Probabilistic Label Tree for Streaming Multi-Label Learning

Tong Wei
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
weit@lamda.nju.edu.cn

Yu-Feng Li
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
liyf@lamda.nju.edu.cn

Guo-Ping Yang
Huawei
yangguoping1@huawei.com

Wei-Wei Tu
4Paradigm Inc., Beijing, China
tuww.cn@gmail.com

ABSTRACT
Multi-label learning aims to predict a subset of relevant labels for each instance, which has many real-world applications. Most extant multi-label learning studies focus on a fixed size of label space. However, in many cases, the environment is open and changes gradually and new labels emerge, which is coined as streaming multi-label learning (SMLL). SMLL poses great challenges in twofolds: (1) the target output space expands dynamically; (2) new labels emerge frequently and can reach a significantly large number. Previous attempts on SMLL leverage label correlations between past and emerging labels to improve the performance, while they are inefficient when deal with large-scale problems. To cope with this challenge, in this paper, we present a new learning framework, i.e., the probabilistic streaming label tree (PSLT). In particular, each non-leaf node of the tree corresponding to a subset of labels, and a binary classifier is learned at each leaf node. Initially, PSLT is learned on partially observed labels, both tree structure and node classifiers are updated while new labels emerge. Using carefully designed updating mechanism, PSLT can seamlessly incorporate new labels by first passing them down from the root to leaf nodes and then update node classifiers accordingly. We provide theoretical bounds for the iteration complexity of tree update procedure and the estimation error on newly arrived labels. Experiments show that the proposed approach improves the performance in comparison with eleven baselines in terms of multiple evaluation metrics. Anonymous source code is available at https://gitee.com/pslt-kdd2021/pslt.

CCS CONCEPTS
• Machine learning → Multi-label learning.

KEYWORDS
multi-label learning, streaming label learning

1 INTRODUCTION
Multi-label learning (MLL) [7, 37] aims to annotate objects with the relevant labels from an extremely large number of candidate labels. MLL recently owns many real-world applications. For example, in webpage categorization [17], labels (categories) are collected in Wikipedia and one needs to annotate a new webpage with all relevant labels from such a big candidate set; in image annotation [4], one wishes to tag each individual picture with multiple labels from such a candidate label set; in recommendation systems [14], one hopes to make informative personalized recommendations from a candidate set of items (labels).

Most extant multi-label learning studies focus on a fixed set of labels [1, 13, 20, 26–29, 34, 36, 37]. That is, they assume that all the labels in the learning process are given at once. In the opposite, this paper investigates an important problem in data streams, i.e., multi-label classification under streaming emerging labels, since in many real-world applications, the environment is open and changes gradually. For instance, in webpage categorization, new labels (categories) are created for historic events in Wikipedia, such as the "2022 Winter Olympics"; in recommendation systems, many new labels (items) are becoming available to users. If multi-label models do not take new labels into account, it deteriorates the performance.

The main challenges of SMLL are in twofolds: (1) the target output space expands dynamically; (2) new labels emerge frequently and can reach a significantly large number. To cope with these problems, one of previous work SLL [33] assumes that each label can be represented as a linear combination of other labels and it learns a linear classifier for each new label incorporating the relationship with past labels. However, the linear representation across labels limits the SLL performance, and the training knowledge from classifiers of past labels is neglected. Recently, DSLL [25] proposes a novel DNN-based framework to effectively model the emerging new labels. DSLL has the ability to explore and utilize deep correlations between new labels and past labels without computationally intensive retraining. However, DSLL learns a separate model for newly arrived labels based on initial teacher model and has difficulties to deal with large-scale problems. Additionally, a unified
model is desired in many cases, which evolves using emerging data stream.

In this paper, we propose a tree-based streaming multi-label learning framework, i.e., the Probabilistic Streaming Label Tree (PSLT), which can seamlessly take advantage of emerging new labels. In PSLT, labels are organized in the tree with the hope of capturing hierarchical correlations among labels. Our algorithm works by first partitioning the label space, such that each label corresponds to a leaf of the tree. In particular, a clustering algorithm is employed to determine the partition of labels and PSLT learns a binary classifier for each child node for the purpose of inference. This procedure repeats recursively until the stop condition is meet. While dealing with emerging labels, PSLT evolves itself by considering both tree structure and node classifiers. It first performs the label insertion procedure which passes new labels from the root to leaf nodes guided by label proximity. Noted that if the size of a leaf node meets the size limit, an additional partition is triggered. After label insertion, node classifiers are learned or modified using emerging labels. In the experiments, we observe that PSLT outperforms eleven baselines on several datasets and it is far more effective than the deep learning based method DSLL. We demonstrate the studied SMLL setup in Figure 1.

We summarize our main contributions as follows.

• We study a general problem setup, i.e., the streaming multi-label learning. A new framework based on the probabilistic label tree is proposed to effectively update the model by leveraging label correlations.
• Analyses on the iteration complexity of the model update procedure and estimation error on emerging labels are provided. Memory complexity of the model is also presented.
• The proposed approach achieves state-of-the-art performance in comparison with eleven baselines in terms of multiple performance metrics on four benchmark multi-label datasets.

The rest of the paper is organized as follows. Section 2 introduces the previous related work. Section 3 gives the problem and describes the proposed framework in detail. Section 4 presents the theoretical analysis. Section 5 presents the experimental results validating the model and providing several insights. We conclude in Section 6.

2 RELATED WORK

Tree-based Multi-label Learning. Most extant multi-label learning approaches focus on fixed label space. Tree-based methods greatly reduce inference time, which generally scales logarithmically in the number of labels. There are typically two types of trees including instance trees [18, 21] and label trees [3, 10, 32], depending whether instance or label is partitioned in tree nodes. These methods can readily scale up to problems with hundreds of thousands of labels. Recently, OPLTs [8, 9] is proposed to learn in the online manner. In OPLTs, new labels traverse down from the root to leaves by randomly choosing a child node or calculating score that maximizes balancedness of tree and fitness of data, in which label correlations are neglected.

Learning with Emerging Classes. One of previous work SLL [33] assumes that: (1) a label is represented as a linear combination of other labels, and (2) the relationship (linear combination) between labels can be inherited by classifiers of different labels. Constrained by these hypotheses, SLL trains a linear classifier for new labels with the relationship between past labels and new labels. However, the linear representation across labels limits the SLL performance, and the training knowledge from classifiers of past labels is neglected. To alleviate this issue, DSLL [25] propose a novel DNN-based framework to effectively model the emerging new labels by using leveraging knowledge of previous trained model to learn high-order representations. From the point of view that new classes are outliers, [38] and [15] use anomaly detection to detect new classes and update models with emerging data, this is different from the streaming multi-label learning. [39] further extends this problem to semi-supervised learning for the presence of unlabeled samples in emerging data. In this paper, we show that it is natural and efficient to incorporate emerging new labels in the probabilistic label tree framework, which takes advantage of label correlations in the meanwhile.

3 PROBABILISTIC STREAMING LABEL TREE

3.1 Problem Formulation

We first introduce the streaming multi-label learning setup and some notations. Suppose that an initial training data set is denoted by $D = \{(x_1, y_1); \ldots; (x_N, y_N)\}$, where $x_i \in \mathbb{R}^D$ is a real vector representing an input feature (instance) and $y_i \in \{0,1\}^L$ is the corresponding output label vector for the $i$-th example. The input feature matrix is denoted as $X = [x_1; \ldots; x_N] \in \mathbb{R}^{D \times N}$ and the initial output label matrix is $Y = [y_1; \ldots; y_N] \in \{0,1\}^{L \times N}$. Note that, $Y_{ij} = 1$ if the $j$-th label is relevant to sample $x_i$ and $Y_{ij} = -1$ otherwise.

By observing $D$, multi-label learning aims to derive a proper model that generates the prediction for a test sample. Then new labels arrive batch by batch in a streaming fashion. For simplicity, we denote $y^{new}_i = [y^{L+1}_i; \ldots; y^{L+m}_i]$ as the newly observed $m$ labels for sample $x_i$, where $m \geq 1$. The goal of streaming multi-label learning is to derive a unified model by taking care of the continually emerging new labels. We summarize definition of notations used in the rest of this paper in Table 1.

![Figure 1: Illustration of the streaming multi-label learning setup. At each timestamp $T_i$, the label matrix is expanded by a batch of emerging new labels, where each row represents the label vector for a sample and $+1$s or $-1$s respectively denote labels are relevant or irrelevant to the sample. $M_i$ denotes the model at time $T_i$ which evolves from $M_{i-1}$ such that its output label space expands accordingly. We use tree-based models for the purpose of demonstration.](image-url)
Table 1: Notations.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>size of training set</td>
</tr>
<tr>
<td>(D)</td>
<td>feature dimension</td>
</tr>
<tr>
<td>(L)</td>
<td>size of label set</td>
</tr>
<tr>
<td>(X \in \mathbb{R}^{N \times D})</td>
<td>feature matrix, (X = [x_1; \ldots; x_N])</td>
</tr>
<tr>
<td>(Y \in {0, 1}^{N \times L})</td>
<td>label matrix, (Y = [y_1; \ldots; y_N])</td>
</tr>
<tr>
<td>(V)</td>
<td>label representation matrix, (V = [v_1; \ldots; v_L])</td>
</tr>
<tr>
<td>(I_j)</td>
<td>(j)-th label</td>
</tr>
<tr>
<td>(T)</td>
<td>a probabilistic streaming label tree</td>
</tr>
<tr>
<td>(K)</td>
<td>branch factor for tree node</td>
</tr>
<tr>
<td>(V_T)</td>
<td>the set of nodes of (T)</td>
</tr>
<tr>
<td>(L_T)</td>
<td>the set of leaf nodes of (T)</td>
</tr>
<tr>
<td>(r_T)</td>
<td>the root of (T)</td>
</tr>
<tr>
<td>(L_0)</td>
<td>number of leaf nodes in the subtree rooted at (v)</td>
</tr>
<tr>
<td>(c_v)</td>
<td>the set of labels in node (v)</td>
</tr>
<tr>
<td>(w \in \mathbb{R}^D)</td>
<td>node classifier</td>
</tr>
<tr>
<td>(m)</td>
<td>number of emerging new labels</td>
</tr>
</tbody>
</table>

3.2 Tree Construction

In this subsection, we describe the procedure of building probabilistic streaming label tree. Our approach processes one node of the tree at a time, starting with the root node. At each non-leaf node, it partitions the labels into a fixed number of disjoint subsets as child nodes and learns a binary classifier for each of the children for the purpose of inference. The procedure might repeat for multiple times.

Label Partitioning. During label partitioning, representations for labels are necessary, based on which label proximities are calculated. Ideally, labels with similar semantic meaning are desired to be grouped in the same clusters. Since label semantic representations are usually unaccessible in many real-world applications. To this end, we provide a simple and effective walk around. Specifically, label representations are constructed by averaging all the training examples for which it is relevant to the corresponding labels. Formally, the label representation matrix \(V = [v_1; \ldots; v_L]\) is given as \(V = \frac{1}{|L|} \sum_{i \in L} V_i\). Using the mean vector of positive instances as label representation has been used in many research fields, e.g., few-shot learning [22], which has demonstrated desirable performance.

Label Partitioning. The aim of label partitioning is to split \(S\) into disjoint subsets by taking label similarity into account. This is achieved by \(k\)-means clustering, which also presents many choices such as number of clusters and degree of balancedness among the clusters. The same process is repeated on each of the newly-created \(K\) child nodes in an iterative manner. Let \(c_k\) denotes the set of labels of the \(k\)-th cluster, the objective function of label partitioning can be formulated as

\[
\min_{c_1, \ldots, c_K} \sum_{k=1}^{K} \sum_{i \in c_k} \text{dist} (v_i, \mu_k) \tag{1}
\]

where \(\text{dist}(\cdot, \cdot)\) represents a distance function and \(\mu_k\) denotes the center of the \(k\)-th cluster. The distance function is defined in terms of the cosine similarity as \(\text{dist} (v_i, \mu_k) = 1 - \frac{v_i^\top \mu_k}{|v_i||\mu_k|}\) \Problem{1} is NP-hard and an approximate solution can be found using the standard \(K\)-means algorithm [12].

Learning Node Classifiers. Once the label space is partitioned into a tree structure, we learn a \(K\)-way One-vs-All classifier [6, 34] at each node of the tree. These classifiers are learned independently by considering only the training samples that have at least one positive label in the current node labels. We distinguish the leaf nodes and non-leaf nodes in the following way: (i) for non-leaf nodes, the classifier learns \(K\) linear classifiers separately, each maps to one of the \(K\) children. During prediction, the output of each classifier determines whether the test point should traverse down the corresponding child. (ii) for leaf nodes, the classifier learns to predict the actual labels on the node. The One-vs-All procedure is shown as \textsc{One-vs-All} in Algorithm 1. Formally, at each node, we consider the following optimization problem for learning linear classifier for each of the \(K\) child:

\[
\min_{w} \ell (y, w) := \mathcal{R}(w) + C \sum_{i=1}^{N} \ell (y_i, w^\top x_i) \tag{2}
\]

where \(\{x_i, y_i\}_{i=1}^{N}\) are constructed training data for a particular child node. For non-leaf nodes, \(y_i = +1\) iff at least one relevant label of \(x_i\) is in the child node, otherwise \(-1\). For leaf nodes, \(y_i\) denotes the true label of \(x_i\) with respect to label \(l_i\). Concretely, if squared hinge loss, i.e., \(\ell (y, \hat{y}) = \max (0, 1 - y \cdot \hat{y})^2\), and \(L_2\)-regularization, i.e., \(\mathcal{R}(w) := ||w||_2^2\), are used, the objective of linear SVM becomes

\[
\min_{w} \ell (y, w) := ||w||_2^2 + C \sum_{i=1}^{N} \max \{0, 1 - y_i w^\top x_i\}^2 \tag{3}
\]

The above problem can be readily solved using off-the-shelf LIBLINEAR [5] which generally scales linearly with the number of training samples.

Algorithm 1 \textsc{TSSLTrain}(X, Y, L, V, K, d_{\text{max}})

1. \(n = \text{new \ NODE}\)
2. \(n.L = [L]\)
3. \(n.V = \frac{1}{|L|} \sum_{i \in L} V_i\)
4. \(c_1, \ldots, c_K = K\text{-MEANS}(L, V, K)\)
5. for \(i = 1, \ldots, K\) do
6. \(\text{child}\_L = c_i\)
7. \(\text{child}\_V = \frac{1}{|c_i|} \sum_{j \in c_i} V_j\)
8. \(n.Ch = 0\)
9. if \(d_{\text{max}} > 0\) then
10. \(n.isLeaf = \text{FALSE}\)
11. \(\text{child} = \text{TRAIN}(c_i, V, K, d_{\text{max}} - 1)\)
12. \(n.Ch = n.Ch \cup \{\text{child}\}\)
13. else
14. \(n.isLeaf = \text{TRUE}\)
15. \(n.W = \text{ONE-VS-ALL}(X, Y, n.L)\)
16. end if
17. end for
18. return \(r\)
3.3 Inference

With the tree structure and node classifiers learned in the above manner, testing points are passed down from the root to leaf nodes. Typically, at each node, the testing point $x$ is evaluated by $K$ binary classifiers corresponding to $K$ child nodes. Let $z_v = \mathbb{I}(\sum_{j \in \mathcal{W}_v} y_j \geq 1)$ denote a label if at least one relevant label of $x$ is in the node labels $\mathcal{W}_v$, provided that the true label $y$ of $x$ is observed. Then, it holds based on the chain rule that for any node $v \in \mathcal{V}_T$:

$$
\eta_v(x) = P(z_v = 1 \mid \text{Path}(v)) = \prod_{v' \in \text{Path}(v)} \eta(x, v'),
$$

where $\text{Path}(v)$ denotes the path from root to node $v$ and $\eta(x, v) = P(z_v = 1 \mid \text{pa}(v) = 1, x)$ for non-root nodes, where $\text{pa}(v)$ denotes the parent node of $v$, and $\eta(x, r_T) = P(z_{r_T} = 1 \mid x) = 1$, that is, $x$ is relevant to at least one label in the root labels. To calculate Equation (4), $P(z_v = 1 \mid \text{pa}(v) = 1, x)$ is estimated by node classifiers of $v$. Notice that for leaf nodes we get the conditional probabilities of labels, that is, $\eta_j(x) = P(y_j = 1 \mid x), \forall l_j \in \mathcal{L}_T$ and $j \in [L]$.

3.4 Tree Update

In this subsection, we describe the way of updating pre-trained tree, which involves two parts, i.e., label insertion and node classifier modification.

**Label Insertion.** While inserting a new label $y_{\text{new}}$, we first calculate its representation from observed positive samples, which is hereby denoted as $\gamma_{\text{new}}$. Then $y_{\text{new}}$ is traversed downwards start from root of $T$. The passing direction is decided by the similarity between $y_{\text{new}}$ and child nodes. In specific, the most proximate child node $j$ is obtained by

$$
\sigma_j = \arg \min_{i \in [K]} \frac{\sigma^T_{\gamma_{\text{new}}} \gamma_{\text{new}}}{||\gamma_{\text{new}}||}.
$$

We use cosine similarity as the proximity measure which is in consistent with the label partitioning procedure while building the tree. The label insertion procedure is done recursively until a leaf node is reached. To take case of newly arrived labels, we update node labels by inserting the new label to nodes which it passes by and node representations are updated accordingly. To do this on-the-fly, sum of representations of labels is maintained and the summation is updated by each newly arrived label. Note that it is not desired to have too many labels in a single leaf node because the time complexity of training a $K$-way One-Vs-All classifier is linear in the number of labels. To this end, the leaf node is further partitioned when the node label size meets a threshold. We illustrate two tree variants in Figure 2.

**Node Classifier Modification.** With emerging new labels inserted in proximate tree nodes, it is desired to update the node classifiers using collected data related to new labels. However, training classifiers for each node from scratch is computational prohibitive. We update node classifiers using previous trained ones as initialization. In particular, for node $v$, we use each of its $K$ classifier $\gamma$ for initialization as $\gamma = \tilde{\gamma}$ and update $\gamma$ using newly arrived samples via gradient-based optimization methods. After that, $\gamma$ is substituted by $\tilde{\gamma}$ for future use. For the newly inserted label, a binary classifier is desired for the purpose of inference.

4 THEORETICAL ANALYSIS

In this section, we first show that the node classifier modification procedure can be done effectively with an iteration complexity bound. Then, we establish an estimation error bound, which guarantees the learning performance of the tree. Finally, we analyze the total time and memory complexities of the proposed approach.

4.1 Analysis on the Iteration Complexity of Node Classifier Modification

With a little bit of notation abuse, Let $y$ be meta-label vector used to train $\gamma$ and $\tilde{y}$ be the modified label vector after inserting new labels. We further denote the discrepancy between $y$ and $\tilde{y}$ as $\ell_f(y, \tilde{y}) = \sum_{i=1}^N \mathbb{I}(y_i \neq \tilde{y}_i)$, and derive an upper bound for the classifier update procedure for tree node as follows.
Algorithm 2 TSLL.PREDICT(rT, x)
1: \( Q = \emptyset \)
2: \( Q.add((rT, 1)) \)
3: \( \hat{y} = 0 \)
4: while \( Q \neq \emptyset \) do
5: (n, \( \hat{y}_n(x) \)) = \( Q.pop() \)
6: if \( n.isLeaf \) then
7: \( \hat{y}_n = \hat{y}_n(x) \)
8: else
9: for child \( \in n.Ch \) do
10: \( \hat{y}_{child}(x) = g(z, W_{child}^T x) \)
11: \( Q.add((\text{child}, \hat{y}_n(x) \times \hat{y}_{child}(x))) \)
12: end for
13: end if
14: end while
15: return \( \hat{y} \)

Algorithm 3 TSLL.UPDATE(n, \( y_{new} \), \( q_{new} \), \( \mathcal{A} \))
1: if \( n.isLeaf \) then
2: \( n.L = n.L \cup \{ y_{new} \} \)
3: \( w_{new} = \text{trainBinaryClassifier}(y_{new}) \)
4: \( n.W = n.W \cup \{ w_{new} \} \)
5: else
6: \( \text{cosSim} = 0 \)
7: nextCh = NONE
8: for \( \text{child} \in n.Ch \) do
9: \( \text{cosSim} = \frac{\text{child.w} \cdot w_{new}}{||\text{child.w}|| \cdot ||w_{new}||} \)
10: if \( \text{cosSim} \geq \text{cosSim} \) then
11: \( \text{cosSim} = \text{cosSim} \)
12: nextCh = child
13: end if
14: end for
15: for \( w \in n.W \) do
16: \( \text{Initialize} \ w = n.w \)
17: Solve subproblem by \( \mathcal{A} \) using \( y_{new} \) and \( \hat{w} \)
18: Set \( n.w = w \)
19: end for
20: \( \text{UPDATE}(\text{nextCh}, y_{new}, q_{new}, \mathcal{A}) \)
21: end if

Lemma 1. Assume \( \ell(\cdot) \) in Problem (2) is \( \alpha \)-Lipschitz and the optimal solution before and after incorporating emerging labels be \( \mathbf{w}^* \) and \( \mathbf{w}^* \) respectively, which satisfy \( ||\mathbf{w}^*||_2 \leq B \) and \( ||\mathbf{w}^*||_2 \leq B \). Then

\[
\ell(\mathbf{y}, \mathbf{w}^*) - \ell(\mathbf{y}, \mathbf{w}^*) \leq 4\alpha B \sum_{i=1}^{N} I(y_i = 1) \wedge I(y_i = -1).
\]

To go a step further based on Lemma 1, we develop the following theorem to show that we could significantly reduce runtime upper bound with the initialization.

Theorem 1. Assume \( \ell(\cdot) \) is \( \alpha \)-Lipschitz and we have a solver \( \mathcal{A} \) with sublinear convergence rate that can solve each subproblem \( k \) with \( \epsilon \) precision in \( T_k = O \left( \frac{\sum_{i=1}^{K} \ell_i(y_i, w_i)}{\epsilon} \right) \) iterations. We can bound the total number of iterations of the node classifier modification procedure by \( T_{total} = \sum_{i=1}^{K} T_k = O \left( \frac{\sum_{i=1}^{K} \ell_i(y_i, w_i)}{\epsilon} \right) \), where \( \Delta_{y,y} = \sum_{i=1}^{N} I(y_i = 1) \wedge I(y_i = -1) \). However, under a naive zeros initialization with \( \mathbf{w}^* = 0 \), the upper bound is \( T_{total} = \sum_{i=1}^{K} T_k = O \left( \frac{NK}{\epsilon} \right) \).

4.2. L1 Estimation Error Bound

In addition to the effectiveness of node classifier modification, we are interested in multi-label classifiers that estimate conditional probabilities of labels, \( \eta_j = \mathbb{P}(y_j = 1 | x) \), \( j \in [m] \), as accurately as possible, that is, with possibly small \( L_1 \)-estimation error, \( ||\eta_j(x) - \hat{\eta}_j(x)|| \). According to [8], we show that the weighted expected \( L_1 \) estimation error averaged over all labels can be bounded by a weighted sum of expected \( L_1 \) error of node classifiers divided by the number of labels.

Theorem 2 (Estimation Error Bound [8]). For any tree \( T \), distribution \( \mathbb{P}(x, y) \), a set of emerging new label \( \{ y_i^{new} \}_{i=1}^{m} \), the expected \( L_1 \) estimation error bound holds:

\[
\frac{1}{m} \sum_{j=1}^{m} \mathbb{E}_{x \sim \mathbb{P}(x)} \left[ ||\eta_j(x) - \hat{\eta}_j(x)|| \right] 
\leq \frac{1}{m} \sum_{v \in \mathbb{V}} \mathbb{E}_{x \sim \mathbb{P}(x)} \left[ ||\eta(x, o') - \hat{\eta}(x, o')|| \right]
\]

where for the root node \( \mathbb{P}(\mathbb{z}_{pa(rt)} = 1) = 1 \).

4.3. Time and Memory Complexity Analysis

In this subsection, we provide bounds for the time and memory complexities of PSLT.

Lemma 2. Under the case specified in Theorem 1, for an emerging new label set \( \{ y_i^{new} \}_{i=1}^{m} \) and a \( k \)-ary tree \( T \), the time complexity of the tree update is bounded by \( \Omega \left( \sum_{i=1}^{K} dK + dK \frac{N}{\epsilon} + N \mathcal{D} + |\mathcal{L}_{rT}| \tau T K \right) \).

Proof. The tree update time complexity of a new label \( y_{new} \) is the sum of the complexities of the label insertion, existing node classifier fine-tuning, and the training of its classifier. On label insertion, \( y_{new} \) traverses from the root to a leaf node with at most \( d \) steps where \( d \) is the depth of the tree \( T \). At each step, cosine similarity between the representation of \( y_{new} \) and tree node is calculated with time complexity of \( O(dKd) \) where \( d \) is the number of non-zero features of node representations on average. The second part involves fine-tuning node classifier along the path. It is obvious to see that at most \( d \) node classifiers need to be updated. As stated in Theorem 1, a good estimation of best parameters can be obtained in \( \Delta_{y,y} = \sum_{i=1}^{N} I(y_i = 1) \wedge I(y_i = -1) \) iterations, where \( \Delta_{y,y} \) denote the averaged discrepancy between \( y \) and \( \hat{y} \) in the path. Hence the time complexity reaches \( O(dK) \). Additionally, to train a new classifier for \( y_{new} \), \( O(\mathcal{D}) \) is afforded for a linear classifier. Lastly, we consider the time complexity when leaf node partition is triggered during label insertion. In the worst case, all leaf nodes with size \( |\mathcal{L}_{rT}| \) meet the set node labels threshold \( \tau \), which leads to time complexity \( O \left( |\mathcal{L}_{rT}| \tau T K \right) \). Overall, the total time complexity of tree update procedure gives \( \Theta \left( \sum_{i=1}^{K} dKd + dK \frac{N}{\epsilon} + N \mathcal{D} + |\mathcal{L}_{rT}| \tau T K \right) \).
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>#Train</th>
<th>#Test</th>
<th>#Feature</th>
<th>#Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>MirFlickr</td>
<td>image</td>
<td>10,417</td>
<td>2,083</td>
<td>1,000</td>
<td>38</td>
</tr>
<tr>
<td>Delicious</td>
<td>web</td>
<td>12,920</td>
<td>3,185</td>
<td>500</td>
<td>983</td>
</tr>
<tr>
<td>EURlex</td>
<td>text</td>
<td>15,479</td>
<td>3,869</td>
<td>5,000</td>
<td>3,993</td>
</tr>
<tr>
<td>Wiki10</td>
<td>text</td>
<td>14,146</td>
<td>6,616</td>
<td>101,938</td>
<td>30,938</td>
</tr>
</tbody>
</table>

**Table 2: Statistics of datasets.**

**Lemma 3.** Under the case specified in Lemma 2, if the $K$-ary tree $T$ is balanced, the tree update time complexity is bounded by $O \left( m \left( K \hat{D} \log K L + K \hat{D} \log K \hat{L} + N \hat{D} \right) + \left| L_T \right| \left| \tau T K D \right| \right)$.

**Proof.** Given that $T$ is balanced, we have its depth $d = \log_K L$, which is small if a wide tree is constructed ($K = 100$ in our implementation). The rest part the bound is the same as Lemma 2. □

**Proposition 1.** The memory complexity of a $K$-ary tree $T$ is bounded by $O \left( (K + 1) \left( \left| V_T \right| - \left| L_T \right| \right) \hat{D} + \left| L_T \right| \bar{D} \right)$, where $\left| V_T \right|$ is the total number of nodes and $\left| L_T \right|$ is the number of leaves in $T$.

**Proof.** Each **internal node** of $T$ stores the mean vector of its associated label representations and $K$ classifiers corresponding to $K$ child nodes. Moreover, each **leaf node** of $T$ stores the classifiers of corresponding labels. Firstly, let $\left| V_T \right| - \left| L_T \right|$ be the number of internal nodes of $T$. Consequently, it takes $O \left( \left( \left| V_T \right| - \left| L_T \right| \right) \hat{D} \right)$ to store the mean representations and $O \left( K \left( \left| V_T \right| - \left| L_T \right| \right) \bar{D} \right)$ to store node classifiers. In addition, as the number of leafs equals $\left| L_T \right|$, it takes $O \left( \left| L_T \right| \bar{D} \right)$ to store label classifiers on average. In total, the space complexity of $T$ is given as $O \left( (K + 1) \left( \left| V_T \right| - \left| L_T \right| \right) \hat{D} + \left| L_T \right| \bar{D} \right)$. □

5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of our proposed approach (Pslt) and compare with eleven baselines using various multi-label learning metrics.

**Datasets.** We perform experiments on four multi-label datasets which are publicly available from the XML repository. Detailed statistics are summarized in Table 2, where $L$ denotes average labels per sample and $\hat{D}$ denotes average non-zero features per sample.

**Baselines.** In our experiments, we compare the proposed methods with ten well-established or state-of-the-art multi-label learning algorithms:

- **Binary Relevance (BR) [34]** is a set of $m$ independent logistic regression classifiers.
- **Classifier chains (CC) [19]** transforms the multi-label learning problem into a chain of binary classification problems to incorporate label dependencies.
- **RAKEL [24]** considers a set of $k$ labels in a multi-label training set as a new single-label classification task.
- **ML-kNN [36]** identifies each unseen instance's $k$ nearest neighbors in the training set and utilizes the maximum a posteriori principle to determine the label set.
- **SLEEC [2]** is a well-known embedding-based method, which learns the label embedding by preserving the pairwise distances between a few nearest label neighbors.
- **SML [11]** is the state-of-the-art embedding-based method, which further improves the convergence rate of SLEEC.
- **SLL [33]** is a streaming label learning method, which assumes linear relationships between past labels and new labels.
- **Deep-ML [35]** introduces a shallow neural network as the multi-label model trained to minimize a ranking loss. Here, we deepen the network to boost performance.
- **DNN-BCE [16]** makes use of ReLU, AdaGrad, Dropout, and other deep learning techniques to train a DNN-based model.
- **C2AE [31]** is the state-of-the-art deep learning method for multi-label classification, which integrates the DNN architectures of canonical correlation analysis and auto-encoder to learn the deep latent space.
- **DSL1 [25]** is the state-of-the-art deep learning method for streaming multi-label classification, which integrates the knowledge distillation technique.

**Evaluation Metrics.** The predictive accuracy of multi-label learning can be evaluated in different ways. Bipartition-based metrics assess a model’s ability to predict sets of labels, whereas ranking-based metrics evaluate MLL methods, which instead produce rankings of labels. Therefore, our proposed method was evaluated in terms of both bipartition-based metrics, i.e., Hamming loss, macro-F1, micros-F1, and instance-F1, and ranking-based metrics, i.e., Precision@k, coverage, ranking loss, average precision (AP), macro-AUC, and micro-AUC [30]. The details of these metrics can be referred in Appendix C.

**Settings.** To handle newly arrived labels, we investigated the models with different batch sizes following previous settings [33]. In particular, for each dataset, we selected randomly 50% (e.g., 483 on Delicious) labels as past labels and constructed a classifier for the past labels. Then, the remaining labels were treated as new labels with different batch sizes. MLL methods can be extended to handle these new labels through independent modeling. More detailed settings are provided in the Appendix C.

5.1 Results with Emerging New Labels

In this part of experiments, we evaluate Pslt and comparison methods on newly arrived labels. For each method, it either learns a classifier on new labels separately or by incorporating information from previously learned model. After that, performance on new labels of each model is evaluated using test data.

**Ranking performance evaluation.** First, we evaluate Pslt and comparison methods with respect to the ranking-based metrics. Due to the limited space, we report results for two measures, i.e., micro-AUC and ranking loss, in Table 3. Additional results for Precision@k, AP and coverage are reported in Appendix D. Regarding micro-AUC, it can be seen that Pslt achieves the best performance in 17 out of 20 cases. In the other three cases, Pslt achieves the second best results with a minor gap between DSL1. For ranking loss, Pslt outperforms the others in 11 cases. Note that conventional multi-label learning
Table 3: Ranking performance of each comparison method for learning new labels with different batch size (denoted by #label) by treating 50% labels as past labels. ↓(↑) means the smaller (larger) the value, the better the performance.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#label</th>
</tr>
</thead>
<tbody>
<tr>
<td>MirFlickr</td>
<td>3 0.6589 0.5049 0.6572 0.6296 0.5476 0.6123 0.7291 0.7448 0.7592 0.8051 0.8122 0.9070</td>
</tr>
<tr>
<td></td>
<td>6 0.7100 0.5045 0.7187 0.6624 0.6400 0.6959 0.8388 0.8476 0.8666 0.8628 0.8713 0.8909</td>
</tr>
<tr>
<td></td>
<td>9 0.7311 0.5071 0.7170 0.6754 0.7001 0.7331 0.8671 0.8692 0.8778 0.8690 0.8972 0.9174</td>
</tr>
<tr>
<td></td>
<td>12 0.7151 0.5057 0.7223 0.6692 0.6662 0.7116 0.8630 0.8600 0.8763 0.8807 0.8886 0.8974</td>
</tr>
<tr>
<td></td>
<td>15 0.7157 0.5051 0.7253 0.6611 0.6805 0.7015 0.8624 0.8584 0.8787 0.8749 0.8873 0.8900</td>
</tr>
</tbody>
</table>

| Delicious | 100 0.7133 0.5658 0.6382 0.5707 0.7813 0.8274 0.7981 0.8776 0.8615 0.8545 0.8820 0.8889 |
|          | 200 0.7080 0.5666 0.6400 0.5692 0.7815 0.8258 0.7956 0.8626 0.8627 0.8731 0.8888 0.8965 |
|          | 300 0.7177 0.5719 0.6495 0.5925 0.7941 0.8387 0.8068 0.8792 0.8655 0.8777 0.9037 0.9014 |
|          | 400 0.7159 0.5705 0.6515 0.5916 0.7964 0.8141 0.8059 0.8736 0.8909 0.8656 0.9048 0.9628 |
|          | 500 0.7144 0.5679 0.6445 0.5868 0.7926 0.8124 0.8030 0.8762 0.8697 0.8657 0.9011 0.8934 |

| EURlex   | 200 0.6996 0.7308 0.7015 0.6255 0.8591 0.8216 0.8813 0.8130 0.8201 0.8561 0.8884 0.9731 |
|          | 400 0.6738 0.7294 0.6821 0.6228 0.8481 0.8611 0.8905 0.8257 0.8423 0.8633 0.8928 0.9760 |
|          | 600 0.6769 0.7321 0.6743 0.6256 0.8547 0.8735 0.8959 0.8218 0.8472 0.8441 0.9001 0.9758 |
|          | 800 0.6825 0.7349 0.6575 0.6283 0.8548 0.8775 0.9016 0.8289 0.8491 0.8637 0.9034 0.9769 |
|          | 1000 0.6733 0.7361 0.6708 0.6258 0.8406 0.8691 0.9115 0.8246 0.8431 0.8456 0.9106 0.9773 |

| Wiki10   | 1k 0.6293 0.6535 0.6493 0.5694 0.8092 0.8113 0.8345 0.6978 0.7830 0.8274 0.8406 0.9321 |
|          | 2k 0.5928 0.6244 0.6029 0.5510 0.7505 0.7557 0.7856 0.6545 0.6968 0.7917 0.8013 0.9132 |
|          | 3k 0.5703 0.6078 0.5815 0.5405 0.7567 0.7192 0.7417 0.6505 0.7197 0.7939 0.8023 0.9072 |
|          | 4k 0.5567 0.6068 0.5687 0.5364 0.7198 0.7105 0.7346 0.6556 0.7307 0.7944 0.7996 0.8989 |
|          | 5k 0.5502 0.5984 0.5651 0.5345 0.7087 0.7194 0.7350 0.6463 0.7228 0.7824 0.7865 0.8954 |

Figure 3: Performance comparison of learning new labels with different batch sizes by considering 50% of labels as past labels. m indicates the number of new labels.
Table 4: Comparison of modeling new labels with different batch sizes by considering 50% labels as past labels. \#label denotes the number of new labels. (\dagger) means the smaller (larger) the value is, the performance will be the better.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#label</th>
<th>BR</th>
<th>CC</th>
<th>RAKEL</th>
<th>ML-4NN</th>
<th>SLEEC</th>
<th>SML</th>
<th>SLL</th>
<th>Deep-ML</th>
<th>DNN-BCE</th>
<th>C2AE</th>
<th>DSLL</th>
<th>Pslt</th>
</tr>
</thead>
<tbody>
<tr>
<td>MirFlickr</td>
<td>3</td>
<td>0.4550</td>
<td>0.5721</td>
<td>0.4468</td>
<td>0.4570</td>
<td>0.4117</td>
<td>0.3919</td>
<td>0.3568</td>
<td>0.3435</td>
<td>0.3442</td>
<td>0.3534</td>
<td>0.3266</td>
<td>0.2184</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.3983</td>
<td>0.5815</td>
<td>0.3916</td>
<td>0.4249</td>
<td>0.3602</td>
<td>0.3572</td>
<td>0.2242</td>
<td>0.2247</td>
<td>0.2216</td>
<td>0.2213</td>
<td>0.2132</td>
<td>0.2060</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>0.4611</td>
<td>0.6099</td>
<td>0.4552</td>
<td>0.5212</td>
<td>0.3651</td>
<td>0.3256</td>
<td>0.2321</td>
<td>0.2342</td>
<td>0.2286</td>
<td>0.2267</td>
<td>0.2153</td>
<td>0.1391</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>0.5654</td>
<td>0.7882</td>
<td>0.5639</td>
<td>0.6372</td>
<td>0.4714</td>
<td>0.4246</td>
<td>0.2836</td>
<td>0.2859</td>
<td>0.2751</td>
<td>0.2732</td>
<td>0.2601</td>
<td>0.1659</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.5817</td>
<td>0.8029</td>
<td>0.5733</td>
<td>0.6593</td>
<td>0.4954</td>
<td>0.4592</td>
<td>0.2862</td>
<td>0.2916</td>
<td>0.2809</td>
<td>0.2781</td>
<td>0.2663</td>
<td>0.2139</td>
</tr>
<tr>
<td>Delicious</td>
<td>100</td>
<td>0.4587</td>
<td>0.6937</td>
<td>0.5420</td>
<td>0.6992</td>
<td>0.3084</td>
<td>0.2515</td>
<td>0.2208</td>
<td>0.1510</td>
<td>0.1454</td>
<td>0.1499</td>
<td>0.1302</td>
<td>0.1588</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.6147</td>
<td>0.8873</td>
<td>0.7399</td>
<td>0.8554</td>
<td>0.4399</td>
<td>0.3693</td>
<td>0.3408</td>
<td>0.2461</td>
<td>0.2281</td>
<td>0.2383</td>
<td>0.2210</td>
<td>0.2388</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>0.6957</td>
<td>0.9518</td>
<td>0.8500</td>
<td>0.9486</td>
<td>0.5460</td>
<td>0.4566</td>
<td>0.4529</td>
<td>0.3167</td>
<td>0.2854</td>
<td>0.2934</td>
<td>0.2512</td>
<td>0.2910</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>0.7383</td>
<td>0.9679</td>
<td>0.8871</td>
<td>0.9673</td>
<td>0.6044</td>
<td>0.5074</td>
<td>0.5147</td>
<td>0.3669</td>
<td>0.3277</td>
<td>0.3266</td>
<td>0.3245</td>
<td>0.3394</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.7716</td>
<td>0.9782</td>
<td>0.9232</td>
<td>0.9776</td>
<td>0.6626</td>
<td>0.4664</td>
<td>0.5708</td>
<td>0.3868</td>
<td>0.4150</td>
<td>0.3959</td>
<td>0.3189</td>
<td>0.3735</td>
</tr>
<tr>
<td>Wiki10</td>
<td>1k</td>
<td>0.1517</td>
<td>0.1356</td>
<td>0.1461</td>
<td>0.1823</td>
<td>0.0729</td>
<td>0.0813</td>
<td>0.0223</td>
<td>0.0932</td>
<td>0.0374</td>
<td>0.0234</td>
<td>0.0160</td>
<td>0.0063</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>0.2588</td>
<td>0.2211</td>
<td>0.2529</td>
<td>0.2978</td>
<td>0.1250</td>
<td>0.1036</td>
<td>0.0306</td>
<td>0.1176</td>
<td>0.0442</td>
<td>0.0325</td>
<td>0.0270</td>
<td>0.0101</td>
</tr>
<tr>
<td></td>
<td>800</td>
<td>0.4457</td>
<td>0.3950</td>
<td>0.4839</td>
<td>0.5200</td>
<td>0.2317</td>
<td>0.1817</td>
<td>0.0866</td>
<td>0.1874</td>
<td>0.0612</td>
<td>0.0461</td>
<td>0.0174</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.5408</td>
<td>0.4801</td>
<td>0.5679</td>
<td>0.6228</td>
<td>0.3102</td>
<td>0.2190</td>
<td>0.0986</td>
<td>0.3000</td>
<td>0.1075</td>
<td>0.0938</td>
<td>0.0213</td>
<td></td>
</tr>
</tbody>
</table>

Classification performance evaluation. Second, we compare the performance of each method evaluated by bipartition-based metrics, where the prediction of each label belongs to \{0, 1\} by thresholding rather than a continuous values. Figure 3 demonstrates the results with respect to micro-F1 with different batch size. Results for several other bipartition-based metrics, e.g., hamming loss, macro-F1, and instance-F1, can be found in the supplementary materials. It can be observed that Pslt achieves the best performance in most cases. Especially, Pslt outperforms others with a large margin on Delicious and Wiki10 datasets. However, traditional multi-label learning approaches, such as BR and ML-kNN, perform poorly on datasets with a relatively large number of labels because interrelationships among labels are not effectively explored.

6 CONCLUSION

In this paper, we study a novel problem setting called streaming multi-label learning, in which most extant multi-label learning approaches cannot directly applied. To solve this learning problem, we propose a new learning framework based on the probabilistic label tree. The benefits are mainly in twofolds. First, it can naturally
incorporate emerging new labels into the tree by taking advantage of label correlations between past and new labels. Second, the node classifiers modification procedure is efficient with an iteration complexity bound and we further theoretically derive an estimation error bound, which guarantees the learning performance of the proposal. Experiments in comparison with eleven baselines on four benchmark datasets verify the efficacy of our method.

REFERENCES


[17] Xin Mu, Kai Ming Ting, and Zhi-Hua Zhou. 2017. Classification under stream-plexity bound and we further theoretically derive an estimation error bound, which guarantees the learning performance of the proposal. Experiments in comparison with eleven baselines on four benchmark datasets verify the efficacy of our method.

Appendices

Appendix A shows the proof of Lemma 1, Appendix B details the proof of Theorem 1, and Appendix C shows definition of evaluation metrics. Appendix D presents additional empirical results.

A PROOF OF LEMMA 1

Lemma 1. Assume $\epsilon(\cdot)$ in Problem (2) is $\alpha$-Lipschitz and the optimal solution before and after incorporating emerging labels be $\mathbf{w}^*$ and $\mathbf{w}'$ respectively, which satisfy $|\mathbf{w}^*|_2 \leq B$ and $|\mathbf{w}'|_2 \leq B$. Then

$$f(\mathbf{y}, \mathbf{w}) - f(\mathbf{y}, \mathbf{w}') \leq \frac{4\epsilon B}{\alpha} \sum_{i=1}^{N} (y_i = 1) \wedge (y_i = -1) \tag{6}$$

Proof. We can rewrite the LHS of Inequality (6) as

$$f(\mathbf{y}, \mathbf{w}) - f(\mathbf{y}, \mathbf{w}') = f(\mathbf{y}, \mathbf{w}) - f(\mathbf{y}, \mathbf{w}') + f(\mathbf{y}, \mathbf{w}') - f(\mathbf{y}, \mathbf{w}) \tag{7}$$

where

$$f(\mathbf{y}, \mathbf{w}) = \sum_{i=1}^{N} (y_i = 1) \wedge (y_i = -1)$$

and

$$f(\mathbf{y}, \mathbf{w}') = \sum_{i=1}^{N} (y_i = 1) \wedge (y_i = -1)$$

Thus, we have

$$f(\mathbf{y}, \mathbf{w}) - f(\mathbf{y}, \mathbf{w}') \leq \frac{4\epsilon B}{\alpha} \sum_{i=1}^{N} (y_i = 1) \wedge (y_i = -1) \tag{6}$$

Hence, the proof is complete.
Note that the second term in Equation (7) is less than 0 because \( w^* \) minimizes the objective in Problem (2) with respect to \( y \). We first derive an upper bound for the first term and the bound for the last term can be similarly achieved.

\[
\begin{align*}
f(\hat{y}, w^*) - f(y, w^*) = & \mathcal{R}(w^*) - \mathcal{R}(w^*) \\
+ & C \sum_{i=1}^{N} \{ (\hat{y}_i x_i^T w^*) - (y_i x_i^T w^*) \} \\
\leq & C \sum_{i=1}^{N} \| (\hat{y}_i x_i^T w^*) - (y_i x_i^T w^*) \| \\
= & C \sum_{i=1}^{N} \{ (y_i \neq \hat{y}_i) \} \{ (x_i^T w^*) - \{ -x_i^T w^* \} \} \\
\leq & 2C \sum_{i=1}^{N} \{ (y_i \neq \hat{y}_i) \} aB \\
= & 2CaB \sum_{i=1}^{N} \{ (\hat{y}_i = 1) \} \land \{ (y_i = -1) \} \\
\end{align*}
\]

The first inequality uses the fact \( a - b \leq |a - b| \) and the second inequality uses the definition of \( f(\cdot) \) as \( \alpha \)-Lipschitz. The third inequality is since \( |x_i^T w^*| \leq \|x_i\| \|w^*\| \), and the assumption \( |w^*| \leq B \). By considering that only additional positive samples would be appended to \( \hat{y} \), we have \( \Delta_{\mathcal{F}} (\hat{y}, \tilde{y}) = \mathbb{I}(\hat{y}_i = 1) \land \mathbb{I}(y_i = -1) \). Lastly, we derive the same result for the third term in Equation (7) and complete the proof. \( \square \)

**B PROOF OF THEOREM 1**

**Theorem 1.** Assume \( f(\cdot) \) is \( \alpha \)-Lipschitz and we have a solver \( \mathcal{A} \) with sublinear convergence rate that can solve each subproblem \( k \) with precision in \( T_k = O \left( \frac{f(\hat{y}_i, w^*_k) - f(\hat{y}_i, \tilde{w}^*_k)}{\varepsilon^p} \right) \) iterations. We can bound the total number of iterations of the node classifier modification procedure by \( T_{\text{total}} = \sum_{k=1}^{K} T_k = O \left( \frac{\sum_{k=1}^{K} \{ |\mathbb{I}(y_k = 1) \} \land |\mathbb{I}(y_k = -1) \} \} }{\varepsilon^p} \right) \), where \( \Delta_{\mathcal{F}} = \sum_{k=1}^{N} \{ (\hat{y}_i = 1) \} \land \{ (y_i = -1) \} \). However, under a naive zeros initialization with \( \tilde{w}^*_k = 0 \), the upper bound is \( T_{\text{total}} = \sum_{k=1}^{K} T_k = O \left( \frac{N}{\varepsilon^p} \right) \).

**Proof.** According to Lemma 1, we have

\[
T_{\text{total}} = \sum_{k=1}^{K} T_k = O \left( \frac{\sum_{k=1}^{K} \{ |\mathbb{I}(y_k = 1) \} \land |\mathbb{I}(y_k = -1) \} \} }{\varepsilon^p} \right)
\]

By denoting \( \Delta_{\mathcal{F}} = \sum_{k=1}^{N} \{ (\hat{y}_i = 1) \} \land \{ (y_i = -1) \} \), we complete our proof for the first iteration complexity. To prove the bound for naive zeros initialization, recall that \( f(\hat{y}, 0) - f(\hat{y}, \tilde{w}^*) \leq 4\alpha \{ (\hat{y}, \tilde{w}^*) \} \) \( \alpha \) is upper bounded by \( \leq C \mathcal{N}(\mathcal{F}, 0) \). Since \( C \) and \( \ell(\hat{y}, 0) \) are typical far less than \( N \) in multi-label learning, the best upper bound for \( f(\hat{y}, 0) - f(\hat{y}, \tilde{w}^*) \) is \( O(N) \). Therefore, combining the bound for \( T_k \), we have \( T_{\text{total}} = \sum_{k=1}^{K} T_k = O \left( \frac{N}{\varepsilon^p} \right) \).

**C EVALUATION METRICS**

Given a data set \( \mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\} \), where \( x_i \in \mathbb{R}^{d \times 1} \) is a real vector representing features and \( y_i \in \{0, 1\}^m \) is the corresponding output label vector. For notational simplicity, we use \( Y_i \) to denote the index set of associated labels of \( y_i \). Formally, \( Y_i = \{ j | y_{ij} = 1 \} \) and \( Y^c_i = \{ j | y_{ij} = 0 \} \). With respect to \( j \)-th column of label matrix, \( Y_i^j = \{ i | y_{ij} = 1 \} \) denotes the index set of associated instance of the \( j \)-th label and \( Y_i^c = \{ i | y_{ij} = 0 \} \) denotes the set of non-associated instances. Table 5 summarizes evaluation metrics used in this paper. Let \( H : \mathbb{R}^d \rightarrow \{0, 1\}^m \) be a multi-label classifier and can be decomposed as \( h^1, ..., h^m \). \( H \) can be evaluated by bipartition-based metrics. \( \mathcal{F} \) is the multi-label predictor, whose predicted value could be regarded as the confidence of relevance and can be evaluated by ranking-based metrics.

**Table 5: Definitions of multi-label performance measures.**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming loss</td>
<td>( h_{\text{loss}}(H) = \frac{1}{nm} \sum_{i=1}^{nm} { y_i } )</td>
</tr>
<tr>
<td>ranking loss</td>
<td>( r_{\text{loss}}(\mathcal{F}) = \frac{1}{n} \sum_{i=1}^{n} \left</td>
</tr>
<tr>
<td>coverage</td>
<td>( c_{\text{coverage}}(\mathcal{F}) = \frac{1}{n} \sum_{i=1}^{n} { \max_{\mathcal{F}}(x_i) } )</td>
</tr>
<tr>
<td>Precision@k</td>
<td>( \text{Precision@k}(\mathcal{F}) = \frac{1}{k} \sum_{i=1}^{k} \left</td>
</tr>
<tr>
<td>macro-F1</td>
<td>( \text{macro-F1}(\mathcal{F}) = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} \left</td>
</tr>
<tr>
<td>micro-F1</td>
<td>( \text{micro-F1}(\mathcal{F}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left</td>
</tr>
<tr>
<td>instance-F1</td>
<td>( \text{instance-F1}(\mathcal{F}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \left</td>
</tr>
<tr>
<td>micro-AUC</td>
<td>( \text{micro-AUC}(\mathcal{F}) = \frac{1}{n} \sum_{i=1}^{n} { Y_i^+ } \mathcal{F}_{\mathcal{F}}(x_i) )</td>
</tr>
</tbody>
</table>

**D ADDITIONAL EXPERIMENTAL RESULTS**

- Figure 4 shows the macro-F1 and instance-F1 results for learning new labels with different batch size. Note that SLEEC, SML and SLL could not properly generate bipartite classification results, hence it is not possible to evaluate the results with the F1 score.
- Table 6 shows the performance in terms of hamming loss. Note that SLEEC, SML, and SLL could not properly generate bipartite classification results, hence it is not possible to
evaluate the results with hamming loss. Additionally, due to the sparsity of feature space, hamming loss is homogenized on EURlex and Wiki10.
Figure 4: Comparison of modeling new labels by considering 50% labels as past labels. $m$ indicates the number of new labels.

Table 6: Comparison of modeling new labels with different batch sizes by considering 50% labels as past labels. #label denotes the number of new labels. ↓(↑) means the smaller (larger) the value is, the performance will be the better.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#label</th>
<th>Hamming loss ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BR</td>
</tr>
<tr>
<td>MirFlickr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.3807</td>
<td>0.2744</td>
</tr>
<tr>
<td>6</td>
<td>0.3201</td>
<td>0.1695</td>
</tr>
<tr>
<td>9</td>
<td>0.2749</td>
<td>0.1553</td>
</tr>
<tr>
<td>12</td>
<td>0.2839</td>
<td>0.1679</td>
</tr>
<tr>
<td>15</td>
<td>0.2845</td>
<td>0.1569</td>
</tr>
<tr>
<td>Delicious</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0.4042</td>
<td>0.0140</td>
</tr>
<tr>
<td>200</td>
<td>0.4069</td>
<td>0.0131</td>
</tr>
<tr>
<td>300</td>
<td>0.4035</td>
<td>0.0156</td>
</tr>
<tr>
<td>400</td>
<td>0.4103</td>
<td>0.0167</td>
</tr>
<tr>
<td>500</td>
<td>0.4139</td>
<td>0.0173</td>
</tr>
<tr>
<td>EURlex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>0.0095</td>
<td>0.0016</td>
</tr>
<tr>
<td>400</td>
<td>0.0069</td>
<td>0.0014</td>
</tr>
<tr>
<td>600</td>
<td>0.0077</td>
<td>0.0015</td>
</tr>
<tr>
<td>800</td>
<td>0.0074</td>
<td>0.0014</td>
</tr>
<tr>
<td>1000</td>
<td>0.0078</td>
<td>0.0015</td>
</tr>
<tr>
<td>Wiki10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1k</td>
<td>0.0010</td>
<td>0.0011</td>
</tr>
<tr>
<td>2k</td>
<td>0.0007</td>
<td>0.0008</td>
</tr>
<tr>
<td>3k</td>
<td>0.0007</td>
<td>0.0008</td>
</tr>
<tr>
<td>4k</td>
<td>0.0007</td>
<td>0.0008</td>
</tr>
<tr>
<td>5k</td>
<td>0.0007</td>
<td>0.0008</td>
</tr>
</tbody>
</table>