Self-Paced Unified Representation Learning for Hierarchical Multi-Label Classification

Zixuan Yuan[1], Hao Liu[1], Haoyi Zhou[2], Denghui Zhang[3], Xiao Zhang[4], Hao Wang[5], Hui Xiong[1]
[1]Hong Kong University of Science and Technology (Guangzhou)
[2]Beihang University,
[3]Stevens Institute of Technology,
[4]Shandong Univerisity,
[5]Alibaba Group
{zixuanyuan, xionghui}@hkust-gz.edu.cn, liu@ust.hk, haoyi@buaa.edu.cn, dzhang42@stevens.edu,
xiaozhang@sdu.edu.cn, cashenry@126.com

Abstract

Hierarchical Multi-Label Classification (HMLC) is a well-established problem that aims at assigning data instances to multiple classes stored in a hierarchical structure. Despite its importance, existing approaches often face two key limitations: (i) They employ dense networks to solely explore the class hierarchy as hard criterion for maintaining taxonomic consistency among predicted classes, yet without leveraging rich semantic relationships between instances and classes; (ii) They struggle to generalize in settings with deep class levels, since the mini-batches uniformly sampled from different levels ignore the varying complexities of data and result in a non-smooth model adaptation to sparse data. To mitigate these issues, we present a *Self-Paced Unified Representation* (SPUR) learning framework, which focuses on the interplay between instance and classes to flexibly organize the training process of HMLC algorithms. Our framework consists of two lightweight encoders designed to capture the semantics of input features and the topological information of the class hierarchy. These encoders generate unified embeddings of instances and class hierarchy, which enable SPUR to exploit semantic dependencies between them and produce predictions in line with taxonomic constraints. Furthermore, we introduce a dynamic hardness measurement strategy that considers both class hierarchy and instance features to estimate the learning difficulty of each instance. This strategy is achieved by incorporating the propagation loss obtained at each hierarchical level, allowing for a more comprehensive assessment of learning complexity. Extensive experiments on several empirical benchmarks demonstrate the effectiveness and efficiency of SPUR compared to state-of-the-art methods, especially in scenarios with missing features.

Introduction

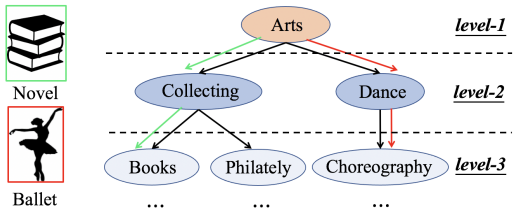
Hierarchical Multi-Label Classification (HMLC) problems have been extensively studied in a wide range of domains, e.g., functional genomics (Wehrmann, Cerri, and Barros 2018), image annotation (Dimitrovski et al. 2011), and text classification (Lewis et al. 2004). As illustrated in Figure 1a, the general focus of HMLC task is to assign objects with correct labels from an enormous label space, where classes are hierarchically constructed as a taxonomy graph or a tree (Giunchiglia and Lukasiewicz 2020; Yu et al. 2022).

Recent decades have witnessed numerous attempts (Cesa-Bianchi et al. 2004; Cerri et al. 2016; Wehrmann, Cerri, and Barros 2018; Giunchiglia and Lukasiewicz 2020) to develop plausible HMLC solutions that are trained to respect the hierarchy or taxonomic constraint, i.e., any instance associated with a given set of subclasses must belong to their related superclasses (Wehrmann, Cerri, and Barros 2018). To achieve this, most approaches require a repeated graph-level exploration over class hierarchy at both training and inference times, thus making models hard to scale to large label spaces (Patel et al. 2021). Furthermore, in many real-world scenarios, the number of instances per class is highly sparse at the bottom levels (Giunchiglia and Lukasiewicz 2020), which inevitably causes the learning obstacles for these approaches to deliver satisfactory inference.

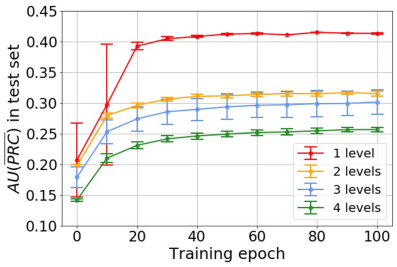
In this paper, we innovate new directions to tackle the aforementioned problems, in terms of the encoding of class-class and class-instance relations, and the prioritizing of instances' learning orders. We detail our research insights from the following two perspectives.

First, most studies are solely dependent on instance features to perform HMLC task, while treating the class-class connectivity as hard criterion to ensure the constraint satisfaction. On the contrary, we find it worthwhile to exploit rich semantic correlations between class nodes and data instance. For example, in Figure 1a, the instance "Ballet" should be classified by the class "Dance" other than "Collecting" at the second level because of their strong semantic affiliation. Such affiliation can also be revealed from their closeness in the latent space. Undoubtedly, associating "Dance" with "Ballet" can boost the success of next-level prediction by considering both class-class connectivity (e.g., "Dance"-"Choreography") and class-instance relationship (e.g., "Ballet"-"Dance"). To implement the above idea, one promising solution is to incorporate graph representation learning to project the class-class relations into latent space, and then integrate with instance features for final prediction. However, the direct adoption of popular graph learning schemes (e.g., GCN (Kipf and Welling 2016)) is not well qualified to (i) learn a set of expressive class rep-

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.



(a) Illustrative examples of HMLC problem. Green and red arrow lines denote the groundtruth class paths that correspond to the inputs "novel" and "ballet", respectively.



(b) The HMLC results on the CELLCYCLE FUN dataset of C-HMCNN(h) (Giunchiglia and Lukasiewicz 2020) that considers different hierarchies.

Figure 1: Examples and challenges of HMLC problem.

resentations that can precisely capture the topological information of class hierarchy, and (ii) avoid the parameter explosion problem for large output spaces.

Second, the common assumption of HMLC methods is that each instance's learning priority remains the same across all class hierarchies. In other words, these methods neglect to explore the hierarchical inductive bias that varies greatly with the growing learning complexity at deeper levels. As a result, the state-of-the-art method (Giunchiglia and Lukasiewicz 2020) fails to achieve equivalent scores at the fourth level as it did at the first level, as depicted in Figure 1b. Inspired by the great success of Self-Paced Learning (SPL) in improving the generalizability of classifiers (Meng, Zhao, and Jiang 2017), we may enhance the state-of-the-art performance at deep hierarchies by organizing the instances' learning sequence from easy to hard. Therefore, we highly need a robust self-paced regularizer to perform effective difficulty measurement and training data sampling.

Along these lines, we propose a *Self-Paced Unified Representation* (SPUR) learning framework for general HMLC. The core of SPUR is to model the class-class connectivity and class-instance dependency into the embedding space, and prioritize the learning sequence of instances from the joint perspectives. Specifically, we first devise two separate encoder modules to project the graph-structured class hierarchy and input features into latent space. Here, to reduce the computational cost, we pre-train the class node embeddings via graph reconstruction loss, and only use a parameter-free pooling function to generate the graph-level representation of class hierarchy. Driven by the theoretical framework that analyzes the representational power of graph learning (Xu et al. 2019), we prove that the aggregated representations are distinguishable to discriminate the patterns of class topology associated with instance features using the Weisfeiler-Lehman graph isomorphism test (Leman and Weisfeiler 1968). To gradually increase learning complexity for model update, we design a self-paced regularizer to adaptively measure the instances' learning hardness based on the discrepancies between historical and current level-wise propagation losses, and adjust training mini-batches accordingly. Finally, we conduct extensive experimental studies on 20 real-world HMLC datasets. The results showcase that our approach can search hierarchical labels more accurately and efficiently than existing methods, especially in the missing feature scenario.

Related Work

Hierarchical Multi-Label Classification Prior studies dedicated to HMLC problems can be roughly divided into three categories: (i) The local methods (Radivojac et al. 2013; Cerri et al. 2016) follow the divide-and-conquer strategy to decompose the main HMLC task into level-wise or node-wise classification sub-tasks. (ii) The global methods usually consist of a single classifier capable of aligning objects with their corresponding classes in the hierarchy at once (Barros et al. 2013; Masera and Blanzieri 2018). (iii) The hybrid methods (Wehrmann, Cerri, and Barros 2018) develop the hybrid recurrent and non-recurrent neural networks to consider both local and global losses. One major limitation of the above methods is that they have to frequently process the entire class hierarchy for constraint checking in both training and inference periods. In contrast, our SPUR only requires a parameter-free graph-level pooling operation based on the pre-trained class representations, which improves the overall model scalability under high-dimensional environments.

While some recent studies (Mittal et al. 2021; Zhang et al. 2021; Yu et al. 2022) also explore the easy-to-hard learning scheme or encoding of label graphs for large output space, our work is different from these studies. For instance, (Mittal et al. 2021) applied random walks on label co-occurrence links to obtain a label correlation graph for label encoding, while we directly encode the class hierarchy into distinguishable class representations based on the WL test. (Zhang et al. 2021) used a tree-based class hierarchy to generate fixed coarse-to-fine learning schedule for transformer fine-tuning, while our method derives an adaptive self-paced regularizer to generate the time-evolving training priorities of instances conditioned on their learning difficulties.

Weisfeiler-Lehman Test. The Weisfeiler-Lehman (WL) test (Leman and Weisfeiler 1968) refers to a computational effective algorithm that checks whether two graphs are topologically identical, a.k.a., the graph isomorphism problem. General design of WL test is to aggregate the labels of nodes and their neighborhoods, and then hash the aggregated labels into unique new labels. This enables to decide that two graphs are non-isomorphic if the labels of the nodes between the two graphs differ at certain iterations. Motivated by this, (Shervashidze et al. 2011) proposed the WL subtree kernel to estimate the similarity between graphs using the counts of node labels at different iterations of the WL test as the feature vector of a graph, while (Xu et al. 2019) analyzed and characterized the expressive power of GNN variants in capturing different graph structures through the WL test.

Self-Paced Learning By mimicking through the human learning principle, curriculum learning optimizes the model's learning objectives from easiness to hardness. Instead of designing a heuristic hardness measurement based on manual experience (Meng, Zhao, and Jiang 2017), self-paced learning (SPL) introduces a regularization term to the objective functions, and formulates an ad-hoc concise protocol to dynamically measure training hardness in terms of instance gradients. Recently, MLSPL (Li et al. 2017) adopts SPL to regularize model learning for multi-label classification task. However, MLSPL considers a different problem setting from our work, since it assumes class hierarchy is unknown, while our work follows the general HMLC methods that exploit the given class hierarchy. Also, to measure learning hardness, MLSPL uses a bi-class SVM to classify the label-wise losses, while our SPUR utilizes a loss-based dynamic threshold to distinguish the level-wise losses.

Methodology

In this section, we first review the formal problem definition of HMLC. Then we introduce one basic neural approach for HMLC problem, discussing their strengths and weaknesses. To improve its performance, we later propose our *Self-Paced Unified Representation* (SPUR) learning framework, where (i) the structure-aware learner involves two separate encoder modules to explore the instance feature and class hierarchy for label prediction, and (ii) the self-paced learning paradigm smoothly updates the model from easy samples to hard ones.

Problem Definition

Consider a set of $|C|$ hierarchical classes $C = \{C_1, \ldots, C_{\mathrm{L}}\}$ organized as a Directed Acyclic Graph (DAG) $\mathcal{G}$, where C_l denotes the collection of classes at the l-th level of hierarchy. For an arbitrary datapoint $\mathbf{x} \in \mathbb{R}^d$, $d \geq 1$, and each class i, a standard HMLC objective is to find a neural function $\mathcal{F}_i : \mathbf{x} \to [0, 1]$, such that $\mathbf{x}$ is predicted to belong to class i if $\mathcal{F}_i(x)$ is larger than or equal to a pre-defined threshold. If there exists a path from a class i to a class j in $\mathcal{G}$, then j is considered as a subclass of i. We assume each class is a subclass of itself. Throughout the paper, all $\mathbf{W}_s$ and $\mathbf{b}_s$ are the learnable weights and biases.

Base Learner

Inspired by recent global approaches (Senge, Coz, and Hüllermeier 2014; Giunchiglia and Lukasiewicz 2020), we develop a neural learner to be trained for mapping relevant classes to each output. Specifically, the information flow inside the base learner traverses from J Fully-Connected (FC) layers to reach the ultimate prediction $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = \mathrm{Sigmoid}(\mathbf{W}_J \mathbf{z}^{\mathrm{F}} + \mathbf{b}_J), \tag{1}$$

where $\mathbf{z}^{\mathrm{F}} = \phi_{J-1}(\phi_{J-2}(\cdots \phi_1(\mathbf{x})))$, which stands for the refined instance embeddings. $\phi_j(\star) = \mathrm{ReLU}(\mathbf{W}_j \star + \mathbf{b}_j)$, $j \in \{1, \ldots, J-1\}$. $\mathbf{W}_1 \in \mathbb{R}^{d_h \times d}$, $\mathbf{W}_J \in \mathbb{R}^{|C| \times d_h}$. $\mathbf{W}_j \in \mathbb{R}^{d_h \times d_h}$, in which $j \in \{2, \ldots, J-1\}$, and d_h is the dimension of latent vector. Unlike conventional tree-based classifiers (Bi and Kwok 2011; Dimitrovski et al. 2012; Schietgat et al. 2010) that naturally respect class hierarchy, most neural networks require additional post-processing steps to ensure such hierarchical constraints (Cerri, Barros, and De Carvalho 2014; Valentini 2010). Likewise, we introduce a concise training trick, named max constraint loss (MCLoss) (Giunchiglia and Lukasiewicz 2020), which adds extra penalties when the hierarchical satisfaction is broken. Since the classes of different hierarchies are predicted as a whole, it would save huge memories for other neural learners (like HMLCN-F (Wehrmann, Cerri, and Barros 2018)) that need to specify extra FC layers to generate local outputs at each hierarchical level.

While conceptually useful, such base learner reflects two potential weaknesses without being addressed. First, total reliance on instance features and hard loss penalty is a risky choice in many HMLC applications, since they suffer from severe data-sparse or feature-absent issues at the deeper levels of class hierarchy. A promising alternative is to capture rich semantic dependencies among input features and flattened class labels, with the purpose of introducing extra useful information for better classification results. Second, as illustrated in Figure 1b, the learning difficulty varies greatly from instance to instance, and usually increases when the instances are labeled as deeper hierarchical classes. In order to better avert unreasonable local minima, it is more desirable to embed gradual learning from simple to hard samples into model optimization.

Self-Paced Unified Representation Learning

Now we describe how our approach SPUR addresses the above limitations of base learner. Figure 2 presents the overall architecture of our SPUR. We first propose a *Structure-Aware Learner* to exploit both class-to-class connectivity and class-instance semantic dependency to perform HMLC. Based on the level-wise classification losses, we then develop a *Self-Paced Learning* paradigm to gradually train the structure-aware learner from easiness to hardness.

Structure-Aware Learner Compared to the sole-input design of base learner, our SPUR considers a joint perspective of class-class and class-instance relations for HMLC task. To achieve this, our structure-aware learner first employs two separate encoder modules, named $\mathrm{Enc\_F}(\cdot)$ and

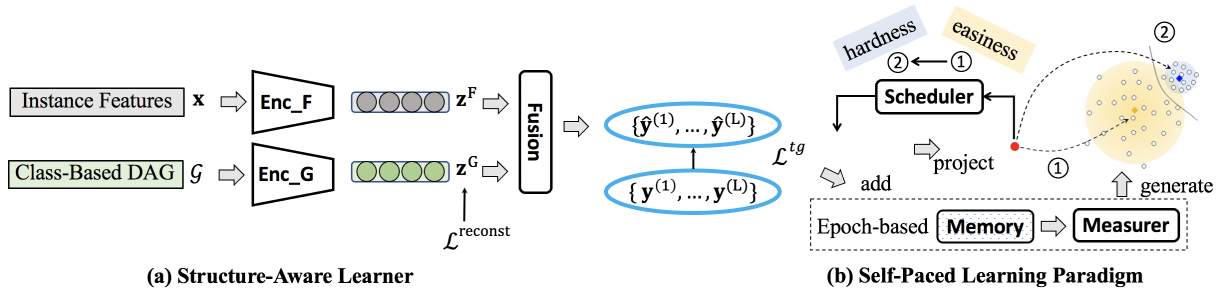


Figure 2: The framework overview of SPUR.

$\mathrm{Enc\_G}(\cdot)$, to map the instances and class into the embedding space. In particular, for $\mathrm{Enc\_F}(\cdot)$, we apply the same architecture as base learner to update the instance embeddings $\mathbf{z}^{\mathrm{F}}$. For $\mathrm{Enc\_G}(\cdot)$, inspired by GIN (Xu et al. 2019), we first develop a node-wise encoding function to convolve over the class hierarchy and output the representations of class nodes in $\mathcal{G}$:

$$\mathbf{v}^{(m)} = \mathrm{MLP}^{(m)}(\mathbf{W}_m \mathbf{v}^{(m-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{u}^{(m-1)}), \quad (2)$$

in which at the m-th iteration, $\mathbf{v}^{(m)} \in \mathbb{R}^{|C| \times d_h}$ and $\mathrm{MLP}^{(\mathrm{m})}(\cdot)$ denotes the embeddings of class node v and multi-layer perceptrons, respectively. $\mathbf{W}_m \in \mathbb{R}^{d_h \times d_h}$. $\mathcal{N}(v)$ is the 1-hop neighbors of node v. Based on the learned class embeddings, we design another parameter-free pooling function to finalize the graph-wise representation $\mathbf{z}^{\mathrm{G}} \in \mathbb{R}^{d_h}$:

$$\mathbf{z}^{\mathrm{G}} = \mathrm{ReLU}(\mathrm{Merge}(\{\mathbf{v}^{(m)} | \ v \in \mathcal{G}, m \in \mathcal{M}\})), \quad (3)$$

where $\mathrm{Merge}(\cdot)$ stands for the element-wise average or summation pooling (Kipf and Welling 2016), operated over the neighborhood $\mathcal{N}(v)$ of any given class node v. $\epsilon^{(m)}$ is a learnable parameter. $\mathcal{M} = \{0, 1, \ldots, M\}$. Afterwards, we utilize the Eq. (1) to output the predicted classes through the aggregation of two updated features $\mathbf{z}^{\mathrm{F}}$ and $\mathbf{z}^{\mathrm{G}}$:

$$\hat{\mathbf{y}} = \mathrm{Sigmoid}(\mathbf{W}_{H_2}(\mathbf{W}_{H_1}[\mathbf{z}^{\mathrm{F}} || \, \mathbf{z}^{\mathrm{G}}] + \mathbf{b}_{H_1}) + \mathbf{b}_{H_2}), \quad (4)$$

where $[\star||\star]$ denotes the concatenation operation along the latent dimension. $\mathbf{W}_{H_1} \in \mathbb{R}^{d_h \times 2d_h}$ and $\mathbf{W}_{H_2} \in \mathbb{R}^{|C| \times d_h}$.

Loss function: Since $\mathbf{z}^{\mathrm{G}}$ encompasses entire hierarchical information, we can easily skip the post-processing step (e.g., MCLoss) to examine the taxonomic constraints, and directly minimizes the classification error of the l-th level via:

$$\mathcal{L}^{tg} = \mathrm{CE\_Loss}(\mathbf{y}^{(\mathrm{L})}, \hat{\mathbf{y}}^{(L)}), \quad (5)$$

in which $\mathbf{y}^{(l)}$, $\hat{\mathbf{y}}^{(l)}$ denote the groundtruth and predicted labels at all layers before the l-th layer (self-included), $l \in \{1, \ldots, \mathrm{L}\}$, respectively. $\mathrm{CE\_Loss}(\mathbf{a}, \mathbf{b}) = -\mathbf{a} \ln \mathbf{b} - (1 - \mathbf{a}) \ln (1 - \mathbf{b})$ represents the cross entropy loss (Zhang and Sabuncu 2018). Notice that, the above graph-based encoder may cause the parameter size to explode with increasing output space, and thus suffer from high computational cost. To mitigate this, we pre-train the representations of class nodes $\mathbf{z}^{\mathrm{G}}$ via the graph reconstruction loss $\mathcal{L}^{reconst} = -\frac{1}{|C|^2} \sum_{u \in C} \sum_{v \in C} (\mathbf{v}^{(M)} \mathbf{u}^{(M)^\top} - \mathbf{A}_{uv})$, which allows the learned node embeddings $\mathbf{v}^{(M)}$, $\mathbf{u}^{(M)}$ to capture the class-wise patterns stored in original DAG's adjacency matrix $\mathbf{A}$. With the pre-trained class representations, we directly regard the parameter-free Eq. (3) as $\mathrm{Enc\_G}(\cdot)$, and enable the structure-aware learner to avoid redundant computations involved in Eq. (1).

In general, producing disparate labels for distinctive inputs is an essential property of good HMLC algorithms (Sorower 2010). Since these predicted labels correspond to multiple paths in $\mathcal{G}$, and can be transformed into new DAGs of different structures, we follow the graph isomorphism theory (McKay and Piperno 2014) to analyze how the topology of predicted DAG is jointly influenced by the input feature and original DAG.

Theorem 1. *Given an original DAG and two training instances* $\mathbf{x}$, $\mathbf{x}'$. *If* $Enc\_F(\mathbf{x})$ *is distant from* $Enc\_F(\mathbf{x}')$ *in the latent space, their predicted DAGs are non-isomorphic, which can be validated by Weisfeiler-Lehman graph isomorphism test.*

The detailed proof is given in the Appendix A. This theorem implies that the unified encoding of class hierarchy and instance feature endows our SPUR with the above important property, mainly because the learned graph-level representations, associated with instance features, are distinguishable enough to discriminate the topological information of predicted DAG.

Self-Paced Optimization Another critical problem of the base learner is regarding its tendency to arrive at an unsatisfactory local minimum with high training and generalization error for a non-convex HMLC objective. As illustrated in Figure 1b, since the model's learning barriers become more severe with the growing hierarchical levels, we introduce the concept of Self-Paced Learning (SPL) (Kumar, Packer, and Koller 2010; Meng, Zhao, and Jiang 2017) that iteratively selects instances from easy to hard and the number of instances at each iteration depends on the gradually annealed weights. To better distinguish easy instances from hard ones, we first create the level-wise loss vector $\mathcal{L}^{gd} = \{\mathcal{L}^{(l)}\}_{l \in \{1, \ldots, \mathrm{L}\}}$, where $\mathcal{L}^{(l)} = \mathrm{CE\_Loss}(\mathbf{y}^{(l)}, \hat{\mathbf{y}}^{(l)})$. Basically, this loss vector can be used to estimate how well the model behaves for local prediction, because of the following good attribute:

Proposition 1. *Given one training instance* $\mathbf{x}$*, any two distinct subgraphs* $\mathcal{G}_i, \mathcal{G}_j$ *of hierarchies* $i \geq j$ *in the predicted class graph* $\mathcal{G}$*, their corresponding losses are distinguishable from each other.*

Proof: The above result can be easily derived since Theorem 1 proves the distinguishability of the learned representations of predicted graph, it should also apply to local hierarchies, thus making the corresponding losses identifiable.

Therefore, such level-wise losses can reveal the uniqueness of graph-based prediction and offer an isomorphism-based estimation to measure the localized learning hardness of structured classes. To globally assess the instance's learning difficulty and adjust training mini-batches, we later develop a self-paced regularizer based on the past and present level-wise losses. In particular, we set the number of learning groups to be two, which correspond to the majority (easy) and minor (hard) categories, respectively. Since the model is being updated continuously, the hardness measurement threshold should be altered periodically (e.g., every K epochs) as well. To be specific, we store all loss vectors w.r.t. previous training records as the epoch-based memory. After every K epochs, we perform the following steps to update the level-wise threshold of learning hardness:

- For each class c within a specific level l, we first compute the current class-wise learning threshold $\hat{\mathcal{L}}_c^{(l)}$ by averaging the losses of its associated training instances.
- These class-wise learning thresholds are then aggregated to form the level-wise learning threshold $\hat{\mathcal{L}}^{(l)} = \sum_{c \in C^{(l)}} \hat{\mathcal{L}}_c^{(l)}$, where $C^{(l)}$ represents a set of classes located at the l-th level.

When a new instance x arrives, we compare its level-wise losses $\mathcal{L}^{(l)}$ against the corresponding learning thresholds $\hat{\mathcal{L}}^{(l)}$, and retrieve the possibility $\mathbf{w}_x$ of being assigned to majority category, which helps control the degree of involvement of this instance in the subsequent model optimization:

$$\mathbf{w}_x = \prod_{l=1}^{L} [1 - \text{Sigmoid}(\mathcal{L}^{(l)} - \hat{\mathcal{L}}^{(l)})]. \tag{6}$$

The intuitive explanation behind Eq. (6) is that when the current loss $\mathcal{L}^{(l)}$ is much larger than the average previous losses $\hat{\mathcal{L}}^{(l)}$ at most levels, the instance x would have a lower probability of being assigned as "easy" sample. Later, we follow standard SPL (Kumar, Packer, and Koller 2010) to embed gradual learning from easy to hard samples as:

$$\min_{\mathbf{W} \in [0,1]^{|C|}} \mathbb{E}(\mathcal{Y}, \mathcal{X} | \mathbf{W}, \lambda) = \sum_{x \in \mathcal{X}} \mathbf{w}_x \mathcal{L}_x^{tg} - \lambda \sum_{x \in \mathcal{X}} \mathbf{w}_x, \tag{7}$$

where $\mathcal{X}$, $\mathcal{Y}$ stand for the sets of input features and groundtruths in the training set, respectively. $\mathbf{w}_x \in \mathbf{W}$, and λ is a hyper-parameter for controlling the learning pace. N is the number of training instances. It is obvious that by increasing λ throughout training, the self-paced learning algorithm allows more difficult samples into the training process. Note that, we randomly sample multiple "easy" instances to participate in each mini-batch optimization. Due to page limit, we put the detailed training procedure in Algorithm 1 of Appendix B.

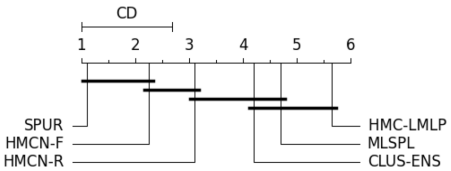


Figure 3: Critical diagram for the Nemenyi's statistical test.

Experiments

Experiment Setup

We evaluate the generic efficacy of our SPUR on 20 publicly-available datasets, which include protein function prediction (Ruepp et al. 2004; Ashburner et al. 2000), annotation of medical (Dimitrovski et al. 2011) or microalgae images (Dimitrovski et al. 2012), and text categorization (Klimt and Yang 2004). These datasets are commonly used for comparison among HMLC approaches (Giunchiglia and Lukasiewicz 2020; Bi and Kwok 2011; Nakano, Lietaert, and Vens 2019; Wehrmann, Cerri, and Barros 2018). For data pre-processing, we follow (Giunchiglia and Lukasiewicz 2020; Wehrmann, Cerri, and Barros 2018) to transform its categorical features into one-hot vectors, and replace missing values with the mean of normalized numeric features, or with a vector of all zeros in the categorical cases. Our SPUR[1] and all baselines are implemented with Pytorch framework (Paszke et al. 2019) and run on a single 3090 Ti GPU. Due to the page limit, we also include the parameter sensitivity and the efficiency test in Appendix C.

Performance Metrics. For HMLC evaluation, we use the mostly used metric, i.e., $AU(\overline{PRC})$ (Bi and Kwok 2011; Wehrmann, Cerri, and Barros 2018; Giunchiglia and Lukasiewicz 2020), to predict when a datapoint belongs to a particular class. Formally, the $AU(\overline{PRC})$ measures the area under the average precision recall curve, of which points $(\overline{P}, \overline{R})$ are calculated as follows:

$$\overline{P} = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|} TP_i + \sum_{i=1}^{|\mathcal{C}|} FP_i} \qquad \overline{R} = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|} TP_i + \sum_{i=1}^{|\mathcal{C}|} FN_i},$$

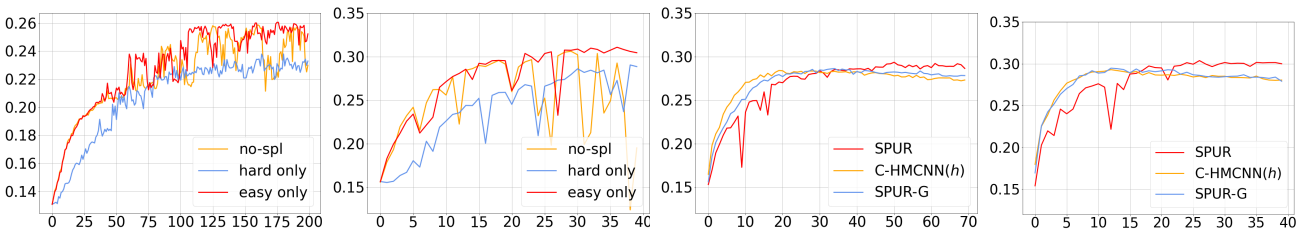
in which TP_i, FP_i, and FN_i stand for the quantities of true positives, false positives, and false negatives for class i, respectively. $|\mathcal{C}|$ is the number of classes.

Baselines. We compare our SPUR with six state-of-the-art HMLC algorithms, which include MLSPL (Li et al. 2017), C-HMCNN(h) (Giunchiglia and Lukasiewicz 2020), HMLC-LMLP (Cerri et al. 2016), CLUS-ENS (Schietgat et al. 2010), HMLCN-R (Wehrmann, Cerri, and Barros 2018), and HMLCN-F (Wehrmann, Cerri, and Barros 2018). We run SPUR, C-HMCNN(h) (Giunchiglia and Lukasiewicz 2020), Clus-Ens (Schietgat et al. 2010), and HMC-LMLP (Cerri et al. 2016) 10 times on each dataset

[1]Code released at https://github.com/yuanzx33033/SPUR/

DATASET	SPUR	MLSPL	C-HMCNN(h)	HMCN-R	HMCN-F	HMC-LMLP	CLUS-ENS
CELLCYCLE FUN	**0.261**	0.227	0.255	0.247	0.252	0.207	0.227
DERISI FUN	**0.198**	0.192	0.195	0.189	0.193	0.182	0.187
EISEN FUN	**0.311**	0.301	0.306	0.298	0.298	0.245	0.286
EXPR FUN	**0.313**	0.300	0.302	0.300	0.301	0.242	0.271
GASCH1 FUN	**0.292**	0.285	0.286	0.283	0.284	0.235	0.267
GASCH2 FUN	**0.262**	0.247	0.258	0.249	0.254	0.211	0.231
SEQ FUN	**0.302**	0.288	0.292	0.290	0.291	0.236	0.284
SPO FUN	**0.221**	0.197	0.215	0.210	0.211	0.186	0.211
CELLCYCLE GO	**0.420**	0.387	0.413	0.395	0.400	0.361	0.387
DERISI GO	**0.374**	0.365	0.370	0.368	0.369	0.343	0.361
EISEN GO	**0.461**	0.427	0.455	0.435	0.440	0.406	0.433
EXPR GO	**0.456**	0.431	0.442	0.450	0.452	0.373	0.422
GASCH1 GO	**0.444**	0.408	0.436	0.416	0.428	0.380	0.415
GASCH2 GO	0.422	0.396	0.414	0.463	**0.465**	0.371	0.395
SEQ GO	**0.450**	0.425	0.446	0.443	0.447	0.370	0.438
SPO GO	**0.387**	0.357	0.382	0.375	0.376	0.342	0.371
DIATOMS	**0.776**	0.718	0.758	0.514	0.530	-	0.501
ENRON	**0.770**	0.732	0.756	0.710	0.724	-	0.696
IMCLEF07A	**0.960**	0.872	0.956	0.904	0.950	-	0.803
IMCLEF07D	**0.933**	0.901	0.927	0.897	0.920	-	0.881
AVERAGE RANKING	1.10	4.70	2.25	4.20	3.10	7.00	5.65

Table 1: Performance comparison of our SPUR with all baselines. The best results are highlighted in bold.



(a) CELLCYCLE FUN. (b) EXPR FUN. (c) GASCH1 FUN. (d) SEQ FUN.

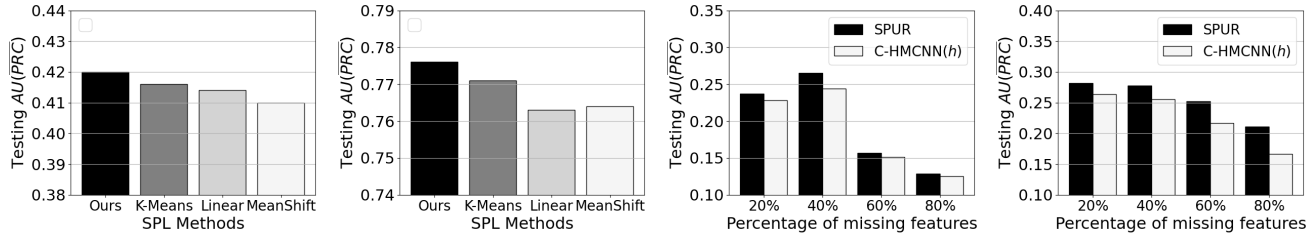
Figure 4: Learning curves of three training strategies (a-b) and three model variants (c-d) on four FunCat tasks. For each figure, the x-axis denotes the training epochs, and the y-axis denotes the $AU(\overline{PRC})$ metric.

to report the average $AU(\overline{PRC})$of those runs. For simplicity, we omit the standard deviations which for SPUR vary in a small range $[0.9 \times 10^{-3}, 7.2 \times 10^{-3}]$, indicating its high degree of stability.

Overall Performance

Table 1 reports the performance of our SPUR and all baselines on 20 different datasets of HMLC tasks. We first analyze the performance of SPUR when compared with the current state-of-the-art methods. As can be seen, our SPUR outperforms the state-of-the-art by achieving superior performances on all datasets except Gasch2 GO, and obtaining the best average ranking (1.10) with the largest number of wins. As the runner-up model, C-HMCNN(h) is the prototype of base learner that applies a feedforward neural network with hierarchical loss constraints. There are two important aspects overlooked by C-HMCNN(h), i.e., the modeling of hierarchy information and prioritizing the training order. Careful design of these two aspects in SPUR has contributed significant gains relative to the C-HMCNN(h) method. Similar to HMLCN-F (Wehrmann, Cerri, and Barros 2018) that scales up with larger class hierarchy, the introduction of graph-based encoder inevitably raises the trade-off between performance and computational cost. Furthermore, our SPUR performs much better than its closest baseline MLSPL which purely relies on the input features to determine the learning hardness. This indicates the necessity of taxonomic modeling to regularize the SPL process.

We next perform the Friedman test (Friedman 1937) to examine the statistical significance of the above results, where we obtain the p-value 5.71×10^{-17} showing the existence of significant differences. We continue with the post-hoc Nemenyi test (Nemenyi 1963), and demonstrate its corresponding critical diagram in Figure 3. The diagram plots the average rank of each approach on a horizontal axis, and the critical difference CD $=$ 1.686. Notice that, the



(a) CELLCYCLE GO. (b) DIATOMS. (c) CELLCYCLE FUN. (d) SEQ FUN.

Figure 5: Performance comparisons among various difficulty estimators (a-b) and different levels of missing features (c-d).

group of methods that are connected through a horizontal line do not differ significantly at the significance level of 0.05. The results of the Nemenyi test allow us to claim that our SPUR outweighs all the other baselines with statistical significance except C-HMCNN(h). Hence, we follow (Giunchiglia and Lukasiewicz 2020) to conduct additional comparison between SPUR and C-HMCNN(h) based on the Wilcoxon test (Gehan 1965), which not only considers the performance rankings, but also quantifies the differences in performances of two approaches. Based on the Wilcoxon test, we reach a final conclusion that there exist a statistically significant difference between the performance of SPUR and C-HMCNN(h) with p-value of 1.91×10^{-6}.

Ablation Study

We further analyze the impact of structure-aware SPL and DAG encoding in terms of model convergences, difficulty estimator design, handling missing value, and time complexity (Appendix C).

Convergences of different training modes. Figures 4 (a-b) plot the learning curves of three training strategies employed on the FunCat dataset: (i) "ours" mode, i.e., the proposed SPUR trained with SPL, (ii) "no-spl" mode, which trains the structure-aware learner without SPL, and (iii) "hard" mode, which only allows hard instances to be trained. As can be seen, for each task, the performance curves of "mixed" and "hard" modes basically form the lower and upper bounds on the expected result of "no-spl" strategy, indicating that a good prioritization in the learning order would push forward the limit of HMLC performance.

Necessity of DAG encoding. To demonstrate the usefulness of learned DAG representations, we report the learning curves of three approaches (i.e., SPUR, C-HMCNN(h), and SPUR-G) in Figures 4 (c-d). Note that, SPUR-G is a variant of SPUR, which only adopts $\mathrm{Enc\_F}(\cdot)$ with SPL. We observe that SPUR-G exhibits slightly better performances than C-HMCNN(h), yet clearly outperformed by SPUR. This is not surprising because SPL relies on the joint modeling of class-class connectivity and class-instance dependencies, which highlights the irreplaceable role of DAG encoding for generating reliable estimation of learning hardness.

Various designs on difficulty estimators. We then evaluate the impacts of various difficulty estimators could impose on the HMLC performances. Here, we consider four types of estimators, including a loss-based dynamic threshold (i.e., ours), a non clustering-based method (i.e., linear) and two clustering-based ones (i.e., K-Means (Hartigan and Wong 1979), MeanShift (Cheng 1995)). For the linear estimator, we periodically update the threshold by multiplying a decaying coefficient γ by the average value of previous losses after every epoch. For the rest of two clustering-based approaches, we cluster the training instances into two groups based on their level-wise losses, where we treat the larger and smaller groups as easy and hard samples, respectively. We report their corresponding performances on two HMLC tasks in Figures 5 (a-b). It is noticeable that the clustering-based estimators are generally better than the linear method, possibly because the variations of level-wise losses are hard to be approximated with linear measurement. Also, our loss-based dynamic threshold achieves significantly better values compared to K-Means and MeanShift on two datasets. This is because the clustering-based methods can hardly scale with high-dimensional data.

Handling missing values. We conduct the analysis to discuss the necessity of exploiting the class-instance semantic dependencies as supplementary knowledge for HMLC task. Taking it into account can not only improve the overall performance, but also help alleviate the influence of missing feature problem on the final prediction. For effective evaluation, we compare the performances of our model with C-HMCNN(h) on the data of two datasets with missing features. In particular, for each dataset, we first routinely train the model using all instances, and report the prediction results regarding the samples of different missing levels in Figures 5 (c-d). It can be seen that the results of C-HMCNN(h) are inferior to our SPUR on two datasets in terms of all levels. This again validates the effectiveness of modeling the class hierarchy as the second identification of datapoint.

Conclusion

We proposed a self-paced unified representation (SPUR) learning framework, which first employs two neural encoders to study the semantic dependencies between input features and class hierarchy, and then incorporates a self-paced learning paradigm to smoothly update the model parameter from easier data to more difficult ones. Experimental results on several HMLC benchmarks demonstrate the superiority and efficiency of our approach.

Algorithm 1: Training procedure for SPUR framework.

Input: The training set $\{\mathcal{X}, \mathcal{Y}\}$, structure-aware learner $F(\cdot;\theta)$, learning rate η, update frequency K of difficulty measurement
Output: the well-trained parameters θ of structure-aware learner
while not converge **do**
  epoch $= 1$
  for $(x, y) \in \{\mathcal{X}, \mathcal{Y}\}$ **do**
    initialize $\mathcal{L}^{gd} = \emptyset$
    for $l = 1, \ldots, L$ **do**
      Obtain the training loss $\mathcal{L}^{(l)}$ based on the Eq. (5)
      $\mathcal{L}^{gd}$.add($\mathcal{L}^{(l)}$)
    end for
    if epoch! $= 1$ **then**
      Compute the likelihood $\mathbf{w}_x$ of $\mathcal{L}^{gd}$ being assigned to the majority category
      $\theta \leftarrow \theta - \eta(\sum_{x\in\mathcal{X}} \mathbf{w}_x \mathcal{L}_x^{tg} - \lambda \sum_{x\in\mathcal{X}} \mathbf{w}_x)$ based on Eq. (7)
    end if
  end for
  if epoch $==$ 1 or epoch%K $==$ 0 **then**
    Update $\hat{\mathcal{L}}^{(l)}, l \in \{1, \ldots, L\}$
  end if
  epoch $=$ epoch $+ 1$
end while

A. Proof for Theorem 1

According to (Xu et al. 2019), if two graphs are projected to different real-value vectors through a graph neural network, the WL test would confirm their non-isomorphism. Our setting is the special case where we always feed the identical DAG to the graph-based encoder. That is, for each class node v or u on the same graph $\mathcal{G}$, if WL node labels $l_v^{(m)} = l_u^{(m)}$, GNN node features $\mathbf{v}^{(m)} = \mathbf{u}^{(m)}$ will hold for all iteration M. The aggregation process $\mathrm{Aggregate}(\cdot,\cdot)$ between distinct input features and class node representation is the only factor that could discriminate the predicted DAGs. This actually creates a input-conditioned mapping Φ such that $\mathbf{v}^{(M)} = \Phi(l_u^{(M)}; \mathbf{x})$ for any $v \in \mathcal{G}$:

$$\{(\mathbf{v}^{(M)}, \{\mathbf{u}^{(M)} : u \in \mathcal{N}(v)\})\} = \{(\Phi(l_v^{(M)}), \{\Phi(l_u^{(M)}; \mathbf{x}) : u \in \mathcal{N}(v)\})\} \quad (8)$$

Note that such aggregation function is permutation invariant w.r.t. the set of graph nodes. Hence we conclude that if the input features are distinguishable in the latent space, the WL test would have obtained different collections of node labels at the final iteration.

B. Algorithm of Model Optimization

The whole training routine is detailed in Algorithm 1. In the lines 5-10, we generate the level-wise loss for each training instance. To smoothly regularize the training procedure, lines 11-14 prioritize the weights of those easier instances to be trained by introducing the likelihood of instance being assigned to the majority (easy) category. Lines 17-19 update the loss-based threshold of learning hardness based on all stored loss vectors at every K epochs. An interesting finding is that, the size of majority category becomes larger as the training continues, which demonstrates the success of SPL teaching our SPUR to gradually learn useful knowledge by studying simpler instances.

Dataset	Model	Total training time (min)
CELLCYCLE FUN	SPUR	**7.2**
	MLSPL	7.3
	C-HMCNN(h)	7.4
DERISI FUN	SPUR	**3.2**
	MLSPL	4.3
	C-HMCNN(h)	4.5
CELLCYCLE GO	SPUR	8.2
	MLSPL	**4.7**
	C-HMCNN(h)	4.8
DERISI GO	SPUR	6.2
	MLSPL	5.9
	C-HMCNN(h)	**4.7**
DIATOMS	SPUR	**20.1**
	MLSPL	22.7
	C-HMCNN(h)	21.6

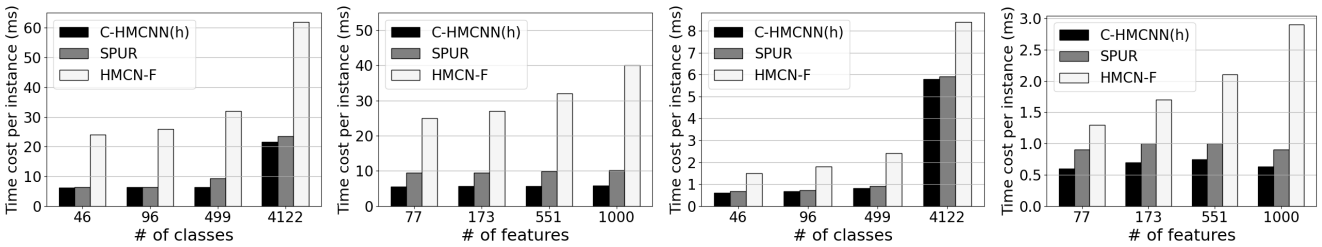
Table 2: Total training time.

C. Supplementary Results

Sensitivity test. Table 3 summarizes the $AU(\overline{PRC})$ scores of SPUR under various λ during training on five different datasets. As can be seen, the model achieves the optimal performance when we set a faster model learning speed with $\lambda = 0.0$. This indicates that taking small learning pace can smooth out the model learning process, which enables our SPUR to efficiently capture rich semantics of input features and taxonomic information for HMLC tasks.

Efficiency test. We also check the total training times of SPUR and two other baselines (i.e., MLSPL and C-HMCNN(h)) on five different datasets. As reported in Table 2, the training time of our SPUR is comparable to the ones of other graph-free baselines on most of the datasets. This implies that the training time and model latency do not change substantially with larger output spaces.

Complexity analysis. We compare time complexity of our SPUR and SOTA methods on HMLC tasks with different number of classes and instances. First, we report the average training time of all models in Figures 6 (a-b). As can be seen, with a larger class number or feature size, the training time of HMLCN-F is increasing much faster than our SPUR and C-HMCNN(h), as HMLCN-F adopts level-wise neural units to output local prediction for each hierarchical level, having the amount of parameters scale with the number of classes or feature dimensions. In contrast, both SPUR and C-HMCNN(h) utilize the post-hoc, parameter-free loss to



(a) The number of classes. (b) The number of features. (c) The number of classes. (d) The number of features.

Figure 6: Analysis of time complexity. We plot the average (i) training time per instance in Figures (a-b), and (ii) inference time per instance in Figures (c-d). 46, 96, 499, 4122 denote the class number of IMCLEF07D, IMCLEF07A, SPO FUN, CELLCYCLE GO datasets, while 77, 173, 551, 1000 represent the feature number of CELLCYCLE FUN, GASCH1 FUN, EXPR GO, ENRON datasets, respectively.

Dataset	λ	$AU(\overline{PRC})$
CELLCYCLE FUN	0.8	0.249
	0.4	0.255
	0.2	0.257
	0.0	**0.260**
DERISI FUN	0.8	0.190
	0.4	0.196
	0.2	0.195
	0.0	**0.198**
CELLCYCLE GO	0.8	0.410
	0.4	0.415
	0.2	0.418
	0.0	**0.420**
DERISI GO	0.8	0.360
	0.4	0.360
	0.2	0.364
	0.0	**0.374**
DIATOMS	0.8	0.761
	0.4	0.763
	0.2	0.768
	0.0	**0.776**

Table 3: Sensitivity test on λ of SPUR.

impose the global hierarchy violation. Besides, the introduction of pre-trained class node representations does facilitate improving the overall performances of our SPUR, while slightly burdening the computational cost compared to C-HMCNN(h). Similar observations can also be found in the average inference time cases, as depicted in Figures 6 (c-d). Notably, the time discrepancy between SPUR and C-HMCNN(h) becomes negligible, which indicates that such graph-based self-paced learning protocol will not incur high latency for real-life scenarios with large output space.

Acknowledgements

This research is supported in part by the National Natural Science Foundation of China (Grant No.92370204), the Foshan HKUST Projects (FSUST21-FYTRI01A), the Guangdong Science and Technology Department, the Science and Technology Planning Project of Guangdong Province (Grant No. 2023A0505050111), Guangzhou Quwan Network Technology Co., Ltd, and the Guangzhou-HKUST(GZ) Joint Funding Program (Grant No. 2024A03J0630, No. 2023A03J0008).

References

Ashburner, M.; Ball, C. A.; Blake, J. A.; Botstein, D.; Butler, H.; Cherry, J. M.; Davis, A. P.; Dolinski, K.; Dwight, S. S.; Eppig, J. T.; et al. 2000. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*, 25(1): 25–29.

Barros, R. C.; Cerri, R.; Freitas, A. A.; and de Carvalho, A. C. 2013. Probabilistic Clustering for Hierarchical Multi-Label Classification of Protein Functions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 385–400. Springer.

Bi, W.; and Kwok, J. T. 2011. Multilabel Classification on Tree-And Dag-Structured Hierarchies. In *International Conference on Machine Learning*.

Cerri, R.; Barros, R. C.; and De Carvalho, A. C. 2014. Hierarchical Multi-Label Classification Using Local Neural Networks. *Journal of Computer and System Sciences*, 80(1): 39–56.

Cerri, R.; Barros, R. C.; PLF de Carvalho, A. C.; and Jin, Y. 2016. Reduction Strategies for Hierarchical Multi-Label Classification in Protein Function Prediction. *BMC Bioinformatics*, 17(1): 1–24.

Cesa-Bianchi, N.; Gentile, C.; Tironi, A.; and Zaniboni, L. 2004. Incremental Algorithms for Hierarchical Classification. *Advances in Neural Information Processing Systems*, 17.

Cheng, Y. 1995. Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8): 790–799.

Dimitrovski, I.; Kocev, D.; Loskovska, S.; and Džeroski, S. 2011. Hierarchical Annotation of Medical Images. *Pattern Recognition*, 44(10-11): 2436–2449.

Dimitrovski, I.; Kocev, D.; Loskovska, S.; and Džeroski, S. 2012. Hierarchical Classification of Diatom Images Using Ensembles of Predictive Clustering Trees. *Ecological Informatics*, 7(1): 19–29.

Friedman, M. 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32(200): 675–701.

Gehan, E. A. 1965. A Generalized Wilcoxon Test for Comparing Arbitrarily Singly-Censored Samples. *Biometrika*, 52(1-2): 203–224.

Giunchiglia, E.; and Lukasiewicz, T. 2020. Coherent Hierarchical Multi-Label Classification Networks. *Advances in Neural Information Processing Systems*, 33: 9662–9673.

Hartigan, J. A.; and Wong, M. A. 1979. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1): 100–108.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*.

Klimt, B.; and Yang, Y. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In *European Conference on Machine Learning*, 217–226. Springer.

Kumar, M.; Packer, B.; and Koller, D. 2010. Self-Paced Learning for Latent Variable Models. *Advances in Neural Information Processing Systems*, 23.

Leman, A.; and Weisfeiler, B. 1968. A Reduction of a Graph to a Canonical Form and an Algebra Arising During This Reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9): 12–16.

Lewis, D. D.; Yang, Y.; Russell-Rose, T.; and Li, F. 2004. Rcv1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5(Apr): 361–397.

Li, C.; Wei, F.; Yan, J.; Zhang, X.; Liu, Q.; and Zha, H. 2017. A Self-Paced Regularization Framework for Multi-label Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6): 2660–2666.

Masera, L.; and Blanzieri, E. 2018. AWX: An Integrated Approach to Hierarchical-Multilabel Classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 322–336. Springer.

McKay, B. D.; and Piperno, A. 2014. Practical Graph Isomorphism, II. *Journal of Symbolic Computation*, 60: 94–112.

Meng, D.; Zhao, Q.; and Jiang, L. 2017. A Theoretical Understanding of Self-Paced Learning. *Information Sciences*, 414: 319–328.

Mittal, A.; Sachdeva, N.; Agrawal, S.; Agarwal, S.; Kar, P.; and Varma, M. 2021. ECLARE: Extreme Classification with Label Graph Correlations. In *Proceedings of the Web Conference 2021*, 3721–3732.

Nakano, F. K.; Lietaert, M.; and Vens, C. 2019. Machine Learning for Discovering Missing or Wrong Protein Function Annotations. *BMC Bioinformatics*, 20(1): 1–32.

Nemenyi, P. B. 1963. *Distribution-Free Multiple Comparisons.* Princeton University.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32.

Patel, D.; Dangati, P.; Lee, J.-Y.; Boratko, M.; and McCallum, A. 2021. Modeling Label Space Interactions in Multi-Label Classification Using Box Embeddings. In *International Conference on Learning Representations*.

Radivojac, P.; Clark, W. T.; Oron, T. R.; Schnoes, A. M.; Wittkop, T.; Sokolov, A.; Graim, K.; Funk, C.; Verspoor, K.; Ben-Hur, A.; et al. 2013. A Large-Scale Evaluation of Computational Protein Function Prediction. *Nature Methods*, 10(3): 221–227.

Ruepp, A.; Zollner, A.; Maier, D.; Albermann, K.; Hani, J.; Mokrejs, M.; Tetko, I.; Güldener, U.; Mannhaupt, G.; Münsterkötter, M.; et al. 2004. The FunCat, A Functional Annotation Scheme for Systematic Classification of Proteins from Whole Genomes. *Nucleic Acids Research*, 32(18): 5539–5545.

Schietgat, L.; Vens, C.; Struyf, J.; Blockeel, H.; Kocev, D.; and Džeroski, S. 2010. Predicting Gene Function Using Hierarchical Multi-Label Decision Tree Ensembles. *BMC Bioinformatics*, 11(1): 1–14.

Senge, R.; Coz, J. J. d.; and Hüllermeier, E. 2014. On the Problem of Error Propagation in Classifier Chains for Multi-Label Classification. In *Data Analysis, Machine Learning and Knowledge Discovery*, 163–170. Springer.

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(9).

Sorower, M. S. 2010. A Literature Survey on Algorithms for Multi-Label Learning. *Oregon State University, Corvallis*, 18: 1–25.

Valentini, G. 2010. True Path Rule Hierarchical Ensembles for Genome-Wide Gene Function Prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(3): 832–847.

Wehrmann, J.; Cerri, R.; and Barros, R. 2018. Hierarchical Multi-Label Classification Networks. In *International Conference on Machine Learning*, 5075–5084. PMLR.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful Are Graph Neural Networks? *International Conference on Learning Representations*.

Yu, H.-F.; Zhong, K.; Zhang, J.; Chang, W.-C.; and Dhillon, I. S. 2022. PECOS: Prediction for Enormous and Correlated Output Spaces. *Journal of Machine Learning Research*, 23(98): 1–32.

Zhang, J.; Chang, W.-C.; Yu, H.-F.; and Dhillon, I. 2021. Fast Multi-Resolution Transformer Fine-Tuning for Extreme Multi-Label Text Classification. *Advances in Neural Information Processing Systems*, 34: 7267–7280.

Zhang, Z.; and Sabuncu, M. 2018. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. *Advances in Neural Information Processing Systems*, 31.