ADDRESSING CLASS IMBALANCE IN HIERARCHICAL MULTILABEL CLASSIFICATION WITH NODE-WISE WEIGHTING

by

Ayushi Sharma

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
June 2025

I dedicate this thesis to my parents for their support and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

Class imbalance is a persistent challenge in machine learning, particularly in hierarchical multilabel datasets where classes are organized in a hierarchically structured taxonomy and multiple labels can be assigned to a single sample. In general, class imbalance negatively affects model learning, as the majority classes dominate the training process while minority classes are often overlooked, leading to biased predictions and poor generalization. These challenges in model learning are further amplified in hierarchical datasets due to the dependencies between parent and child labels. To address this, we propose an adapted node-wise class weighting method specifically designed for hierarchical multilabel data, where we use a hierarchical constraint mechanism to preserve the hierarchical structure of the dataset and reimagine hierarchical multilabel imbalance as defined by node frequencies in a dataset rather than by its sample distribution. We evaluate the effectiveness of the proposed method on multiple hierarchical multilabel datasets, including functional genomics datasets and an oceanographic dataset. We analyze which evaluation metrics provide a comprehensive assessment that is specifically suited for assessing hierarchical multilabel tasks. The experimental results show that our node-wise weighting method consistently improves recall for minority classes without sacrificing much precision on majority classes, outperforming prior techniques for handling class imbalance. These findings show the potential of our method to address class imbalance in hierarchical settings, with implications for a wide range of hierarchical multilabel classification tasks encountered in real-world applications.

List of Abbreviations

AP Average Precision

AUPRC Area Under the Precision-Recall Curve

AUROC Area Under the Receiver Operating Characteristic Curve

Bin. AP Binarized Average Precision

CVIR Coefficient of Variation of IRLbl

DAG Directed Acyclic Graph

FD Full Depth

FMA Free Music Archive

FN False Negatives

FP False Positives

FUN Functional Catalogue

GO Gene Ontology

HMC Hierarchical Multilabel Classification

HMD Hierarchical Multilabel Dataset

HMeanIR Hierarchical Mean Imbalance Ratio

HML Hierarchical Multilabel Label

HROS-FD Hierarchical Random Over-Sampling for Full Depth

HROS-PD Hierarchical Random Over-Sampling for Parital Depth

HRUS-FD Hierarchical Random Under-Sampling for Full Depth

HRUS-PD Hierarchical Random Under-Sampling for Parital Depth

IRLbl Imbalance Ratio per Label

IRLbP Imbalance Ratio per Label Path

LP Label Powerset

LPROS Label Powerset Random Oversampling

LPRUS Label Powerset Random Undersampling

MeanIR Mean Imbalance Ratio

MLC Multilabel Classification

MLD Multilabel Dataset

MLeNN Multilabel Edited Nearest Neighbor

MLROS Multilabel Random Oversampling

MLRUS Multilabel Random UnderSampling

MLSMOTE Multilabel Synthetic Minority Over-sampling Technique

PD Partial Depth

REMEDIAL Resampling Multilabel datasets by Decoupling Imbalanced Labels

TP True Positives

Acknowledgements

I would like to convey my deepest gratitude to my supervisor, Dr. Thomas Trappenberg, for his invaluable guidance throughout my master's journey. The successful completion of this work is due to his constant support and mentorship, his patience and constructive criticism, and the consistent feedback he offered.

I sincerely thank my research project collaborator, Isaac Xu, who designed the experiments and supervised the implementations. His collaborative spirit and technical acumen greatly improved the quality of this work. He was consistently available to clarify my questions and responded promptly. He gave constructive feedback on my work while remaining consistently patient throughout. His clear explanations and genuine kindness helped me through the most challenging parts of the project.

I really appreciate the help I always received from Dr. Martin Gillis, who was ready to help whenever I asked. His willingness to address my doubts or simply share a quick dose of optimism brought fresh energy to the Hierarchical Anticipatory Learning (HAL) lab at Dalhousie University and my work.

Lastly, I extend my thanks to all members of the HAL Lab. The exchange of ideas fostered a supportive environment in which I learned a great deal. My time in the lab was both rewarding and memorable, and I will always value the experience.

Chapter 1

Introduction

This thesis addresses the challenge of class imbalance [15] [27] in machine learning classification tasks involving HMDs [2] [34] [42]. In HMDs, labels are organized in a structured format having a parent-child relationship with other labels, forming a hierarchy as shown in Figure 1.1. For example, in a dataset, an image may be annotated not only with a specific child label but also with all of its ancestor labels in the hierarchy. When such a hierarchically labelled sample contains multiple labels, the dataset is referred to as an HMD. For example, an image dataset containing a sample with a cat and a dog, would be annotated with multiple hierarchical labels such as "Animal/Cat, Animal/Dog," indicating that it possesses more than one label, each reflecting its position within the label hierarchy, where "Animal" is their parent label.

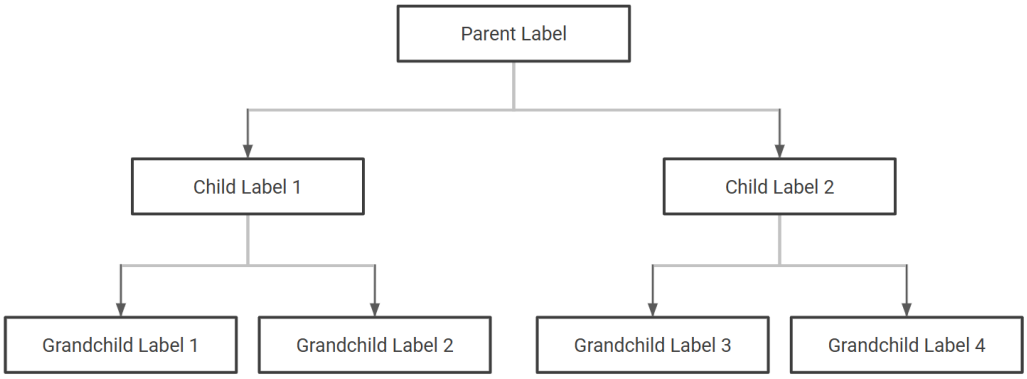


Figure 1.1: Parent–Child Label Relationships in a Hierarchical Structure. The parent label branches into multiple child labels, each of which may further branch into more specific grandchild labels, reflecting a multi-level classification hierarchy.

Certain classes that are deeper in the label hierarchy have fewer samples than other classes that are on top of the label hierarchy. This is because the labels at the top of the hierarchy are more general categories, while those at the bottom are more specific. In an HMD, this results in class imbalance, a condition where the distribution

of samples across categories is highly unequal, leading to one or more categories being significantly underrepresented compared to the others. The imbalance leads to biased models that favor more frequent classes, resulting in poor precision and recall for underrepresented categories. These underrepresented categories might carry more importance for scientific analysis, such as identifying specific marine species for biodiversity monitoring or detecting low-frequency gene functions in biological data. Therefore, it is essential for the model to learn both underrepresented classes and well-represented general classes.

Handling class imbalance in multilabel and hierarchical multilabel datasets is challenging for several reasons. In multilabel datasets, some unique combination of labels occurs only once or twice, which requires different ways to address the class imbalance [8]. The problem becomes more complex in hierarchical multilabel datasets because the label structure must follow parent-child relationships. Simply duplicating samples can worsen imbalance at higher levels of the hierarchy, and converting hierarchical data into a flat format often results in significant combinatorial complexity, as it requires tracking all possible root-to-leaf label paths in the hierarchy. Moreover, many hierarchies are not simple trees but more complex graphs, which makes standard techniques less effective. There are methods that address class imbalance in HMDs using resampling strategies [37] [40]. These resampling strategies involve either duplicating rare classes in the training data, known as oversampling [7], or reducing the number of samples containing common classes, known as undersampling [33]. For example, in multilabel classification [3] [54], a technique helps duplicate samples associated with less frequent label combinations, that is, unique sets of labels that co-occur in a single sample [37]. When resampling HMDs, such a technique is used as part of the process of resampling, along with additional steps to account for the hierarchical structure of the labels. There are also some resampling techniques [40] specifically designed for HMC tasks. These methods can help to some extent, but they also have their drawbacks. They make the dataset larger, which increases the training time and computational complexity. Some HML resampling techniques are unsuitable for addressing imbalance in non-tree hierarchical structures. Moreover, some of these approaches do not incorporate hierarchical dependencies into the learning objective.

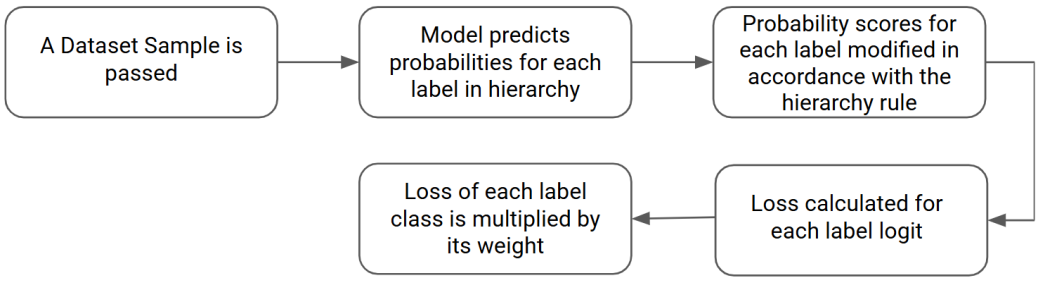


Figure 1.2: High-Level Workflow of the Proposed Node-Wise Weighting Method. The model processes a dataset sample to predict probabilities for each label in the hierarchy. These probabilities are adjusted based on hierarchical constraints. A loss is computed for each label logit, which is then weighted according to the frequency of the corresponding label class.

To overcome these challenges, we aim to find a technique that addresses the imbalance in a way that makes models more confident for rare classes. To address the issue, this thesis proposes a novel node-wise weighting approach, the working mechanism of which is illustrated in Figure 1.2. It involves reducing the imbalance directly into the loss function of the model, rather than requiring changes to the dataset. It computes class-specific weights based on node frequency and applies them during training, giving more importance to rare labels and less importance to common ones. We apply a hierarchical constraint mechanism [25] to ensure that the parent-child relationships between nodes are respected. In doing so, the model learns more balanced representations without compromising the hierarchical structure of the data. This method has several advantages. It does not increase the size of the dataset or require duplicating samples, which helps avoid extra computational costs and overfitting [51]. It can help the model learn to predict labels that were previously ignored, especially those that are rare and deep in the hierarchy. This leads to better recall for rare classes, meaning the model is more likely to correctly identify them, even if it sacrifices some precision on common ones.

We test our method on multiple HMDs. These include a set of non-image datasets from the biological domain [2] [42] and an image dataset containing oceanographic images [34]. These datasets vary in size and complexity, where the relationship between labels are hierarchical. The hierarchy of labels can be either tree hierarchies or non-tree hierarchies where labels can have more than one parent. This helps test whether our approach works consistently across different types of hierarchical datasets

such as feature-based non-image and image-based datasets. Our results show that node-wise weighting significantly improves performance on rare labels and allows the model to predict labels that were previously missed, in both types of datasets. Although the improvements are not as strong when using larger and more powerful models especially after they are pre-trained on in-domain dataset.

We explore whether the commonly used evaluation metrics for model performance in a multilabel setting are equally good for evaluation in HMC, especially for rare classes. Following this, we also introduce additional metrics for evaluating model performance and discuss whether relying on a single metric is comprehensive in analyzing the overall model performance.

Chapter 2

Background

2.1 Foundations of Hierarchical Multilabel Learning

To understand our research, we need to initially understand the structure and nature of hierarchical multilabel datasets. For this, we need to understand what is meant by 'hierarchy' and 'multilabel'.

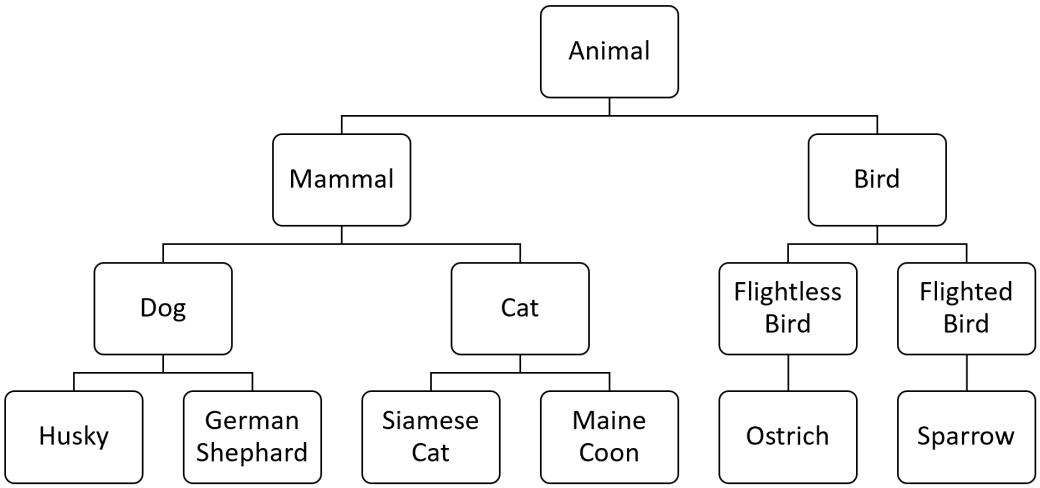


Figure 2.1: Hierarchy of Animal Classification. The general category like 'Animal' branch into specific subcategories such as 'Mammal' and 'Bird', and further into leaf nodes like 'Dog' or 'Ostrich'. This shows how child labels are connected to their ancestors, a key concept in hierarchical structure.

A hierarchy refers to an arrangement of categories or labels at different levels. Each element in a hierarchy has one or more parent elements, except for the top-most element, also called the root. Each element may have one or more child elements. A hierarchy can be defined as a Directed Acyclic Graph (DAG) that can be either a tree structure, where a child node has only one parent node, or a non-tree, where a child node can have more than one parent node. In a tree, the nodes at the top represent broad categories, and as one moves downward, the nodes become increasingly specific. When the labels in a dataset are organized in such a hierarchical structure, they

are called hierarchical datasets. An example of a hierarchical structure is shown in Figure 2.1. In this figure, the Animal category is the topmost element and has subcategories, such as Mammal and Bird, which have their own subcategories, and so on. Each category or element, also referred as a node or a class, in the hierarchy inherits the contextual meaning and properties of its parent element, also known as its ancestor, for instance, the class Husky inherits associations from all of its parent nodes such as 'Dog', 'Mammal', and 'Animal'.

In a multilabel setting, each data sample can belong to multiple classes simultaneously, with labels assigned from a fixed set of possible classes. When a dataset contains such multilabeled samples, it is referred to as a multilabel dataset. For example, an image can contain several animals, such as a 'Dog', a 'Cat', and a 'Bird', that coexist independently, making it a multilabeled data sample.

When hierarchical datasets contain any sample with multiple labels, they are referred to as HMDs. For instance, in an HMD, an image sample might be labeled with two classes, 'Dog' and 'Ostrich', whose hierarchical label paths look like 'Animal' > 'Mammal' > 'Dog' and 'Animal' > 'Bird' > 'Flightless Bird' > 'Ostrich'.

Class imbalance is a challenge not only in non-hierarchical datasets but also in hierarchical ones, including HMDs, due to two inherent characteristics of natural hierarchies. Firstly, the natural distribution of different classes is not balanced, as some categories are naturally rare. For example, 'German Shepherds' are found in greater numbers compared to 'Otterhounds'. Another reason is that classes appearing deeper in the hierarchy, which represent more specific categories, usually have far fewer samples compared to those closer to the top of the hierarchy. For instance, while there may be many samples for a general category like 'Flower', there are likely to be very few for a specific type, such as 'White Orchid'. So, every time we include a rare, specific class, we also automatically include its broader parent classes, which further emphasizes the imbalance. This constraint is discussed in Subsection 2.2.2.

We are interested in exploring how to handle class imbalance in HMDs because it is a significant problem in our domain of interest, the BenthicNet dataset [34]. Since minority classes of BenthicNet often correspond to rare but ecologically critical marine species, accurately identifying these species is essential for biodiversity monitoring and ecological assessments. However, class imbalance hinders this by making it difficult

for models to learn meaningful representations of minority classes. Models then favor majority classes, leading to biased predictions, poor precision and recall for rare categories, and ultimately, a reduced ability to capture vital ecological information. In HML learning, rare categories sometimes never even get predicted in a sample. This is typically due to their low confidence scores, which fall below the set probability threshold, and this results in the model failing to predict them altogether.

2.2 Related Work

2.2.1 Class Imbalance in Prior Research

We initially explore how class imbalance is addressed in MLDs [59] before moving on to how it is addressed in HMDs.

In the study [8] of class imbalance in MLDs, the authors mention specialized metrics to quantify imbalance in multilabel data and resampling algorithms tailored to the unique structure of Multilabel Classification (MLC) as proposed in [11]. To quantify imbalance, the metrics are the Imbalance Ratio per Label (IRLbl) metric, which measures how skewed the frequency of a given label is compared to the most frequent label, and the Mean Imbalance Ratio (MeanIR) metric, which provides the average imbalance across all labels, summarizing the overall imbalance in the dataset. Finally, the Coefficient of Variation of IRLbl (CVIR) metric reveals how much the imbalance varies between labels, highlighting whether the skew is uniform or concentrated in a few problematic labels. The resampling algorithms are grouped into two categories. The first category is based on the Label Powerset (LP) transformation [4] [14], where each unique combination of labels is treated as a distinct class. From this, the authors propose Label Powerset Random Undersampling (LPRUS) [11] and LPROS [11], which respectively remove or duplicate samples. Before defining what LPROS is, we define what a labelset is. A labelset, in a multilabel dataset, represents a distinct combination of multiple labels assigned to a sample, effectively treating that combination as a single, unified label. For example, a sample annotated with labels ‘A’, ‘B’, and ‘C’ would belong to the labelset ‘ABC’. LPROS oversamples samples based on unique labelsets rather than individual labels and treats each distinct labelset as a single class. This helps increase the representation of minority labelsets

by duplicating samples associated with them. LP-based methods can, however, fail in cases where most labelsets are unique, a common occurrence in high-dimensional MLDs. To overcome this limitation, the second category of multilabel resampling methods addresses imbalance at the individual label level. These are Multilabel Random UnderSampling (MLRUS) [8] and Multilabel Random Oversampling (MLROS) [8], which apply random undersampling or oversampling by focusing only on samples that contain underrepresented labels, as identified using the IRLbl and MeanIR metrics. These algorithms ensure that the resampling process directly targets label imbalance rather than indirectly through label combinations.

The authors conduct an experimental evaluation across 13 widely used MLDs and four prominent MLC algorithms — CLR [22], RAkEL [55], IBLR [16], and HOMER [53] — testing each of the four proposed resampling methods under different configurations. Performance is measured using metrics such as accuracy, micro-FMeasure, and macro-FMeasure. Macro-FMeasure calculates F1 score per label and then averages the results, giving equal weight to rare and frequent labels, making it sensitive to how well the model handles minority labels. In contrast, Micro-FMeasure aggregates true positives, false positives, false negatives across all labels before computing the F1 score, which tends to emphasize performance on frequent labels. The experimental findings show that MLROS, especially with 10% oversampling, yields the best performance across most evaluation metrics.

There is other literature, a survey [51] that focuses on the problem of class imbalance in MLC. Firstly, the authors discuss three distinct types of imbalance. The first is within-label imbalance, where individual labels are unevenly distributed across samples. This means that for any given label, there are often fewer positive samples where the label is present compared to negative ones where the label is absent [38] [46] [47]. For example, assuming the label 'Dog' is present in only 200 out of 10,000 samples, meaning it has 200 positive cases and 9,800 negative ones. This shows a strong within-label imbalance for the Dog label. The second is among-label imbalance, which occurs when some labels appear in the dataset far more frequently than others [10] [20]. For example, the label 'Animal' appears in 7,000 samples, while 'Cat' appears in only 500, and 'Dog' appears in 200. This difference in overall frequency between labels illustrates among-label imbalance. The third form is label-set imbalance

[8], which refers to the rarity of certain label combinations. In multilabel datasets, each instance can be associated with a labelset. Due to the combinatorial nature of multilabel assignments, most of these labelsets occur only once or a few times in the entire dataset. For example, in a medical dataset, the combination of 'Cough, Fever, Seizure' might appear in only one patient record, while 'Cough, Fever' might appear in hundreds. These imbalances can degrade model performance, especially when rare labels are also important to predict, as is often the case in real-world domains like medical diagnostics or rare object detection.

Secondly, they categorize existing solutions into four main groups: resampling methods, classifier adaptations, ensemble methods, and cost-sensitive learning. The resampling approaches include LPRUS, LPROS, MLRUS, and MLROS. LPRUS and LPROS are limited by the sparseness of the labels in the MLDs, that is, there are MLDs with as many distinct label combinations as instances. Another approach to tackle this limitation is to evaluate the individual imbalance level of each label. MLRUS and MLROS are examples of such an approach that looks at the frequency of individual labels instead of full labelsets. Although their limitation is that some of the minority samples selected by MLROS may contain the most frequent labels along with minority labels. Therefore, the oversampling will include both the majority and minority labels. This limitation is solved by Resampling Multilabel datasets by Decoupling Imbalanced Labels (REMEDIAL) [10], which improves sampling by separating majority and minority labels within the same instance before resampling, addressing the issue of co-occurrence that can otherwise dilute the impact of oversampling. Heuristic resampling techniques remove or duplicate samples heuristically instead of randomly. For instance, Multilabel Edited Nearest Neighbor (MLeNN) [12] removes majority class samples based on how dissimilar they are to their neighbors, reducing the risk of losing important information. Multilabel Synthetic Minority Over-sampling Technique (MLSMOTE) [13] generates synthetic samples by interpolating between existing samples that share rare labels. The classifier adaptation methods focus on modifying the learning algorithm to address the imbalance. In contrast, the ensemble approaches combine multiple models to make a final prediction in MLC. Techniques such as BR-IRUS [50] use inverse random undersampling with

binary relevance classifiers, while heterogeneous ensembles [48] combine diverse algorithms. Other strategies include stacking, for example, MLKNN-based [39], ensemble classifier chains, ECCRU3 [32], ensemble of multilabel classifiers [49], and traditional approaches like bagging and adaptive boosting [57]. Cost-sensitive approaches incorporate imbalance directly into the learning objective by assigning higher penalties for misclassifying minority labels, making the model more attentive to rare classes. In addition to these solutions, the survey paper discusses several metrics for assessing the severity of imbalance and model performance. To measure imbalance, metrics such as IRLbl [11], MeanIR [11], that were also mentioned in the earlier paper [8], and SCUMBLE [9] are discussed. SCUMBLE quantifies how often rare and frequent labels appear together in the same sample. For evaluating models, the authors discuss three types of metrics. Example-based metrics [26] are computed individually for each sample, then averaged to obtain the final value, using measures such as Hamming Loss [52], which measures the proportion of labels that are predicted incorrectly for each sample, while Subset Accuracy [54] evaluates the percentage of correctly predicted labels among all predicted and true labels. It requires the predicted set of labels to match the true set of labels, Precision, Recall, and F-Measure. Label-based metrics [36] evaluate the performance for each individual label across the dataset and then average the results using either micro-averaging or macro-averaging. Micro-averaging aggregates the true positives, false positives, true negatives, and false negatives across all labels before computing the metric, effectively giving more weight to frequent labels. In contrast, macro-averaging calculates the metric separately for each label and then takes the average. Ranking-based metrics [28] evaluate the ranking of labels such as one-error [35], coverage, ranking loss, and average precision.

After exploring how class imbalance is handled in MLDs, we look at literature that addresses the same issue in HMDs. One of the studies we found tackles class imbalance by directly resampling HMDs. The same authors had previously investigated this problem in their earlier work, using an alternative approach that did not employ resampling directly on HMDs. We are interested in these approaches because we want to compare how well our method performs against them and evaluate the effectiveness of our model compared to both of theirs. We shall first look at their initial approach to tackling class imbalance [37] and then discuss the approach presented in their later

work [40].

In the study by Pereira et al. [37], the Imbalance Ratio per Label Path (IRLbP) and Hierarchical Mean Imbalance Ratio (HMeanIR) metrics are used to quantify the degree of imbalance in a dataset, that are catered for HMDs. A label path is the sequence of nodes from the root to a specific leaf in the hierarchy, as shown in Figure 2.2. As described in this study, the IRLbP metric measures the degree of imbalance for each label path.

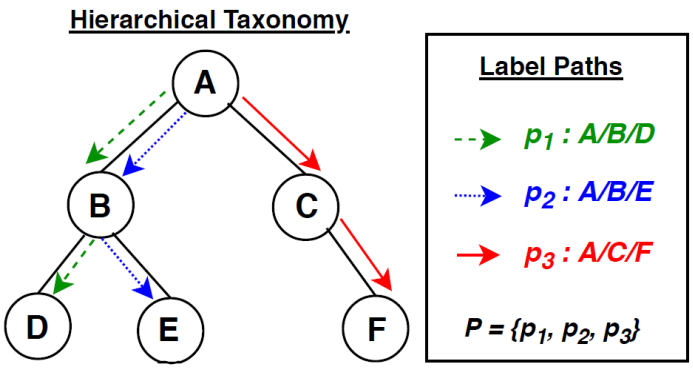


Figure 2.2: Label Paths in a Hierarchical Taxonomy. Each path corresponds to a sequence of labels describing the classification hierarchy for individual instances (reproduced from [37]).

The Equation 2.1 provides the formula of IRLbP for a given path p, where the *max(Count of all paths)* is the number of samples associated with the most frequent label path in the dataset, and *Count of path p* is the number of samples associated with a specific label path p. This metric quantifies the imbalance for each individual label path relative to the most frequent path.

$$\text{IRLbP}(p) = \frac{\max(\text{Count of all paths})}{\text{Count of path } p} \tag{2.1}$$

$$\text{HMeanIR} = \frac{1}{|P|} \sum_{p \in P} \text{IRLbP}(p) \tag{2.2}$$

The HMeanIR metric captures the average imbalance of the entire dataset across all label paths in the hierarchy. It is computed as the mean of all IRLbP values as shown in Equation 2.2, where P is the set of all label paths in the hierarchy and *IRLbP(p)* is the imbalance ratio [29] for each path p. A higher HMeanIR value indicates greater overall imbalance in the dataset.

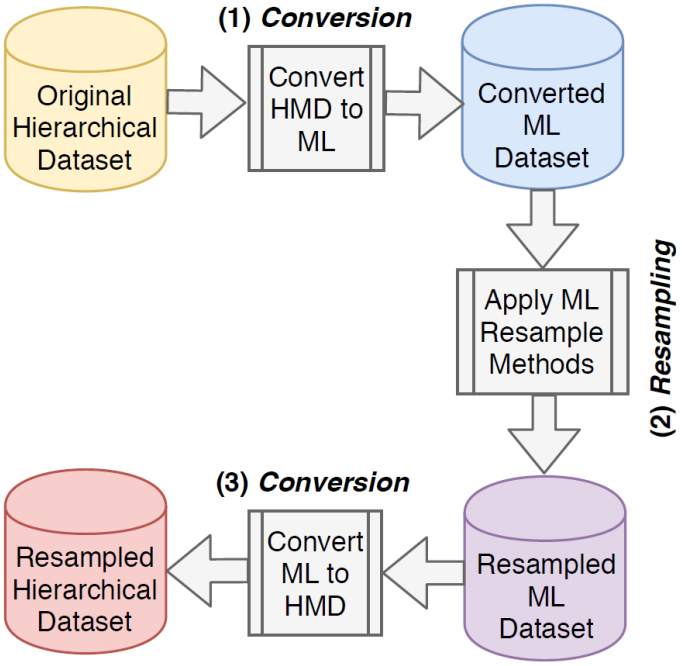


Figure 2.3: Overview of the Proposed Resampling Framework for Handling Class Imbalance in HMDs. The approach consists of three main phases: (1) converting the original HMD into a flat MLD, (2) applying existing multilabel resampling techniques to address label imbalance, and (3) transforming the resampled MLD back into its hierarchical form to obtain a balanced HMD (reproduced from [37]).

The approach to addressing class imbalance in HMDs that involves first converting the data into an MLD, enabling resampling techniques to be applied at that level rather than directly on the hierarchical structure, is illustrated in Figure 2.3. They first convert HMD into flat MLD by taking each hierarchical path and just breaking it down into all the individual labels involved, dropping the parent-child relationships. For example, there is an HMD with one of the labels as 'A/B/E', where 'A' is the root, 'B' is the child of 'A', and 'E' is the child of 'B'. This label is converted to MLD by flattening its path. Therefore, 'A/B/E' becomes the labels 'A', 'B', and 'E'. Then, all the unique labels across all paths for a sample are grouped together into one flat list of labels. In the end, each data point that originally had hierarchical labels now just has a set of flat labels. This makes the data usable for traditional multilabel classification models that do not consider label hierarchies. This is illustrated in Figure 2.4 along with its algorithmic procedure shown in Algorithm 1.

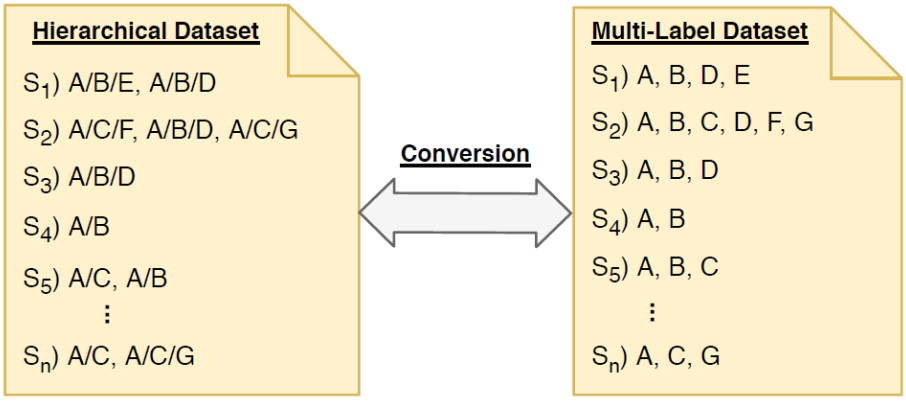


Figure 2.4: Illustration of the Conversion Process Between HMDs and MLDs. This figure shows how label paths in HMDs are flattened into standard multilabel sets during conversion (reproduced from [37]).

Algorithm 1 Algorithm for conversion of a HMD to a MLD. For each instance, the hierarchical label paths are flattened into a unique labelset by aggregating all labels from the paths (reproduced from [37]).

Inputs: D: A Hierarchical Dataset

Output: D': A Multilabel Dataset

$D' \leftarrow$ new empty list of instances
for each h_sample **in** D **do**
    $ml\_sample_{features} \leftarrow h\_sample_{features}$
    $labels \leftarrow$ empty set
    for each $label\_path$ **in** h_sample_{labels} **do**
        $labels \leftarrow labels \cup label\_path_{labels}$
    end for
    $ml\_sample_{labels} \leftarrow labels$
    $D' \leftarrow D' + ml\_sample$
end for
return D'

After converting HMD to MLD, the next step involves applying the traditional multilabel resampling techniques [51] as shown in Figure 2.3. One of the multilabel resampling techniques used is LPROS. The IRLbP and HMeanIR metrics are calculated before applying the resampling techniques to assess the imbalance in the dataset, and

then recalculated after resampling to evaluate how effective the multilabel resampling methods are in reducing the imbalance.

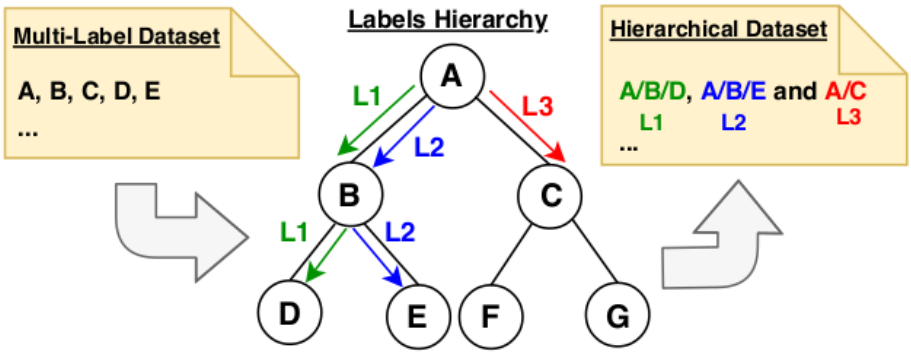


Figure 2.5: Demonstration of the Process to Identify the Longest Label Paths for Restoring the Hierarchical Structure from Resampled Multilabel Instances. The hierarchy is traversed to locate the deepest paths associated with each label (reproduced from [37]).

After resampling the MLD with any one of the multilabel resampling algorithms, the resampled MLD is converted back to HMD. For example, MLD contains a list of labels, such as 'A', 'B', 'C', 'D', and 'E', assigned to a sample as shown in Figure 2.5. Subsequently, they utilize the hierarchical class structure to explore the hierarchy and find the longest path for each label in the multilabel list that leads to it from the root node. For example, for label 'E', the algorithm will trace back up the tree and find that the longest path is 'A/B/E'. It does this for every label in the sample. Then, once it has all those paths, it combines them into multiple hierarchical labels for the sample. But to avoid redundancy, it removes any paths that are already covered by a longer one, so if there is already a path 'A/B/E', there does not have to be a path such as 'A/B' by itself anymore. After processing all the flat labels for each sample, we obtain a set of HMLs from the multilabels. The algorithmic procedure is shown in Algorithm 2.

Using the Free Music Archive (FMA) dataset, in this study [37], the authors demonstrated that out of all the other multilabel resampling techniques they experimented with, LPROS significantly improved classification performance for minority paths, achieving the highest improvements in both Area Under the Receiver Operating Characteristic Curve (AUROC) and Area Under the Precision-Recall Curve

(AUPRC) metrics. Therefore, we choose to compare our method with their methodology that uses LPROS as the resampling technique.

Algorithm 2 Algorithm for reconstruction of a HMD from a flat MLD. The algorithm retrieves the longest paths for each label using the predefined label hierarchy and removes any redundant paths to preserve the hierarchical structure (reproduced from [37]).

Inputs: D: A Multilabel Dataset, H: Labels Hierarchy

Output: D': A Hierarchical Dataset

$D' \leftarrow$ new empty list of instances
for each h_sample **in** D **do**
    $ml\_sample_{features} \leftarrow h\_sample_{features}$
    $labels \leftarrow$ empty set
    for each $label$ **in** h_sample_{labels} **do**
        $path \leftarrow \text{FindLongestPath}(H, label)$
        $labels \leftarrow labels \cup path$
    end for
    $labels \leftarrow \text{RemoveRepeatedPaths}(labels)$
    $ml\_sample_{labels} \leftarrow labels$
    $D' \leftarrow D' + ml\_sample$
end for
return D'

In their later work [40] to handle class imbalance, the authors proposed four novel algorithms that operate directly on HMD, removing the need to convert them into MLD prior to resampling, while preserving the hierarchical relationships between classes. The methods achieve this by either oversampling samples with minority label paths or undersampling samples with majority label paths, ensuring that parent-child dependencies remain intact.

To understand the four algorithms, we shall first look at what Partial Depth (PD) and Full Depth (FD) are. PD can be thought of as missing precision, which is a lack of complete labeling in the dataset. This means that PD samples can be labeled at different depths and do not necessarily reach the deepest leaf level in the hierarchy for

every label path. Whereas, FD provides a complete and detailed path for every label, capturing all labels from root to leaf at the same depth level for every label path. In an ideal hierarchical structure, each sample should be fully annotated from the root to the most specific leaf node. However, due to labeling inconsistencies, many samples lack these complete annotations. Therefore, imbalance is handled differently in FD and PD, which is why the algorithms in this study are categorized to address FD and PD separately, with one algorithm for oversampling and another for undersampling in each category.

Algorithm 3 Pseudocode for computing the imbalance ratio (IRLbP) for each label path and the overall hierarchical mean imbalance ratio (HMeanIR) within a hierarchical dataset. These metrics form the basis for identifying minority and majority paths used in subsequent resampling steps (reproduced from [40]).

Inputs: D: The hierarchical dataset

Output: IRLbP: The imbalance ratio per label paths in D, HMeanIR: The average imbalance ratio in D

labelPaths ← label paths from dataset D
for each path **in** labelPaths **do**
    countDict[path] ← number of samples in D labelled with *path*
end for
maxCount ← max number of labelPaths in countDict
IRLbP ← empty dictionary
for each path **in** labelPaths **do**
    pathCount ← countDict[path]
    IRLbP[path] ← maxCount / pathCount
end for
HMeanIR $\leftarrow \left(\sum_{i=1}^{|\text{labelPaths}|} \text{IRLbP}[path_i] \right) / |\text{labelPaths}|$
return IRLbP, HMeanIR

The four algorithms are Hierarchical Random Over-Sampling for Full Depth (HROS-FD), Hierarchical Random Under-Sampling for Full Depth (HRUS-FD), HROS-PD, and Hierarchical Random Under-Sampling for Parital Depth (HRUS-PD).

Since the datasets we will use in our study follow a PD structure, and this study

found oversampling to be more effective than undersampling, we focused on HROS-PD and compared the performance of our method only with that of this algorithm, among the four proposed. This study also utilizes the imbalance metrics IRLbP and HMeanIR, originally introduced in their earlier work [37], to measure imbalance at various levels of the hierarchy within HMD and evaluate the effectiveness of the proposed algorithms. We now present an explanation of the HROS-PD algorithm to illustrate its underlying mechanism.

The HROS-PD algorithm starts by calculating the IRLbP for each label path, which is then used to derive the HMeanIR, as outlined in the pseudocode shown in Algorithm 3. A label path is identified as a minority path if its IRLbP exceeds the HMeanIR, making it a candidate for oversampling. The procedure for identifying these minority paths is detailed in the pseudocode in Algorithm 4.

Algorithm 4 Pseudocode for extracting the set of minority label paths from a hierarchical dataset. Label paths with an imbalance ratio above the computed HMeanIR are selected as minority paths to guide the oversampling process (reproduced from [40]).

Inputs: D: The hierarchical dataset

Output: minPaths: The label paths from the set of minority paths

```
IRLbP, HMeanIR ← Calculate Imbalance Ratios
minPaths ← empty list
for each path in labelPaths do
    if IRLbP[path] > HMeanIR then
        append path into minPaths
    end if
end for
return minPaths
```

To control the extent of oversampling, they provide a hyperparameter, S, which is a percentage that determines the total number of new samples to be added relative to the dataset size. To avoid oversampling any single minority label path disproportionately, the total number of new samples to be added is divided equally among all the minority paths. This defines the maximum number of samples that can be added

to each individual minority path, referred to as maxIncrease. The algorithm also computes the average number of samples per label path in the dataset. This value serves as a secondary limit, ensuring that no label path—regardless of its minority status, is oversampled beyond what is typical for the dataset as a whole.

The resampling itself is carried out in a bottom-up traversal of the hierarchy, starting from the leaf nodes in the deepest level and moving upward. In each iteration, the algorithm identifies the current set of leaf nodes in the label tree. For each leaf node that belongs to the minority path, it checks how many samples are currently labeled with that path. It then begins duplicating existing samples labeled with that path, preferably selecting samples where the target label path is the deepest node in the label sequence, to avoid unintentionally affecting the distribution of child nodes or overlapping paths elsewhere in the hierarchy. Once a node has been processed, that is, it has reached its duplication limit or its sample count has reached the average, it is removed from the tree. The algorithm then recalculates the set of minority paths at every level, using the original HMeanIR as a fixed threshold throughout the process, and repeats the procedure. This ensures that label paths ending at leaf nodes, that is, the deepest points in the hierarchy, as well as those ending at internal nodes, that is, intermediate levels that may still serve as final labels for some samples, are both adequately represented, without the resampling process favoring one depth level over another.

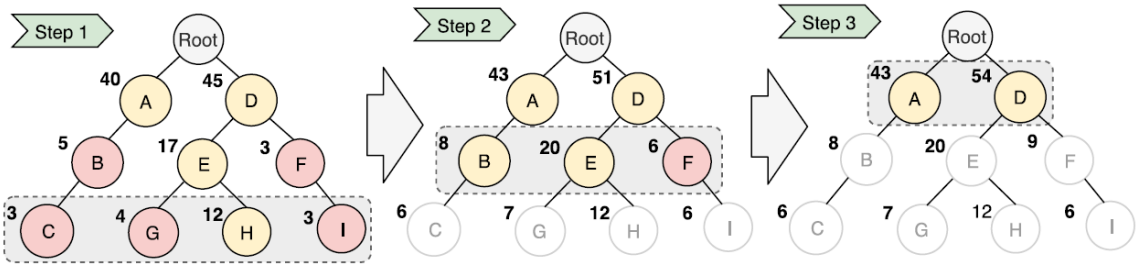


Figure 2.6: Visual Representation of the Hierarchical Resampling Process Using the HROS-PD Algorithm on a Dataset with 85 Instances. Dashed-circle nodes indicate the label paths selected for resampling in the current iteration, while red-highlighted nodes correspond to those identified as part of the minority class set based on the imbalance ratio (reproduced from [40]).

Figure 2.6 illustrates an example of the HROS-PD algorithm applied to a hierarchical dataset containing 85 instances. The resampling process is shown in three

Algorithm 5 Pseudocode for HROS-PD. This algorithm incrementally oversamples the dataset by processing minority label paths in a leaf-first traversal of the label hierarchy, while dynamically updating imbalance statistics to avoid over-representation of internal nodes (reproduced from [40]).

Inputs: D: The hierarchical dataset,
S: Percentage of samples to increase (default = 10%)

Output: D': An oversampled dataset

```
samplesToCreate ← |D| × S
labelTree ← retrieve label tree from D
minPaths, HMeanIR ← getMinorityPaths(D)
meanSize ← calculate the average number of samples per labelPaths
maxIncrease ← samplesToCreate / |minPaths|
while labelTree is not empty do
    for each leafNode in labelTree do
        if leafNode in minPaths then
            numSamples ← samplesWithLabelPath(D, leafNode)
            increased ← 0
            while increased < maxIncrease and numSamples < meanSize do
                D' ← randomly duplicate sample (last labeled with leafNode)
                numSamples ← numSamples +1
                increased ← increased +1
            end while
        end if
        remove leafNode from labelTree
    end for
    minPaths ← getMinorityPaths(D, HMeanIR)
end while
return D'
```

iterative steps. In each step, nodes enclosed in a dashed rectangle represent the current leaf nodes being evaluated for resampling. The nodes highlighted in red identify as minority classes associated with minority label paths, that is, those whose imbalance ratio exceeds the HMeanIR. In Step 1, the algorithm begins by targeting the minority leaf nodes 'C', 'G', and 'I', and duplicates samples associated with those paths. This resampling affects the distribution of their parent nodes as well, for instance, duplicating node 'I' also increases the counts of 'F' and 'D' due to hierarchical inheritance. After removing the processed leaf nodes, Step 2 proceeds with the next level of leaf nodes, 'B', 'E', and 'F'. Again, only those still classified as minority paths based on the original HMeanIR are selected for oversampling. Notably, although 'B' was previously a minority path, it is no longer considered one in this iteration because the oversampling of its child node 'C' in Step 1 indirectly increased the sample count of 'B' through hierarchical inheritance, bringing its updated IRLbP below the original HMeanIR threshold. Finally, in Step 3, the remaining nodes are processed. At this stage, no node belongs to the minority path. The algorithm continues this bottom-up process until the entire tree has been traversed, ensuring that resampling is conducted in a way that respects the hierarchy while improving the representation of minority paths. The pseudocode for the HROS-PD algorithm is shown in Algorithm 5.

2.2.2 Hierarchical Constraint Mechanism

We have seen how previous HMC work ensures that hierarchical consistency is maintained in their approaches to handling class imbalance in HMC. Similarly, we also want to ensure that hierarchical consistency is maintained in our work when we solve the issue of class imbalance. For this, we adopted a constraint mechanism mentioned in a study [25], designed to enforce hierarchical consistency in HMC by incorporating hierarchical constraints into the learning process through the use of an adjacency matrix. This matrix defines ancestor-descendant relationships, allowing for the processing of model outputs to ensure that the constraint is met.

$$A_{ij} = \begin{cases} 1, & \text{if } j \in D_i, \\ 0, & \text{else.} \end{cases} \tag{2.3}$$

$$P(D) \leq P(A) \tag{2.4}$$

Mathematically, a *descendant matrix A* is defined as shown in Equation 2.3, where D_i represents the set of all descendant nodes for node i. This matrix is then applied to the predicted outputs of the model, ensuring that the probability of each ancestor node is always at least as high as that of its descendants. This requirement is formally expressed in Equation 2.4, where *P(D)* represents the probability of a descendant node being assigned to a sample, and *P(A)* represents the probability of an ancestor node being assigned to a sample.

2.3 Evaluation Metrics

To compare the performance of various methods tackling the class imbalance challenge, it is essential to evaluate the correctness of predictions and how well the model captures minority classes. Evaluating how well a model performs is tricky because the accuracy-based metric, such as accuracy, does not work well due to class imbalance and the structure of hierarchical labels. This makes the choice of evaluation metrics particularly important, as they must reflect these different aspects of performance.

2.3.1 Average Precision Score Metric

Research in HML literature has commonly relied on AP, an approximation of the AUPRC, as a standard benchmark for evaluating model performance. We will discuss the reason behind its use and its challenges in Chapter 5. Its challenges prompted an investigation into the Binarized Average Precision (Bin. AP) score as a potential additional metric for evaluation, which is further explained in Chapter 5.

2.3.2 Node-Wise Precision, Node-Wise Recall, Node-Wise F1 Score

In addition to AP and Bin. AP scores, our study also uses node-wise precision, node-wise recall, and node-wise F1 scores to evaluate model performance in HMC. The need for more than one metric other than AP and Bin. AP for evaluation will be discussed in Chapter 5. These node-wise metrics are distinct from the standard precision, recall, and F1 score metrics. As discussed in Section 2.2 in the survey

paper [51], in MLC the standard micro-averaged precision, recall, and F1 scores are calculated by combining all true positives, false positives, and false negatives across all classes, which can be dominated by majority classes and may overlook performance on minority classes. Whereas node-wise metrics are computed individually per node and then averaged, giving equal importance to all nodes in the dataset, including rare ones. To do this, first, the model's sigmoidal outputs are thresholded to convert them into binary labels, 0 or 1. This step ensures that the predictions can be directly compared against the ground truth. For each data sample, predictions are compared against the true labels to determine True Positives (TP), False Positives (FP), and False Negatives (FN). Subsequently, the TP, FP, and FN values are summed across all dataset samples for each node. The precision, recall, and F1 scores are calculated thereafter. Following this, these scores are macro-averaged across all nodes to obtain an overall performance metric.

Chapter 3

Proposed Node-Weighting Method: Detailed Methodology

3.1 Mechanism of Node-Wise Weighting

In contrast to previous works that propose resampling techniques to address class imbalance, our method leaves the dataset unchanged. Instead, it incorporates imbalance handling directly into the training process of the model by factoring it into the loss calculation. This approach forms the basis of our proposed method, which we refer to as node-wise weighting.

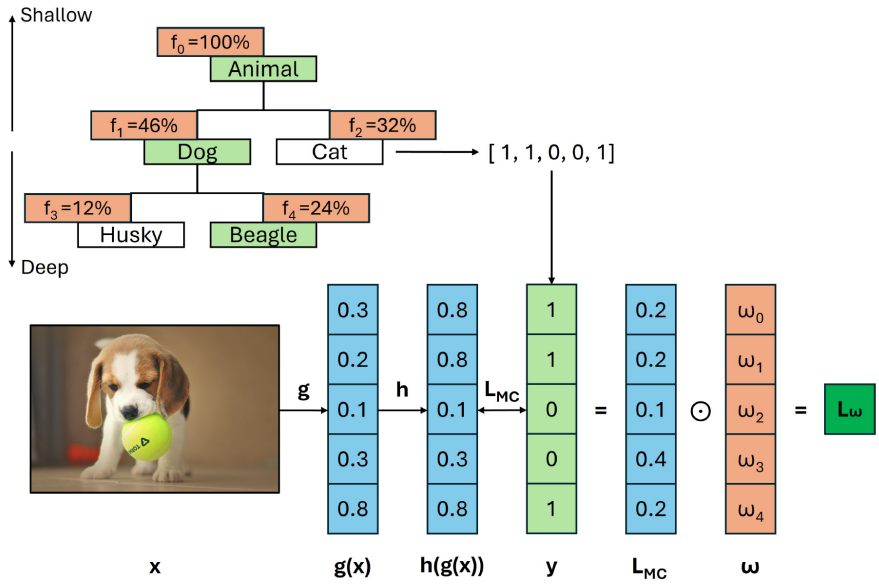


Figure 3.1: An Overview of Hierarchical Classification with Node Weighting. The green boxes in the hierarchy represent the active label for the given input sample. The orange boxes denote the node weights used to compute the weighted loss. An input sample x is fed into the encoder g and processed by a hierarchical constraint mechanism h. The output and annotation y produce the subsequent hierarchical loss L_{MC}. L_{MC} is subsequently multiplied on an element-by-element basis with node weights ω, derived from node frequencies f or the raw counts. The final averaged loss L_ω is computed thereafter and backpropagated.

To begin with, it is essential to clarify that node-wise weighting should not be confused with traditional class weighting. Our node-wise weighting method adjusts the learning process by assigning weights to each individual node, also known as a class. For a sample, the model outputs a sigmoid prediction for each node in the hierarchy - one logit per node. Each node gets its own independent weight based on how often it appears in the dataset. The loss is calculated per node, and the loss of each node is multiplied by its own weight before being summed. So, even if a sample is annotated with multiple nodes, including a majority and a minority node, the model can penalize mistakes on the minority node if needed.

Figure 3.1 presents an overview of the proposed hierarchical classification framework incorporating node-wise weighting. This illustrative example highlights the key components of our method, including feature encoding, hierarchical constraint mechanism enforcement, and loss computation. The input sample x is first passed through the encoder g, where feature extraction takes place, transforming the input into a learned feature representation suitable for hierarchical processing. These encoded features are subsequently processed by a hierarchical constraint mechanism h, which is designed to ensure that the model predictions respect the hierarchical relationships between nodes. To ensure the hierarchical relationships are maintained, the probability of predicting a descendant node should never be higher than that of its ancestor. This is done with the help of a descendant matrix, which is a filter applied to the predicted sigmoidal outputs of the model. For this, the matrix is multiplied by the model's predicted probabilities, modifying predictions to ensure they respect hierarchical dependencies. Each row in the resulting matrix, which is the filtered output matrix, represents the probability of a node and all its descendants being present in a given sample. The maximum is taken from each row across descendants, ensuring that the probability of any node i is always greater than or equal to the probability of its descendants. The entire process of ensuring hierarchical relationships between nodes is illustrated in Figure 3.2. The result of this step is a set of predicted outputs, $h(g(x))$.

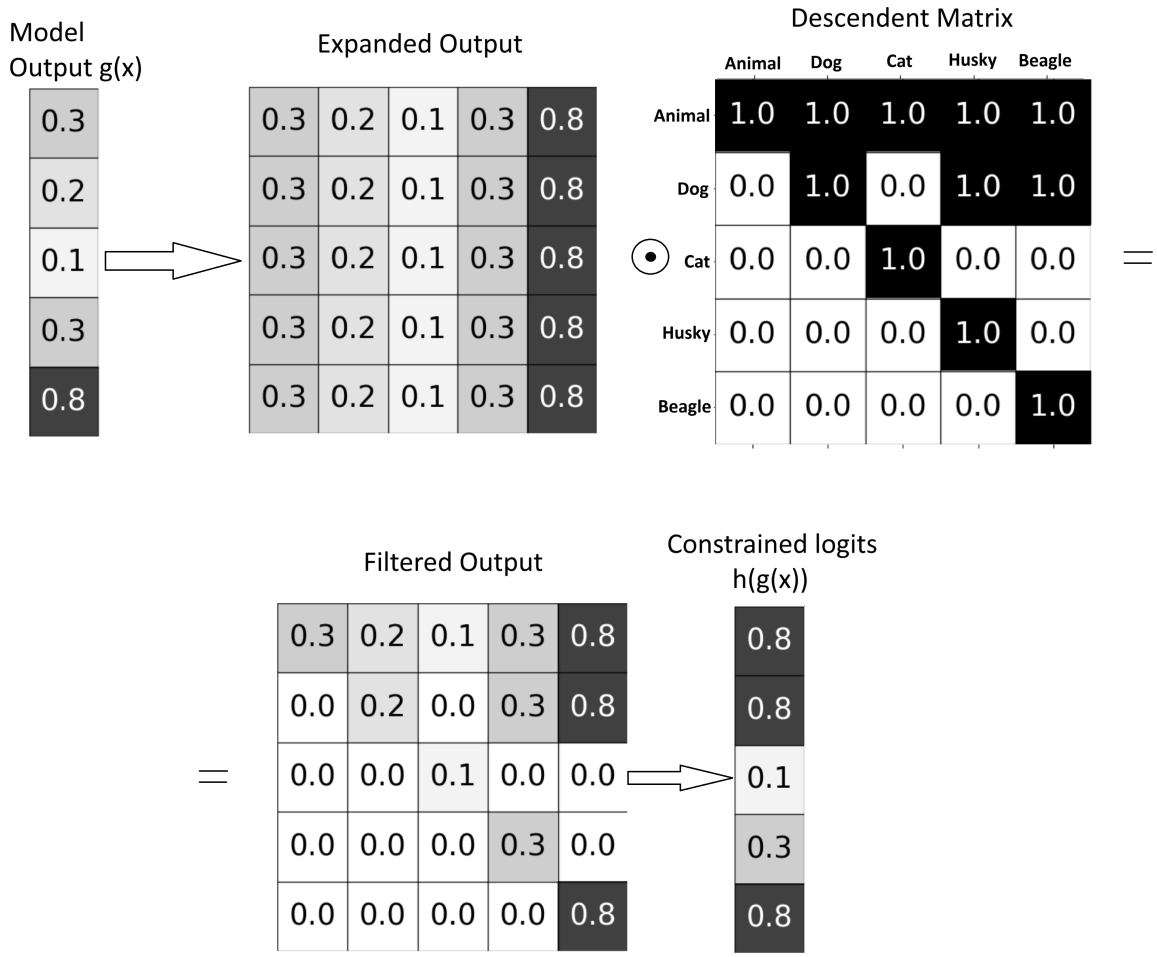


Figure 3.2: Demonstration of C-HMCNN's Hierarchically Constrained Model Output [25]: The model output g(x) is the prediction when the image sample of a beagle is supplied to a trained model. In this example, it violates hierarchy, since the BEAGLE logit is greater than the ANIMAL and DOG logit. This output is subsequently expanded and filtered through the descendant (adjacency) matrix, which represents the hierarchy, via the Hadamard product. Finally, the maximum along each row is taken from the filtered output to obtain a constrained model output, preserving hierarchy.

The core idea of node-weighting is realized in the next step. The output $h(g(x))$ and annotation y produce the subsequent hierarchical loss L_{MC}. L_{MC} is multiplied on a node-by-node basis with each node's respective node weights ω, derived from node frequencies f or the raw counts. The final averaged loss L_{ω} is then computed and backpropagated. This means that each node in the hierarchy is assigned a specific weight, such that common nodes receive lower weights and rare nodes receive higher weights. This approach ensures that the learning process is not dominated by the

majority nodes and that the model gives more attention to rare and deeper nodes in the hierarchy, which would otherwise be overlooked.

By integrating node-specific weights directly into the loss calculation, we get several benefits. Multiplying the loss by node weights prevents majority nodes from overshadowing minority ones during training. Assigning higher weights to minority nodes increases their impact on the final loss, which encourages the model to improve the recall on these underrepresented classes.

3.2 Limitations of Previous Approaches and How Node-Wise Weighting Addresses Them

The resampling technique in prior literature [37] is not scalable or practical for HMDs because the number of possible labelsets increases exponentially ($O(2^n)$) with the number of hierarchical nodes, leading to high computational costs and potential overfitting [51]. Moreover, resampling methods like HROS-PD [40] not only increase the storage requirements but also lead to higher computational costs during training, especially when the dataset is large. In large hierarchical datasets, it can slow down the learning process and may require additional resources. Additionally, HROS-PD works only for tree-based DAG hierarchies, not for non-tree DAG hierarchies. In HMDs, samples often have multiple labels that contain both majority and minority nodes. Prior resampling methods that duplicate entire samples inadvertently reinforce the presence of majority nodes even when the goal is to address imbalance. This leads to the over-representation of already frequent nodes, making it harder for minority nodes to be accurately learned. By duplicating samples, traditional resampling can worsen overfitting [6] [23], especially when the duplicated samples are from minority classes. This makes the model memorize specific samples instead of generalizing the features of the minority class, reducing its ability to perform well on unseen data.

These limitations highlight the need for more sophisticated approaches that do not increase compute power and storage requirements to balance HMDs while respecting the underlying hierarchical relationships. The node-wise weighting helps address these limitations. It directly integrates into the loss calculation, offering a more efficient and scalable solution for complex or large HMDs.

Chapter 4

Datasets

For our experiments, we use various HMDs, such as the Functional Catalogue annotated datasets, which have a tree DAG hierarchy, and the Gene Ontology annotated datasets, which have a non-tree DAG hierarchy. We have also experimented with an image dataset, EchinodermNet, a subset of the BenthicNet dataset [34].

4.1 Functional Catalogue and Gene Ontology Annotated Datasets

The FUN and GO datasets [2] are used in bioinformatics for the hierarchical classification of genes and proteins. FUN [42] is a manually curated classification system that organizes gene expression data from various experimental conditions into a tree hierarchy comprising 1,362 functional categories across six levels [56]. Each dataset contains an average of 492 nodes, 500 per dataset, except for Eisen, which contains 462 nodes. The FUN datasets used in the study are Cellcycle, Derisi, Eisen, Expr, Gasch-1, Gasch-2, Seq, and Spo. Each dataset represents a different subset of gene expression data, with varying degrees of class imbalance at different hierarchical levels. The FUN annotated datasets include yeast gene expression profiles from experiments exploring diverse biological conditions. Cellcycle captures the gene expression of cell-cycle-regulated genes [45]. Derisi [18], Gasch1, and Gasch2 [24] focus on stress response genes, providing insights into how yeast reacts to environmental changes. Spo [17] covers sporulation and structural gene-related studies.

GO is a more comprehensive classification scheme with deeper hierarchical structures compared to FUN. The GO datasets used in this study have the same names as those in the FUN datasets, such as Cellcycle, Derisi, Eisen, Expr, Gasch-1, Gasch-2, Seq, and Spo. Key characteristics of the GO annotated datasets are that their hierarchy contains 22,960 classes, organized into 14 levels of depth, and each dataset has an average of 3,997 functional categories [56]. The GO annotations provide a challenging benchmark due to the large number of classes, more depth levels, and dense labeling.

GO datasets test the scalability and efficiency of HMC methods. Both FUN and GO datasets exhibit label imbalance, where certain classes are underrepresented. This imbalance is more pronounced in GO, where a small subset of functional categories accounts for the majority of labels. This dual annotation allows for a comparison of HMC algorithms across two hierarchical schemes with varying levels of hierarchical depths.

4.2 BenthicNet and EchinodermNet Dataset

The BenthicNet dataset [34] is an oceanographic dataset comprising a large collection of seafloor imagery, designed for research in underwater image classification. Encompassing over 11 million images, it integrates contributions from diverse global sources, including government agencies, research institutions, and individual scientists. Approximately 12%, that is, 188,435 images, are labeled, while the remaining 88% are unlabeled, providing opportunities for supervised, semi-supervised, and self-supervised learning. This dataset is structured in a hierarchical multilabel taxonomy that adheres to the CATAMI classification scheme [1]. This scheme provides a framework for describing benthic habitats, with labels that span biota, substrate, bedforms, and relief features.

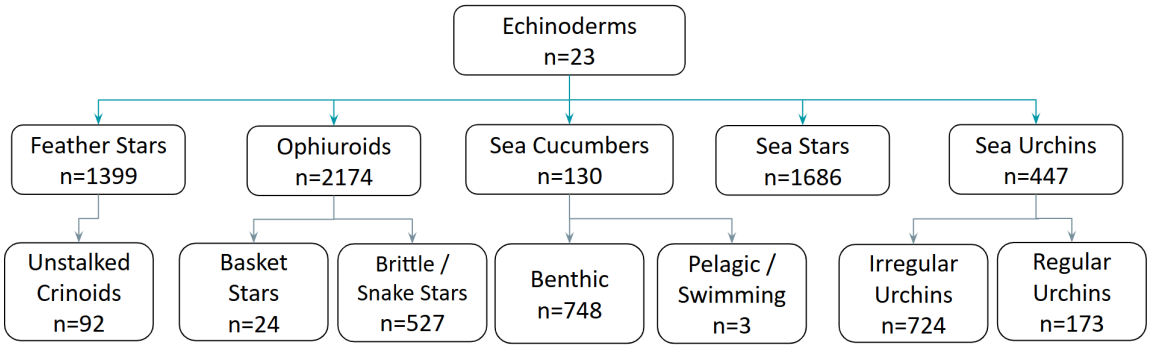


Figure 4.1: Hierarchical Structure of Classes in the EchinodermNet Dataset. The bottom-most labels represent the most specific categories. The value of n indicates the number of individual sample images for each specific label, without aggregation across levels.

We perform experiments on only a subset of the BenthicNet dataset, specifically the EchinodermNet category, which are ecologically vital components of marine ecosystems because the full BenthicNet dataset comprises of 188,435 images,

making it computationally intensive to process the entire dataset. In contrast, the Echinoderm subset contains 8,150 images, allowing for more manageable and efficient experimentation. The hierarchical taxonomy of classes within EchinodermNet is depicted in Figure 4.1. These include species such as sea stars, sea urchins, and brittle stars. Their hierarchical annotations range from general labels like 'Biota > Echinoderm' to more specific ones, such as 'Sea Stars' or 'Unstalked Crinoids'.

Chapter 5

Challenges With Average Precision and the Proposal of Binned Average Precision as an Improved Metric

To evaluate HML models properly, we need metrics suitable for HMC tasks. The way we measure performance plays a crucial role in how we understand model behavior and determine what to improve. Therefore, in this chapter, we explore the reasons behind using the AP score in HMC literature, the challenges associated with this metric, and the need for additional metrics to ensure a more comprehensive and well-rounded evaluation.

5.1 AP Score as a Common Evaluation Metric in HMC Research

In MLC, models output a confidence score for each label. Therefore, each label for a sample is predicted with a probability between 0 and 1, for example, 0.2 or 0.8. Deciding whether a label should be considered present or absent requires setting a threshold, typically 0.5. The challenge arises because there is no single threshold that works well across all labels. Some labels, especially rare ones, may require a lower threshold to avoid missing true positives. In contrast, others, especially those who are frequent, may need a higher threshold to avoid too many false positives. For this reason, Bi Kwok [3] emphasized that a fixed threshold is often suboptimal and varies significantly depending on the dataset and label distribution. Because of this variability, AP became popular in the HMC literature since it evaluates performance across all prediction scores, also referred to as thresholds, thereby bypassing the need to pick one threshold. We can interpret the AUPRC we plot by using all unique prediction scores as thresholds, as the AP score.

5.2 Limitations of AP Score: An Overly Optimistic Measure, and Introducing the Binary AP Score

Even with a clear understanding of why AP has become a preferred metric in HML tasks, which is due to its ability to evaluate performance without committing to a single threshold, it comes with certain trade-offs. In particular, we have observed that AP may sometimes give an overly optimistic view of model performance when calculated from raw sigmoidal outputs. To demonstrate this effect, we perform a separate controlled experiment that highlights the optimistic nature of AP and its implications on performance evaluation.

We simulated a multi-label dataset, where $y = [y_0,\ y_1]$ is the ground truth annotation for a truth array of batch size two, containing two samples y_0 and y_1, such that:

$$y_0 = [0, 0, 1, 1]; \quad y_1 = [0, 0, 0, 1].$$

For each sample, the model predicts the following sigmoid outputs $\hat{y}$:

$$\hat{y}_0 = [0.1, 0.7, 0.75, 0.8]; \quad \hat{y}_1 = [0.3, 0.6, 0.2, 0.8].$$

These are the continuous predicted scores, that is, the confidence for each label, generated randomly by adding noise without a model. The higher the score, the more confident the model is in predicting a label as positive. We can see that the closest correct label predictions for y_0 are the third and fourth labels, and for y_1, it is only the fourth label. After thresholding at 0.5 to the sigmoid outputs, we get the following binarized output $\tilde{y}$:

$$\tilde{y}_0 = \hat{y}_0 \geq 0.5 = [0, 1, 1, 1]; \quad \tilde{y}_1 = \hat{y}_1 \geq 0.5 = [0, 1, 0, 1].$$

For y_0, the model predicts the second, third, and fourth labels as positive or 1, and for y_1, it is the second and fourth labels that are predicted as positive or 1. In both samples, the model predicts the second label as positive with high confidence, that is, 0.7 for y_0 and 0.6 for y_1, despite the ground truth being negative, that is, 0, for that label in both samples. This is a false positive and is not penalized by the AP metric when applied to sigmoid outputs.

We calculated AP scores predicted using Algorithm 6 for $\hat{y}$ and $\tilde{y}$. The AP score we get for $\hat{y}$ is 1.0, whereas the score we get for $\tilde{y}$ is 0.6.

Algorithm 6 Average Precision Computation

This function is written explicitly for clarity and reproduces the results of scikit-learn's average precision calculation with the averaging method set to "micro".

function AVERAGEPRECISION($y_{\text{true}}, y_{\text{pred}}$)
    flatPredicted $\leftarrow$ Flatten(y_{pred})
    uniquePredicted $\leftarrow$ Unique(*flatPredicted*)
    thresholds $\leftarrow$ SortDescending(*uniquePredicted*)
    precisions $\leftarrow$ list()
    recalls $\leftarrow$ list(0)
    for all threshold $\in$ thresholds **do**
        $\gamma_{\text{pred}} \leftarrow$ {if (pred $\geq$ threshold) then 1 else 0 for each pred in y_{pred}}
        $tp \leftarrow$ Sum({1 for $(b_{\text{pred}}, b_{\text{true}})$ in $(\gamma_{\text{pred}}, y_{\text{true}})$ if $b_{\text{pred}} = 1$ and $b_{\text{true}} = 1$})
        $fp \leftarrow$ Sum({1 for $(b_{\text{pred}}, b_{\text{true}})$ in $(\gamma_{\text{pred}}, y_{\text{true}})$ if $b_{\text{pred}} = 1$ and $b_{\text{true}} = 0$})
        $fn \leftarrow$ Sum({1 for $(b_{\text{pred}}, b_{\text{true}})$ in $(\gamma_{\text{pred}}, y_{\text{true}})$ if $b_{\text{pred}} = 0$ and $b_{\text{true}} = 1$})
        precision $\leftarrow tp/(tp + fp)$
        recall $\leftarrow tp/(tp + fn)$
        Append(precisions, precision)
        Append(recalls, recall)
    end for
    recallRange $\leftarrow$ list()
    for $i \leftarrow 1$ **to** Length(recalls) **do**
        Append(recallRange, recalls[i] $-$ recalls[$i - 1$])
    end for
    $ap \leftarrow$ Sum({$p \cdot r$ for (p, r) in (precisions, recallRange)})
    return ap
end function

We can see that the prediction values are very confident when we treat the sigmoid outputs as probabilities, even when they are wrong compared to the actual labels in $y = [y_0, y_1]$. Despite the prediction being wrong, the score for the sigmoid output shows a perfect result, namely an AP score of 1.0, which makes the model appear better than it actually is.

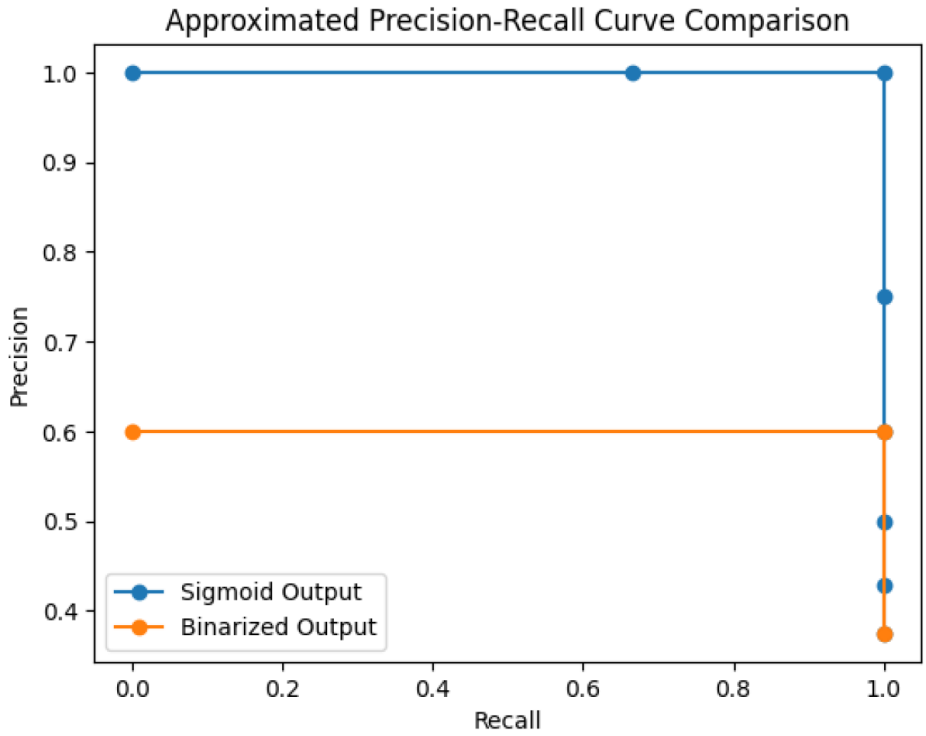


Figure 5.1: Comparison Between Approximated PR Curves. In our example, the precisions and recalls obtained in the intermediate steps of Algorithm 6 produce the displayed PR curves. In this particular example, both curves are comprised of only a single rectangle with non-zero width. Generally, this observation does not hold true.

Another way we can understand the results is by calculating the rectangular approximation of the PR curves for $\hat{y}$ and $\tilde{y}$. The PR curves are calculated using the precision and recall values at each sigmoidal output score for $\hat{y}$ and $\tilde{y}$. The precision and recall values we get for each sigmoidal output are as follows:

For $\hat{y}$:

$$\text{Precision} = [1.00, 1.00, 1.00, 0.75, 0.60, 0.50, 0.43, 0.38]$$

$$\text{Recall} = [0.00, 0.67, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00]$$

For $\tilde{y}$:

$$\text{Precision} = [0.6, 0.6, 0.38]$$

$$\text{Recall} = [0.00, 1.00, 1.00]$$

When these values are plotted, they produce the precision-recall curves shown in Figure 5.1.

From these results, we can see that applying AP to raw sigmoidal outputs can lead to an overly optimistic view of performance. Only the binarized output, after thresholding the sigmoidal outputs at 0.5, gives a score that fairly penalizes the model for its mistakes.

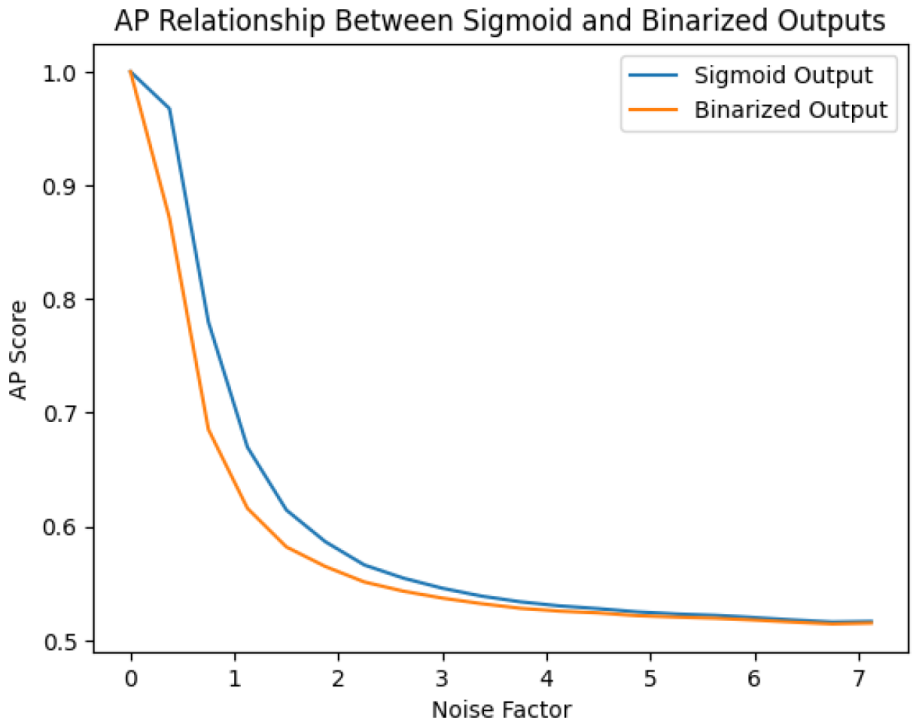


Figure 5.2: Comparison Between Calculated AP Scores with Increasing Noise. The plotted displays the output of Algorithm 7. In this experiment, as Gaussian noise is supplied to the system to mimic the descent of a perfect model into a random one, both the sigmoid and the binarized output AP scores approach 0.5, as expected. Most notably, during this entire process, the sigmoid AP scores are lower-bounded by the binarized AP scores outside of small fluctuations near convergence.

Also, we might wonder if the AP score for the binarized output is always lower than the AP score for the sigmoid output. Interestingly, that is not always the case.

Algorithm 7 Noise Factor Experiment

This sampling process entails generating 100,000 synthetic annotations. The goal is to add increasing amounts of Gaussian noise to these annotations to mimic the predictions of a "learned model". As the amount added of noise approaches infinity, the predictions approach that of a random model. In this manner, we can simulate and compare the AP scores of sigmoid and binarized outputs from a spectrum ranging from the perfect model to a random model.

$nSamples \leftarrow 100000$
$totalIncrements \leftarrow 20$
$multiplier \leftarrow 7.5$
$noiseFactors \leftarrow \text{list}()$
$sigScores \leftarrow \text{list}()$
$binScores \leftarrow \text{list}()$
for $i \leftarrow 0$ to $totalIncrements$ **do**
    $noiseFactor \leftarrow i \cdot multiplier / totalIncrements$
    $annotations \leftarrow \textsc{RandomIntegers}(0, 1, (nSamples, 10))$
    $annotationsShape \leftarrow \textsc{Shape}(annotations)$
    $pred \leftarrow annotations + \textsc{RandomNormal}(0, noiseFactor, annotationsShape)$
    $sigPred \leftarrow \textsc{Clip}(pred, 0, 1)$
    $binPred \leftarrow sigPred > 0.5$
    $sigAP \leftarrow \textsc{AveragePrecision}(annotations,\ sigPred)$
    $binAP \leftarrow \textsc{AveragePrecision}(annotations,\ binPred)$
    $\textsc{Append}(noiseFactors,\ noiseFactor)$
    $\textsc{Append}(sigScores,\ sigAP)$
    $\textsc{Append}(binScores,\ binAP)$
end for

In rare situations, especially with learned models, this assumption does not hold. For example, if the predictions of a model are largely the mirror opposite of the ground truth, then the AP score of the binarized output may exceed that of the sigmoid output. But in general, when we are evaluating real or random models, it is almost always the case that the sigmoid output's AP scores are lower-bounded by the binarized output's AP scores. We show this result in Figure 5.2, using data generated from a sampling procedure explained in Algorithm 7.

5.3 Beyond AP: The Need for Additional Evaluation Metrics

While the AP-based metrics are suited in HML settings for evaluating model performance, it is worth questioning whether they alone are sufficient to provide a thorough and well-rounded assessment. Relying solely on a single metric may lead to an incomplete or even misleading interpretation of model performance. This position is reinforced by empirical observations and illustrative experiments, which demonstrate the limitations of AP in capturing the true nature of model performance, as discussed further.

To investigate this issue, we conduct a separate toy experiment. We generate a synthetic dataset of 1000 samples. Each sample is annotated using five categorical labels corresponding to five output logits from a model. For every sample, two indices are randomly selected with resampling to be annotated as positive. To simulate imperfect model predictions, uniform noise is added to the annotations, with noise drawn from the range [-0.5, 0.5] so that the predictions can be generated without a model. The intensity of this noise is scaled by a multiplicative factor k=1.5, and centered around a zero offset (c=0). Subsequently, we scale the number of nodes from 5 to 1,000 to demonstrate the decoherence for AP as a measure of model performance in large sparsely annotationed hierarchies, such as those seen in life sciences [1] [2] [42]. As might be expected, we find that regardless of k or the maximum number of indices we choose to be positive, as we increase the number of nodes in our dataset, similar effects as illustrated in Figure 5.3 are observed. While all metrics agree when the label space is small, discrepancies arise as the number of nodes increases. Initially, when only five nodes are present, metrics such as AP, Bin. AP, node-level F1, and recall all indicate high performance. This agreement is expected, as the added symmetric

noise has a probability below 50% of flipping any particular annotation, keeping the model's outputs relatively reliable.

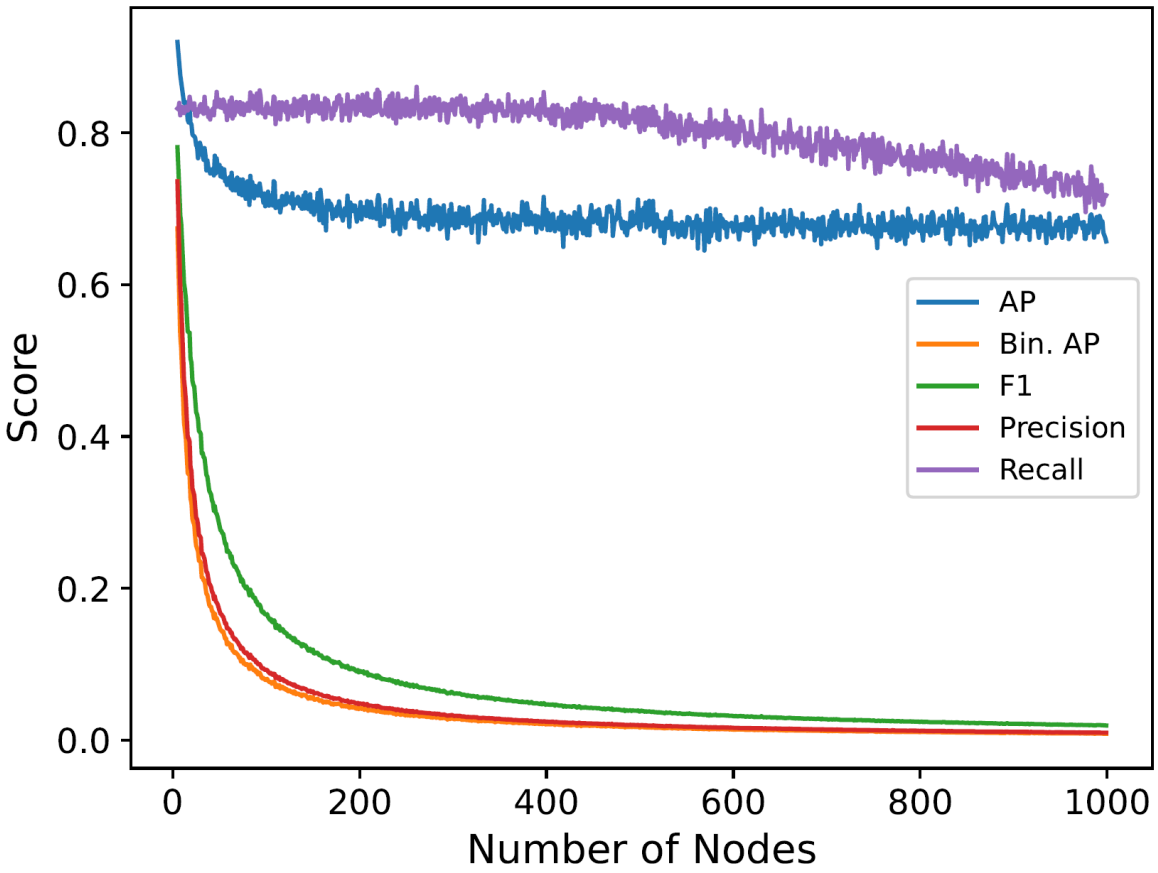


Figure 5.3: A Comparison of HML Metrics on Semi-Random Predictions Across Increasingly Larger Hierarchies. Each data point represents simulated scores for a noisy model where annotations and predictions have the number of dimensions equal to the number of nodes. All examined HML metrics range from 0 to 1, with 1 being the ideal score.

However, as the number of nodes increases, the metrics begin to diverge, as shown in Figure 5.3. While AP remains relatively high, often plateauing, other metrics such as Bin. AP and node F1 drop sharply, revealing a decline in model performance. The only exception is recall. The node-wise F1 score provides a clearer picture, showing that the number of false positives for each node increases as the number of nodes increases. Since the noise characteristics remain unchanged, the model continues to detect many true positives, thus maintaining its recall, only slowly dropping near the higher end. The Bin. AP and node-wise F1 show that the semi-random model

performs poorly across all nodes. The high scores AP assigns to our semi-random model seem to misrepresent its performance. We can imagine AP and Bin. AP as a type of agreement measure between prediction and annotation. There is no immediate way to transform these metrics into a portrayal of node-wise accuracies. On the other hand, the node-wise F1 and its component metrics provide an intuitive aggregate performance of a model for each node within a hierarchy. Given this demonstration, we advocate using multiple metrics to enhance performance interpretability, rather than using AP alone.

Chapter 6

Experimental Design

6.1 The Standard Weighting Formula

To formalize the concept of our node-weighting method, we considered using the standard weighting formula [5]. The standard weighting formula is, $w_i = \frac{N_{samples}}{N_{classes} \cdot n_i}$, where, w_i is the weight assigned to node i, $N_{samples}$ represents the total number of samples in the dataset, $N_{classes}$ is the total number of nodes in the hierarchy, and n_i is the number of positive annotations for node i, that is, how frequently node i appears in the dataset.

We can observe that as n_i increases, the corresponding weight decreases, and vice versa. This has the effect of assigning higher weights to rarer nodes, encouraging the model to focus more on them during training, while assigning lower weights to more frequent nodes, preventing them from dominating the learning process.

6.1.1 Node Frequency Count in a HML setting

In an HML setting, we can define $N_{samples}$ in two ways, that is, node-specific, where we count how often a particular node appears in the data and the other is an aggregated count, where we also include the counts of all descendant nodes in the total count for that node. For example, in EchinodermNet, when we count the total number of samples for the node 'Sea Urchins', we also include the sample count from all of its children in the hierarchy, such as 'Irregular Urchins' and 'Regular Urchins'. For our experiments, we use the aggregated count for $N_{samples}$ rather than the node-specific count which only considers samples labeled exclusively with that node and non of its child categories. This choice is motivated by the observation that aggregated counts generally led to improved mean performance across metrics, even though the improvement was not statistically significant. More importantly, using the aggregated count for $N_{samples}$ better reflects the hierarchical structure of the data, where annotations

naturally imply the presence of ancestor classes.

6.2 Positive and Negative Weightings

In the context of HML, each node from the hierarchy is either present, that is, annotated in a given sample, or absent, that is, not annotated. Because of this binary setup, assigning separate weights for such positive and negative cases becomes a practical way to address class imbalance more effectively during training. Therefore, the standard weighting formula can be applied separately to positive samples, where a node is present, a strategy called Positive Weighting, and to negative samples, where the node is absent, known as Negative Weighting. This approach ensures that each weight of the node reflects its annotation frequency, enabling the model to better handle imbalances arising from both positive and negative occurrences in the dataset.

To illustrate how we compute the positive and negative weights used in our experiments, we provide an example based on the zoology dataset. The dataset contains nodes 'Animal', 'Dog', and 'Beagle' in a hierarchy. 'Animal' contains 1000 positive and 5000 negative samples. 'Dog' contains 200 positive and 5800 negative samples. 'Beagle' contains 50 positive and 5950 negative samples. This is mentioned as within-label imbalance in the survey paper [51]. We can calculate separate positive and negative weights for each node. The positive and negative weights for a given node i are calculated using formulas in Equation 6.1 and Equation 6.2, respectively. In these equations, w_i^+ denotes the positive weight for node i, and w_i^- denotes the negative weight for node i, n_i^+ denotes the positive frequency for node i, n_i^- denotes the negative frequency for node i, and $N_{classes}$ equals 2.

$$w_i^+ = \frac{N_{samples}}{N_{classes} \cdot n_i^+} \tag{6.1}$$

$$w_i^- = \frac{N_{samples}}{N_{classes} \cdot n_i^-} \tag{6.2}$$

So when we calculate the positive and negative weights, for the zoology dataset, it would be as follows:

- **Animal:** $w_{animal}^+ = \frac{6000}{2 \cdot 1000} = 3$,
  $w_{animal}^- = \frac{6000}{2 \cdot 5000} = 0.6$

- **Dog:** $w^{+}_{dog} = \frac{6000}{2 \cdot 200} = 15$,
  $w^{-}_{dog} = \frac{6000}{2 \cdot 5800} = 0.52$

- **Beagle:** $w^{+}_{beagle} = \frac{6000}{2 \cdot 50} = 60$,
  $w^{-}_{beagle} = \frac{6000}{2 \cdot 5950} \approx 0.5$

In this case, the weightings we obtain would reflect the imbalance between the 'binary' instances of each node as $N_{classes} = 2$. If we were to address this binary imbalance, we could run into some issues. For example, nodes belonging to even the majority class can have significantly more negative samples than positive ones, in binary node weighting. This imbalance can lead to a distribution of weights, where positive samples receive weights greater than 1 and negative samples receive weights less than 1. As a result, minority nodes with only a few annotated positive samples may be assigned extremely large positive weights.

We may think of applying the stratification technique for multilabel data that tackles the problem of uneven label frequencies across cross-validation folds by iteratively assigning samples so that each fold mirrors the original distribution of every label [43]. It ensures balanced, representative splits even when samples carry multiple labels, something standard stratified techniques can't achieve. However, these iterative stratification techniques only handle flat multi-label datasets and do not account for the hierarchical relationships between labels. Therefore, we cannot ensure the parent–child label structures to remain balanced across folds in hierarchical multi-label settings.

When we consider class-wise imbalance, where $N_{classes}$ is the total number of nodes in a dataset, we would define our weights by comparing between different nodes. For example, we would compute weights by comparing the number of samples of 'Animal' node to the number of samples of the 'Dog' node in our datasets. In such a case, we have potential to obtain a more mixed range of positive weights. However, negative weights in this situation can remain high, particularly even for majority nodes that continue to receive many negative annotations. Another issue arises with node frequencies in a multi-modal distribution, where groups of majority nodes often cluster at similar counts. If this is the case, it is possible to obtain a set of weights that essentially ignores theses majority nodes when learning. This leads to the model

never learning to predict any node and perform poorly. We are therefore motivated to process our weights further before applying them in learning.

6.3 Normalization of Positive Weights in Hierarchical Multilabel Node-Wise Weighting

One approach to managing extreme weights in both binary and class-wise weighting cases is to normalize them, ensuring they remain within a controlled range and contribute to a more stable and balanced learning process. We could also manage extreme weights by applying a hard floor or ceiling to the weights, but that could obscure subtle frequency differences between nodes. Therefore, we prefer to choose normalization, as it preserves relative differences while bounding weights within a manageable range.

$$\omega_i = \omega_0 + \frac{w_i - w_{min}}{w_{max} - w_{min}} \tag{6.3}$$

The normalization formula is shown in Equation 6.3 where, ω_i is the normalized weight for node i, w_i is the original computed weight for node i from the standard weighting formula, w_{min} is the minimum weight in the dataset, w_{max} is the maximum weight in the dataset and ω_0 is a parameter that sets the minimum value for the normalized weights. The normalization formula ensures that all normalized weights fall within a specific range, starting from ω_0 and scaling up based on the relative spread of the original weights. It helps ensure that weights for rare nodes do not become extremely large and prevents frequent nodes from getting extremely small weights.

As a side note, we found that negative weights did not contribute to improved learning, regardless of whether they were used independently, alongside positive weights, or after additional processing. Therefore, our analysis from here on focuses exclusively on positive weights. Additionally, in our experimentations, class-wise weighting significantly outperformed binary interpretations, due to which we focus only on class-wise weightings further on.

6.3.1 Role of ω_0

The minimum bound ω_0 serves as the baseline weight, representing the minimum normalized value that all nodes get after normalization. By adjusting ω_0, we control the range of variation in the weights. It ensures that even the majority nodes retain some influence during training. Without ω_0, that is, $\omega_0 = 0$, the node with the smallest original weight of w_{min} would receive a normalized weight of zero as w_i would equal w_{min} in such a case, $\omega_i = 0 + \frac{w_{min} - w_{min}}{w_{max} - w_{min}} = 0$.

A value of zero for ω_0 implies that the model will completely ignore the corresponding node during training, resulting in no learning. This is undesirable, as even the majority nodes, should contribute to the learning process. Therefore, ω_0 gives control over how much one wants the model to focus on rare nodes versus frequent nodes. It helps fine-tune the balance between recall and precision, ensuring that no node is entirely ignored during training. The effects of varying ω_0 are summarized in Table 6.1.

Table 6.1: Effect of Varying Initial Weight Value ω_0. This table illustrates the effect of varying initial weight value ω_0 on model behavior and how it affects the precision and recall values.

HIGH ω_0 (e.g., 1.0)	**LOW ω_0 (e.g., 0.0)**
The model focuses more on frequent nodes	The model focuses more on rare nodes
Lower recall, lower precision	Higher recall, lower precision
Fewer false positives	More false positives

6.4 Scheduling ω_0 Decay

We experiment with various scheduling strategies where node-wise weighting, ω_i, is updated over time to shift emphasis away from the weighted loss toward the unweighted one in hopes to retain precision as much as possible as recall increased. The hypothesis is that early emphasis on rare nodes helps the model learn to recall them effectively, while a later shift to an unweighted objective allows it to retain precision. We aim to investigate whether scheduling decay can achieve this. In particular, we

want to determine which type of scheduling, epoch-based or batch-based, is most effective in achieving this balance.

6.4.1 Experimenting With Various Scheduling Techniques

In epoch-based scheduling, the node-wise weighting, ω_i is scheduled to transition toward an unweighted objective in later epochs. Specifically, every ω_i is updated progressively across epochs to approach 1. If a node starts with a low weight, it increases to approach 1; if it starts with a high weight, it decreases toward 1. This allows each node to gradually shift toward unweighted learning at its own pace, with ω_0 acting as the minimum allowable starting point. In the batch-based scheduling method we approach the unweighted objective every epoch.

We also experimented with four different scheduling techniques to gradually transition from weighted to unweighted learning, that is, linear, exponential, alternating, and mixed loss scheduling. Each of these methods attempts to gradually transition from weighted to unweighted learning. In linear scheduling, ω_i update steps are evenly spaced out to the number of training steps (N_{steps}) in each epoch. In exponential scheduling, ω_i decays exponentially. This implies that the shift from weighted to unweighted learning is slower at first but accelerates later in training. The exponential scheduler divides exponential steps as, $\Delta\omega_i = \frac{1-\omega_i}{(N_{\text{steps}}-1)^k}$. During each batch update, exponential updates are performed as, $\omega_{i,t+1} = \omega_{i,t} + t^k\Delta\omega_i$, where t represents the batch index in an epoch, and k is the exponential factor that determines how aggressively ω_i decays over time. In our experiments, we set $k = 3$. The alternating scheduling switches between weighted and unweighted learning in every batch. The Mixed loss approach, instead of transitioning or switching between objectives, combines weighted and unweighted learning into a single function. This is done by introducing a parameter λ. It imposes a λ and 1 - λ relationship between the two objectives. For example, with $\lambda = 0.7$, the model would assign 70% to the unweighted objective and 30% to the weighted objective.

The experimental results of these scheduling strategies, along with an evaluation of their impact on model performance, are presented in Chapter 7.

6.5 Combining LPROS Resampling With Node-Wise Weighting

Later, we show that LPROS achieves similar precision among the other evaluated HML approaches, while node-weighting contributes to improved recall. Their combination could yield better overall results than either technique alone. To explore this possibility, we conduct experiments that combine existing resampling techniques such as LPROS with node-weighting to assess their joint impact on model performance.

We chose just the LPROS resampling technique to combine with our node-weighting approach because it performed the best among other multilabel resampling techniques to tackle imbalance [37]. Unlike hierarchical resampling methods such as HROS-PD [40] that can only be applied to HMDs that are tree DAG hierarhcies, LPROS can be applied to MLDs that are converted from HMDs that are either tree or non-tree DAG hierarchies.

6.6 Experiments on the EchinodermNet Dataset

The previously discussed experimental setups are targeted at datasets such as FUN and GO. To evaluate the generalizability of our method, we also assess its performance on the EchinodermNet image dataset. By applying node-wise weighting to the EchinodermNet dataset, we want to examine if similar results are observed with image-based datasets, that is, whether node-wise weighting helps models improve recall and detect underrepresented nodes. We also want to understand whether the effects differ when using randomly initialized models trained from scratch versus pretrained models leveraging prior knowledge from BenthicNet dataset. We tested how node-wise weighting affects hierarchical image classification using two state-of-the-art deep learning model architectures, ResNet-50 and Vision Transformer (ViT-B), with a patch size of 16 [19]. As ViT is known for its ability to capture global dependencies across an image, it might help with hierarchical classification. These experiments help to highlight the effects of node weighting and are motivated in part by the continued relevance of CNNs in ocean sciences [21] [30] [58].

Chapter 7

Experimental Results

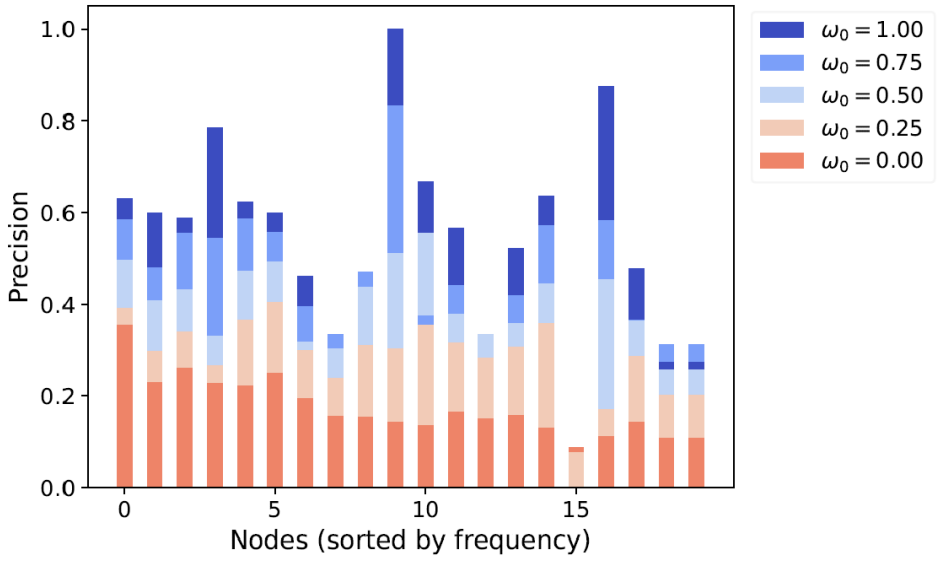
7.1 FUN and GO

7.1.1 Evaluation of Various HML Imbalance Handling Methods

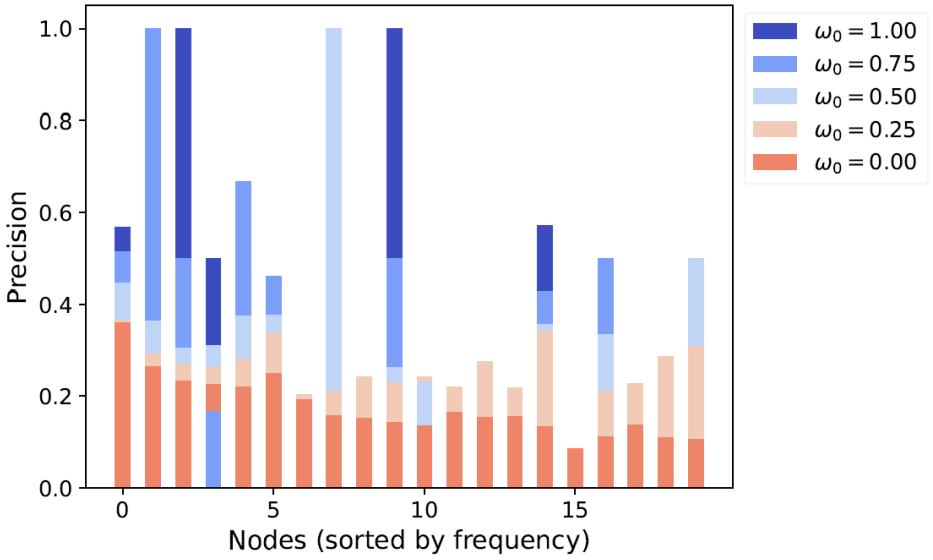
The performance of various methods addressing class imbalance on the FUN datasets, including no weighting represented as 'NONE', converting HMD to MLD to apply resampling using the LPROS technique, HROS-PD, and node-wise weighting with different ω_0 values, is shown in Table 7.1. Their performance is evaluated using several metrics such as AP, Binarized AP, node-wise F1, node-wise Precision, node-wise Recall, with top significant results highlighted in bold. The statistical significance is determined using a threshold of two standard deviations from the mean. It is seen from the table that while the AP score is not significantly affected by the imbalance method, significant improvements are observed in Bin. AP, F1 score, and especially recall from node-wise weighting, with recall improving up to a factor of six over the control, on certain datasets with $\omega_0 = 0.25$. Therefore, node-wise weighting helps the model better identify minority and deep nodes within the hierarchy, which are typically underrepresented and harder to predict. However, in a few datasets, increase in recall comes at the expense of trade in precision, though the difference is within two standard deviations indicating minor variations. Interestingly, in certain datasets such as Derisi, Gasch-2, and SPO, node-wise weighting not only boosts recall but also improves precision. We generally observe that as we decrease ω_0 from starting value of 1 down to 0, the precision drops while recall increases significantly. This results in an increase in F1 score since recall improves faster than the drop in precision until eventually when the loss of precision overpowers further lesser gains in recall. For most datasets, we found an optimal balance near $\omega_0 = 0.25$, where we can maximize Bin. AP and F1 such that either no or little precision is lost. Figure 7.1 illustrates the effect of decreasing ω_0 on precision.

Table 7.1: HML Method Comparisons (FUN). Different strategies in addressing HML imbalance are compared for the FUN datasets: no method used (None), HROS-PD, and node weighting using ω_0 values of 0.25 and 0.50. The metrics examined are average precision (AP), binarized average precision (Bin. AP), and node-wise F1, precision, and recall.

DATASET	METHOD	AP	Bin. AP	F1	PRECISION	RECALL
CELLCYCLE (FUN)	NONE	25.68±0.13	6.26±0.11	2.61±0.09	9.81±0.70	1.74±0.06
	LPROS	24.81±0.25	6.42±0.10	2.98±0.18	9.70±0.80	2.13±0.19
	HROS-PD	25.57±0.16	6.18±0.12	2.70±0.25	9.57±0.90	1.88±0.22
	$\omega_0 = 0.25$	25.45±0.10	**10.21±0.08**	**4.85±0.11**	9.13±0.36	**4.59±0.11**
	$\omega_0 = 0.50$	25.69±0.12	8.20±0.08	3.54±0.08	9.34±0.36	2.68±0.07
DERISI (FUN)	NONE	19.58±0.10	2.67±0.08	0.46±0.03	1.75±0.21	0.34±0.03
	LPROS	19.18±0.10	2.66±0.08	0.50±0.02	1.96±0.16	0.35±0.02
	HROS-PD	19.39±0.11	2.70±0.09	0.44±0.02	1.82±0.41	0.33±0.01
	$\omega_0 = 0.25$	19.61±0.09	**7.55±0.07**	**1.82±0.05**	**2.88±0.20**	**2.16±0.07**
	$\omega_0 = 0.50$	19.66±0.09	4.34±0.08	0.90±0.06	2.31±0.33	0.74±0.04
EISEN (FUN)	NONE	30.70±0.07	9.36±0.14	4.82±0.25	14.03±0.62	3.29±0.20
	LPROS	29.69±0.14	9.32±0.10	5.87±0.32	15.15±0.78	4.11±0.30
	HROS-PD	30.70±0.07	9.36±0.14	4.82±0.25	14.03±0.62	3.29±0.20
	$\omega_0 = 0.25$	30.51±0.11	**12.64±0.10**	**7.16±0.16**	11.27±0.37	**6.69±0.12**
	$\omega_0 = 0.50$	30.73±0.07	10.91±0.17	5.92±0.21	12.79±0.54	4.55±0.15
EXPR (FUN)	NONE	30.13±0.23	10.11±0.12	5.84±0.26	13.49±0.19	4.26±0.24
	LPROS	28.45±0.13	9.90±0.17	7.16±0.26	14.84±0.73	5.40±0.22
	HROS-PD	30.03±0.22	10.18±0.15	6.31±0.40	14.17±0.44	4.71±0.44
	$\omega_0 = 0.25$	30.30±0.09	**12.50±0.12**	**8.50±0.30**	11.92±0.35	**8.10±0.42**
	$\omega_0 = 0.50$	30.40±0.09	11.47±0.07	6.95±0.33	12.86±0.46	5.59±0.34
GASCH-1 (FUN)	NONE	28.44±0.24	8.27±0.16	4.98±0.52	12.56±1.57	3.64±0.37
	LPROS	26.71±0.16	8.15±0.27	5.64±0.24	13.10±0.83	4.16±0.16
	HROS-PD	28.28±0.13	8.27±0.17	5.63±0.16	13.36±0.48	4.13±0.10
	$\omega_0 = 0.25$	28.57±0.22	**11.64±0.20**	**7.42±0.36**	11.08±0.50	**6.94±0.36**
	$\omega_0 = 0.50$	28.58±0.22	10.19±0.21	6.03±0.36	12.19±1.02	4.77±0.29
GASCH-2 (FUN)	NONE	25.83±0.06	5.47±0.13	2.61±0.12	8.72±0.61	1.80±0.08
	LPROS	24.97±0.10	5.49±0.15	2.86±0.10	9.92±0.50	1.98±0.09
	HROS-PD	25.54±0.14	5.38±0.13	2.58±0.10	8.55±0.22	1.79±0.06
	$\omega_0 = 0.25$	25.51±0.06	**10.24±0.08**	**4.92±0.16**	9.19±0.33	**4.62±0.09**
	$\omega_0 = 0.50$	25.76±0.06	7.66±0.20	3.47±0.12	9.21±0.59	2.63±0.11
SEQ (FUN)	NONE	29.24±0.15	9.95±0.14	4.14±0.19	11.85±0.69	2.91±0.15
	LPROS	28.56±0.27	10.44±0.16	5.38±0.16	**14.31±0.44**	3.84±0.12
	HROS-PD	29.40±0.26	10.29±0.11	4.78±0.20	13.29±0.36	3.43±0.18
	$\omega_0 = 0.25$	28.91±0.20	**12.13±0.15**	**6.60±0.12**	10.57±0.61	**6.29±0.20**
	$\omega_0 = 0.50$	29.17±0.16	11.12±0.20	5.29±0.11	11.83±0.63	4.18±0.15
SPO (FUN)	NONE	21.54±0.07	3.39±0.11	0.61±0.04	3.03±0.17	0.43±0.02
	LPROS	21.17±0.12	3.53±0.08	0.80±0.04	3.20±0.18	0.54±0.03
	HROS-PD	21.38±0.06	3.37±0.10	0.60±0.04	3.09±0.19	0.42±0.02
	$\omega_0 = 0.25$	21.21±0.09	**7.82±0.05**	**1.94±0.04**	3.21±0.05	**2.17±0.02**
	$\omega_0 = 0.50$	21.42±0.08	4.84±0.07	1.11±0.04	2.98±0.11	0.87±0.03



(a) Cellcycle (FUN)



(b) Derisi (FUN)

Figure 7.1: Comparison of the Effects of Varying ω_0 Across Datasets. Plotted are node-wise precision scores across the top-20 most frequent nodes in their respective datasets. Except for a minor switch between the 13th and 14th nodes, Cellcycle (FUN) and Derisi (FUN) share identical ranks for their top-20.

We can see from Figure 7.1 that lowering ω_0 appears to increase the node-wise precision for previously unpredicted nodes, whereas decrease the precision for previously predicted nodes. Table 7.1 shows higher values for these datasets because larger ω_0 values, or no weighting, cause recall to drop so significantly that the model

stops making predictions for some nodes, resulting in a score of zero and lowering the overall average. It may seem surprising that smaller positive weights for majority nodes should result in an increase in recall and lower precision. We would expect assigning lower priority to positive annotations for a node typically reduces the model's tendency to predict positively, except in the most certain cases, thereby decreasing recall and increasing precision. The reason could be due to the hierarchical constraint, where the presence of a descendant node implies the presence of its ancestor node as well. Thus, the ancestor nodes, which are the majority nodes, are still indirectly predicted through their descendants due to this hierarchical structure. Since the model does not learn the majority nodes directly, its recall increases at the cost of precision. Therefore, the more we lower ω_0, the more we encourage indirect node learning in such a manner.

Table 7.2: Overview of FUN and GO Datasets Before and After Oversampling. This table illustrates the sample counts of FUN and GO datasets before and after oversampling using HROS-PD. The algorithm performs poorly on GO datasets due to their non-tree DAG structure, resulting in only a 3% increase in samples. In contrast, the FUN datasets exhibits a 10% increase, aligning with our predefined resampling specifications.

DATASET	**SAMPLE COUNT**	**SAMPLE COUNT POST RESAMPLING**	**% OF LABEL INCREASE**	**ORIGINAL HMeanIR**
cellcycle_FUN	1628	2120	9.35	246.8
derisi_FUN	1608	2104	9.5	247.35
eisen_FUN	1058	1451	10.85	184.62
expr_FUN	1639	2131	9.31	247.4
gasch1_FUN	1634	2126	9.33	247.1
gasch2_FUN	1639	2131	9.31	247.04
seq_FUN	1701	2193	9.08	244.96
spo_FUN	1600	2096	9.57	248.1
cellcycle_GO	1625	2849	3.26	13069.2
derisi_GO	1605	2834	3.31	12989.69
eisen_GO	1055	1687	2.26	10509.1
expr_GO	1636	2862	3.25	13089.07
gasch1_GO	1631	2858	3.26	13074.11
gasch2_GO	1636	2862	3.25	13089.07
seq_GO	1692	2922	3.17	13432.09
spo_GO	1597	2831	3.34	12984.18

Table 7.3: HML Method Comparisons (GO). Results presented are for the GO datasets. The following methods are reported: no method used (None), HROS-PD, and node weighting using ω_0 values of 0.25 and 0.50. The maximum mean results per dataset and metric across five trials with different seeds, considering statistical significance, are highlighted in bold. The metrics examined are average precision (AP), binarized average precision (Bin. AP), and node-wise F1, precision, and recall.

DATASET	METHOD	AP	Bin. AP	F1	PRECISION	RECALL
CELLCYCLE (GO)	NONE	41.34±0.10	16.31±0.46	1.16±0.07	3.23±0.19	0.86±0.05
	LPROS	40.70±0.14	16.45±0.44	1.38±0.05	**4.05±0.13**	1.01±0.04
	HROS-PD	40.29±0.11	11.37±0.60	0.59±0.06	2.26±0.31	0.43±0.04
	$\omega_0 = 0.25$	40.93±0.08	**19.84±0.09**	**1.94±0.07**	3.22±0.26	**1.86±0.05**
	$\omega_0 = 0.50$	41.24±0.12	19.49±0.14	1.47±0.04	3.29±0.14	1.23±0.04
DERISI (GO)	NONE	37.01±0.06	12.88±0.36	0.34±0.01	0.69±0.12	0.30±0.01
	LPROS	37.02±0.07	12.77±0.78	0.33±0.03	0.63±0.07	0.28±0.03
	HROS-PD	36.38±0.04	8.79±0.39	0.19±0.02	0.55±0.05	0.15±0.02
	$\omega_0 = 0.25$	36.86±0.05	17.38±0.08	**0.77±0.05**	0.97±0.22	**0.95±0.04**
	$\omega_0 = 0.50$	36.95±0.06	17.22±0.08	0.53±0.02	0.78±0.13	0.57±0.02
EISEN (GO)	NONE	45.34±0.15	19.57±0.61	1.90±0.18	4.50±0.32	1.45±0.15
	LPROS	44.70±0.15	19.25±0.10	2.15±0.10	4.90±0.34	1.62±0.07
	HROS-PD	44.66±0.16	16.19±0.51	1.41±0.09	4.17±0.19	1.02±0.07
	$\omega_0 = 0.25$	44.75±0.14	22.47±0.13	**3.17±0.07**	4.78±0.11	**3.04±0.09**
	$\omega_0 = 0.50$	45.14±0.14	22.38±0.16	2.40±0.22	4.40±0.36	2.04±0.18
EXPR (GO)	NONE	38.13±0.25	16.93±0.25	3.80±0.17	7.41±0.37	2.98±0.20
	LPROS	37.47±0.30	16.48±0.25	3.75±0.05	7.31±0.24	2.93±0.07
	HROS-PD	38.89±0.31	15.15±0.70	2.35±0.14	5.94±0.63	1.69±0.11
	$\omega_0 = 0.25$	**40.41±0.43**	**19.37±0.28**	**4.61±0.10**	6.87±0.15	**4.13±0.13**
	$\omega_0 = 0.50$	39.21±0.18	18.11±0.23	4.31±0.09	7.42±0.20	3.56±0.12
GASCH1 (GO)	NONE	43.61±0.12	17.79±0.25	1.52±0.03	3.08±0.20	1.19±0.02
	LPROS	43.27±0.06	18.04±0.44	1.97±0.07	4.01±0.17	1.56±0.05
	HROS-PD	42.48±0.19	12.96±0.51	0.86±0.06	2.38±0.22	0.63±0.05
	$\omega_0 = 0.25$	43.40±0.11	**21.48±0.09**	**2.66±0.08**	3.98±0.25	**2.61±0.05**
	$\omega_0 = 0.50$	43.57±0.11	21.21±0.14	2.06±0.06	3.53±0.20	1.80±0.04
GASCH2 (GO)	NONE	41.56±0.10	16.43±0.49	1.19±0.02	3.52±0.10	0.86±0.02
	LPROS	41.45±0.12	16.54±0.48	1.30±0.04	**3.80±0.03**	0.95±0.04
	HROS-PD	40.34±0.10	11.10±0.57	0.49±0.03	1.57±0.10	0.36±0.02
	$\omega_0 = 0.25$	41.13±0.06	20.03±0.12	**1.84±0.09**	3.21±0.26	**1.84±0.08**
	$\omega_0 = 0.50$	41.40±0.07	19.82±0.10	1.42±0.06	3.42±0.30	1.20±0.05
SEQ (GO)	NONE	38.60±0.28	17.77±0.38	3.94±0.25	8.96±0.40	2.91±0.19
	LPROS	37.30±0.81	16.74±0.59	4.07±0.14	9.34±0.38	3.01±0.10
	HROS-PD	39.63±0.33	15.17±0.81	2.46±0.21	6.84±0.37	1.72±0.17
	$\omega_0 = 0.25$	39.90±0.32	**19.69±0.27**	4.46±0.33	7.93±0.66	3.65±0.26
	$\omega_0 = 0.50$	39.22±0.44	18.95±0.37	4.29±0.21	8.63±0.23	3.31±0.19
SPO (GO)	NONE	38.19±0.05	13.65±0.56	0.68±0.10	1.85±0.33	0.54±0.07
	LPROS	37.94±0.06	13.51±0.27	0.78±0.09	2.25±0.42	0.59±0.05
	HROS-PD	36.98±0.11	9.15±0.32	0.31±0.05	0.94±0.37	0.25±0.03
	$\omega_0 = 0.25$	37.88±0.07	18.12±0.04	**1.26±0.02**	2.29±0.24	**1.29±0.01**
	$\omega_0 = 0.50$	38.09±0.05	17.95±0.20	1.01±0.02	2.49±0.11	0.89±0.02

Table 7.3 presents the evaluation results on the GO datasets, covering the same metrics and methods designed to address the class imbalance problem presented for the FUN datasets. The overall performance trend is similar to what we observe in Table 7.1, though we see a notable drop in HROS-PD performance. It is expected

because HROS-PD is not designed for non-tree DAG hierarchies [40]. The evidence that HROS-PD is not designed for the GO datasets is present in Table 7.2. The HROS-PD resampling technique is configured to yield approximately a 10% increase in the resampled dataset relative to the original dataset to mitigate class imbalance, but the GO datasets failed to achieve this target and could only achieve around 3% increase in the resampled dataset relative to the original dataset.

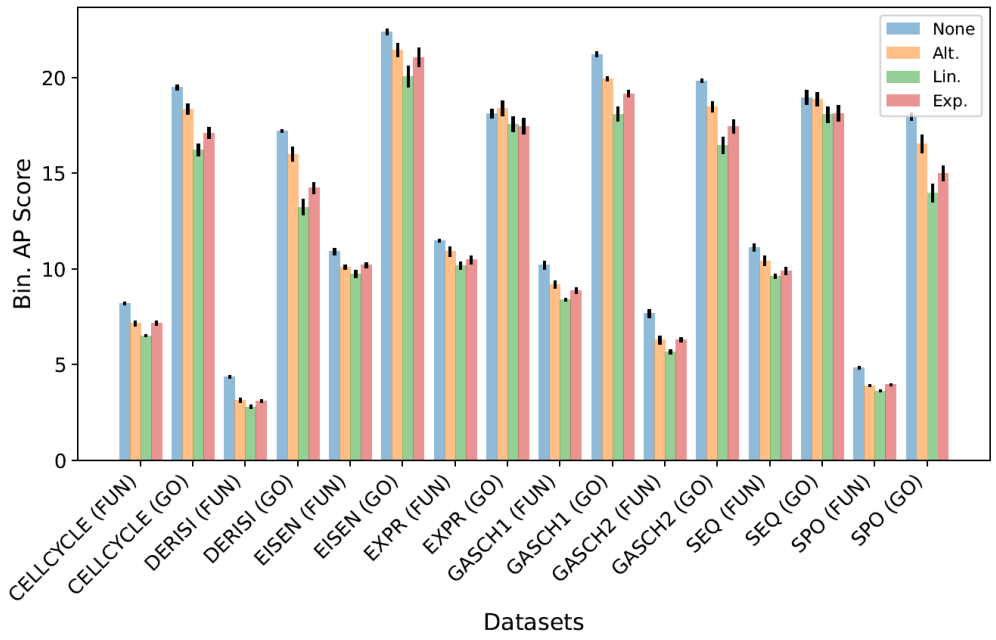
7.1.2 Results on Scheduling ω_0 Decay for Node-Wise Weighting

We observe from the results in Table 7.1 that node-wise weighting improves recall in HMC, but this improvement often comes at the cost of reduced precision. Given this challenge, we considered whether scheduling ω_i over time could help retain the recall benefits while retaining precision.

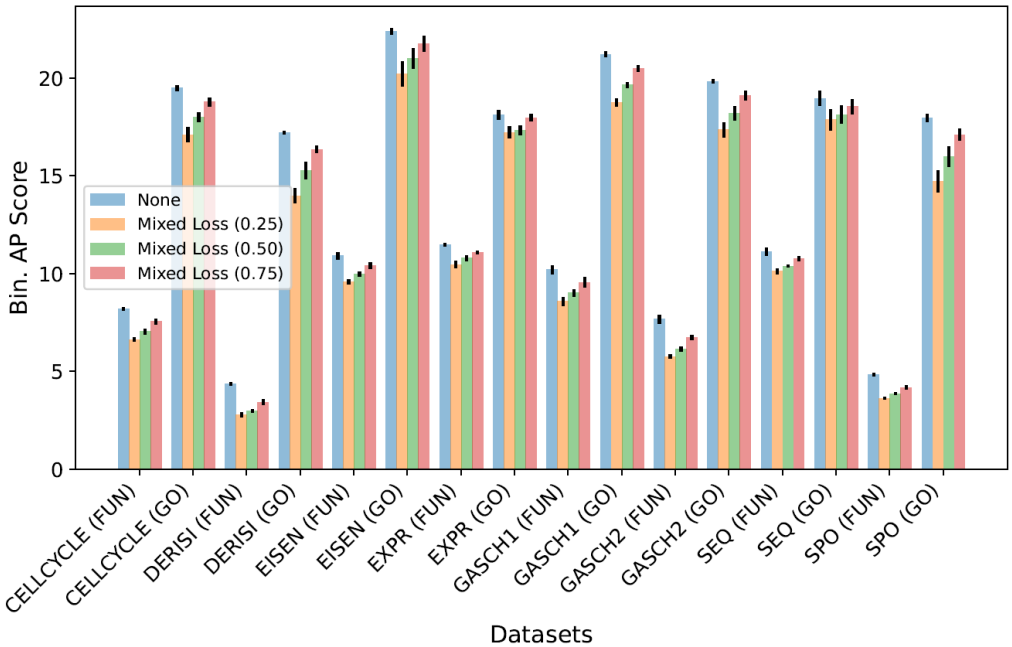
We observed that the batch-based scheduler outperformed the epoch-based scheduler significantly. While this result aligns with findings from lifelong learning and multi-task learning [41] [44], it seems to suggest that the loss landscapes for an HML imbalanced weighted objective and an unweighted objective do not easily transition from one to the other and are likely conflicting.

We also experimented with four different scheduling techniques to transition from weighted to unweighted learning, linear, exponential, alternating, and mixed loss scheduling. Each of these methods attempts to transition from weighted to unweighted learning, but none has proven more effective than a fully weighted loss approach, as shown in Figure 7.2.

While we analyzed performance across different datasets using the Bin. AP metric, the impact remained consistent across all evaluation metrics. The two objectives, weighted and unweighted, appear to have distinct and conflicting loss landscapes since merging these objectives did not lead to an improved performance, as we had hoped to gain.



(a) Schedulers



(b) λ-Mixed Loss

Figure 7.2: Combining Weighted and Unweighted Objectives. In Figure 7.2(a), we display the effect of our implemented schedulers on Bin. AP score across all FUN and GO datasets. In Figure 7.2(b), a mixed loss objective is used instead of a scheduler, with varying λ.

7.1.3 Combining Resampling Technique LPROS With Node-Wise Weighting

As scheduling approaches failed to deliver meaningful improvements, we chose to explore an alternative strategy, that is, integrating resampling techniques with node weighting to better manage the trade-off between precision and recall. This idea comes from observing that LPROS often achieves the highest precision compared to other approaches, as shown in Table 7.1, and node weighting helps enhance recall.

When we examined individual node performances across datasets, we observed that the LPROS impacted the nodes differently. Some nodes improved precision at the expense of recall, while others improved both precision and recall. The LPROS-trained model started predicting previously unpredicted nodes but stopped predicting some previously predicted nodes. We subsequently performed resampling, followed by the calculation of node weights.

Table 7.4: Combined LPROS and Node-Weighting F1 Performance. Results which exceed two σ's in comparison to the previously best performing method are bolded. Entries that fall within this threshold are depicted in gray.

DATASET	FUN		GO	
	COMBINED	ω_0 **= 0.25**	**COMBINED**	ω_0 **= 0.25**
CELLCYCLE	**5.22±0.16**	4.85±0.11	**2.33±0.18**	1.94±0.07
DERISI	**2.01±0.08**	1.82±0.05	**0.94±0.05**	0.77±0.05
EISEN	**8.28±0.29**	7.16±0.16	**3.52±0.12**	3.17±0.07
EXPR	**9.29±0.28**	8.50±0.30	4.40±0.11	4.61±0.10
GASCH1	8.05±0.16	7.42±0.36	**3.09±0.04**	2.66±0.08
GASCH2	**5.39±0.22**	4.92±0.16	**2.09±0.05**	1.84±0.09
SEQ	**7.57±0.35**	6.60±0.12	4.60±0.14	4.46±0.33
SPO	**2.24±0.03**	1.94±0.04	1.35±0.08	1.26±0.02

The node-wise weights with $\omega_0 = 0.25$, were computed on the data resampled using LPROS. The result of this experiment can be seen in Table 7.4. A significant improvement in the F1 Score can be seen across almost all datasets. It was initially expected that the combined method would improve F1 due to higher precision. However, recall also increased, indicating that the model was able to predict even more rare nodes. While AP scores did not show significant differences between methods, the mean Bin. AP score increased, but often within two standard deviations (σ),

indicating that it was not statistically significant.

7.2 EchinodermNet

The performance of different node weighting strategies on the EchinodermNet dataset, including no weighting ('None') and node-wise weighting with varying ω_0 values, is presented in Table 7.5. These methods are evaluated across both ResNet-50 and ViT-B architectures under randomly initialized state and using a frozen pre-trained encoder provided by BenthicNet [34]. The evaluation highlights the impact of these methods on performance metrics, with the best significant values emphasized in bold.

Table 7.5: Performance on EchinodermNet. Tested methods include no node weighting performed (None) and different ω_0 settings for node weighting.

METHODS	ResNet-50		ViT-B	
	INIT.	PRE-TRAINED	INIT.	PRE-TRAINED
None	18.19±2.10	62.61±1.12	44.46±0.47	63.83±1.23
$\omega_0 = 0.25$	**26.28±1.93**	57.76±2.35	46.18±1.64	63.88±1.45
$\omega_0 = 0.50$	20.45±0.89	59.39±1.42	44.04±0.94	64.25±1.81
$\omega_0 = 0.75$	16.23±1.66	60.07±1.51	42.48±1.03	60.99±3.30

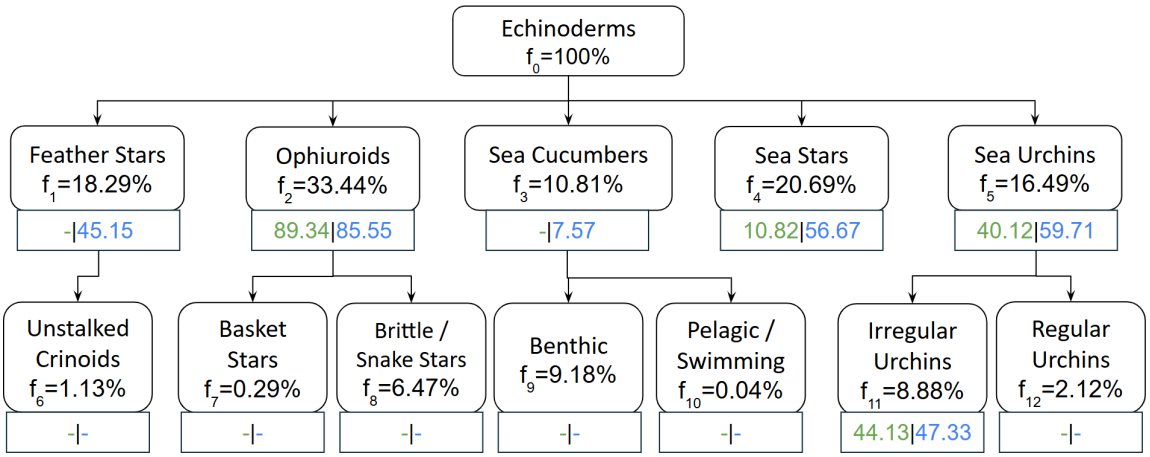


Figure 7.3: Randomly Initialized ResNet-50 Performance on Echinoderms. Node F1 scores are presented under each hierarchical node in the CATAMI echinoderm tree. On the left, in green, the performance without node weighting is stated, while the blue value on the right indicates node weighting with $\omega_0 = 0.25$ is applied. Entries marked with "-" imply that the model never made a prediction for the node in question and consequently attained an F1 score of zero.

For the pre-trained ResNet-50, the highest performance was achieved without applying node weighting, suggesting that node weighting offers limited benefit when the model has already learned informative features during pre-training. The node-wise performance of ResNet-50 across individual classes in the EchinodermNet dataset, comparing training from initialization with and without node-weighting on, is shown in Figure 7.3. It compares the results obtained from training ResNet-50 from scratch, both with and without node weighting, to assess the impact of the weighting strategy on model performance. The results show the same effects as seen in the FUN and GO datasets, that is, the recall increases, and we detect some previously unpredicted nodes. However, these improvements are not as strong when we use larger, more powerful models, especially after they have been pre-trained on data from the same domain. For example, in some cases, with the pre-trained ResNet-50, the boost in recall was insufficient to make up for the loss in precision.

Since the combination of LPROS and node weighting showed better results with the FUN and GO datasets, we examined the effect of this approach on the EchinodermNet dataset. LPROS resampling on EchinodermNet resulted in significantly worse performance under all conditions, regardless of whether node weighting was applied or not. We are unsure why LPROS performed poorly in this setting, which highlights the need for further investigation. Despite these issues, the mean scores in randomly initialized cases remain significant in cases where node-weighting is applied.

The ‘Sea stars’ node showed a significant improvement, that is, around a five times increase in F1 score after applying node-weighting, as shown in Figure 7.3. Nodes without children, like ‘Sea stars’, still benefited from node-weighting, even though it was originally thought to help hierarchical relationships. When we look at precision and recall, we see that precision only drops slightly, especially compared to nodes that have children. However, recall goes up by a factor of ten, which is notable given that this node has no descendants. Since it is not a minority node, its assigned weight is actually below one, that is, 0.46. This suggests that the value of the weight may be less important than its comparison to the weights of other nodes at the same level. In other words, giving slightly lower weights to sibling nodes that appear more frequently may help the model better identify positive samples of less frequently seen majority nodes, even if they have no child nodes. Some nodes, like ‘Regular Urchins’,

still got zero F1 score, indicating that the model never predicted them.

Chapter 8

Conclusion

This thesis presents a novel approach for addressing class imbalance in HMC tasks, particularly targeting datasets with class imbalance such as FUN, GO, and a subset of the BenthicNet dataset called EchinodermNet dataset. In HMDs, class imbalance is a prevalent issue due to the hierarchical dependencies between classes, where rare or deep nodes are underrepresented. Addressing imbalance while ensuring that the hierarchical constraints remain intact is important to preserve the true structure of the data.

We address the challenge of class imbalance by reimagining the problem at the node level, rather than focusing solely on sample distribution. By redefining imbalance in terms of individual node frequencies, our approach shifts attention from the overall sample distribution to the specific challenges that the model faces in learning the minority and deeper nodes. Furthermore, this work supports the use of evaluation metrics such as Binarized Average Precision in addition to node-wise precision, node-wise recall, node-wise F1 scores and Average Precision to evaluate HMC tasks. The empirical analysis reveals that Average Precision, when calculated on raw sigmoid outputs, often gives an overly optimistic view of performance, particularly in sparse label settings. In contrast, Bin. AP and node-level F1 scores offered more reliable insights into model behavior across both majority and minority nodes.

Experimental results across multiple datasets demonstrate that node-wise weighting, especially when combined with normalization strategies, leads to improvements in recall, enabling the model to detect nodes that were previously unpredicted and improving overall node-level performance without compromising hierarchical consistency. Although there is a drop in precision, the overall node-wise F1 score improves. Notably, $\omega_0 = 0.25$ consistently emerged as an optimal trade-off point between precision and recall, enhancing the model's ability to detect minority nodes while keeping

the decline in precision minimal. Additionally, the combination of node-wise weighting with the LPROS resampling technique yielded better results than using node-wise weighting alone, particularly in feature-based non-image datasets. Furthermore, we attempted to blend weighted and unweighted objectives through scheduling or mixed loss functions, but they did not outperform the pure node-weighted approach, affirming the efficacy of our method.

Finally, the method's generalizability was validated on the image-based EchinodermNet dataset using both ResNet-50 and Vision Transformer models, affirming the practical applicability of node-wise weighting in diverse HML scenarios beyond textual or biological data. Since EchinodermNet has some degree of sparseness and our method could recall a few previously unpredicted nodes, this suggests that our approach could generalize well to varying degrees of sparseness in HMDs.

There are a few things we could explore in the future. We could experiment with developing an adaptive, node-based weighting system—similar to node-wise focal loss [31] but for HMDs. Additionally, the current node-weighting approach primarily considers aggregate node frequencies. An alternative method that also accounts for a node's number of ancestors and its depth within the hierarchy could potentially improve the accuracy of predictions, especially for deeper, minority nodes. Although we do not currently incorporate features such as geospatial location, leveraging such contextual information in the future could further enhance the performance. Additionally, we could explore applying node-wise re-weighting as a post-training adjustment to refine predictions further.

Bibliography

[1] F. Althaus, N. Hill, R. Ferrari, et al., "A standardized vocabulary for identifying benthic biota and substrata from underwater imagery: The CATAMI classification scheme," *PLOS ONE*, vol. 10, no. 10, pp. 1–18, 2015.

[2] M. Ashburner, C. A. Ball, J. A. Blake, et al., "Gene ontology: tool for the unification of biology," *Nature Genetics*, vol. 25, no. 1, pp. 25–29, 2000.

[3] W. Bi and J. T. Kwok, "Multilabel classification on tree- and DAG-structured hierarchies," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, Bellevue, WA, USA, 2011.

[4] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern Recognition*, vol. 37, no. 9, pp. 1757–1771, Sep. 2004, doi: https://doi.org/10.1016/j.patcog.2004.03.009.

[5] L. Buitinck, G. Louppe, M. Blondel, et al., "API design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.

[6] J. Burez and D. Van den Poel, "Handling class imbalance in customer churn prediction," *Expert Systems with Applications*, vol. 36, no. 3, pp. 4626–4636, Apr. 2009, doi: https://doi.org/10.1016/j.eswa.2008.05.027.

[7] F. J. Castellanos, J. J. Valero-Mas, J. Calvo-Zaragoza, and J. R. Rico-Juan, "Oversampling imbalanced data in the string space," *Pattern Recognition Letters*, vol. 103, pp. 32–38, Feb. 2018, doi: https://doi.org/10.1016/j.patrec.2018.01.003.

[8] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," *Neurocomputing*, vol. 163, pp. 3–16, 2015.

[9] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera, "Concurrence among Imbalanced Labels and Its Influence on Multilabel Resampling Algorithms," *Lecture Notes in Computer Science*, pp. 110–121, 2014, doi: https://doi.org/10.1007/978-3-319-07617-1_10.

[10] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Dealing with difficult minority labels in imbalanced mutilabel data sets," *Neurocomputing*, vol. 326–327, pp. 39–53, Jan. 2019, doi: https://doi.org/10.1016/j.neucom.2016.08.158.

[11] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera, "A First Approach to Deal with Imbalance in Multi-label Datasets," *Lecture Notes in Computer Science*, pp. 150–160, 2013, doi: https://doi.org/10.1007/978-3-642-40846-5_16.

[12] F. Charte, A. J. Rivera, del Jesus, and F. Herrera, "MLeNN: A First Approach to Heuristic Multilabel Undersampling," *Lecture notes in computer science*, pp. 1–9, Jan. 2014, doi: https://doi.org/10.1007/978-3-319-10840-7_1.

[13] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation," *Knowledge-Based Systems*, vol. 89, pp. 385–397, Nov. 2015, doi: https://doi.org/10.1016/j.knosys.2015.07.019.

[14] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "REMEDIAL-HwR: Tackling multilabel imbalance through label decoupling and data resampling hybridization," *Neurocomputing*, vol. 326–327, pp. 110–122, Jan. 2019, doi: https://doi.org/10.1016/j.neucom.2017.01.118.

[15] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, p. 1, Jun. 2004, doi: https://doi.org/10.1145/1007730.1007733.

[16] W. Cheng and E. Hüllermeier, "Combining instance-based learning and logistic regression for multilabel classification," *Machine Learning*, vol. 76, no. 2–3, pp. 211–225, Jul. 2009, doi: https://doi.org/10.1007/s10994-009-5127-5.

[17] S. Chu et al., "The Transcriptional Program of Sporulation in Budding Yeast," *Science*, vol. 282, no. 5389, pp. 699–705, Oct. 1998, doi: https://doi.org/10.1126/science.282.5389.699.

[18] J. L. DeRisi, "Exploring the Metabolic and Genetic Control of Gene Expression on a Genomic Scale," *Science*, vol. 278, no. 5338, pp. 680–686, Oct. 1997, doi: https://doi.org/10.1126/science.278.5338.680.

[19] A. Dosovitskiy, L. Beyer, A. Kolesnikov, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Int. Conf. Learn. Representations (ICLR)*, 2021.

[20] M. Fang, Y. Xiao, C. Wang, and J. Xie, "Multi-label Classification: Dealing with Imbalance by Combining Labels," pp. 233–237, Nov. 2014, doi: https://doi.org/10.1109/ictai.2014.42.

[21] J. I. Fincham, C. Wilson, J. Barry, et al., "Developing the use of convolutional neural networking in benthic habitat classification and species distribution modelling," *ICES Journal of Marine Science*, vol. 77, no. 7-8, pp. 3074–3082, 2020.

[22] J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker, "Multilabel classification via calibrated label ranking," *Machine Learning*, vol. 73, no. 2, pp. 133–153, Aug. 2008, doi: https://doi.org/10.1007/s10994-008-5064-8.

[23] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, Jul. 2012, doi: https://doi.org/10.1109/tsmcc.2011.2161285.

[24] A. P. Gasch et al., "Genomic Expression Programs in the Response of Yeast Cells to Environmental Changes," *Molecular Biology of the Cell*, vol. 11, no. 12, pp. 4241–4257, Dec. 2000, doi: https://doi.org/10.1091/mbc.11.12.4241.

[25] E. Giunchiglia and T. Lukasiewicz, "Coherent hierarchical multilabel classification networks," in *Adv. Neural Inform. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9662–9673.

[26] S. Godbole and Sunita Sarawagi, "Discriminative Methods for Multi-labeled Classification," *Springer eBooks*, pp. 22–30, May 2004, doi: https://doi.org/10.1007/978-3-540-24775-3_5.

[27] H. He and E. A. Garcia, "Learning from Imbalanced Data - IEEE Journals Magazine," *Ieee.org*, 2009. https://ieeexplore.ieee.org/document/5128907

[28] F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus, *Multilabel Classification*. Cham: Springer International Publishing, 2016. doi: https://doi.org/10.1007/978-3-319-41111-8.

[29] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, Nov. 2002, doi: https://doi.org/10.3233/ida-2002-6504.

[30] D. Langenkämper, T. Möller, D. Brün, and T. Wilhelm Nattkemper, "Efficient visual monitoring of offshore windmill installations with online image annotation and deep learning computer vision," in *Global Oceans 2020: Singapore – U.S. Gulf Coast*, 2020, pp. 1–6.

[31] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, 2020.

[32] B. Liu and G. Tsoumakas, "Making Classifier Chains Resilient to Class Imbalance," *PMLR*, pp. 280–295, Nov. 2018, Accessed: May 16, 2025. [Online]. Available: https://proceedings.mlr.press/v95/liu18c.html

[33] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory Undersampling for Class-Imbalance Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, Apr. 2009, doi: https://doi.org/10.1109/tsmcb.2008.2007853.

[34] S. C. Lowe, B. Misiuk, I. Xu, et al., "Benthicnet: A global compilation of seafloor images for deep learning applications," *arXiv preprint arXiv:2405.05241*, 2024.

[35] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, Sep. 2012, doi: https://doi.org/10.1016/j.patcog.2012.03.004.

[36] O. Maimon and L. Rokach, Eds., *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2010. doi: https://doi.org/10.1007/978-0-387-09823-4.

[37] R. Miranda Pereira, Y. Maldonado e Gomes da Costa, and C. Nascimento Silla, "Dealing with imbalanceness in hierarchical multilabel datasets using multilabel resampling techniques," in *Proc. 30th Int. Conf. Tools Artif. Intell. (ICTAI)*, 2018, pp. 818–824.

[38] W. W. Y. Ng, G. Zeng, J. Zhang, D. S. Yeung, and W. Pedrycz, "Dual autoencoders features for imbalance classification problem," *Pattern Recognition*, vol. 60, pp. 875–889, Dec. 2016, doi: https://doi.org/10.1016/j.patcog.2016.06.013.

[39] A. Pakrashi and B. M. Namee, "Stacked-MLkNN: A stacking based improvement to Multi-Label k-Nearest Neighbours," *PMLR*, pp. 51–63, Oct. 2017, Accessed: May 16, 2025. [Online]. Available: https://proceedings.mlr.press/v74/pakrashi17a.html

[40] R. M. Pereira, Y. M. Costa, and C. N. Silla Jr., "Toward hierarchical classification of imbalanced data using random resampling algorithms," *Information Sciences*, vol. 578, pp. 344–363, 2021.

[41] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995. doi: 10.1080/09540099550039318.

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

[43] K. Sechidis, G. Tsoumakas, and I. Vlahavas, "On the Stratification of Multi-label Data," Machine Learning and Knowledge Discovery in Databases, pp. 145–158, 2011, doi: https://doi.org/10.1007/978-3-642-23808-6_10.

[44] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms," in *AAAI Spring Symposium: Lifelong Machine Learning*, 2013. Available: https://api.semanticscholar.org/CorpusID:18548239.

[45] P. T. Spellman et al., "Comprehensive Identification of Cell Cycle–regulated Genes of the Yeast Saccharomyces cerevisiae by Microarray Hybridization," *Molecular Biology of the Cell*, vol. 9, no. 12, pp. 3273–3297, Dec. 1998, doi: https://doi.org/10.1091/mbc.9.12.3273.

[46] Z. Sun, Q. Song, X. Zhu, H. Sun, B. Xu, and Y. Zhou, "A novel ensemble method for classifying imbalanced data," *Pattern Recognition*, vol. 48, no. 5, pp. 1623–1637, May 2015, doi: https://doi.org/10.1016/j.patcog.2014.11.014.

[47] Y. SUN, A. K. C. WONG, and M. S. KAMEL, "CLASSIFICATION OF IMBALANCED DATA: A REVIEW," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 04, pp. 687–719, Jun. 2009, doi: https://doi.org/10.1142/s0218001409007326.

[48] M. A. Tahir, J. Kittler, and A. Bouridane, "Multilabel classification using heterogeneous ensemble of multi-label classifiers," *Pattern Recognition Letters*, vol. 33, no. 5, pp. 513–523, Apr. 2012, doi: https://doi.org/10.1016/j.patrec.2011.10.019.

[49] M. A. Tahir, J. Kittler, K. Mikolajczyk, and F. Yan, "Improving Multilabel Classification Performance by Using Ensemble of Multi-label Classifiers," *Lecture notes in computer science*, pp. 11–21, Jan. 2010, doi: https://doi.org/10.1007/978-3-642-12127-2_2.

[50] M. A. Tahir, J. Kittler, and F. Yan, "Inverse random under sampling for class imbalance problem and its application to multi-label classification," *Pattern Recognition*, vol. 45, no. 10, pp. 3738–3750, Oct. 2012, doi: https://doi.org/10.1016/j.patcog.2012.03.014.

[51] A. N. Tarekegn, M. Giacobini, and K. A. Michalak, "A review of methods for imbalanced multilabel classification," *Pattern Recognition*, vol. 118, p. 107965, 2021.

[52] G. Tsoumakas and I. Katakis, "Multi-Label Classification," *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1–13, Jul. 2007, doi: https://doi.org/10.4018/jdwm.2007070101.

[53] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Effective and Efficient Multilabel Classification in Domains with Large Number of Labels." Accessed: May 14, 2025. [Online]. Available: http://www.ecmlpkdd2008.org/files/pdf/workshops/mmd/4.pdf

[54] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining Multi-label Data," *Springer eBooks*, pp. 667–685, Jan. 2009, doi: https://doi.org/10.1007/978-0-387-09823-4_34.

[55] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-Labelsets for Multilabel Classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 7, pp. 1079–1089, Jul. 2011, doi: https://doi.org/10.1109/tkde.2010.164.

[56] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel, "Decision trees for hierarchical multi-label classification," *Machine Learning*, vol. 73, no. 2, pp. 185–214, Aug. 2008, doi: https://doi.org/10.1007/s10994-008-5077-3.

[57] G. I. Winata and M. L. Khodra, "Handling imbalanced dataset in multi-label text categorization using Bagging and Adaptive Boosting," *arXiv (Cornell University)*, pp. 500–505, Aug. 2015, doi: https://doi.org/10.1109/iceei.2015.7352552.

[58] O. Zavala-Romero, A. Bozec, E. P. Chassignet, and J. R. Miranda, "Convolutional neural networks for sea surface data assimilation in operational ocean models: test case in the gulf of mexico," *Ocean Science*, vol. 21, no. 1, pp. 113–132, 2025.

[59] M.-L. Zhang and Z.-H. Zhou, "A Review on Multi-Label Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837, Aug. 2014, doi: https://doi.org/10.1109/tkde.2013.39.