}$'s first two nodes, i.e., B $\to$ D.
- $\boldsymbol{\bar{\ell}_{r,h}(\mathbf{S})}$: the average value of scores (e.g., $\widehat{mLPR}$s) of the sub-chain $\mathcal{C}_r(h)$, i.e., $\bar{\ell}_{r,h}(\mathbf{S}) = \frac{1}{|\mathcal{C}_r(h)|}\sum_{i\in\mathcal{C}_r(h)} S_i$. Here we use $C_r(h)$ to denote the set of nodes in the sub-chain $\mathcal{C}_r(h)$ when there is no ambiguity. .

We first consider ranking nodes from multiple disjoint single-child chains $\mathcal{C}_{r_1}, \ldots, \mathcal{C}_{r_p}$. This is equivalent to merging these chains into a single chain. The relative position along the final chain will reflect the rank. To this end, we introduce Algorithm 1 which is illustrated in Figure 3:

(a) Initialize the ranked list $\mathcal{L} = \emptyset$.

(b) For the chains $I \to J$ and $G \to H$, there are four sub-chains: G, $G \to H$, I, $I \to J$ with average scores 0.8, 0.45, 0.3, 0.6 respectively. The sub-chain G has the largest average, so we remove it from the original graph and attach it to $\mathcal{L}$.

(c) In the remaining graph, there are three sub-chains: $I \to J$, I and H with average scores 0.6, 0.3, 0.1. The sub-chain $I \to J$ has the largest average, so we remove it from the remaining graph and attach it to $\mathcal{L}$.

(d) There remains a single node H. We attach it to $\mathcal{L}$. Since there is no node in the remaining graph, $\mathcal{L}$ is the final ranking.

The produced ranking of Algorithm 1 satisfies the Hierarchy-Consistency property, because it preserves the relative ordering of the nodes in each chain. This ranking also maximizes (4.2). The heuristics is that this algorithm essentially sorts the average scores in a descending order. The detailed argument is part of the proof of Theorem 5.

For a general tree case, we introduce Algorithm 2 (HierRank), which uses Algorithm 1 repeatedly. Figure 4 is used to illustrate this algorithm:

(a) Identify $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$. In Figure 4 (a), $\mathcal{P}_1 = \{B, D, E, G, H, I, J, K, L\}$. $\mathcal{P}_2 = \{F\}$, $\mathcal{P}_3 = \{C\}$.

(b) For each node in $\mathcal{P}_2$, we apply Algorithm 1 to merge the chains attached to it. Then attach the resulting single chain to this node, and update $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$. In Figure 4 (a), $\mathcal{P}_2$ only contains Node F. Apply Algorithm 1 to merge the two sub-chains $G \to H$ and $I \to J$ attached to node F. Attach the resulting chain $G \to I \to J \to H$ to node F, and we get Figure 4 (b). Update $\mathcal{P}_1 = \{B, D, E, C, F, G, I, J, H, K, L\}$, $\mathcal{P}_2 = \{A\}$, $\mathcal{P}_3 = \emptyset$.

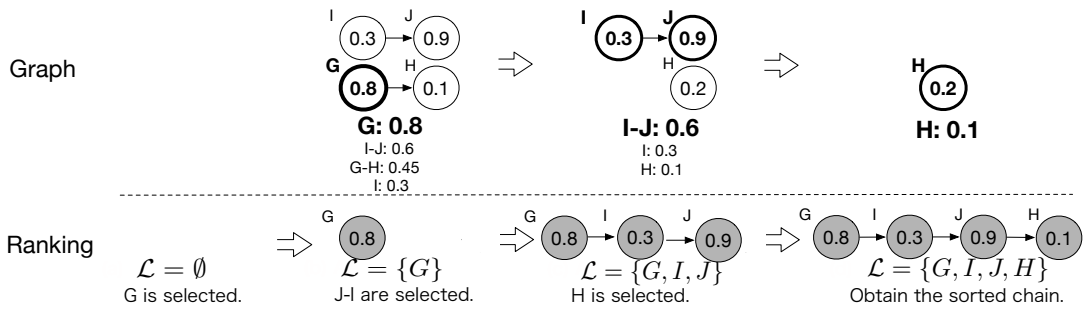


Figure 3: An example of the merging process in Algorithm 1: merge the two sub-chains $G \to H$ and $I \to J$ in Figure 2. The nodes in bold form a sub-chain of the highest averaging $\widehat{mLPR}$, and the nodes filled in grey gives a ranking produced by the merging procedure.

(c) Repeat step (b) until $\mathcal{P}_2$ is empty (then $\mathcal{P}_3$ is empty as well). In Figure 4 (b), $\mathcal{P}_2$ only contains node A. Apply Algorithm 1 to merge the two sub-chains $B \to D \to E$ and $C \to F \to G \to I \to J \to H$ that are attached to node A. Attach the resulting chain $C \to F \to G \to B \to D \to I \to J \to E \to H$ to node A, and we obtain Figure 4 (c). Update $\mathcal{P}_1 = \{\text{all nodes}\}$, $\mathcal{P}_2 = \emptyset$, $\mathcal{P}_3 = \emptyset$. Since $\mathcal{P}_2$ is empty now, we terminate the loop.

(d) Apply Algorithm 1 to merge the remaining single-child chains. In Figure 4 (c), there remain two sub-chains $K \to L$ and $A \to C \to F \to G \to B \to D \to I \to J \to E \to H$. Apply Algorithm 1 to merge them, and we obtain the final ranking $A \to K \to C \to F \to G \to B \to D \to I \to J \to L \to E \to H$.

In the very beginning, some nodes are put in $\mathcal{P}_1$, $\mathcal{P}_2$ or $\mathcal{P}_3$, and the other nodes are left out. As the algorithm proceeds, the nodes in $\mathcal{P}_2$ and $\mathcal{P}_3$ are transferred to $\mathcal{P}_1$, and some left-out nodes are transferred to $\mathcal{P}_2$ and $\mathcal{P}_3$ until all the nodes are put in $\mathcal{P}_1$ and there remains a single chain. It can be seen that $\mathcal{P}_2$ are updated upwards along the graph, so HierRank works in a bottle-up fashion. As we note above, when we repeat using Algorithm 1 in HierRank, the local ranking satisfies the hierarchy constraint and attains the maximum of Equation (4.2), conditional on the sub-graph involved. We show in Theorem 5 that HierRank enjoys the desired optimality that it produces a topological ordering of $\mathcal{G}$ with the maximum of Equation (4.2). The detailed proof is deferred to Appendix E.1.

Theorem 5 *Finding out the topological ordering for $\mathcal{G}$ that maximizes ceAUHC* (3.4) *is equivalent to finding out the optimal topological ordering of $\mathcal{G}$ that replaces any sub-tree with the corresponding merged chain by Algorithm 2. Hence, Algorithm 2 leads to the optimal topological ordering by merging all the trees into a single chain.*

Algorithm 1 The Chain-Merge algorithm.

Input: p chains $\mathcal{D} = \{\text{node} \in \mathcal{C}_r : r = r_1, \ldots, r_p\}$, the node scores $\tilde{\mathbf{S}}$ (e.g., classifier scores, or $\widehat{mLPRs}$).
Procedure:
1: Set $\mathcal{L} = \emptyset$;
2: Compute $\{\bar{\ell}_{r,i}(\tilde{\mathbf{S}}) : i = 1, \ldots | \mathcal{C}_r |, r = r_1, \ldots, r_p\}$.
3: **while** $\mathcal{D} \neq \emptyset$ **do**
4: $\quad (r', h') = \arg \max_{C_r(h) \subset \mathcal{D}} \bar{\ell}_{r,h}(\tilde{\mathbf{S}})$.
5: $\quad \mathcal{L} \leftarrow \mathcal{L} \oplus C_{r'}(h')$, where $\oplus$ indicates the concatenation of two sequences.
6: $\quad \mathcal{D} \leftarrow (\mathcal{D} \backslash C_{r'}) \cup (C_{r'} \backslash C_{r'}(h'))$.
7: $\quad$ Update the average scores of the remaining nodes as step 2.
8: **end while**
Output: $\mathcal{L}$.

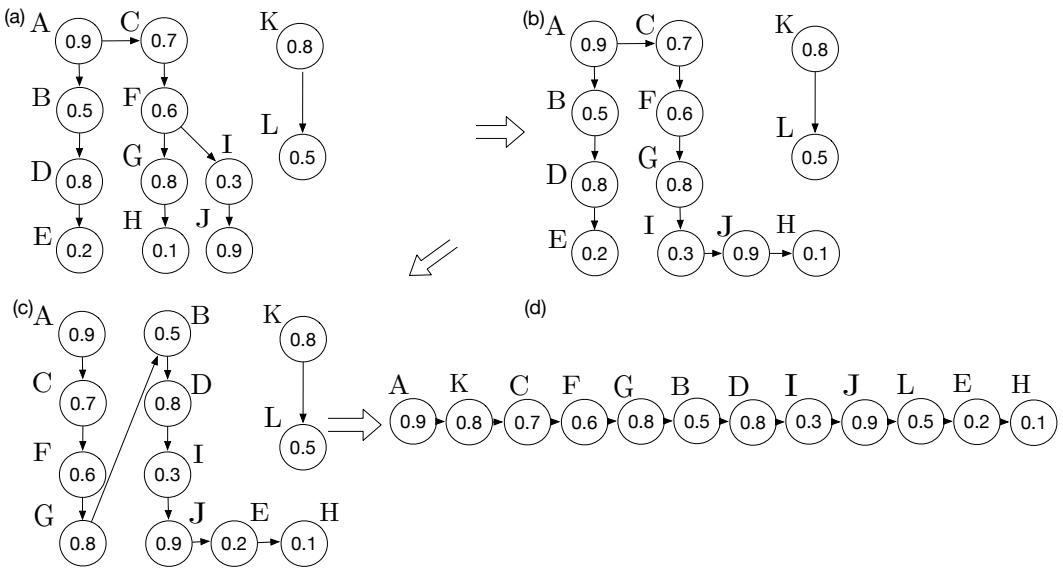


Figure 4: An example of the merging process in Algorithm 2: (a)→(b) merge $G \to H$ and $I \to J$ into $G \to I \to J \to H$; (b)→(c) merge $B \to D \to E$ and $C \to F \to G \to I \to J \to H$ to $D \to F \to G \to B \to D \to I \to J \to E \to H$; (c)→(d) merge all nodes to $A \to K \to C \to F \to G \to B \to D \to I \to J \to L \to E \to H$.

Algorithm 2 The HierRank algorithm for the tree hierarchy.

Input: The tree graph $\mathcal{G}$, node scores $\tilde{\mathbf{S}}$ (e.g., classifier scores, or $\widehat{mLPRs}$).
Procedure:

Figure out $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$.
while $\mathcal{P}_2 \neq \emptyset$ **do**
  Pop out one v from $\mathcal{P}_2$. Take two children of v, r_1 and r_2.
  Feed C_{r_1} and C_{r_2} into Algorithm 1 and obtain $\mathcal{L}(r_1, r_2)$.
  Replace C_{r_1} and C_{r_2} with $\mathcal{L}(r_1, r_2)$.
  Update $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$.
end while
if There remain multiple chains **then**
  Apply Algorithm 1 to these chains.
end if
Let $\mathcal{L}$ be the resulting chain.
[1] **Output:** a ranking $\mathcal{L}$.

We have three remarks for HierRank. First, the property stated in Proposition 4 is not guaranteed when using $\widehat{mLPR}$s instead of mLPRs. If we have good estimations of $\mathbb{P}(Y_i)$, $\mathbb{P}(Y_i|Y_{pa(i)})$ and LPR_i, the naive sorting behaves similarly to HierRank when ranking $\widehat{mLPRs}$. If the estimations are way off the true values, $\widehat{mLPRs}$ would miss the hierarchy information in $\mathcal{G}$, thus HierRank outperforms naive sorting significantly. Second, the time complexity of HierRank reaches up to $\mathcal{O}(K^3)$ for each individual. It implies that the ranking over the K nodes in $\mathcal{G}$ across M individuals costs $\mathcal{O}(nK^2 + n \log M)$ computations ($n = MK$). This high complexity stems from the exhaustive merging and repeated computations of the moving average at each iteration. To improve the efficiency, we modify Algorithm 2 by segmenting a chain into blocks, which are defined by the maximal running average. In this fashion, we eliminate the redundant computations and obtain Algorithm 6 [using 2']. Algorithm 6 is equivalent to Algorithm 2 in light of the output and only costs $\mathcal{O}(n \log n)$ operations. In addition, we note that there is an existing algorithm Condensing Sort and Select Algorithm (CSSA) (Bi and Kwok, 2011) that has the same time complexity as Algorithm 6. CSSA was designed to make the first L decisions (L is a positive integer) and can be applied to the ranking problem as well. It produces almost the same ranking as HierRank except for the tail part; see Appendix C.2 for the details of Algorithm 6 and CSSA. Third,

we can extend Algorithm 2 that is designed for the tree graph to handle the DAG graph.The details of this algorithm is deferred to Appendix C.3.

5 Evaluation

5.1 Setup

In the sequel, a variety of rankings can be produced based on $\widehat{mLPR}s$ (**$\widehat{mLPR}$-based ranking**), which differs in how to estimate $mLPR$s and how to rank $\widehat{mLPR}s$. To obtain $\widehat{mLPR}s$, we estimate $\mathbb{P}(Y_i|Y_{pa(i)})$ and $\mathbb{P}(Y_i)$ using SVM with covariates or the empirical proportions without covariates. We estimate LPR by the local polynomial regression as Jiang et al. (2014), or estimate $\mathbb{P}(S_i|Y_i)$ by modelling $\mathbb{P}(S|Y=0)$ and $\mathbb{P}(S|Y=1)$ as two Gaussian densities (DeCoro et al., 2007). Then the mLPR is estimated in terms of the indpt approximation, the nbh approximation and the full version. Note that for the indpt approximation, it does not rely on the estimators of $\mathbb{P}(Y_i|Y_{pa(i)})$ and $\mathbb{P}(Y_i)$, so it performs the same regardless of how these quantities are estimated. Given $\widehat{mLPR}s$, the ranking is produced via either naive sorting or HierRank.

We compare $\widehat{mLPR}$-based rankings to three methods of different variants. The first one is simply ranking the raw classifier scores (**Raw-scores-based ranking**). Next, we consider **HIROM** (Bi and Kwok, 2015) which is the state-of-the-art local HMC classifier. It produces Bayes-optimal predictions that minimize a series of hierarchical risks with a general learning model that is independent of the loss functions. Here we use the hierarchical ranking loss and the hierarchical hamming loss for HIROM, which extends the classic ranking loss and hamming loss to the HMC scenario by considering the hierarchy information. Moreover, we consider another line of efforts for the HMC problem, i.e, the "global" classifier. It solves the classification issue and the hierarchy issue mentioned above simultaneously. Unlike local classifiers, global classifiers simultaneously make predictions for the graph rather than on a node by node basis. Here we use **CLUS-HMC** and its variants (Blockeel et al., 2002, 2006; Vens et al., 2008), which extends the decision tree for multi-label classification on both tree and DAG label hierarchies. It is the state-of-the-art global HMC classifier among non-deep-learning methods. The details of all the above methods are summarized in Table 1.

Table 1: Details of the competing methods.

Method	Raw-score-based ranking	ClusHMC	HIROM	$\widehat{mLPR}$-based ranking
Type	local classifier (2nd stage)	global classifier	local classifier (1st and 2nd stage)	local classifier (2nd stage)
Input	Classifier Scores	Labels Y_i's; Covariates	Labels Y_i's; Covariates	Labels Y_i's; Classifier Scores
Estimators of $\mathbb{P}(Y_i)$, $\mathbb{P}(Y_i\|Y_{pa(i)})$	N/A	N/A	SVM, empirical estimator	SVM, empirical estimator
Ranking	naive sorting, HierRank	N/A	HierRank	naive sorting, HierRank
Other variants	N/A	version: vanilla, bagging	loss: Hierarchical Ranking, Hierarchical Hamming	Approximation: indpt, nbh, full; Estimator of LPR: local polynomial regression, Gaussian Mixture

We evaluate $\widehat{mLPR}$-based rankings using three HMC datasets: 1) A synthetic dataset with three trees that are comprised of 25 nodes; 2) the disease-gene-expression data (Huang et al., 2010); 3) the RCV1v2 data (Lewis et al., 2004). We use the truncated area under the hit curve as the evaluation metric. To be specific, we convert the result into a ranking where the top ones are more likely to be positive. Then we take top $\kappa \times 100\%$ of the samples as positive and the remaining as negative, where $\kappa \in \{0.1, 0.2, 0.3, 0.4, 0.5, 1\}$. For each κ, we get the corresponding recall rate, and compute the area under the hit curve truncated at this recall rate.

After obtaining an $\widehat{mLPR}$-based ranking, the next step is to cut the ranking to make the final decisions. One immediate solution is to resort to the validation set. Suppose the individual classifier for each node in $\mathcal{G}$ and the estimation methods of $\mathbb{P}(Y_i)$'s, $\mathbb{P}(Y_i|Y_{pa(i)})$s and LPRs are trained on the training set. Then we can determine the cutoffs on the validation set. Specifically, the classifiers and the estimation methods of LPRs, $\mathbb{P}(Y_i)$, $\mathbb{P}(Y_i|Y_{pa(i)})$ learned from the the training set can be applied to the validation set to produce $\widehat{mLPR}$s. Then HierRank/naive sorting takes these $\widehat{mLPR}$s to

produce a ranking of nodes for the validation set. Since the truth is known on this dataset, metrics like F-measure or precision (1 - FDR) can be computed at an arbitrary cutoff. Then a desired cutoff on this ranking can be chosen to control the FDR or to attain the maximal F-measure. The evaluation on the goodness of this cutoff is performed on the testing set. For other methods in Table 1, the same strategy can be used to select a cutoff.

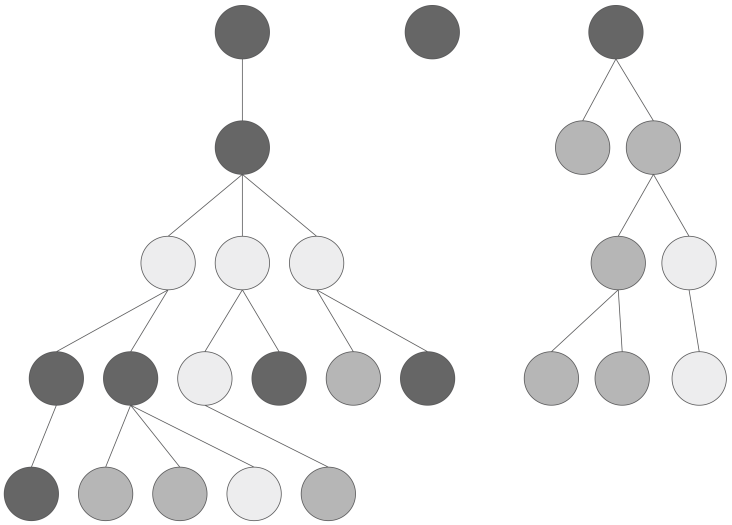
5.2 A synthetic data with a complicated tree structure

For the synthetic dataset, the setting is comprised of three trees with mixes of high- and low-quality nodes and varying levels of dependence between the nodes (Figure 5). The quality of a node refers to the ability of the corresponding classifier to distinguish between the positive and the negative. The simulation dataset consists of 5,000 training samples and 1,000 test samples. We generate the true instance status as follows. The conditional probabilities $\mathbb{P}(Y_i = 1|Y_{pa(i)} = 1)$ are randomly generated from a uniform distribution, with the constraint that each dataset has to have a minimum of 15 positive instances in the training set, which amounts to a minimum prevalence of 0.3% for any class. Given the instance status, the simulated classification score is sampled from the status-specific distribution — data are generated from a Beta(η, 6) distribution for the negative case and a Beta(6, η) distribution for the positive case, where $\eta = 2,\ 4,\ 5.5$ for the high, medium, low node quality respectively. Details of the score generation mechanism can be found in Table 2.

Table 2: Score distribution in terms of the node quality.

Quality	Positive instance	Negative instance	Node color
High	Beta$(6, 2)$	Beta$(2, 6)$	light grey
Medium	Beta$(6, 4)$	Beta$(4, 6)$	medium grey
Low	Beta$(6, 5.5)$	Beta$(5.5, 6)$	black

Figure 5: A 25-nodes tree-hierarchy. Light grey, medium grey, and black correspond to high, medium, and low quality, respectively.



Since there is no covariate for each sample, we just use empirical proportions for $\mathbb{P}(Y_i|Y_{pa(i)})$ and $\mathbb{P}(Y_i)$. From Table 3, we see that the $\widehat{mLPR}$-based ranking works best for the HMC task. Particularly for the $\widehat{mLPR}$-based ranking, the full version outperforms that of the neighborhood approximation. The independence approximation is the worst. We interpret this result as that the sample size is sufficiently large and the data quality is sufficiently good to learn

Table 3: The truncated area under the PR curve and the recall rate (inside the bracket) for the synthetic data. Here κ refers to the proportion of objects in the top that are taken as positive.

	κ	0.05	0.1	0.2	0.3	0.5
Raw	naive sorting	1.4 (5.3)	1.8 (8.5)	2.3 (14.0)	2.9 (20.2)	4.3 (35.4)
	HierRank	0.5 (5.1)	1.8 (13.5)	5.1 (30.4)	8.1 (45.5)	12.7 (69.1)
ClusHMC	vanilla	30.0 (32.7)	48.5 (54.4)	61.8 (76.6)	65.8 (85.8)	68.4 (93.9)
	bagged	31.7 (33.9)	50.8 (56.7)	63.6 (76.8)	67.9 (86.5)	70.4 (94.3)
HIROM	hier.ranking	33.7 (35.5)	56.1 (35.5)	70.6 (81.1)	71.9 (84.1)	73.2 (88.7)
	hier.hamming	34.4 (35.7)	54.9 (59.7)	73.0 (85.4)	75.1 (89.6)	76.1 (92.6)
$\widehat{mLPR}$-based ranking (mLPR + HierRank)	indpt	34.7 (35.7)	58.5 (62.3)	72.8 (82.8)	76.1 (89.9)	78.1 (95.8)
	nbh	36.0 (36.5)	61.3 (64.0)	77.0 (85.7)	80.4 (92.8)	82.1 (97.6)
	full	**36.2 (36.6)**	**62.1 (64.7)**	**78.3 (86.8)**	**81.8 (93.9)**	**83.4 (98.6)**
$\widehat{mLPR}$-based ranking (Gaussian + HierRank)	indpt	33.6 (34.7)	49.5 (53.7)	62.7 (76.7)	67.8 (87.8)	69.9 (94.3)
	nbh	33.9 (34.0)	54.8 (59.8)	69.8 (81.7)	73.5 (89.5)	75.4 (95.3)
	full	34.4 (35.2)	56.1 (60.2)	70.8 (81.6)	74.7 (89.9)	76.8 (96.3)
$\widehat{mLPR}$-based ranking (mLPR + naive sorting)	indpt	34.5 (35.5)	56.6 (60.7)	71.4 (82.2)	74.7 (89.2)	76.8 (95.6)
	nbh	34.9 (36.4)	60.9 (63.8)	76.7 (85.5)	80.1 (92.6)	81.8 (97.5)
	full	**36.2 (36.6)**	**62.1 (64.7)**	**78.3 (86.8)**	**81.8 (93.9)**	**83.4 (98.6)**

the estimators of the LPRs, $\mathbb{P}(Y_i|Y_{pa(i)})$'s and $\mathbb{P}(Y_i)$'s accurately. Therefore, the hierarchy information can be well learned when estimating the mLPRs. One direct evidence, based on Proposition 2, is that the $\widehat{mLPR}$-based ranking of the full version using the naive sorting is almost as good as that using HierRank. Finally, note that the result of the $\widehat{mLPR}$-based ranking with LPRs learned by modelling $\mathbb{P}(S|Y)$ as Gaussian densities is inferior to the other methods. It shows the advantage of using LPRs in Formula (d) instead of $\mathbb{P}(S|Y)$ in Formula (b) of Equation (4.1).

Given the $\widehat{mLPR}$-based ranking, the next step is to determine the cutoff to make the final decisions. To this end, we split the original training set into a training set and a validation set of equal sizes (2500 samples), and then use the cutoff selection approach discussed at the end of Section 5.1. In Table 4, we show the performance of this strategy. The cutoff is taken to attain a $(1-\alpha)\times 100\%$ recall rate ($\alpha = 0.01, 0.05, 0.1, 0.2$) or the maximal F-score on the validation set. Then the same cutoff is applied to the testing set. We see that the recall rate on the testing set is close to the target one for every method. For the F-score, the strategy also finds out the nearly maximal value for each method. These results indicate the reliability of the cutoff selection strategy. On the other hand, the results also corroborate the conclusion drawn from Table 3. The $\widehat{mLPR}$-based ranking of the full version makes the least discoveries to attain the desired recall rate, thus the least false positive discoveries against other methods. It also produces a ranking corresponding to the best maximal F-score.

5.3 Disease Diagnosis

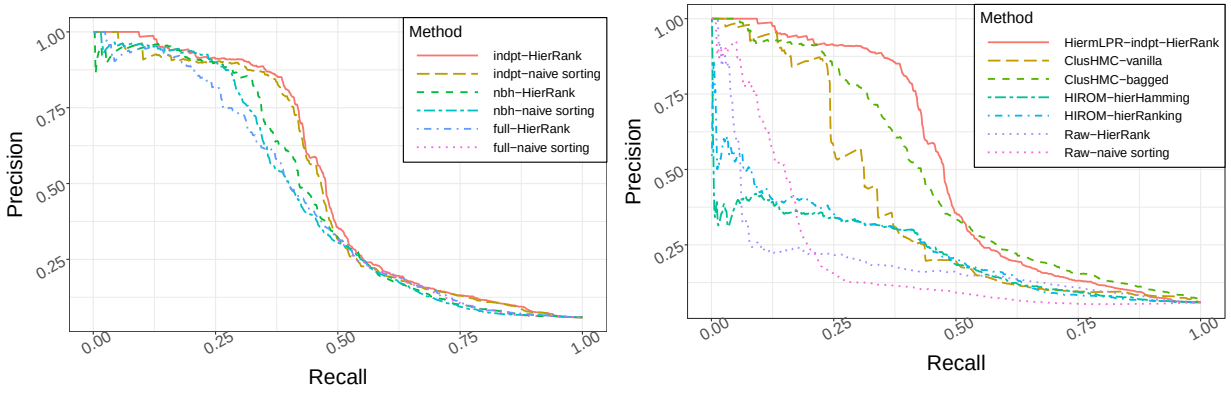
Huang et al. (2010) developed a classifier for predicting disease along the UMLS directed acyclic graph, trained on public microarray datasets from the National Center for Biotechnology Information (NCBI) Gene Expression Omnibus (GEO). They collected 100 studies, including a total of 196 datasets and 110 disease labels. The 110 nodes are grouped into 24 connected DAGs; see Figure 10 and 11 in Appendix D.2 for the detailed graphs. In general, the graphs have three properties: 1) It is shallow rather than deep; 2) It is scattered rather than highly connected; 3) Data redundancy occurs as an artifact of the label mining: the positive instances for a label are exactly the same as for its ancestors.

We follow the training procedure used in Huang et al. (2010) to obtain the binary Bayesian classifiers. Specifically, we use the leave-one-out cross validation to get the Bayesian classification scores for each sample. It guarantees that the classification scores have are identically distributed for all samples (see the related discussion in Section 5.4). Next, we use another round of leave-one-out cross validation to compute the mLPRs. Since the disease-gene-expression data has very limited sample size, we just use the empirical proportion for $\mathbb{P}(Y_i|Y_{pa(i)})$ and $\mathbb{P}(Y_i)$. Finally, we apply HierRank on these mLPRs to produce the ranking.

We compare the performance of the $\widehat{mLPR}$-based ranking against other competing methods. We get the truncated area under the PR curve in Table 5 and the cutoff results in Table 6 of Appendix D.4. In addition, the resulting precision-recall curve is shown in Figure 6. The $\widehat{mLPR}$-based ranking performs better than all of the other methods overall, and it performs significantly better in the initial portion of the precision-recall curve. It is not a surprise that the $\widehat{mLPR}$-based ranking of the indpt approximation performs better than the nbh approximation and the full version, since

Table 4: The tested value on the synthetic dataset for a given recall or the maximal F-score, which is obtained by the cutoff determined at the target value on the validation set. All the values are in percentage. The number inside the bracket is the proportion of objects in the top that are taken as positive. For the recall metrics, those that use less proportion to attain the target recall rate are highlighted in bold. For the F-max metrics (i.e., the maximal F-score), A/B means the attained F-score is A while the maximal F-score of the ranking is B. Those with the maximal A are highlighted in bold.

	Target Metric	Recall-99	Recall-95	Recall-90	Recall-80	F-max
Raw Scores	naive sorting	99.1 (98.8)	95.2 (94.0)	90.1 (84.8)	80.4 (64.3)	29.0/29.1 (47.3)
	HierRank	99.0 (98.4)	95.1 (95.5)	89.8 (92.3)	80.0 (86.3)	23.3/23.3 (98.8)
ClusHMC	vanilla	99.1 (81.8)	94.9 (54.1)	89.5 (36.8)	79.3 (22.8)	64.3/64.4 (14.9)
	bagged	99.1 (81.4)	94.9 (53.0)	89.7 (36.4)	79.4 (22.4)	66.0/66.1 (11.1)
HIROM	hier.ranking	99.0 (97.7)	94.9 (78.1)	89.4 (54.3)	79.5 (17.0)	73.2/73.2 (13.6)
	hier.hamming	99.2 (95.6)	94.7 (66.5)	89.6 (31.2)	79.8 (16.3)	71.9/72.0 (13.7)
$\widehat{mLPR}$-based ranking (LPR + HierRank)	indpt	98.9 (75.2)	94.9 (44.4)	89.5 (29.7)	79.6 (17.7)	71.4/71.5 (11.9)
	nbh	98.9 (65.0)	94.7 (35.8)	89.2 (24.5	79.6 (15.8)	74.2/74.2 (12.4)
	full	**99.0 (55.6)**	**94.6 (32.2)**	**89.3 (22.9)**	**79.8 (15.3)**	**74.8/74.9 (12.8)**
$\widehat{mLPR}$-based ranking (Gaussian + HierRank)	indpt	98.8 (83.7)	94.7 (54.1)	89.8 (34.0)	79.7 (21.6)	61.4/61.5 (18.1)
	nbh	98.8 (82.5)	94.8 (48.1)	89.4 (30.7)	79.5 (18.5)	70.2/70.2 (12.9)
	full	99.1 (69.2)	94.8 (43.3)	89.7 (29.9)	79.5 (18.4)	70.1/70.2 (13.7)
$\widehat{mLPR}$-based ranking (LPR + naive sorting)	indpt	99.0 (77.6)	94.4 (46.1)	89.4 (31.0)	79.8 (18.1)	70.4/70.4 (12.2)
	nbh	98.9 (64.1)	94.7 (36.3)	89.2 (24.8)	79.7 (15.9)	73.9/73.9 (12.4)
	full	**99.0 (55.6)**	**94.6 (32.2)**	**89.3 (22.9)**	**79.8 (15.3)**	**74.8/74.9 (12.8)**



(a) Variants of $\widehat{mLPR}$-based ranking

(b) All Methods [Replace HiermLPR by $\widehat{mLPR}$-based ranking]

Figure 6: Precision recall curve for several classifiers run on the real dataset of Huang et al. (2010).

it is hard to estimate $\mathbb{P}(Y_i|Y_{pa(i)})$ and $\mathbb{P}(Y_i)$ due to the limited sample size. In this case, the mLPR estimation does not fully incorporate the hierarchy information, and the assumption of Proposition 2 is violated. Thus, HierRank plays an important rule in making use of the hierarchy. This can be justified by Table 5, where the $\widehat{mLPR}$-based ranking gets significant gains via HierRank over the naive sorting. In contrast, the Raw-score-based ranking via HierRank behaves badly because the classification scores differ in distribution between classes. HIROM does not work well due to the poor estimators of $\mathbb{P}(Y_i|Y_{pa(i)})$'s and $\mathbb{P}(Y_i)$'s. ClusHMC is better but inferior to the $\widehat{mLPR}$-based ranking since it is not able to handle disconnected hierarchies.

5.4 RCV example

The RCV1v2 dataset contains $30,000$ samples in total and 103 categories. The data has a good quality and the categorization task is not difficult. We use this example mainly to illustrate a subtlety in the training process of the method that produces the $\widehat{mLPR}$-based ranking. In this method, we have two training stages: train the binary

Table 5: The truncated area under the PR curve and the recall rate (inside the bracket) for the disease dataset. Here κ refers to the proportion of objects in the top that are taken as positive.

	κ	0.01	0.1	0.2	0.3	0.5
Raw Scores	naive sorting	9.1 (11.0)	14.1 (25.5)	15.7 (38.4)	16.8 (48.9)	17.8 (60.6)
	HierRank	4.9 (6.2)	10.8 (31.9)	14.1 (51.5)	16.1 (65.8)	17.7 (80.5)
ClusHMC	vanilla	14.4 (15.1)	30.0 (42.9)	32.1 (53.7)	33.1 (61.0)	35.0 (79.7)
	bagged	15.0 (15.7)	39.9 (52.7)	43.2 (66.7)	45.0 (77.1)	46.5 (89.0)
HIROM	hier. ranking	4.6 (8.3)	17.1 (43.7)	19.7 (56.4)	20.7 (63.4)	21.6 (72.1)
	hier. hamming	2.6 (6.9)	15.1 (43.7)	17.4 (54.8)	18.5 (62.4)	20.0 (77.0)
$\widehat{mLPR}$-based ranking (LPR + HierRank)	indpt	**15.7 (16.0)**	**42.6 (52.1)**	**45.3 (63.5)**	**46.7 (72.2)**	**48.2 (85.1)**
	nbh	15.1 (16.1)	39.8 (51.4)	42.1 (61.1)	43.3 (68.6)	44.3 (77.8)
	full	14.0 (15.4)	37.4 (51.2)	40.2 (63.4)	41.3 (70.4)	42.2 (78.1)
$\widehat{mLPR}$-based ranking (LPR + naive sorting)	indpt	15.0 (15.7)	42.1 (50.9)	44.7 (62.2)	46.2 (71.8)	47.7 (84.6)
	nbh	15.2 (16.0)	39.9 (52.7)	41.0 (61.9)	41.9 (67.7)	42.8 (75.9)
	full	14.9 (15.7)	37.9 (50.9)	40.7 (63.0)	41.8 (69.8)	42.7 (77.5)

classification scores using SVM and train the LPR model using the SVM scores. For a fair evaluation, we split the dataset into three partitions: Partition I (25% of the samples), Partition II (25% of the samples), and Partition III (50% of the samples). First, we train the SVM model on partition I, and then predict the classification scores on each partition (Figure 7 (a)). The distributions of the classification scores on Partition I differs far from those on Partition II and Partition III, while the latter two distributions are quite similar. This is reasonable since Partition I is the training dataset and Partition II& III are the testing dataset for the SVM stage. For the LPR estimation, there are two possible training strategies:

1. Train the model for the LPR on Partition I and then predict the LPR scores on Partition II & III.
2. Train the model for the LPR on Partition II and then predict the LPR scores on Partition III.

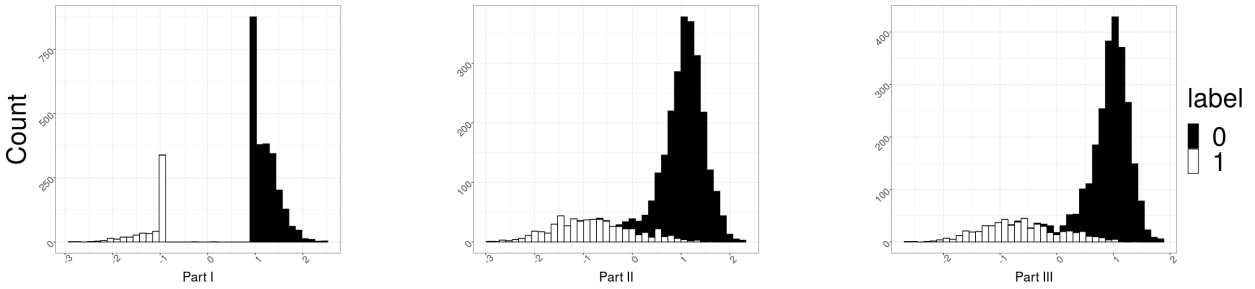
We suggest using the second strategy. To elaborate this point, we investigate the predicted LPR scores for both strategies. As shown in Figure 7 (b), for the first strategy, the distributions of LPRs are mixed between the positive and the negative groups on Partition II & III, while the distributions are clearly separated on Partition I. This results from the fact that distribution of the input SVM scores on Partition I deviates far from that on Partition II & III. This leads to a bad generalization from the training data to the testing data. In contrast, The second strategy overcomes this issue by training on Partition II and testing on Partition III, which have similar distributions of the SVM scores. As a result, the distributions of the LPRs between two groups are well separated on both Partition II and III.

By using the second strategy for data splitting, we evaluate the $\widehat{mLPR}$-based ranking against the competing methods on the RCV1v2 dataset, as shown in Table 7 and 8 of Appendix D.4. Almost all the methods can find out a majority of correct positives in the very beginning, since it is easy to classify documents in this dataset. It can be seen that the $\widehat{mLPR}$-based ranking outperforms all the other methods, which justifies our argument that the full consideration of the hierarchy is significantly beneficial. Also, we observed that the full version performs better than the indpt approximation and the nbh approximation.

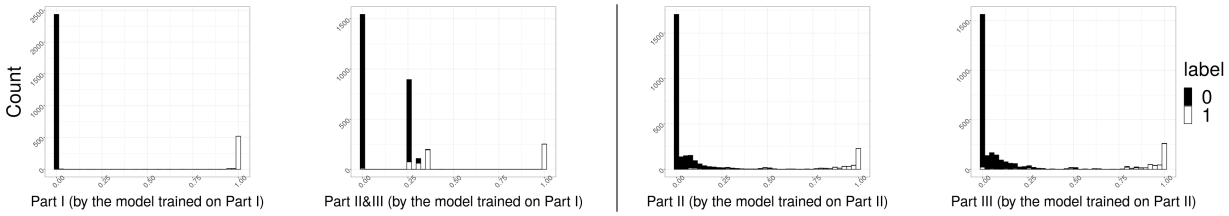
Finally, we need to point out that here we trim the factor $1/\mathbb{P}(Y_i)$ since it can be quite unstable if $\mathbb{P}(Y_i)$ is close to 0. This strategy has been commonly adopted in statistics and machine learning, e.g., the Iterated Probability Weights method in causal inference (Lee et al., 2011) and varieties of deep neural networks (Pascanu et al., 2013) use the clipping trick.

6 Conclusions

In this article, we present a method that produces the $\widehat{mLPR}$-based ranking for the second-stage decision in the two-stage HMC method. It is developed with the intent of maximizing the objective function ceAUHC (3.4) while respecting the hierarchy. We derive the statistics mLPR from ceAUHC, which takes into account both the hierarchical information and the statistical differences between classes. The maximization of ceAUHC is equivalent to sorting mLPRs from the largest to the smallest if the true mLPRs are available. The resulting ranking satisfies the Hierarchy-Consistency and has a nice property that the front node is more likely to be positive the tail node (Proposition 4). In reality, we have to resort to the estimation of mLPRs, which might no longer enjoy these properties. To this end, the ranking algorithm HierRank is proposed. It has been theoretically shown to optimize ceAUHC under the hierarchical constraint.



(a) SVM scores.



(b) LPR scores.

Figure 7: The predicted SVM/LPR scores. (a) The SVM model is trained on Partition I, and predicted on all partitions. (b) The LPR models are respectively trained on Partition I then predicted on all partitions (left), and trained on Partition II then predicted on Partition II & III (right).

We provide a faster versions of HierRank of $\mathcal{O}(n \log n)$ complexity, one developed by reducing the redundant computations of the original algorithm. We also extend the HierRank to the DAG structure. Furthermore, we show its advantage over the competing methods in one simulation study and two real data studies. Finally, we provide an approach for selecting a cutoff for the final decision on the $\widehat{mLPR}$-based ranking. It has been shown that this cutoff selection method can attain the target recall rate or maximize the F-score. For the above reasons, we recommend our method as a computationally efficient, statistically driven approach that produces a ranking for the second-stage decision in a local classification framework.

Despite the merits of the $\widehat{mLPR}$-based ranking mentioned above, there remains large room for improvement. First, we note that the $\widehat{mLPR}$-based ranking of the independent approximation, neighborhood approximation and the full version perform differently in terms of the data quality and sample size. It can be very useful to have a deep theoretical understanding how these factors affect the choice of the three types of mLPRs. Second, rather than using ceAUHC as the objective function in the second stage, it is of great interest to use it as an objective function to train an end-to-end classification system while taking into account the graph hierarchy. This can be embedded into a deep learning architecture to make a powerful classifier. [CITATION on the differential ranking paper]

7 Acknowledge

We thank Xinwei Zhang, Calvin Chi, Jianbo Chen for their suggestions on this paper.

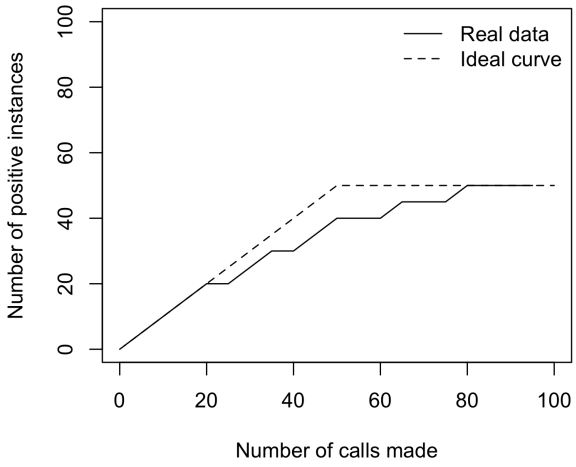


Figure 8: The hit curve.

Appendices

A Hit Curve

The hit curve has been explored in the information retrieval community as a useful alternative to the ROC curve and the PR curve. In a hit curve, the x-axis represents the number of discoveries and the y-axis represents the number of true discoveries (i.e., the hit number); see Figure 8. The hit curve is widely used in situations where the users are more interested in the top-ranked instances. For example, in evaluating the performance of a web search engine, the relevance of the top-ranked pages is more important than those that appear lower in search results because users expect that the most relevant results appear first. The hit curve can serve well in this situation as a graphic representation of the ranker's performance, since it would plot the results in order of decreasing relevance and the y-axis would indicate the results' relevance to the target. On the other hand, the ROC curve, which plots the true positive rates (TPR) against the false positive rates (FPR) at various threshold settings, does not depend on the prevalence of positive instances(Davis and Goadrich, 2006; Hand, 2009). In the case of search results, the number of relevant pages is tiny compared to the size of the World Wide Web (i.e., low prevalence of positive instances), which would result in an almost zero FPR for the top ranked pages. That is to say, with very few true positives, the early part of the ROC curve would fail to meaningfully visualize the search ranking performance. In the case of many hierarchical multi-label classification problems, like disease diagnosis problems, this issue exists as well; there are many candidate diseases to consider while few are actually relevant to the patient. Although the PR curve accounts for prevalence to a degree (i.e., showing the trade-off between precision and recall for different threshold), Herskovic et al. (2007) provided a simple example where the hit curve can be the more informative choice: with only five positive cases out of 1000, the hit curve's shape clearly highlighted the call order of a method that had called 100 instances before the five true positives, whereas the corresponding PR curve was uninformative (i.e., both the recall and precision rates are zero for the first 100 called instances).

B The local precision rate

B.1 Problem setting and notation

For consistency, we use the same notation as Jiang et al. (2014). Assume that classifiers have been learned for K labels connected in an acyclic graph and that there are M instances to be classified. We impose no requirements on class membership other than being hierarchically consistent: an instance could belong to none of the classes, and those that do belong to a class are not required to have leaf-level membership.

We assume that each label's classifier was trained on M instances and produces a score $s_{k,m}$ ($m = 1, \ldots M$, $k = 1 \ldots, K$) that can be thresholded to produce label assignments: without loss of generality we take larger scores to indicate the positive class, i.e. all instances with $s_{k,m} > \lambda_k$ are said to have label k. For example, if a logistic regression is used for predicting label k, a standard choice for $s_{k,m}$ is the estimated posterior probability that instance m belongs to label k.

Our classification framework begins with the scores for each instance, for which we assume the following generative model. If $Q_{k,m}$ is a binary indicator for whether instance m truly has label k, $Q_{k,m} = 1$ with probability π_k. We require that label membership implies membership in all of its ancestors: $P(Q_{Par(k),m} = 1|Q_{k,m} = 1) = 1$, where $Par(k)$ is the parent of label k. Also, we assume conditional independence of labels at the same hierarchical level: if labels k and j share a parent i, $Q_{k,m}$ and $Q_{j,m}$ are independent conditional on the parent status $Q_{i,m}$.

Given a threshold λ_k, the chance that the instance does not belong to the label k is given by $F_k(\lambda_k) = P(s_{k,m} \leq \lambda_k)$, the cumulative distribution function (cdf) for the scores of classifier k. This CDF can be expressed as a mixture of the score distributions for the two classes: $F_k = \pi_k F_{1,k} + (1 - \pi_k)F_{0,k}$, where $F_{1,k}$ is the CDF for those having the label k, and $F_{0,k}$ is the CDF for those without. Analogously, the respective density functions are denoted by $f_{1,k}$ and $f_{0,k}$, and the mixture density by f_k.

B.2 Definition and optimality result

Jiang et al. (2014) developed the local precision rate with the intention of maximizing precision with respect to recall in the multi-label setting. Specifically, they maximized an expected population version of the micro-averaged precision and recall rate given by Pillai et al. (2013). The micro-averaged precision rate has the form $\frac{\sum_k TP_k}{\sum_k (TP_k + FP_k)}$, where TP_k and FP_k are the number of true and false positives for label k, respectively.

We can write the expected pooled precision and recall rate, we first write the expected precision of the classifier for class k with threshold λ_k as

$$G_k(\lambda_k) = P(Q_{k,\cdot} = 1|s_{k,\cdot} > \lambda_k) = \frac{\pi_k(1 - F_{1,k}(\lambda_k))}{1 - F_k(\lambda_k)}. \tag{B.1}$$

From rearranging we also have that the joint probability $P(s_{k,\cdot} > s$ and $Q_{k,\cdot} = 1)$ is $(1 - F_k(s))G_k(F_k(s))$.

Then, we can pool decisions across all K labels using the thresholds $\lambda_1, \ldots, \lambda_k$ to obtain the expected pooled precision rate (ppr).

$$ppr = \frac{\sum_k (1 - F_k(\lambda_k))G_k(\lambda_k)}{\sum_k 1 - F_k(\lambda_k)} \tag{B.2}$$

The denominator represents the *a priori* expected number of times a given instance will be assigned to a label if the decision thresholds $\lambda_1, \ldots, \lambda_k$ are used. The pooled recall rate (prr) has the same form, except with $\sum_k Q_{k,\cdot}$ as the denominator instead.

Jiang et al. (2014) observed that to maximize the expected pooled precision with respect to pooled recall, it was enough to maximize $\sum_k (1 - F_k(\lambda_k))G_k(\lambda_k)$ while holding $\sum_k 1 - F_k(\lambda_k)$ fixed since $\sum_k Q_{k,\cdot}$ was a constant. The local precision rate (LPR) was then defined as

$$LPR_k(s) = -\frac{d}{dF_k(s)}\{(1 - F_k(s))G_k(s)\} = G_k(s) - (1 - F_k(s))\frac{dG_k(s)}{dF_k(s)} \tag{B.3}$$

In their main theoretical result Theorem 2.1, they showed that if the LPR for each class is monotonic, then ranking the KM LPRs calculated for each instance/class combination and thresholding the result produces a classification that maximizes the expected pooled precision with respect to a fixed recall rate. The monotonicity requirement is equivalent to having monotonicity in the likelihood of the positive class, and it is satisfied when higher classifier scores correspond to a greater likelihood of being from the positive class–this rules out poorly behaved classifiers, for example a multi-modal case where the positive class scores lie in the range $[0, 0.3) \cup (0.7, 1]$, and the negative class scores in $[0.3, 0.7]$.

B.3 Connection to local true discovery rate

After substituting expressions for the derivatives $\frac{dG_k(s)}{dF_k(s)} = \frac{dG_k(s)}{ds}\frac{ds}{dF_k(s)}$, the LPR can be shown to be equivalent to the local true discovery rate, $l$tdr.

$$LPR_k(s) = G_k(s) - (1 - F_k(s))\frac{dG_k(s)}{dF_k(s)} \tag{B.4}$$

$$= G_k(s) - (1 - F_k(s))\left[\frac{\pi_k f_{1,k}(s)}{(1 - F_k(s))f_k(s)} + \frac{\pi_k(1 - F_{1,k}(s))}{(1 - F_k(s))^2}\right] \tag{B.5}$$

$$= G_k(s) - \frac{\pi_k f_{1,k}(s)}{f_k(s)} - G_k(s) \tag{B.6}$$

$$= \frac{\pi_k f_{1,k}(s)}{f_k(s)} = l\text{tdr} \tag{B.7}$$

The local false discovery rate, $l$fdr $= 1 - l$tdr is its more well known relative; it has been studied extensively for Bayesian large-scale inference. This connection between a statistic used for hypothesis testing and the LPR, which was developed for classification, suggests the possibility that methodological developments on the LPR in classification could have meaningful implications for statistical inference. We elaborate on this connection in Section B.5.

B.4 Methods for estimating LPR

The optimality result in Jiang et al. (2014) was derived using true LPR values, which are generally unknown in practice. The authors discussed two methods for estimating the LPR. In the first method, estimates for $f_{0,k}$, f_k, and π_k are plugged in after expressing $LPR_k(s)$ as the local true discovery rate. In the second method, a local quadratic kernel smoother is used to simultaneously estimate $G_k(s)$ and $G'_k(s)$ in the definition of LPR.

Theoretically, Jiang et al. (2014) showed that under certain conditions, the first method converges to the true result faster than the second. However, in simulation studies the second method performed better than the first. The difference is due to the difficulty in estimating the densities $f_{0,k}$ and f_k on real data: any situation which would make kernel density estimation difficult would result in poor estimates of $l$tdr. For example, if the data are observed densely in one or two short intervals and sparsely elsewhere, the kernel density estimate of f_k would have bumps in the sparse regions, making $l$tdr unreliable. Further, because $f_{0,k}$ and f_k are estimated separately, they have different levels of bias and variance; in particular $f_{0,k}$ has larger variance (since it is only estimated from the negative class cases. In comparison, the functions $G_k(u)$ and $G'_k(u)$ are estimated jointly in the second method and $G_k(u)$ is always densely observed, as its domain is score percentiles rather than the scores themselves.

For both estimation methods, it is possible to obtain LPR estimates that are negative. Users must adopt a heuristic to handle these cases.

Lee (2013) suggested an alternative based on the second method that averages the estimates obtained via a weighted spline fit on bagged samples of the data. The addition of weights and bagging were introduced in order to estimate the precision function more robustly in regions supported by less data.

B.5 Connection to statistical inference

The key distinction between inference and classification is the presence of training data, which allows users to estimate distributions that are assumed unknown in statistical inference. If we choose to ignore the available class distribution information, one can reframe a two-class classification problem as a hypothesis testing problem where the null corresponds to membership in the negative class. The classifier score could be used as a statistic, although this means that one would need to train the classifier on the available data while pretending that they cannot estimate the class distributions. This approach clearly fails to take full advantage of the available data, but is meant to highlight the connection between these two problems.

The local precision rate is closely related to another statistic used in Bayesian large-scale inference, the local false discovery rate (Jiang et al., 2014). The local false discovery rate was motivated by the insight that in large-scale inference, enough data is available to estimate class distributions with some accuracy. As a result, it is possible to use point-wise statistics based on $f_0(s)/f_1(s)$, which may contain more information than their more popular tail-probability counterparts. Most of the literature on this statistic has come from Bradley Efron, who laid the groundwork theory and

provided interesting applications of the local false discovery rate in microarray gene expression experiments in Efron (2005, 2007, 2012). Cai and Sun (2009) proved an optimality result similar to that of Jiang et al. (2014) for a multiple inference procedure for grouped hypotheses that uses local false discovery rates: their procedure minimizes the false non-discovery rate subject to a constraint on the false discovery rate.

Research on hierarchical hypothesis testing is limited but growing. Yekutieli et al. (2006) first defined different ways to evaluate FDR when testing hypotheses along a tree, and gave a simple top-down algorithm for controlling these error types in Yekutieli (2008). In that work, the hypotheses at each level of the tree were assumed independent. More recently, Benjamini and Bogomolov (2014)'s work on selective inference provided an algorithm for testing on hypotheses arranged in families along a two-level tree where the parent and child are permitted to be highly dependent, although in so doing they give up control on the global FDR. Beyond the connection with the local false discovery rate, it remains to be seen whether other concepts from classification with LPRs can also be applied to inference. Most of the literature is concentrated on theoretical results that show that certain testing procedures can effectively bound a measure of Type I error. One possibility is that sorting algorithms with origins in computer science, like the one presented in this work, could have meaningful applications as testing procedures.

C Discussion on HierRank

C.1 An equivalent algorithm

We find an existing algorithm called Condensing Sort and Select Algorithm (CSSA) (Baraniuk and Jones, 1994) that is also of $\mathcal{O}(n \log n)$ complexity and can be adapted to solve (3.5). Bi and Kwok (2011) first extended CSSA in their proposed decision rule for the HMC problem. In their paper, CSSA was used to provide an approximate solution to the integer programming problem

$$\max_{\Psi} \quad \sum_{k \in \mathcal{T}} B(k)\Psi(k) \tag{C.1}$$

$$s.t. \quad \Psi(k) \in \{0,1\}, \forall k, \quad \sum_{k \in \mathcal{T}} \Psi(k) = L, \tag{C.2}$$

$$\Psi \text{ is } \mathcal{T}\text{-non-increasing},$$

where $\mathcal{T}$-nonincreasing means that $\Phi(k) \leq \Phi(k')$ if node k' is the ancestor of node k; $B(k)$ is a score produced by kernel dependency estimation (KDE) approach (Weston et al., 2003). Instead of directly solving (C.1) with (C.2), Bi and Kwok (2011) tackles a relaxed problem by replacing the binary constraint (C.2) by

$$\Psi(k) \geq 0, \forall k, \quad \Psi(0) = 1, \sum_{k \in \mathcal{T}} \Psi(k) \leq L. \tag{C.3}$$

Bi and Kwok (2011) proposed Algorithm 3 to solve this problem.

Algorithm 3 The CSSA algorithm

Input: A forest $\mathcal{S}$
Denote $Par(S_i)$ as the parent of supernode S_i, $n(S_i)$ as the number of nodes in S_i, and Ψ as a vector indicating which nodes are selected.

Initialize $\Psi(0) \leftarrow 1$; $\Gamma \leftarrow 1$.
Initialize all other nodes as supernodes with $\Psi(k) \leftarrow 0$ and sort them according to the LPR value.
while $\Gamma < L$ **do**
  Find $i = \arg\max_i \ \frac{1}{n(S_i)} \sum_{j \in S_i} LPR_j$
  if $\Psi(Par(S_i)) = 1$ **then**
    $\Psi(S_i) \leftarrow \min\{1, (L - \Gamma)/n(S_i)\}$
    $\Gamma \leftarrow \Gamma + n(S_i)$
  else
    Condense S_i and $Par(S_i)$ as a new supernode.
  end if
end while
Output: Vector Ψ.

It turns out that CSSA can be modified as Algorithm 4 that is shown to generate the same result as Algorithm 2 (see Theorem 6). On the other hand, we note CSSA and Algorithm 2 differ in the following aspects.

First, Algorithm 2 is independently introduced and interpreted in the context of ceAUHC, with a statistical justification for ordering nodes using LPR in particular. CSSA originates in signal processing and has been successfully used in wavelet approximation and model-based compressed sensing (Baraniuk and Jones, 1994; Baraniuk, 1999; Baraniuk et al., 2010).

Second, the optimality results of Bi and Kwok (2011) cannot be directly applied to derive the optimality of Algorithm 4.To be specific, we revisit the objective function in (C.1). Note that Algorithm 3 has a property that $\Psi(k) = 1$ for L implies $\Phi(k) = 1$ for L' when $L < L'$. It implies this algorithm produces a ranking list regardless of the choice of L. If this property also holds for the solution to (C.1) with (C.2), summing up these objective function values over $L = 1, \ldots, n$ maximizes (3.5) (replace $B(k)$ with LPR_k). Unfortunately, this strategy fails on the objective function (C.1) with (C.3), that is, the weight for the k-th node can be either smaller or larger than $(n - k + 1)$ by summing up the solutions to (C.1) with (C.3). For example, if the last selected supernode $n(S_i)$ in Algorithm 3 contains $n(S_i) > L - \Gamma$, then $\Psi(S_i) = (L - \Gamma)/n(S_i) < 1$. The relaxation constraint (C.3) disables the direct extension of what have been proved in Bi and Kwok (2011) to the optimality of ceAUHC for Algorithm (4). Therefore, our analysis provides a novel insight for CSSA. By showing that Algorithm (2) and it is equivalent to Algorithm (4) in terms of the produced ranking, we connect CSSA to the optimality of ceAUHC.

Finally, Algorithm 2 merges the chains from the bottom up, rather than as in CSSA, constructing ordered sets of nodes called supernodes by starting from the node with the largest value in the graph and moving outward. It is easy to see that the blocks defined in Algorithm 6 (the faster version of HierRank introduced in Appendix C.2) is exactly the same as the supernodes taken off in Algorithm 4. Hence, our independently proposed algorithm provides some novel insight into CSSA under the HMC setting.

Theorem 6 *Algorithm 2 and Algorithm 4 yield the same ordering, so Algorithm 4 maximizes ceAUHC as well.*

Algorithm 4 An equivalent algorithm modified from CSSA.

Input: A forest $\mathcal{S}$

Denote $Par(S_i)$ as the parent of supernode S_i, $n(S_i)$ as the number of nodes in S_i, and $\mathcal{L}$ as a vector for holding sorted LPR values.

Procedure:

1: Initialize with one node per LPR value, and each node as its own supernode, $\mathcal{L} = []$ (empty vector).
2: **while** $|\mathcal{L}| < n$ **do**
3: Find $i = \arg\max_i \frac{1}{n(S_i)} \sum_{j \in S_i} LPR_j$
4: **if** $Par(S_i) = \emptyset$ **then**
5: Take the nodes in S_i off the graph and append them to $\mathcal{L}$.
6: **else**
7: Condense S_i and $Par(S_i)$ into a supernode.
8: **end if**
9: **end while**

Output: A hierarchically consistent ordering of n LPR values.

C.2 A faster implementation of HierRank

To solve the scalability issue of Algorithm 2, we propose a faster version of HierRank by reducing redundant and repetitive computations in Algorithm 2. The speed-up is motivated by the following observations: 1) Algorithm 1 breaks a single chain into multiple blocks via the formula $(r', i') = \arg \max_{C_r(i) \subset \mathcal{S}} \bar{\ell}_{r,i.}$; 2) It can be shown that these blocks can only be agglomerated into a larger block rather than being further partitioned into smaller ones; 3) the agglomeration occurs only between a parent block and its child block in the hierarchy. Thus, HierRank can be implemented at the block level so that the partition is only executed once during multiple merging. By considering the above facts and taking care of other details, we obtain a faster version of HierRank (see Algorithm 6), which costs $\mathcal{O}(n \log n)$ computations.

In Algorithm 1, we note the fact that each sub-chain in the tree can be partitioned into multiple blocks — given a chain C_r, the breaking points are sequentially defined as

$$p_j := \begin{cases} \max_{1 \le i \le |C_r|} \frac{1}{C_r(i)} \sum_{k \in C_r(i)} LPR_k, & \text{if } j == 1 \\ \max_{p_{j-1} < i \le |C_r|} \frac{\sum_{k \in C_r(i)/C_r(p_{j-1})} LPR_k}{|C_r(i)| - |C_r(p_{j-1})|}, & \text{if } j > 1 \end{cases} \tag{C.4}$$

For example, Figure 9 (i) shows a chain of 6 nodes can be partitioned into two blocks. During the merging procedure of Algorithm 2, it turns out that the blocks defined by the above partitions will not be broken into smaller pieces, but can be further agglomerated. To show this, suppose there are two consecutive blocks in a chain, B_1, B_2, and B_1 locates ahead of B_2. Now we reform the blocks from the nodes in B_1 and B_2, using the rule in (C.4). It is obvious that nodes in B_1 will be clustered together. It remains to see which nodes in B_2 will be clustered with the nodes in B_1. Denote by $B_2(i)$ a sub-block consisting of the first i nodes in B_2, and by $\ell_B = \frac{1}{|B|}\sum_{k\in B} LPR_k$ given a block B. Then, the average LPR of the nodes in B_1 and the first i nodes in B_2 is computed as:

$$\ell_{B_1\cup B_2(i)} \quad = \quad \frac{|B_1|\ell_{B_1} + i\ell_{B_2(i)}}{|B_1|+i} = \ell_{B_1} + \frac{\ell_{B_2(i)} - \ell_{B_1}}{|B_1|/i+1}. \tag{C.5}$$

By the definition of block B_2, we have $\ell_{B_2} \geq \ell_{B_2(i)}, \forall i = 1,\ldots,|B_2|$. If $\ell_{B_1} > \ell_{B_2}$, none of the nodes in B_2 will be clustered together will the nodes in B_1. If $\ell_{B_1} \leq \ell_{B_2}$, (C.5) shows that all the nodes of B_1 and B_2 will form a new block. Therefore, blocks will not be broken into pieces, but can be further agglomerated. During the merging of multiple chains whose roots have the same parent, no blocks will be agglomerated since the blocks are sorted in a descending way along the merged chain; see the three descendant blocks of the bold block in Figure 9 (ii). On the contrary, blocks can be agglomerated with those from the parent chain. Figure 9 (iii) shows that after the chain merging, the blocks in the merged chain can be further agglomerated with the parent block (the bold one).

These observations motivate us to propose Algorithm 6, a faster version of Algorithm 2. We avoid repeatedly computing moving averages by partitioning each chain into blocks, storing the size and the average of each block. Specifically, there are three new components we need for Algorithm 6:

- **Detect breaking points.** For a chain C_r, breaking points can be detected by (C.4). Many existing algorithms can be used to this end. For example, recursion leads to an $\mathcal{O}(|C_r|\log|C_r|)$ time complexity. Figure 9 (i) illustrates this step.
- **Merge multiple chains with defined blocks**. Merging m multiple chains with detected blocks can be realized using the k-way merge algorithm. The time complexity is $\mathcal{O}(s\log m)$, where s is the total number of blocks in these chains. Figure 9 (ii) illustrates this step using a tree of five blocks.
- **Agglomerate the upstream chain and the downstream merged chain.** For a node $v \in \mathcal{P}$, denote by $C^{(v)}$ the chain ends with the node v. Suppose the children of v as $r_1,\ldots,r_H$. Denote by C_v the chain output by merging $C_{r_1},\ldots,C_{r_H}$ using the k-way merge algorithm. Denote the blocks of $C^{(v)}$ by $B_1^{(up)},\ldots,B_s^{(up)}$ and the blocks of C_v by $B_1^{(down)},\ldots,B_t^{(down)}$. Algorithm 5 agglomerates the blocks of $C^{(v)}$ and C_v with a time complexity of $\mathcal{O}(|C^{(v)}|+|C_v|)$. Figure 9 (iii) illustrates this step using the output of Figure 9 (ii).

Throughout Algorithm 6, the total time complexity consists of three parts: 1) detecting breaking points requires $\mathcal{O}(n\log K)$ computations; 2) merging multiple chains with defined blocks requires $\mathcal{O}(Dn\log K + n\log M)$ computations, where D is the number of nodes that have multiple children in the graph for one instance. The quantity D upper bounds the number of times each sub-chain merges during the algorithm; 3) agglomerating the upstream chain and the downstream merged chain requires $\mathcal{O}(Dn)$ computations. In total, the time complexity of Algorithm 6 is $\mathcal{O}(Dn\log K)$. In reality, most tree structures are shallow with $D < 10$. For example, the $D = 6$ and $D = 5$ in Figure 10 and Figure 11 respectively. So our algorithm is actually of $\mathcal{O}(n\log K)$ runtime for practical use.

Algorithm 5 Agglomerate the blocks in the upstream chain and the downstream chain

Input: Blocks $B_1^{(up)},\ldots,B_s^{(up)}$ from the upstream chain $C^{(v)}$ and Blocks $B_1^{(down)},\ldots,B_t^{(down)}$ from the downstream chain C_v.

Procedure:

Let b_0 be $B_1^{(down)}$, b_{-1} be the block ahead of it in $C^{(v)}$ and b_{+1} be the block after it. Denote by $\ell_{b_0}, \ell_{b_{-1}}, \ell_{b_{+1}}$ the averaging LPR within b_0, b_{-1} and b_{+1} respectively.
while $\ell_{b_0} > \ell_{b_{-1}}$ or $\ell_{b_{+1}} > \ell_{b_0}$ **do**
  if $\ell_{b_0} > \ell_{b_{-1}}$ **then**
    Agglomerate b_0 and b_{-1}. The new block is still called b_0 and the block ahead of it now is called b_{-1}.
  else
    Agglomerate b_0 and b_{+1}. The new block is still called b_0 and the block after it now is called b_{+1}.
  end if
end while

[1] **Output:** The new sequence of blocks.

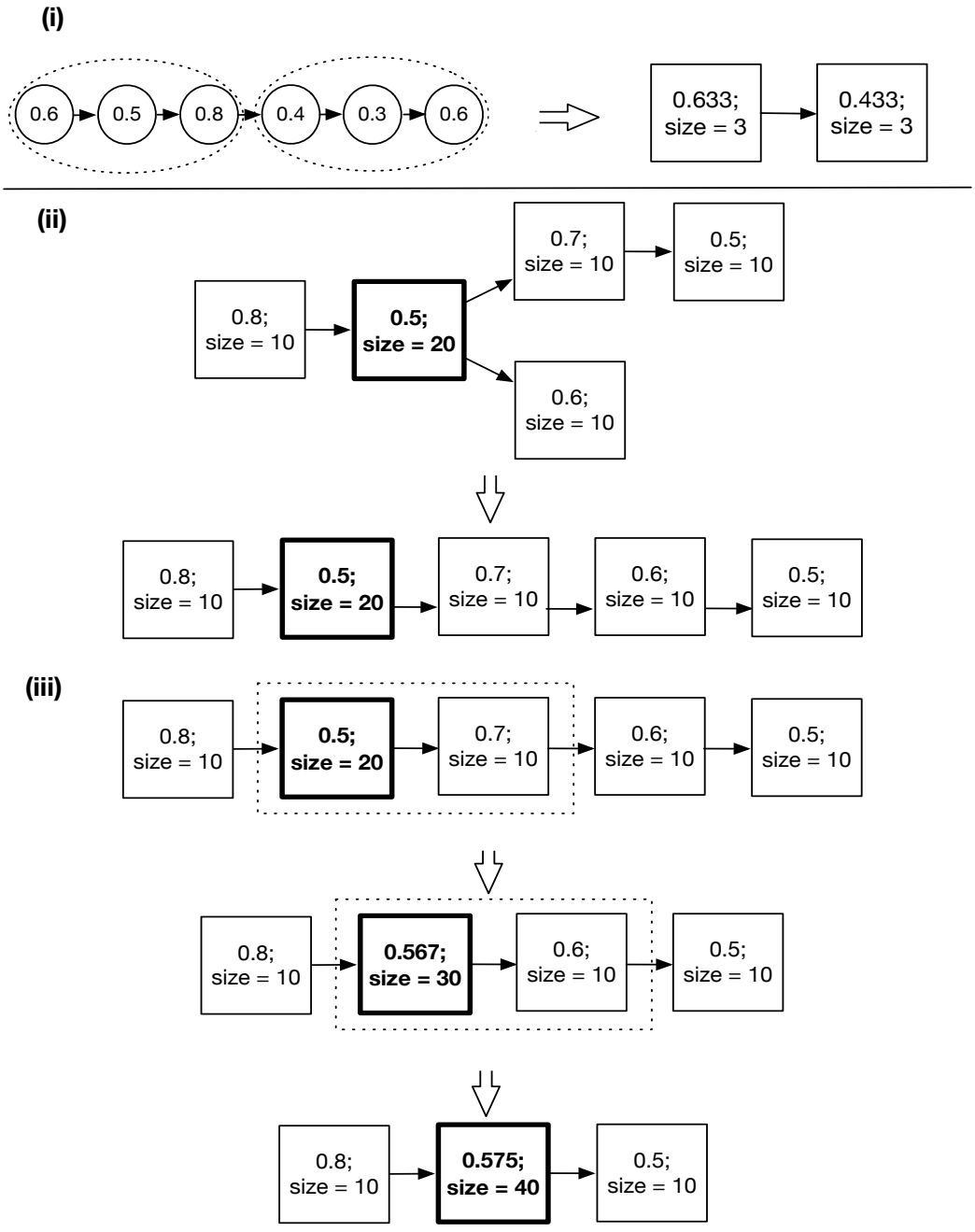


Figure 9: Illustrating the three components in Algorithm 6 using two examples which are separated by the solid line. The first example starts from a tree of six nodes and the second example starts from a tree of five blocks. (i) Detect breaking points of the chain of six nodes and partition them into two blocks. (ii) Merge the two children chains of the bold block. (iii) Agglomerate the upstream chain and the downstream chain around the bold node. The final list of blocks are positioned in a descending way.

Algorithm 6 A faster implementation of the HierRank algorithm.

Input: A forest $\mathcal{S}$

Procedure:

1: Figure out $\mathcal{P}$.
2: **while** $\mathcal{P} \neq \emptyset$ **do**
3: Pop out a v from $\mathcal{P}$. Take out all of its children $r_1, \ldots, r_H$. These children's descendants have at most one child. Denote the chain ends with the node v as $C^{(v)}$.
4: Find the breaking points $p_1^{(h)}, \ldots, p_{S_h}^{(h)}$ for C_{r_h} by (C.4), $h = 1, \ldots, H$.
5: Merge $C_{r_1}, \ldots, C_{r_H}$, in terms of the averaging LPRs of the blocks separated by the breaking points. Denote the new chain as C_v.
6: Agglomerate blocks of $C^{(v)}$ and C_v by Algorithm 5.
7: **end while**
8: **if** There remain multiple chains **then**
9: Merge them use the k-way merge algorithm.
10: **end if**
11: Let $\mathcal{L}$ be the resulting chain.

[1] **Output:** a ranking $\mathcal{L}$.

C.3 Extension to DAG

Directed acyclic graph (DAG) is a more general hierarchy than the tree structure and is more applicable to real data. In the DAG hierarchy structure, one node can have more than one parent, which brings about an additional decision issue – which parent the node and its descendants should respect. We call it the "AND" constraint if the node respects its all parents and call it the "OR" constraint if the node only respects one of its parents. Denote by Q all the nodes that have at least two parents.

It is easy to extend our algorithm for the tree hierarchy to the DAG structure by dynamic programming. For each node in Q, we explore all the possible cases that this node respects to one of its parents and cut its edges to other parents. Such strategy obviously works for the "OR" constraint. Bi and Kwok (2011) has shown that at least one case satisfies the "AND" requirement. Each case boils down to a forest, thus we can use Algorithm 6 to get a ranking. For the "OR" requirement, we just select the case with the highest objective function value. For the "AND" requirement, we select the scenario that attains the highest value among those satisfy the requirement.

Although the above brute-force strategy looks clumsy and time-consuming, it works for most practical scenarios, since most applications have shallow and scattered hierarchy structures. For instance, in Figure 10, there are at most seven nodes that have more than one parents in a connected part, while there are at most two such nodes in Figure 11. So we only need to explore $2^7 = 128$ cases at most. Considering there are only limited number of labels, about 100 for most times, the computation time is definitely acceptable. Finally, we have shown the equivalence between Algorithm 2 and Algorithm 4 under the tree constraint in Section 4. It is easy to adapt the results to the DAG structure. Hence, for complicated and deep structures, we can use the extended version of Algorithm 4 to DAG hierarchy introduced in Bi and Kwok (2011).

Algorithm 7 The HierRank algorithm for the DAG hierarchy.

Input: The DAG graph $\mathcal{G}$, node scores $\mathbf{S}$.

Procedure:

Figure out $\mathcal{P} := \mathcal{P}_1 \cap \mathcal{P}_2$.
Figure out $\mathcal{Q} := \{v : v \text{ has more than one parent}\}$.
while There is a node with more than one children or more than one parent. **do**
  while $\mathcal{P} \neq \emptyset$ **do**
    Pop out one v from $\mathcal{P}$. Take two children of v, r_1 and r_2.
    Feed C_{r_1} and C_{r_2} into Algorithm 1 and obtain $\mathcal{L}(r_1, r_2)$.
    Replace C_{r_1} and C_{r_2} with $\mathcal{L}(r_1, r_2)$.
    Update $\mathcal{P}$.
  end while
  if $\mathcal{Q} \neq \emptyset$. **then**
    Find $v \in \mathcal{Q}$ such that $\min_{u \in pa(v)} S_u = \min_{u \in \cup_{u' \in \mathcal{Q}} pa(u')} S_u$. Find $u := \arg\min_{u' \in pa(v)} S_{u'}$.
    Let $pa(u) = pa(u) \cup pa(v)/u$, $pa(v) = u$.
    Update $\mathcal{P}$ and $\mathcal{Q}$.
  end if
end while
if There remain multiple chains **then**
  Apply Algorithm 1 to these chains.
end if
Let $\mathcal{L}$ be the resulting chain.

[1] **Output:** a ranking $\mathcal{L}$.

D Experiments

D.1 Background of the disease data

Huang et al. (2010) developed a classifier for predicting disease along the Unified Medical Language System (UMLS) directed acyclic graph, trained on public microarray datasets from the National Center for Biotechnology Information (NCBI) Gene Expression Omnibus (GEO). GEO was originally founded in 2000 to systematically catalog the growing volume of data produced in microarray gene expression studies. GEO data typically comes from research experiments where scientists are required to make their data available in a public repository by a grant or journal guidelines. At of July 2008, GEO contained 421 human gene expression studies on the three microarray platforms that were selected for analysis (Affymetrix HG-U95A (GPL91), HG-U133A (GPL96), and HG-U133 Plus 2 (GPL570)). Briefly, 100 studies were collected, yielding a total of 196 datasets. These were used for training the classifier in Huang et al. (2010).

Labels for each dataset were obtained by mapping text from descriptions on GEO to concepts in the UMLS, an extensive vocabulary of concepts in the biomedical field organized as a directed acyclic graph. The mapping resulted in a directed acyclic graph of 110 concepts matched to the 196 datasets at two levels of similarity – a match at the GEO submission level, and a match at the dataset level, with the latter being a stronger match. The disease concepts and their GEO matches are listed in Table S2 in the supplementary information for Huang et al. (2010).

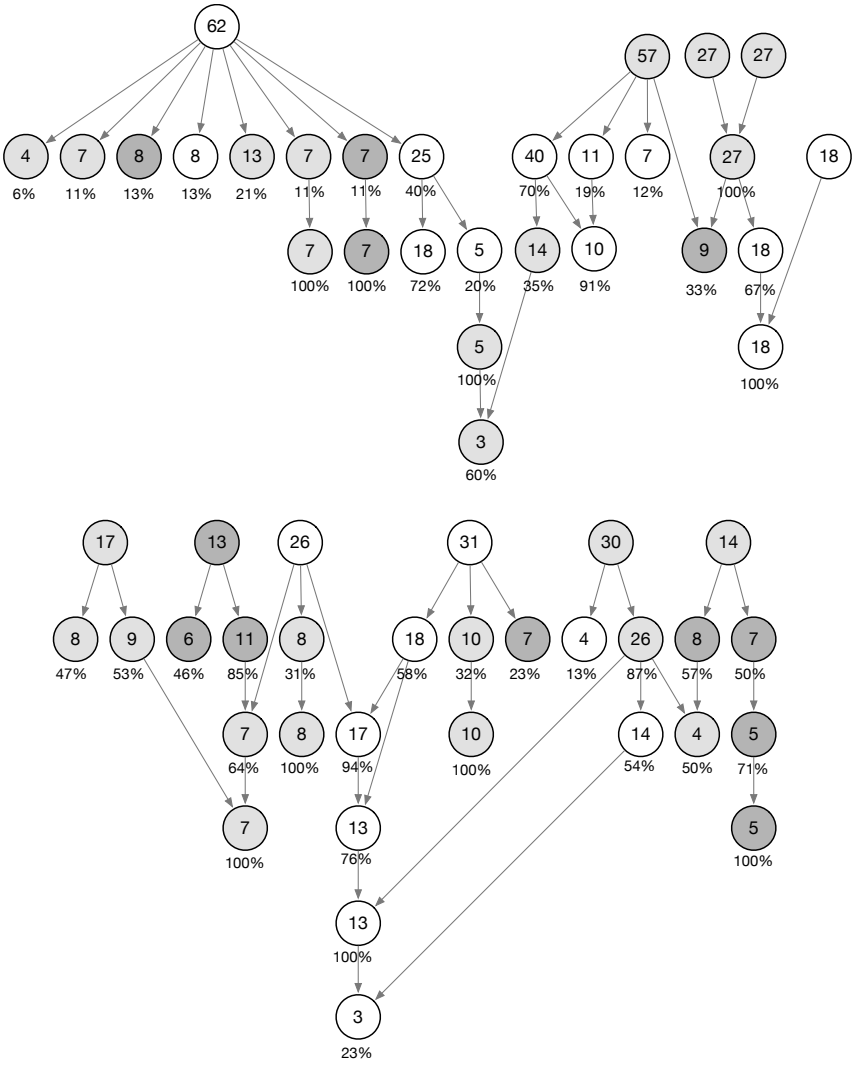
Training a classifier for each label is a complex multi-step process, and is described in detail in the Supplementary Information of Huang et al. (2010). We summarize that process here. In the classifier for a particular disease concept, the negative training instances were the profiles among the 196 that did not match with that disease concept. The principal modeling step involved expressing the posterior probability of belonging to a label in terms of the log likelihood ratio and some probabilities that have straightforward empirical estimates. The log likelihood ratio was modeled with a log-linear regression. A posterior probability estimate was then obtained for each of the 110×196 instances in the data by leave-one-out cross-validation, i.e. estimating the i-th posterior probability based on the remaining ones. These posterior probability estimates were used as the first-stage classifier scores. An initial label assignment for the first-stage was then obtained by finding the optimal score cutoffs for each classifier.

D.2 Characteristics of the disease diagnosis data and hierarchy

The full hierarchy graph is partitioned into two parts as shown in Figures 10 and 11 respectively. As the two figures show, the 110 nodes are grouped into 24 connected sets. In general, the graph is shallow rather than deep: the maximum node depth is 6, though the median is 2. Only 10 nodes have more than one child. This occurs because 11 of the

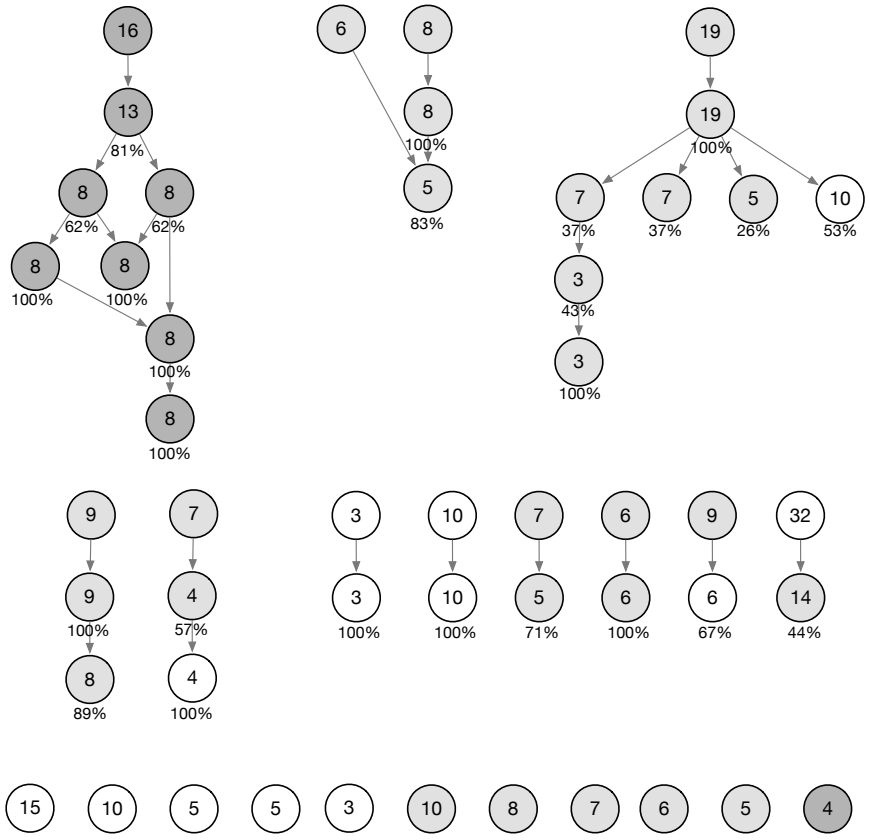
connected sets are standalone nodes, while six are simple two-node trees. The two largest sets consist of 28 and 30 nodes, respectively.

Figure 10: Structure of the disease diagnosis dataset, part 1 of 2. The colors correspond to node quality: white indicates that a node's base classifier has AUC between $(0.9, 1]$; light grey, $(0.7, 0.9]$, dark grey, $(0, 0.7]$. The values inside the circles indicate the number of positive cases, while the value underneath gives the maximum percentage of cases shared with a parent node.



The graph nearly follows a tree structure. Most nodes have only one parent or are at the root level. Only 15 nodes have 2 parents, and 2 nodes have 3 parents. Most nodes do not have a high positive case prevalence. The largest number of samples belonging to a label is 62, or a 32.63% positive case prevalence. The average prevalence is 5.89%, with a minimum prevalence of 1.53%, corresponding to 3 cases for a label. Data redundancy occurs as an artifact of the label mining: usually, the positive instances for a disease concept are the same as for its ancestors. There are few datasets that are tagged with a general label and not a leaf-level one. Twenty six nodes or 23.64% of all nodes share the same data as their parents, so they have the same classifier, and therefore the same classifier scores or LPRs as their parents. If we take the number of nodes that share more than half of their data with their parent, this statistic rises to 50%. A consequence of this redundancy is that the graph is shallower than appears in the figure: for example, the first connected set in the top left of Figure 11 appears to have six levels, but actually only has three because the last three levels do not contain any new information.

Figure 11: Structure of the disease diagnosis dataset, part 2 of 2. The colors correspond to node quality: white indicates that a node's base classifier has AUC between $(0.9, 1]$; light grey, $(0.7, 0.9]$, dark grey, $(0, 0.7]$. The values inside the circles indicate the number of positive cases, while the value underneath gives the maximum percentage of cases shared with a parent node.



D.3 Details of ClusHMC implementation

We use ClusHMC and follow Dimitrovski et al. (2011) by constructing bagged ensembles and used the original settings of Vens et al. (2008), weighting each node equally when assessing distance, i.e. $w_i = 1$ for all i. In addition to node weights, the minimum number of instances is set to 5, and the minimum variance reduction is tuned via 5-fold cross validation from the options 0.60, 0.70, 0.80, 0.90, and 0.95. Following the implementation of Lee (2013), a default of 100 PCTs are trained for each ClusHMC ensemble; each PCT is estimated by resampling the training data with replacement and running ClusHMC on the result.

D.4 Additional Results

Table 6: The tested value on the disease dataset for a given recall or the maximal F-score, which is obtained by the cutoff determined at the target value on the validation set. All the values are in percentage. The number inside the bracket is the proportion of objects in the top that are taken as positive. For the recall metrics, those that use less proportion to attain the target recall rate are highlighted in bold. For the F-max metrics (i.e., the maximal F-score), A/B means the attained F-score is A while the maximal F-score of the ranking list is B. Those with the maximal A are highlighted in bold.

	Target Metric	Recall-99	Recall-95	Recall-90	Recall-80	F-max
Raw Scores	naive sorting	99.4 (99.2)	96.4 (98.6)	91.4 (97.7)	83.0 (89.5)	22.1/22.7 (2.6)
	HierRank	**98.1 (97.0)**	95.3 (86.9)	84.6 (63.7)	71. 2 (41.4)	22.2/24.3 (23.1)
ClusHMC	vanilla	**99.4 (97.0)**	97.6 (77.0)	94.0 (67.6)	80.7 (50.0)	36.4/38.8 (31.0)
	bagged	100 (96.9)	**98.7 (70.2)**	95.6 (57.5)	85.6 (38.9)	46.6/46.7 (3.6)
HIROM	hier. ranking	98.5 (97.2)	97.2 (92.5)	93. 0 (84.2)	84.3 (64.7)	36.1/37.3 (8.5)
	hier. hamming	98.4 (94.6)	96.3 (91.5)	90.9 (81.3)	85.9 (62.4)	36.1/37.2 (8.4)
$\widehat{mLPR}$-based ranking (LPR + HierRank)	indpt	**98.7 (97.0)**	94.3 (87.5)	**87.4 (61.0)**	76.4 (37.6)	**52.6/53.4 (3.3)**
	nbh	99.7 (99.3)	92.6 (87.7)	88.5 (83.1)	73.8 (46.9)	48.3/49.5 (4.2)
	full	98.5 (96.6)	91.6 (86.5)	84.5 (71.9)	71.5 (43.4)	45.0/45.8 (4.0)
$\widehat{mLPR}$-based ranking (LPR + naive sorting)	indpt	98.1 (97.6)	96.1 (91.1)	89.0 (69.2)	**73.6 (36.7)**	**52.6/53.0 (3.3)**
	nbh	99.2 (98.6)	96.3 (92.2)	87.2 (77.9)	74.8 (52.8)	45.7/46.3 (2.8)
	full	99.0 (97.3)	92.9 (87.5)	85.9 (79.6)	74.1 (48.4)	44.7/45.5 (3.9)

Table 7: The truncated area under the PR curve and the recall rate (inside the bracket) for the RCV data. Here κ refers to the proportion of objects in the top that are taken as positive.

	κ	0.01	0.1	0.2	0.3	0.5
Raw Scores	naive sorting	0.5 (2.1)	0.6 (5.3)	0.6 (6.9)	0.6 (8.5)	0.6 (13.4)
	HierRank	0.9 (3.2)	1.3 (11.1)	1.5 (17.5)	1.7 (23.6)	2.0 (35.9)
ClusHMC	vanilla	25.3 (28.3)	59.0 (80.0)	60.6 (88.2)	60.9 (90.8)	61.1 (93.6)
	bagged	28.8 (29.8)	65.6 (83.7)	67.3 (92.0)	67.7 (95.7)	67.9 (98.3)
HIROM	hier. ranking	26.9 (27.8)	62.5 (80.0)	63.6 (85.9)	63.9 (89.1)	64.1 (92.1)
	hier. hamming	26.7 (27.7)	62.2 (88.8)	62.8 (91.7)	63.0 (93.8)	63.2 (96.1)
$\widehat{mLPR}$-based ranking (LPR + HierRank)	indpt	28.5 (29.7)	67.5 (86.6)	68.8 (92.7)	69.1 (95.2)	69.2 (96.3)
	nbh	28.7 (29.9)	70.9 (88.9)	**72.0 (94.1)**	72.3 (96.5)	72.4 (97.1)
	full	**28.9 (29.9)**	**71.0 (88.9)**	**72.0 (93.6)**	**72.4 (97.0)**	**72.5 (97.6)**
$\widehat{mLPR}$-based ranking (LPR + naive sorting)	indpt	28.1 (29.4)	66.3 (85.8)	67.6 (92.4)	67.8 (94.4)	67.9 (95.1)
	nbh	28.7 (29.8)	70.7 (88.8)	71.7 (93.5)	72.2 (97.1)	72.2 (97.8)
	full	28.7 (29.8)	70.9 (88.9)	**72.0 (94.2)**	**72.4 (96.8)**	72.4 (97.6)

Table 8: The tested value on the RCV dataset for a given recall or the maximal F-score, which is obtained by the cutoff determined at the target value on the validation set. All the values are in percentage. The number inside the bracket is the proportion of objects in the top that are taken as positive. For the recall metrics, those that use less proportion to attain the target recall rate are highlighted in bold. For the F-max metrics (i.e., the maximal F-score), A/B means the attained F-score is A while the maximal F-score of the ranking list is B. Those with the maximal A are highlighted in bold.

	Target Metric	Recall-99	Recall-95	Recall-90	Recall-80	F-max
Raw Scores	naive sorting	99.0 (100.0)	95.1 (99.8)	89.9 (99.4)	80.0 (98.7)	6.1/6.1 (100.0)
	HierRank	99.0 (100.0)	95.1 (99.8)	90.2 (98.9)	80.1 (98.0)	6.1/6.1 (100.0)
ClusHMC	vanilla	99.0 (91.7)	94.7 (59.8)	89.7 (24.9)	80.1 (10.0)	62.2/62.2 (2.5)
	bagged	98.9 (68.3)	94.6 (27.0)	89.3 (16.2)	79.4 (7.6)	64.3/64.5 (2.8)
HIROM	hier. ranking	99.0 (94.4)	94.6 (69.1)	89.7 (33.2)	80.1 (10.2)	65.6/65.6 (3.3)
	hier. hamming	99.0 (90.8)	94.8 (37.9)	89.9 (12.4)	79.9 (6.0)	60.4/60.6 (3.5)
$\widehat{mLPR}$-based ranking (LPR + HierRank)	indpt	**99.0 (88.5)**	95.1 (28.8)	90.2 (14.7)	80.0 (6.2)	64.5/64.6 (2.8)
	nbh	99.0 (88.8)	94.9 (23.3)	89.9 (11.1)	**80.0 (5.6)**	67.8/67.9 (2.8)
	full	99.0 (89.3)	95.1 (22.3)	**90.0 (11.1)**	79.9 (5.7)	**68.1/68.1 (2.7)**
$\widehat{mLPR}$-based ranking (LPR + naive sorting)	indpt	99.1 (91.1)	95.0 (40.2)	90.0 (16.0)	80.0 (6.5)	63.7/63.8 (2.9)
	nbh	99.0 (95.1)	95.0 (23.3)	89.9 (11.2)	80.0 (5.7)	67.7/67.7 (2.8)
	full	99.1 (95.2)	**95.1 (22.1)**	**90.0 (11.1)**	79.9 (5.7)	68.1/68.1 (2.7)

E Proofs of Theorems

E.1 Proof of Theorem 5

Proof The simplest case is a graph with only one chain, and the theorem obviously holds for this case. Next, suppose there are two chain branches, $X_{(1)}, \ldots, X_{(m)}$ and $Y_{(1)}, \ldots, Y_{(m')}$, both of which share the same parent node in $\mathcal{P}$. Directly merging these two chain branches by Algorithm 1 yields an ordering denoted as o_{XY}. Now given an arbitrary ordering o_A of all the n nodes in the entire graph, which respects the tree hierarchy. Denote by p_1 the first position of the nodes among $X'_{(i)}s$ and $Y'_{(j)}s$ within o_A, and denote all the nodes, other than $X'_{(i)}s$ and $Y'_{(j)}s$ and located after the p_1 position, by $W_{(1)}, \ldots, W_{(n')}$ (the position of $W_{(l)}$ is ahead of that of $W_{(l')}$ in o_A if $l < l'$). We want to show that

Lemma 7 *There exists an ordering $o_{A'}$ that is at least as good as o_A, in terms of ceAUHC, such that* $\big(\star\big)$ Node B is located ahead of Node C in o_A if it is the case in o_{XY}, where B, C are two distinct nodes from $X'_{(i)}s$ and $Y'_{(i)}s$.

Lemma 7 implies that in order to figure out the optimal ordering of the original tree structure, it boils down to replacing the two branches $X_{(1)}, \ldots, X_{(m)}$ and $Y_{(1)}, \ldots, Y_{(m')}$ by a single chain characterized by o_{XY} and seeking the optimal ordering of the new structure. The ordering $o_{A'}$ mentioned above can be constructed easily with two constraints: (i) fixing the nodes located ahead of the position p_1 as well as their ordering as in o_A; (ii) the position of $W_{(l)}$ is ahead of $W_{(l')}$ if $l < l'$, as in o_A. The first constraint is straightforward, and the second constraint can be satisfied by applying Algorithm 1 to $X_{(1)}, \ldots, X_{(m)}$, $Y_{(1)}, \ldots, Y_{m'}$ and $W_{(1)}, \ldots, W_{(n')}$. Here, we take $W_{(1)}, \ldots, W_{(n')}$ as a chain, regardless of their original structure. Without loss of generality, we assume the first maximal chain branch figured out by Algorithm 1 is $X_{(1)}, \ldots, X_{(t)}$, $t \leq m$ (we can skip the case that a part of $W_{(l)}$'s is the first maximal chain branch since it does not affect $\big(\star\big)$). In order to prove Lemma 7, it is reduced to showing that

Lemma 8 Conditional on (i) and (ii), the ordering of the maximal ceAUHC puts $X_{(1)}, \ldots, X_{(t)}$ at the position $p_1, \ldots, p_1 + t - 1$ respectively.

The detailed proof of Lemma 8 is deferred to Appendix E.2. Note that $X_{(1)}, \ldots, X_{(t)}$ must be located in the first place in the ordering o_{XY}. Therefore, by applying Lemma 8 in an inductive way (exclude $X_{(1)}, \ldots, X_{(t)}$ and apply the same argument on the remaining nodes), we can conclude the constructed $o_{A'}$ is at least as good as o_A and satisfies $\big(\star\big)$. Here, we need to clarify the point that putting $X_{(1)}, \ldots, X_{(t)}$ in such place does not violate the tree hierarchy since $X'_{(i)}s$ and $Y'_{(j)}s$ are the children chains of the same node, and none of $W'_{(l)}s$ can be an ancestor of $X'_{(i)}s$ or $Y'_{(j)}s$, otherwise o_A is not a valid ordering. Thus the proof is completed. ■

E.2 Proof of Lemma 8

Proof
Let a denote the average of these t values. For the sake of simplicity, we further assume $p_1 = 1$ and simply denote by $Z_{(1)}, \ldots, Z_{(n-t)}$ the combination of $X_{(t+1)}, \ldots, X_{(m)}$, $Y'_{(j)}s$ and $W'_{(l)}s$. Let the ordering o_A be as follows:

$$\begin{array}{cccccccccccc} Z_{(1)} & \ldots & Z_{(i_1-1)} & X_{(1)} & Z_{(i_1)} & \ldots & Z_{(i_t-t)} & X_{(t)} & Z_{(i_t-t+1)} & \ldots & Z_{(n-t)} \\ 1 & \ldots & i_1-1 & i_1 & i_1+1 & \ldots & i_t-1 & i_t & i_t+1 & \ldots & n \end{array}$$

where i_c is the position of $X_{(c)}$, $c = 1, \ldots, t$. Note that $i_{c+1} \geq i_c + 1$. Denote by $o_{A'_1}$ the ordering of $(X_{(1)}, \ldots, X_{(t)}) + o_A/(X_{(1)}, \ldots, X_{(t)})$, that is, move $(X_{(1)}, \ldots, X_{(t)})$ to the head of o_A. The difference in the value of the objective function (OF) between $o_{A'_1}$ and o_A can be written as follows:

$$\begin{aligned} \text{OF of } o_{A'_1} =& \sum_{i=1}^{t}(n-i+1)X_{(i)} + \sum_{j=1}^{n-t}(n-t-j+1)Z_{(j)} \\ \text{OF of } o_A =& (n-i_1+1)X_{(1)} + \ldots + (n-i_t+1)X_{(t)} \\ &+ \sum_{j=1}^{i_1-1}(n-j+1)Z_{(j)} + \ldots + \sum_{j=i_t-t+1}^{n-t}(n-(j+t)+1)Z_{(j)} \end{aligned}$$

$$
\begin{aligned}
\text{OF of } o_{A_1'} - \text{OF of } o_A &= \left[(i_1-1)X_{(1)} - \sum_{k=1}^{i_1-1} Z_{(k)}\right] + \ldots + \left[(i_t-t)X_{(t)} - \sum_{k=1}^{i_t-t} Z_{(k)}\right] \\
&= (i_1-1)\left[X_{(1)} - \frac{1}{i_1-1}\sum_{k=1}^{i_1-1} Z_{(k)}\right] + \ldots + (i_t-t)\left[X_{(t)} - \frac{1}{i_t-t}\sum_{k=1}^{i_t-t} Z_{(k)}\right] \\
&= (i_1-1)\left[\big(X_{(1)}-a\big) + \left(a - \frac{1}{i_1-1}\sum_{k=1}^{i_1-1} Z_{(k)}\right)\right] + \ldots \\
&\qquad + (i_t-t)\left[\big(X_{(t)}-a\big) + \left(a - \frac{1}{i_t-t}\sum_{k=1}^{i_t-t} Z_{(k)}\right)\right] \\
&= \big[(i_1-1)\big(X_{(1)}-a\big) + \ldots + (i_t-t)\big(X_{(t)}-a\big)\big] \\
&\qquad + \left[(i_1-1)\left(a - \frac{1}{i_1-1}\sum_{k=1}^{i_1-1} Z_{(k)}\right) + (i_t-t)\left(a - \frac{1}{i_t-t}\sum_{k=1}^{i_t-t} Z_{(k)}\right)\right]
\end{aligned}
$$

It remains to prove both the first term and the second term on the right side are non-negative:

- *The first term.* We can rewrite the sum

  $$(i_1-1)\big(X_{(1)}-a\big) + \ldots + (i_t-t)\big(X_{(t)}-a\big)$$

  as follows

  $$(i_1-1)\sum_{k=1}^{t}(X_{(k)}-a) \;+\; (i_2-i_1-1)\sum_{k=2}^{t}\big(X_{(k)}-a\big) + \ldots + (i_t-i_{t-1}-1)(X_{(t)}-a).$$

  The first sum $\sum_{k=1}^{t}(X_{(k)}-a) = 0$ since a is the average. The other sums being nonnegative follows from the fact that a must be at least as large as the smaller averages in the chain, i.e. $a \geq \frac{1}{c}\sum_{k=1}^{c} X_{(k)}$ where $1 \leq c \leq t$. In detail, we know that

  $$
  \begin{aligned}
  X_{(c+1)} + \ldots + X_{(t)} &= ta - [X_{(1)} + \ldots + X_{(c)}], \quad 1 \leq c \leq t-1 \\
  &\geq ta - ca = (t-c)a, \quad \text{from the fact above} \\
  (X_{(c+1)}-a) + \ldots + (X_{(t)}-a) &\geq 0
  \end{aligned}
  $$

  Therefore, each sum

  $$\sum_{k=c}^{t}\big(X_{(k)}-a\big) \geq 0, \quad c = 1, \ldots, t. \tag{E.1}$$

  It is clear that the expression

  $$(i_1-1)\sum_{k=1}^{t}(X_{(k)}-a) \;+\; (i_2-i_1-1)\sum_{k=2}^{t}\big(X_{(k)}-a\big) + \ldots + (i_t-i_{t-1}-1)(X_{(t)}-a)$$

  is exactly zero only when each $X_{(c)} = a$.

- *The second term.* We claim that each term in the expression

  $$\left[(i_1-1)\left(a - \frac{1}{i_1-1}\sum_{k=1}^{i_1-1} Z_{(k)}\right) + (i_t-t)\left(a - \frac{1}{i_t-t}\sum_{k=1}^{i_t-t} Z_{(k)}\right)\right]$$

  must be nonnegative, and equality holds only if there is a tie. To see this, we notice that $\sum_{k=1}^{i_c-c} Z_{(k)}$ can be separated as three sums: $\sum_{k=t+1}^{t_X} X_{(k)}$, $\sum_{k=1}^{t_Y} Y_{(k)}$ and $\sum_{k=1}^{t_W} W_{(k)}$, where $t_X \leq m$, $t_Y \leq m'$, $t_W \leq n'$ and $c = t_X - t + t_Y + t_W$. In terms of the procedure of Algorithm 1, it follows that

  $$\sum_{k=1}^{t_Y} Y_{(k)} \leq t_Y \cdot a \quad \text{and} \quad \sum_{k=1}^{t_W} W_{(k)} \leq t_W \cdot a.$$

For the same reason, $\sum_{k=t+1}^{t_X} X_{(k)} \leq (t_X - t)a$, otherwise we have $\frac{1}{t_X}\sum_{k=1}^{t_X} X_{(k)} > a$ and it violates the condition that a is the average of the first maximal chain branch. So we have

$$\frac{1}{i_c - c}\sum_{k=1}^{i_c - c} Z_{(k)} = \frac{1}{i_c - c}[\sum_{k=t+1}^{t_X} X_{(k)} + \sum_{k=1}^{t_Y} Y_{(k)} + \sum_{k=1}^{t_W} W_{(k)}] \leq a.$$

■

E.3 Proof of Theorem 6

Proof To establish the bridge between Algorithm 2 and Algorithm 4, we start from a simple case, i.e., a tree consisting of multiple chains with the same root (the root is in $\mathcal{P}$). Specifically, denote by R the root and these children chain by $C_s := \{X_1^{(s)}, \ldots, X_{k_s}^{(s)}\}$, $s = 1, \ldots, \nu$, where ν is the number of chains, and k_s is the length of the s_{th} chain. Without loss of generality, suppose $S_1 := C_1(i_1) = \{X_1^{(1)}, \ldots, X_{i_1}^{(1)}\}$ is the first supernode that has been condensed to R, if R has not been taken off, or the first to be taken off after R. We claim that

Lemma 9 *When merging $C_1, \ldots, C_\nu$, Algorithm 1 puts S_1 in the first place.*

To show Lemma 9, we only need to show $\frac{1}{i_1}\sum_{k\in C_1(i_1)} LPR_k \geq \frac{1}{i}\sum_{k\in C_s(i)} LPR_k$, $\forall\ i \in \{1, \ldots, k_s\}, s \in \{1, \ldots, \nu\}$. The detailed proof is deferred to Appendix E.4. Inductively, it implies that the ordering given by Algorithm 4 on such simple case is the same as Algorithm 2. Therefore, any complicated structure boils down to the above simple case, since we can inductively merge the sub-chains starting from a root in $\mathcal{P}$ using Algorithm 1. This completes the proof showing that the results of Algorithm 2 and Algorithm 4 are the same. ■

E.4 Proof of Lemma 9

Proof We show the proof in three steps:

(i) Along the chain C_1, all the sub-chains starting from $X_1^{(1)}$ with larger length than S_1 have at most as large average LPR as S_1, that is, $\bar{\ell}_{1,i} \leq \bar{\ell}_{1,i_1}, \forall i_1 < i \leq k_1$. In terms of the procedure of Algorithm 4, all the mean LPR values in the supernodes behind S_1 is no larger than ℓ_{1,i_1}, so the argument (i) holds.

(ii) Along the chain C_1, all the sub-chains starting from $X_1^{(1)}$ with smaller length than S_1 have at most as large average LPR as S_1, that is, $\bar{\ell}_{1,i} \leq \bar{\ell}_{1,i_1}, \forall 1 \leq i < i_1$. Otherwise, suppose $i_1' < i_1$ s.t. $\bar{\ell}_{1,i_1'} > \bar{\ell}_{1,i_1}$. By Eq. E.1, we know that $\sum_{i=c}^{i_1'}(X_i^{(1)} - \ell_{1,i_1'}) \geq 0$, $c = 1, \ldots, i_1'$. So to make any supernode right behind the one ending with $X_{i_1'}^{(1)}$ merged with its former supernode, the average LPR value of the former must be at least $\ell_{1,i_1'}$. Thus, we can inductively conclude that $\bar{\ell}_{1,i_1} \geq \bar{\ell}_{1,i_1'}$, which is a contradiction.

(iii) $\bar{\ell}_{s,i} \leq \bar{\ell}_{1,i_1}, \forall i \in \{1, \ldots, k_s\}, s \in \{1, \ldots, \nu\}$. Otherwise, wlog, suppose $\bar{\ell}_{2,i_2} > \bar{\ell}_{1,i_1}$. By Eq. E.1, any super node ending with $X_{i_2}^{(2)}$ has average LPR at least $\bar{\ell}_{2,i_2}$. Then it contradicts with the assumption that S_1 is the first supernode that will be merged with R, if R has not been taken off, or by the time S_1 is taken off.

■

References

Alves, R. T., Delgado, M., and Freitas, A. A. (2010). Knowledge discovery with artificial immune systems for hierarchical multi-label classification of protein functions. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–8. IEEE.

Ananpiriyakul, T., Poomsirivilai, P., and Vateekul, P. (2014). Label correction strategy on hierarchical multi-label classification. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 213–227. Springer.

Baraniuk, R. G. (1999). Optimal tree approximation with wavelets. In *Wavelet Applications in Signal and Image Processing VII*, volume 3813, pages 196–208. International Society for Optics and Photonics.

Baraniuk, R. G., Cevher, V., Duarte, M. F., and Hegde, C. (2010). Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001.

Baraniuk, R. G. and Jones, D. L. (1994). A signal-dependent time-frequency representation: Fast algorithm for optimal kernel design. *Signal Processing, IEEE Transactions on*, 42(1):134–146.

Barutcuoglu, Z., Schapire, R. E., and Troyanskaya, O. G. (2006). Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836.

Benjamini, Y. and Bogomolov, M. (2014). Selective inference on multiple families of hypotheses. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):297–318.

Bi, W. and Kwok, J. T. (2011). Multi-label classification on tree-and dag-structured hierarchies. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 17–24.

Bi, W. and Kwok, J. T. (2015). Bayes-optimal hierarchical multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):2907–2918.

Blockeel, H., Bruynooghe, M., Dzeroski, S., Ramon, J., and Struyf, J. (2002). Hierarchical multi-classification. In *Proceedings of the ACM SIGKDD 2002 workshop on multi-relational data mining (MRDM 2002)*, pages 21–35.

Blockeel, H., Schietgat, L., Struyf, J., Džeroski, S., and Clare, A. (2006). Decision trees for hierarchical multilabel classification: A case study in functional genomics. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 18–29. Springer.

Cai, T. T. and Sun, W. (2009). Simultaneous testing of grouped hypotheses: Finding needles in multiple haystacks. *Journal of the American Statistical Association*, 104(488):1467–1481.

Clare, A. (2003). *Machine learning and data mining for yeast functional genomics*. PhD thesis, The University of Wales.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM.

DeCoro, C., Barutcuoglu, Z., and Fiebrink, R. (2007). Bayesian aggregation for hierarchical genre classification. pages 77–80.

Dimitrovski, I., Kocev, D., Loskovska, S., and Džeroski, S. (2011). Hierarchical annotation of medical images. *Pattern Recognition*, 44(10):2436–2449.

Efron, B. (2005). *Local false discovery rates*. Division of Biostatistics, Stanford University.

Efron, B. (2007). Size, power and false discovery rates. *The Annals of Statistics*, 35(4):1351–1377.

Efron, B. (2012). *Large-scale inference: empirical Bayes methods for estimation, testing, and prediction*, volume 1. Cambridge University Press.

Gauch, S., Chandramouli, A., and Ranganathan, S. (2009). Training a hierarchical classifier using inter document relationships. *Journal of the Association for Information Science and Technology*, 60(1):47–58.

Hand, D. J. (2009). Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine learning*, 77(1):103–123.

Herskovic, J. R., Iyengar, M. S., and Bernstam, E. V. (2007). Using hit curves to compare search algorithm performance. *Journal of biomedical informatics*, 40(2):93–99.

Holden, N. and Freitas, A. A. (2005). A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 100–107. IEEE.

Huang, H., Liu, C.-C., and Zhou, X. J. (2010). Bayesian approach to transforming public gene expression repositories into disease diagnosis databases. *Proceedings of the National Academy of Sciences*, 107(15):6823–6828.

Jiang, C.-R., Liu, C.-C., Zhou, X. J., and Huang, H. (2014). Optimal ranking in multi-label classification using local precision rates. *Statistica Sinica*, 24(4):1547–1570.

Kiritchenko, S., Matwin, S., and Famili, F. (2005). Functional annotation of genes using hierarchical text categorization. In *BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05), Detroit, USA, 2005*.

Kiritchenko, S., Matwin, S., Nock, R., and Famili, A. F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 395–406. Springer.

Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 170–178. Morgan Kaufmann Publishers Inc.

Lee, B. K., Lessler, J., and Stuart, E. A. (2011). Weight trimming and propensity score weighting. *PloS one*, 6(3).

Lee, W. T. (2013). *Bayesian Analysis in Problems with High Dimensional Data and Complex Dependence Structure*. PhD thesis, University of California, Berkeley.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

Mayne, A. and Perry, R. (2009). Hierarchically classifying documents with multiple labels. In *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, pages 133–139. IEEE.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

Pillai, I., Fumera, G., and Roli, F. (2013). Threshold optimisation for multi-label classifiers. *Pattern Recognition*, 46(7):2055–2065.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. (2006). Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7(Jul):1601–1626.

Silla, C. N. and Freitas, A. A. (2009). Novel top-down approaches for hierarchical classification and their application to automatic music genre classification. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 3499–3504. IEEE.

Sun, A. and Lim, E.-P. (2001). Hierarchical text classification and evaluation. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 521–528. IEEE.

Triguero, I. and Vens, C. (2016). Labelling strategies for hierarchical multi-label classification techniques. *Pattern Recognition*, 56:170–183.

Valentini, G. (2009). True path rule hierarchical ensembles. In *International Workshop on Multiple Classifier Systems*, pages 232–241. Springer.

Valentini, G. (2011). True path rule hierarchical ensembles for genome-wide gene function prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(3):832–847.

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

Weston, J., Chapelle, O., Vapnik, V., Elisseeff, A., and Schölkopf, B. (2003). Kernel dependency estimation. In *Advances in neural information processing systems*, pages 897–904.

Wu, F., Zhang, J., and Honavar, V. (2005). Learning classifiers using hierarchically structured class taxonomies. In *Abstraction, Reformulation and Approximation*, pages 313–320. Springer.

Yekutieli, D. (2008). Hierarchical false discovery rate–controlling methodology. *Journal of the American Statistical Association*, 103(481):309–316.

Yekutieli, D., Reiner-Benaim, A., Benjamini, Y., Elmer, G. I., Kafkafi, N., Letwin, N. E., and Lee, N. H. (2006). Approaches to multiplicity issues in complex research in microarray analysis. *Statistica Neerlandica*, 60(4):414–437.

Zhang, M.-L. and Zhou, Z.-H. (2013). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837.