Condensed Filter Tree for Cost-Sensitive Multi-Label Classification

Chun-Liang Li R01922001@CSIE.NTU.EDU.TW
Hsuan-Tien Lin HTLIN@CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering, National Taiwan University

Abstract

Different real-world applications of multi-label classification often demand different evaluation criteria. We formalize this demand with a general setup, cost-sensitive multi-label classification (CSMLC), which takes the evaluation criteria into account during learning. Nevertheless, most existing algorithms can only focus on optimizing a few specific evaluation criteria, and cannot systematically deal with different ones. In this paper, we propose a novel algorithm, called condensed filter tree (CFT), for optimizing any criteria in CSMLC. CFT is derived from reducing CSMLC to the famous filter tree algorithm for cost-sensitive multi-class classification via constructing the label powerset. We successfully cope with the difficulty of having exponentially many extended-classes within the powerset for representation, training and prediction by carefully designing the tree structure and focusing on the key nodes. Experimental results across many real-world datasets validate that CFT is competitive with special purpose algorithms on special criteria and reaches better performance on general criteria.

1. Introduction

The multi-label classification problem allows each instance to be associated with a set of labels simultaneously. It has in recent years attracted much attention among researchers (Tsoumakas et al., 2010; 2012) because the problem setting matches many different real-world applications; these include bio-informatics (Elisseeff & Weston, 2002), text mining (Srivastava & Zane-Ulman, 2005) and multimedia (Turnbull et al., 2008). The different applications often come with different criteria for evaluating the performance of multi-label classification algorithms. Popular criteria include the Hamming loss, the $0/1$ loss, the Rank loss, the F1 score and the Accuracy score (Tsoumakas et al., 2010).

Currently, most algorithms are designed based on none, one, or a few specific criteria. For instance, the label-wise decomposition approaches (Read et al., 2009) aim at optimizing the Hamming loss by decomposing the multi-label classification problem into several binary classification problems, one for each possible label. The label powerset approach aims at optimizing the $0/1$ loss by treating each distinct label-set as a unique extended class and reducing multi-label classification to multi-class classification. The probabilistic classifier chain (PCC) (Dembczynski et al., 2010) approach estimates the probability of each possible label-set given an instance and uses the estimate to make a Bayes-optimal decision for any loss functions, while the structured SVM approach (Petterson & Caetano, 2010; 2011) uses different convex surrogates for different evaluation criteria. Somehow both approaches require either special inference rules or loss maximizers for different evaluation criteria.

The variety of evaluation criteria calls for a more general algorithm that can cope with different criteria systematically and automatically. We formalize this need with a general setup, called cost-sensitive multi-label classification (CSMLC). CSMLC can be viewed as an extension of the popular setup of cost-sensitive multi-class classification. In CSMLC, we feed the multi-label classification algorithm with a cost function that quantifies the difference between a predicted label-set and a desired one. A general CSMLC algorithm operates on the given cost function, with the goal being better performance on that cost function. Compared with the existing methodology that requires specific design for every new application (criterion), general CSMLC algorithms can be used to save those design efforts and be easily adopted towards different application needs.

In this paper, we propose a novel algorithm for CSMLC, called the condensed filter tree (CFT). In contrast to PCC, the proposed CFT directly takes the criterion into account as the cost function during training, thereby averting the

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

need to design a specific inference rule for each new criterion and avoiding the possibly time-consuming inference step during prediction. Inspired by the rich literature of cost-sensitive multi-class classification (Domingos, 1999; Beygelzimer et al., 2008), CFT is derived by first reducing CSMLC to cost-sensitive multi-class classification via the label powerset approach. Nevertheless, the reduction leads to exponentially many extended classes, which makes training, prediction and model representation computationally challenging.

We conquer the challenge of prediction by exploiting tree-based models for cost-sensitive multi-class classification. Tree-based models use a tree structure that is constructed by binary classifiers to make fast predictions. Then we achieve time complexity logarithmic with respect to the number of extended classes, which is linear with respect to the number of possible labels. Furthermore, we conquer the challenge of model representation by proposing *proper ordering* and *K-classifier* tricks. Interestingly, the two tricks reveal a strong connection between the CFT algorithm (which is derived from the label powerset approach) and the label-wise decomposition approaches.

Finally, we conquer the training challenge by modifying the famous Filter Tree algorithm (Beygelzimer et al., 2008) for CSMLC. The modification comes from revisiting the theoretical bound of Filter Tree, which allows the proposed CFT algorithm to only focus on some key tree nodes for training efficiency.

We conduct experiments on nine real-world datasets to validate the proposed CFT algorithm. Experimental results demonstrate that for specific evaluation criteria, CFT is competitive with special-purpose algorithms, such as PCC with specifically designed inference rules or the state-of-the-art MLkNN algorithm (Zhang & Zhou, 2007). For general criteria, for which there is as yet no inference rule for PCC, CFT can reach significantly better performance. The results justify the superiority of the proposed CFT for general CSMLC problems.

2. Problem Setup

In a multi-label classification problem, we denote the feature vector by $\mathbf{x} \in \mathbb{R}^d$ and its relevant label set by $\mathcal{Y} \subseteq \{1, 2, ..., K\}$, where K is the number of classes. The label set $\mathcal{Y}$ is commonly represented as a label vector, $\mathbf{y} \in \{0, 1\}^K$, where $\mathbf{y}[k] = 1$ if and only if $k \in \mathcal{Y}$. Given a dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, which contains N *iid* training examples $(\mathbf{x}_n, \mathbf{y}_n)$ drawn from an unknown distribution $\mathcal{P}$, the goal is to design an algorithm that uses $\mathcal{D}$ to find a classifier $h\colon \mathbb{R}^d \to \{0, 1\}^K$ in the *training* stage, with the hope that $h(\mathbf{x})$ closely predicts $\mathbf{y}$ of an unseen $\mathbf{x}$ in the *prediction* stage when $(\mathbf{x}, \mathbf{y})$ is drawn from $\mathcal{P}$.

For evaluating the closeness of the prediction $\hat{\mathbf{y}} = h(\mathbf{x})$, one of the most common criteria is called the Hamming loss $\mathsf{Hamming}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{K}\sum_{k=1}^K [\![\mathbf{y}[k] \neq \hat{\mathbf{y}}[k]]\!]$. Note that the Hamming loss evaluates each label component separately and equally weighted. In addition to the Hamming loss, there are many other criteria that evaluate the components of $\hat{\mathbf{y}}$ jointly; these include the $0/1$ loss, the Rank loss, the F1 score and the Accuracy score (Tsoumakas et al., 2010). In this paper, we will use loss to denote the criterion that shall be minimized, and score to denote the criterion that shall be maximized.

The variety of criteria calls for a general setup of multi-label classification, called cost-sensitive multi-label classification (CSMLC), which will be the main focus of this work. CSMLC can be viewed as an extension of the popular setup of cost-sensitive multi-class classification (Domingos, 1999). In CSMLC, we assume that there is a known cost function (matrix) $C\colon \{0, 1\}^K \times \{0, 1\}^K \to R$, where $C(\mathbf{y}, \hat{\mathbf{y}})$ denotes the cost of predicting $(\mathbf{x}, \mathbf{y})$ as $\hat{\mathbf{y}}$. The cost matrix is not only part of the prediction stage by using $C(\mathbf{y}, h(\mathbf{x}))$ to evaluate the performance of any classifier h, but also part of the training stage by feeding C as an additional piece of information to guide the learning algorithm.

The CSMLC setup meets the goal of optimizing many existing criteria, such as the (per-example) F1 score, the Accuracy score and the Rank loss (Tsoumakas et al., 2010). Note that the setup above only considers a cost matrix C indexed by a desired vector $\mathbf{y}$ and a predicted vector $\hat{\mathbf{y}}$. Thus, the setup cannot fully cover some more complicated evaluation criteria such the micro-F1 score and the macro-F1 score, which are defined on a set of vectors. Studying the CSMLC setup can be viewed as an intermediate step toward tackling those complicated criteria in the future.

There are many existing algorithms for tackling the multi-label classification, but they either do not seriously take the cost matrix (criteria) into account, or only aim at a few specific cost matrices. That is, general algorithms for CSMLC have not been well studied. One intuitive family of algorithms is label-wise decomposition. For instance, the binary relevance (BR) algorithm (Tsoumakas et al., 2010) decomposes $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ into K binary classification datasets $\mathcal{D}_k = \{(\mathbf{x}_n, \mathbf{y}_n[k])\}_{n=1}^N$, and trains K independent binary classifiers h_k with $\mathcal{D}_k$ for predicting $\mathbf{y}[k]$. One extension of BR is the classifier chain (CC) algorithm (Read et al., 2009), which takes $\mathcal{D}_k = \{(\mathbf{z}_n, \mathbf{y}_n[k])\}_{n=1}^N$ and $\mathbf{z}_n = (\mathbf{x}_n, \mathbf{y}_n[1], ..., \mathbf{y}_n[k-1])$, to train h_k. One practical variant of CC, named CC-P, takes the predicted labels $\hat{\mathbf{y}}[k]$ instead of the true labels $\mathbf{y}[k]$ as the features in $\mathbf{z}_n$.

Because CC-P (as well as BR/CC) predicts $\hat{\mathbf{y}}[k]$ separately by each h_k, arguably their main goal is to minimize the

Hamming loss (Tsoumakas et al., 2010). Extending CC-P for general CSMLC, however, is non-trivial, because it is difficult to embed the $2^K \times 2^K$ possible $C(\mathbf{y}, \hat{\mathbf{y}})$ cost components into K separate steps of training. One algorithm that solves the difficulty for some specific cost matrices is the probabilistic classifier chain (PCC) (Dembczynski et al., 2010). PCC avoids the embedding issue in training by adopting a soft version of CC-P/CC without any cost information to estimate the conditional probability $P(\mathbf{y}|\mathbf{x})$. The probabilistic view allows PCC to interpret CC as greedily maximizing the $0/1$ loss through the chain rule. During prediction, PCC considers the cost matrix for making the Bayes-optimal decision, which is based on an efficient inference rule that has been specifically designed for the cost matrix.

The potential drawback of PCC is that not only is it non-trivial to estimate the conditional probability but it is also challenging to design an efficient inference rule for each cost matrix. Because of the latter challenge, PCC currently can be used only to exactly tackle the Hamming loss, Rank loss, and the F1 score (Dembczynski et al., 2010; 2012a; 2011). PCC can also be used with some search-based inference rule to approximately optimize the $0/1$ loss (Dembczynski et al., 2012b; Kumar et al., 2013), but not other criteria in CSMLC.

Another major algorithm, known as label powerset (LP), reduces multi-label classification to multi-class classification (Tsoumakas et al., 2010). LP treats each unique pattern of the label vector as a single extended class. That is, the K possible labels are encoded to 2^K extended classes via a bijection function $enc\colon \{0,1\}^K \to \{1, ..., 2^K\}$. During training, LP transforms $\mathcal{D}$ into $\mathcal{D}_m = \{(\mathbf{x}_n, c_n)\}_{n=1}^N$, where $c_n = enc(\mathbf{y}_n)$, and trains a multi-class classifier h_m from $\mathcal{D}_m$. Then during prediction, LP takes $h(\mathbf{x}) = enc^{-1}(h_m(\mathbf{x}))$. Trivially, LP focuses on the $0/1$ loss, because the error rate of h_m in the reduced problem is equivalent to the $0/1$ loss of h. The disadvantage is that the exponentially many extended classes makes LP infeasible and impractical in general.

Lo et al. (2011) propose the CS-RAKEL algorithm that optimizes some weighted Hamming loss by extending from RAKEL (Tsoumakas & Vlahavas, 2007), a representative algorithm between the label-wise decomposition and label powerset approaches. Somehow CS-RAKEL is designed for specific application needs and cannot tackle general CSMLC problems.

In summary, some related algorithms and their corresponding criteria are shown below. None of them can tackle general CSMLC problems.

Algorithms	Criteria Being Optimized
CC-P/CC	Hamming loss or $0/1$ loss
PCC	Hamming loss, F1 score, Rank loss, $0/1$ loss
LP	$0/1$ loss
CS-RAKEL	Weighted Hamming loss

3. Tree Model for CSMLC

Inspired by the connection between CSMLC and the rich literatures of cost-sensitive classification (Domingos, 1999; Beygelzimer et al., 2008), we design a general CSMLC algorithm via the connection. Note that LP reduces multi-label classification to multi-class classification to optimize the $0/1$ loss. If we follow the same reduction step but start from a general CSMLC problem, we end up with a cost-sensitive classification problem of 2^K extended classes and (implicitly) a $2^K \times 2^K$ cost matrix. Then any existing cost-sensitive classification algorithms can be used to solve CSMLC. We call this preliminary algorithm cost-sensitive label powerset (CS-LP). As with LP, the exponential number of extended classes presents a computational challenge for CS-LP. For example, using CS-LP to reduce CSMLC to the weighted-all pair approach (Beygelzimer et al., 2005) requires $\frac{2^K(2^K-1)}{2}$ comparisons for making each prediction.

Interestingly, PCC can be viewed as a special case of using CS-LP to reduce CSMLC to the famous Meta-Cost approach (Domingos, 1999). Meta-Cost estimates the conditional probability during training and then makes the Bayes optimal decision with respect to a cost matrix in prediction. Similarly, PCC estimates the probability by CC-P/CC, and then infers the optimal decision with respect to the cost matrix by the specifically designed inference rule.

We take another route that uses CS-LP to reduce CSMLC to tree models for cost-sensitive classification (Beygelzimer et al., 2008). A similar idea based on using the Hamming loss has been discussed in a blog post (Mineiro, 2011), but the idea has not been seriously studied for general CSMLC problems. Tree models form a binary tree by weighted binary classifiers to conduct cost-sensitive classification. Each non-leaf node of the tree is a binary classifier for deciding which subtree to go to, and each leaf node represents a class. Without loss of generality, we assume that the leaf nodes are indexed orderly by $1, 2, ..., \#classes$. Making a prediction for each instance follows the decisions of binary classifiers, starting from the root to the leaf. That is, only $O\left(\log(\#classes)\right)$ decisions are required for making each prediction. In CS-LP, tree models result in $O(K)$ time for each prediction, of the same order as label-wise decomposition approaches.

Nevertheless, the number of nodes on the resulting tree structure is $O(2^K)$, which poses challenges in representation and training. We first tackle the representation challenge in this section, and then study algorithms for training the tree model in Section 4.

Proper Ordering. Recall that CS-LP needs a bijective functions $enc(\cdot)\colon \{0,1\}^K \to \{1, ..., 2^K\}$ for encoding $\mathbf{y}$ to c and decoding the predicted class $\hat{c}$ to the corresponding label vector $\hat{\mathbf{y}}$. Although a prediction $\hat{c}$ can

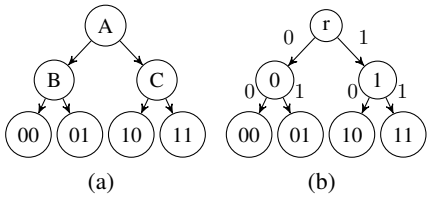


Figure 1. Proper Ordering. (a) Put labels on leaf nodes orderly (b) Index internal nodes by paths.

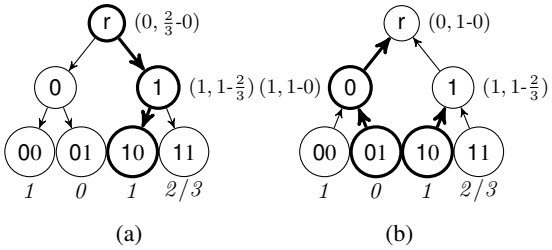


Figure 2. (a) Training of Top-down Tree; (b) Training of Filter Tree. The cost matrix used is $1-$(F1 score). (0/1, w) means the direction based on proper ordering, and the weight for training the instance on the node. The thick edge represents the prediction for the instance by the trained classifier on the parent node.

be made within $O(K)$ time in the tree model, the encoding function requires a careful design to make both enc and enc^{-1} efficient for the 2^K possible inputs to those functions. We first consider the *proper ordering* trick, which lets $enc(\mathbf{y}) = \mathsf{BinaryNumber}(\mathbf{y})+1$. That is, we treat each $\mathbf{y}$ as a binary string, encode it by computing its corresponding integer in $O(K)$, and decode accordingly, as illustrated in Fig. 1(a). Based on proper ordering, if we let $\{0, 1\}$ represent the decision $\{L, R\}$ in each classifier of the tree, then the label vector $\mathbf{y}$ (extended class c) of each leaf node is equivalent to the sequence of binary decisions made from the root to the leaf. More generally, we can index each node of the tree by $\mathbf{t} \in \{0, 1\}^{k-1}$, as shown in Fig. 1(b), where $\mathbf{t}$ is the sequence of binary decisions from the root to the node on layer k.

K-Classifier Trick. Even with proper ordering, there are $2^K - 1$ total internal nodes (classifiers) on the tree. The exponential number makes representing (and training) classifiers infeasible in practice. One existing idea for feasible representations is called the 1-classifier trick (Beygelzimer et al., 2008), which lets all $2^K - 1$ internal nodes $\mathbf{t}$ share one classifier $h(\mathbf{x}, \mathbf{t})$. Nevertheless, using the 1-classifier trick often requires the classifier to be of sufficient power to capture different characteristics of different nodes. The requirement makes the trick less suitable for practical use. Therefore, we propose a trade-off, *K-classifier trick*, between using 1 classifier and $2^K - 1$ classifiers.

The K-classifier trick physically works as follows. After proper ordering, $\hat{\mathbf{y}}[k]$ corresponds to the prediction made by one of the nodes on layer k of the tree. In other words, the purpose of all the nodes located on layer k is similar: predicting $\hat{\mathbf{y}}[k]$. The similar purpose allows us to view each node as a part of a layer classifier of the form $h_k(\mathbf{x}, \mathbf{t})$, which takes an instance $\mathbf{x}$ and a node index $\mathbf{t} \in \{0, 1\}^{k-1}$ for predicting the k-th component of the label vector. Then equivalently only K classifiers (one per each layer) are required for representing the tree.

Connection to CC-P. By the proper ordering and the K-classifier tricks, predicting the extended class $\hat{c}$ by the tree from layer 1 (root) to layer K is equivalent to predicting $\hat{\mathbf{y}}[1], ..., \hat{\mathbf{y}}[K]$ by the K classifiers $\{h_1, ..., h_K\}$ using $\mathbf{x}$ and $\mathbf{t}$. Such a prediction algorithm is exactly the same as those used in CC-P, which uses the classifiers h_k with exactly the same inputs (the instance $\mathbf{x}$ and the predicted labels which form the node index $\mathbf{t}$ in the tree) for predicting the next label.

The use of the two tricks reveals an interesting connection between two very different families of approaches: label-wise decomposition can be viewed as a special case of label powerset (in prediction). In short, label powerset with the tree model, proper ordering and K-classifiers tricks is equivalent to CC-P for prediction. Thus, by studying the role of the cost matrix during training, we can systematically extend CC-P to be cost-sensitive. Next, we will discuss how to train the K classifiers subject to the cost matrix efficiently.

4. Training of Tree Model

There are two major algorithms for training the binary classifiers in the tree, *Top-down Tree* and *Filter Tree* (Beygelzimer et al., 2008).

Top-down Tree (TT). Top-down Tree trains classifiers from layer 1 (root) to layer K. Formally, for each internal node $\mathbf{t}$ on layer k, denote its left child as $\mathbf{t}_0$ and right child as $\mathbf{t}_1$ on layer $(k+1)$. Define $\mathbf{t}^*$ as the leaf node (prediction) with the minimum cost on the subtree $T_{\mathbf{t}}$ rooted at $\mathbf{t}$. Then for each training example $(\mathbf{x}_n, \mathbf{y}_n)$ that reaches $\mathbf{t}$ during top-down training, we form a example $((\mathbf{x}_n, \mathbf{t}), b_n, w_n)$ to train the weighted classifier h_k, where the label $b_n = \arg\min_{i\in\{0,1\}} C(\mathbf{y}_n, \mathbf{t}_i^*)$ represents the optimal decision, and the weight $w_n = |C(\mathbf{y}_n, \mathbf{t}_0^*) - C(\mathbf{y}_n, \mathbf{t}_1^*)|$ represents the cost difference. Then the training examples are split to two sets based on the decision of the trained classifier h_k, and are used to train the child nodes $\mathbf{t}_0$ and $\mathbf{t}_1$, respectively. All the training examples are taken to train the root classifier, and the whole tree is trained recursively with such divide-and-conquer steps. Note that as illustrated in Fig. 2(a), each training example $(\mathbf{x}_n, \mathbf{y}_n)$ only contributes to training the nodes that are on the path from the root to the predicted leaf of the example.

The time complexity of Top-down Tree for CSMLC is the same as CC-P. In fact, if we take the Hamming loss, then $w_n = \frac{1}{K}$ is the same for each instance on every node in the k-th layer, $(\mathbf{x}_n, \mathbf{t}_n) = (\mathbf{x}_n, \hat{\mathbf{y}}_n[1], ..., \hat{\mathbf{y}}_n[k-1])$ and $b_n = \mathbf{y}_n[k]$. Thus, Top-down Tree with the Hamming loss is equivalent to CC-P. That is, general Top-down Tree can be viewed as a systematic extension of CC-P for general CSMLC.

Uniform Filter Tree (UFT). It is known that Top-down Tree may suffer from the weaker theoretical guarantee (Beygelzimer et al., 2008). An alternative algorithm is called Filter Tree, which trains the classifiers in a bottom-up manner starting from the last non-leaf layer, and each example $(\mathbf{x}_n, \mathbf{y}_n)$ is used to train all nodes. As illustrated in Fig. 2(b), the last non-leaf layer of classifiers is trained by forming weighted examples based on the better leaf of the two. After training, each node on the last layer decides the winning leaf of the two by predicting on $\mathbf{x}_n$. Then the winning labels form the new "leaves" of a smaller filter tree, and the classifiers on the upper layer are trained similarly. Due to the bottom-up manner, on layer k, Filter Tree considers all the predictions from layer $k+1$ to layer K. That is, Filter Tree split one original training example to 2^{k-1} examples, one for each possible node $\mathbf{t}$, to train *all* of the 2^{k-1} nodes on the layer. When k is large, training on layer k can thus be challenging. Compared with Top-down Tree, Filter Tree is less efficient by considering all training examples for each node, but enjoys a stronger theoretical guarantee (Beygelzimer et al., 2008).

One possibility for training Filter Tree efficiently is to only train a few nodes for each layer, with the hope that other nodes can also perform decently because of the classifier-sharing in the K-classifier trick. The original Filter Tree work (Beygelzimer et al., 2008) suggests one simple approach that splits one example to train M uniformly chosen nodes on the k-th layer to approximate the full training of 2^{k-1} nodes. We call this algorithm Uniform Filter Tree for CSMLC.

Condensed Filter Tree (CFT). In Filter Tree, there are 2^K possible traversing paths from the root to the leaves for each instance; however, many of them are seldom needed if we have reasonably good classifiers, such as paths that result in high costs. Therefore, we can shift our focus to the important nodes on each layer instead of uniform sampling for each instance. Next, we revisit the regret bound of Filter Tree, and show that the bound can be revised to focus on a *key path* of the nodes on the tree.

In CSMLC, for a feature vector $\mathbf{x}$ and some distribution $\mathcal{P}_{|\mathbf{x}}$ for generating the label vector $\mathbf{y}$, the regret rg of a classifier h on $\mathbf{x}$ is defined as

$$rg(h, \mathcal{P}) = E_{\mathbf{y}\sim\mathcal{P}_{|\mathbf{x}}}\left[C(h(\mathbf{x}), \mathbf{y})\right] - \min_g E_{\mathbf{y}\sim\mathcal{P}_{|\mathbf{x}}}\left[C(g(\mathbf{x}), \mathbf{y})\right].$$

For a distribution that generates weighted binary examples $(\mathbf{x}, b, w)$, the regret can be defined similarly by using w as the cost of a wrong prediction (of b) and 0 as the cost of a correct prediction.

Let $\mathbf{y}^* = \arg\min_{\tilde{\mathbf{y}}} E_{\mathbf{y}\sim P|\mathbf{x}} C(\mathbf{y}, \tilde{\mathbf{y}})$ be the ideal prediction of $\mathbf{x}$ under $\mathcal{P}$. When $\mathbf{t}'$ is an ancestor (prefix) of $\mathbf{t}$ on the tree, denote $\langle \mathbf{t}', \mathbf{t} \rangle$ as a list (path) that contains the nodes on the path from node $\mathbf{t}'$ to $\mathbf{t}$. We call $\langle r, \mathbf{y}^* \rangle$ the *ideal path* of the tree for $\mathbf{x}$, where r is the root of the tree. Similarly, for each node $\mathbf{t}$, we can define the ideal path of the subtree $T_\mathbf{t}$ rooted at $\mathbf{t}$. Beygelzimer et al. (2008) prove that for Filter Tree, the CSMLC regret of any tree-based classifier is upper-bounded by the total regret of all the nodes on the tree. Next, we show that the total regret of the nodes on the ideal path can readily be used to upper-bound the CSMLC regret.

Theorem 1. *Under the proper ordering and K-classifier tricks, for each $\mathbf{x}$ and the multi-label classifier h formed by chaining K binary classifiers $(h_1, ..., h_K)$ as in the prediction procedure of Filter Tree, the regret $rg(h, \mathcal{P})$ is no more than*

$$\sum_{\mathbf{t}\in\langle r, \mathbf{y}^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}) \neq \mathbf{y}[k]]\!] rg\Big(_{h_k(\mathbf{x},\mathbf{t}),\mathsf{FT}_\mathbf{t}(\mathcal{P}, h_{k+1}, ..., h_K)}\Big),$$

where k denotes the layer that $\mathbf{t}$ is on, and $\mathsf{FT}_\mathbf{t}(\mathcal{P}, h_{k+1}, ..., h_K)$ represents the procedure that generates weighted examples $(\mathbf{x}, b, w)$ to train the node at index $\mathbf{t}$ based on sampling $\mathbf{y}$ from $\mathcal{P}_{|\mathbf{x}}$ and considering the predictions of classifiers in the lower layers.

Proof. For each node $\mathbf{t}$ on layer k, h_k directs the prediction procedure to move to either the node $\mathbf{t}_0$ or $\mathbf{t}_1$. Without loss of generality, assume $h_k(\mathbf{x}, \mathbf{t}) = 1$. We denote $\hat{\mathbf{t}}$ as the prediction (leaf) on $\mathbf{x}$ when starting at node $\mathbf{t}$. For each leaf node $\tilde{\mathbf{y}}$, let $\bar{C}(\tilde{\mathbf{y}}) \equiv E_{\mathbf{y}\sim\mathcal{P}|\mathbf{x}} C(\mathbf{y}, \tilde{\mathbf{y}})$. Then the node regret $rg(\mathbf{t})$ is simply $\bar{C}(\hat{\mathbf{t}}_1) - \min_{i\in\{0,1\}} \bar{C}(\hat{\mathbf{t}}_i)$.

In addition to the regret of nodes, we also define the regret of the subtree $T_\mathbf{t}$ rooted at node $\mathbf{t}$. The regret of the subtree $T_\mathbf{t}$ is as defined as the regret of the predicted path (vector) $\hat{\mathbf{t}}$ within the subtree $T_\mathbf{t}$, that is, $rg(T_\mathbf{t}) = \bar{C}(\hat{\mathbf{t}}) - \bar{C}(\mathbf{t}^*)$, where $\mathbf{t}^*$ denotes the optimal prediction (leaf node) in the subtree $T_\mathbf{t}$. By this definition, $rg(h, \mathcal{P})$ is simply $rg(T_r)$.

The proof can be made by replacing the total regret with $rg(T_r)$ in the original Filter Tree work (Beygelzimer et al., 2008). Due to the space limit, we omit the complete proof here. □

In Theorem 2, the bound is related to certain nodes on the ideal path for each training example. The bound inspires us to first consider using each training example to

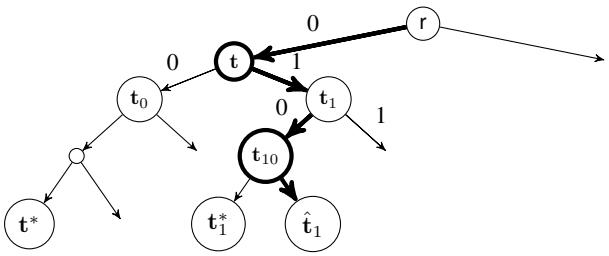


Figure 3. The thick edge represents the predition of the corresponding parent node. The ideal path is $\langle r, \mathbf{t}^* \rangle$ and $\mathbf{t}$ is the first mis-classified node; the ideal path of subtree $T_{\mathbf{t}_1}$ is $\langle \mathbf{t}_1, \mathbf{t}_1^* \rangle$ and $\mathbf{t}_{10}$ is the first mis-classified node. Both nodes $\mathbf{t}$ and $\mathbf{t}_{10}$ are on the predicted path $\langle r, \hat{\mathbf{t}}_1 \rangle$.

only train the K nodes in its ideal path to get the classifiers $h_1, .., h_K$ for each layer. Then we can find the uppermost mis-classified node $\mathbf{t}$ on the ideal path for each example $(\mathbf{x}_n, \mathbf{y}_n)$. Without loss of generality, assume $\mathbf{t}$ is on the layer k, with $\mathbf{y}[k] = 0$ and $h_k(\mathbf{x}, \mathbf{t}) = 1$. According to Theorem 2, we could decrease the regret $rg(h, \mathcal{P})$ (or $rg(T_r)$) by decreasing the node regret $rg(\mathbf{t}) = \bar{C}(\hat{\mathbf{t}}_1) - \min_{i \in \{0,1\}} \bar{C}(\hat{\mathbf{t}}_i)$, which can be done by decreasing $\bar{C}(\hat{\mathbf{t}}_1)$.[1] Because $\bar{C}(\mathbf{t}_1^*)$ is a constant, decreasing $\bar{C}(\hat{\mathbf{t}}_1)$ is equivalent to decreasing the regret, $rg(T_{\mathbf{t}_1}) = \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\mathbf{t}_1^*)$, of the subtree $T_{\mathbf{t}_1}$. We can then recursively adopt the above procedure to optimize the subtree regret $rg(T_{\mathbf{t}_1})$ as shown in Fig. 3.

The procedure suggests decreasing the regret on $\langle r, \mathbf{t} \rangle$ and $\langle \mathbf{t}, \hat{\mathbf{t}} \rangle$, the predicted path of $\mathbf{x}_n$. Therefore, the next key path for $\mathbf{x}_n$ that should be included for training is its predicted path. That is, we can now train Filter Tree by adding the predicted path for each $\mathbf{x}_n$. We call the resulting algorithm Condensed Filter Tree, as shown in Algorithm 1. The path-adding step can be repeated to further zoom into the key nodes. The number of adding step can be treated as a parameter M, and will be further discussed in Section 5.

In summary, we derive three efficient approaches for general CSMLC with trees: TT (a systematic extension of CC-P), UFT and CFT. Next, we compare them with other existing algorithms by experiments.

Algorithm 1 Condensed Filter Tree for CSMLC

$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$; $\mathcal{D}^p = \{((\mathbf{x}_n, \mathbf{y}_n), \mathbf{y}_n)\}_{n=1}^N$
for $m = 1$ **to** M iterations **do**
  for each layer k from layer K to root **do**
    $\mathcal{D}_k = \emptyset$
    for each instance $((\mathbf{x}_n, \tilde{\mathbf{y}}_n), \mathbf{y}_n) \in \mathcal{D}^p$ **do**
      $\mathbf{t} = (\tilde{\mathbf{y}}_n[1], ..., \tilde{\mathbf{y}}_n[k])$; $\mathbf{z}_n = (\mathbf{x}_n, \mathbf{t})$
      $b_n = \arg\min_{i \in \{0,1\}} C(\mathbf{y}_n, \hat{\mathbf{t}}_i)$
      w_n=$|C(\mathbf{y}_n, \hat{\mathbf{t}}_1)$-$C(\mathbf{y}_n, \hat{\mathbf{t}}_0)|$
      $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup (\mathbf{z}_n, b_n, w_n)$
    end for
    $h_k \leftarrow train(\mathcal{D}_k)$
  end for
  if $m < M$ **then**
    for each instance $(\mathbf{x}_n, \mathbf{y}_n) \in \mathcal{D}$ **do**
      $\hat{\mathbf{y}}_n = predict(h_1, ..., h_K, \mathbf{x}_n)$
      $\mathcal{D}^p \leftarrow \mathcal{D}^p \cup ((\mathbf{x}_n, \hat{\mathbf{y}}_n), \mathbf{y}_n)$
    end for
  end if
end for

5. Experiment

We conduct the experiments of different evaluation criteria on nine real-world datasets[2] (Tsoumakas et al., 2011; Read, 2012). In the experiments, we take three kinds of algorithms in our comparison: (a) the label-wise decomposition approaches, including classifier chain (CC), ensemble classifier chain (ECC), and probabilistic classifier chain (PCC); (b) the tree-based models, including top-down tree (TT), uniform filter tree (UFT) and condensed filter tree (CFT); (c) a state-of-the-art algorithm that does not explicitly take cost into account, MLkNN (Zhang & Zhou, 2007). We first consider three cost matrices: **Hamming loss**, **Rank loss**= $\sum_{\mathbf{y}[i] \rhd \mathbf{y}[j]} \left([\![\hat{\mathbf{y}}[i] < \hat{\mathbf{y}}[j]]\!] + \frac{1}{2} [\![\hat{\mathbf{y}}[i] = \hat{\mathbf{y}}[j]]\!] \right)$ and **F1 score**= $\frac{2\|\mathbf{y} \cap \hat{\mathbf{y}}\|_1}{\|\mathbf{y}\|_1 + \|\hat{\mathbf{y}}\|_1}$. The three matrices corresponds to known efficient inference rules for PCC (Dembczynski et al., 2010; 2011). Then we take other criteria for comparison in Section 5.3.

We couple PCC with L2-regularized logistic regression and other algorithms with linear support vector machines implemented in LIBLINEAR (Fan et al., 2008). For MLkNN, we use the implementation in Mulan (Tsoumakas et al., 2011). In each run of the experiment, we randomly sample 50% of the dataset for training and reserve the rest for testing. For UFT and CFT, we restrict the maximum M to 8 for efficiency. For other parameters of each algorithm, we use cross-validation on the training set to search the best choice. Finally, Tables 1, 2 and 3 list the results for the three cost matrices, respectively, with the mean and the standard error over 40 different random runs, and the best result of each dataset is bolded. We also compare CFT with other algorithms based on the t-test at 95% confidence level. The number of datasets that CFT wins, ties and losses are shown in Table 4.

[1] Since $\hat{\mathbf{t}}_0$ relates to regret of other nodes on the ideal path of $\mathbf{t}$, we cannot easily increase $\bar{C}(\hat{\mathbf{t}}_0)$ to decrease $rg(\mathbf{t})$.

[2] `CAL500`, `emotions`, `enron`, `imdb`, `medical`, `scene`, `slash`, `tmc` and `yeast`.

Table 1. The result of Hamming loss (the best (lowest) ones are marked in bold)

Dataset	CC	ECC	MLkNN	PCC	TT(CC-P)	UFT	CFT
CAL.	0.1376 ± 0.002	0.1374 ± 0.002	0.1379 ± 0.002	0.1370 ± 0.002	0.1375 ± 0.002	0.1489 ± 0.005	**0.1368 ± 0.002**
emo.	0.2613 ± 0.029	0.2501 ± 0.022	**0.2122 ± 0.012**	0.2297 ± 0.011	0.2435 ± 0.015	0.2222 ± 0.014	0.2138 ± 0.009
enron	0.0465 ± 0.001	0.0466 ± 0.001	0.0540 ± 0.001	**0.0462 ± 0.001**	0.0467 ± 0.001	0.0551 ± 0.001	0.0467 ± 0.001
imdb	0.0808 ± 0.000	**0.0713 ± 0.000**	0.0714 ± 0.000	0.0714 ± 0.000	0.0715 ± 0.000	0.0715 ± 0.000	0.0715 ± 0.000
medical	0.0109 ± 0.001	0.0113 ± 0.001	0.0176 ± 0.001	0.0110 ± 0.001	0.0108 ± 0.001	0.0119 ± 0.001	**0.0102 ± 0.001**
scene	0.1118 ± 0.004	0.0971 ± 0.004	**0.0942 ± 0.004**	0.0962 ± 0.003	0.0980 ± 0.003	0.1032 ± 0.003	0.1004 ± 0.003
slash	0.0418 ± 0.001	0.0383 ± 0.000	0.0514 ± 0.001	0.0386 ± 0.001	0.0388 ± 0.001	**0.0375 ± 0.001**	0.0383 ± 0.001
tmc	0.0571 ± 0.000	**0.0565 ± 0.000**	0.0669 ± 0.000	0.0576 ± 0.000	0.0575 ± 0.000	0.0574 ± 0.000	0.0572 ± 0.000
yeast	0.2107 ± 0.003	0.2009 ± 0.004	**0.1981 ± 0.003**	0.2006 ± 0.003	0.2000 ± 0.002	0.2008 ± 0.002	0.2013 ± 0.003

Table 2. The result of Rank loss (the best (lowest) ones are marked in bold)

Dataset	CC	ECC	MLkNN	PCC	TT	UFT	CFT
CAL.	1516.0 ± 60.4	1432.6 ± 39.0	1408.9 ± 21.3	967.93 ± 12.57	965.49 ± 11.20	968.40 ± 12.03	**963.13 ± 10.99**
emo.	2.697 ± 0.315	2.350 ± 0.299	1.906 ± 0.120	1.763 ± 0.102	1.868 ± 0.134	1.714 ± 0.131	**1.632 ± 0.093**
enron	44.190 ± 0.736	42.625 ± 0.775	55.959 ± 1.386	**24.379 ± 0.557**	25.144 ± 0.704	25.622 ± 0.576	24.907 ± 0.625
imdb	21.312 ± 0.299	22.559 ± 0.283	24.396 ± 2.345	**12.620 ± 0.044**	12.665 ± 0.047	12.638 ± 0.046	12.637 ± 0.049
medical	5.882 ± 0.595	5.800 ± 0.564	5.826 ± 0.565	2.942 ± 0.327	3.611 ± 0.431	**2.812 ± 0.291**	3.602 ± 0.455
scene	1.022 ± 0.053	0.922 ± 0.030	0.853 ± 0.046	**0.696 ± 0.024**	0.744 ± 0.029	0.764 ± 0.026	0.739 ± 0.028
slash	6.603 ± 0.132	6.467 ± 0.131	8.259 ± 0.259	**3.835 ± 0.080**	4.358 ± 0.152	3.965 ± 0.065	4.289 ± 0.074
tmc	7.704 ± 0.089	7.306 ± 0.139	5.329 ± 0.079	3.952 ± 0.034	3.924 ± 0.042	3.912 ± 0.040	**3.894 ± 0.040**
yeast	9.596 ± 0.224	9.208 ± 0.143	9.735 ± 0.247	8.753 ± 0.140	8.752 ± 0.138	8.813 ± 0.148	**8.747 ± 0.118**

Table 3. The result of F1 score (the best (highest) ones are marked in bold)

Dataset	CC	ECC	MLkNN	PCC	TT	UFT	CFT
CAL.	0.319 ± 0.028	0.368 ± 0.015	0.318 ± 0.010	0.460 ± 0.006	0.447 ± 0.006	0.454 ± 0.005	**0.473 ± 0.004**
emo.	0.416 ± 0.087	0.489 ± 0.068	0.579 ± 0.030	**0.639 ± 0.018**	0.550 ± 0.061	0.619 ± 0.029	0.637 ± 0.016
enron	0.538 ± 0.010	0.547 ± 0.011	0.385 ± 0.021	0.574 ± 0.007	0.580 ± 0.009	0.545 ± 0.011	**0.598 ± 0.010**
imdb	0.256 ± 0.001	0.157 ± 0.015	0.001 ± 0.000	0.352 ± 0.015	0.371 ± 0.001	0.358 ± 0.001	**0.374 ± 0.001**
medical	0.784 ± 0.017	0.779 ± 0.014	0.523 ± 0.038	**0.817 ± 0.015**	0.789 ± 0.021	0.797 ± 0.011	0.796 ± 0.014
scene	0.687 ± 0.012	0.701 ± 0.010	0.655 ± 0.023	**0.735 ± 0.011**	0.721 ± 0.010	0.667 ± 0.007	0.717 ± 0.010
slash	0.489 ± 0.012	0.496 ± 0.007	0.136 ± 0.054	**0.577 ± 0.008**	0.517 ± 0.011	0.540 ± 0.005	0.514 ± 0.007
tmc	0.684 ± 0.003	0.693 ± 0.003	0.606 ± 0.007	0.714 ± 0.002	0.709 ± 0.002	0.687 ± 0.002	**0.714 ± 0.002**
yeast	0.622 ± 0.007	0.634 ± 0.007	0.607 ± 0.012	0.638 ± 0.008	0.639 ± 0.005	0.649 ± 0.006	**0.649 ± 0.006**

Table 5. The result of Acc. score, and Comp. score (best ones are marked in bold)

Dataset	Accuracy(↑)		Composite Score(↑)	
	PCC-F1	CFT	PCC-Ham or F1	CFT
CAL.	0.303 ± 0.008	**0.315 ± 0.004**	−0.362 ± 0.012	**−0.302 ± 0.013**
emo.	0.534 ± 0.021	**0.535 ± 0.015**	−0.566 ± 0.100	**−0.460 ± 0.063**
enron	0.453 ± 0.009	**0.476 ± 0.009**	0.300 ± 0.017	**0.351 ± 0.012**
imdb	0.242 ± 0.010	**0.268 ± 0.001**	−0.263 ± 0.055	**−0.096 ± 0.001**
medical	**0.783 ± 0.015**	0.764 ± 0.018	**0.758 ± 0.016**	0.747 ± 0.018
scene	**0.676 ± 0.011**	0.669 ± 0.010	0.150 ± 0.036	**0.170 ± 0.022**
slash	**0.511 ± 0.009**	0.481 ± 0.006	0.263 ± 0.012	**0.277 ± 0.011**
tmc	0.613 ± 0.004	**0.614 ± 0.002**	0.402 ± 0.007	**0.419 ± 0.004**
yeast	0.518 ± 0.012	**0.539 ± 0.006**	−0.398 ± 0.019	**−0.376 ± 0.019**

Table 4. CFT versus the other algorithms based on t-test at 95% confidence level (#win/#tie/#loss)

criteria	CC	ECC	MLkNN	PCC	TT	UFT
Ham.	7/1/1	2/4/3	5/1/3	4/3/2	5/2/2	6/2/1
Rank.	9/0/0	9/0/0	9/0/0	3/2/4	4/5/0	6/1/2
F1.	9/0/0	9/0/0	9/0/0	4/2/3	7/1/1	6/2/1
Total	25/1/1	20/4/3	23/1/3	11/7/9	16/8/3	18/5/4

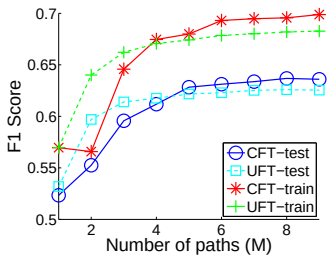
5.1. Cost-insensitive versus Cost-sensitive

Table 1 compares all the algorithms based on the Hamming loss. As discussed in Section 4, CC-P is equivalent to TT with Hamming loss. In Table 1, the five algorithms that can reach the best performance are ECC, MLkNN, PCC, UFT and CFT. Moreover, ECC successfully improves the performance of CC. The state-of-the-art algorithm, MLkNN, often achieves the best results. When looking at Table 4 for t-test results, CFT is competitive to ECC and PCC, while often being better than MLkNN.

For the other two criteria, as shown in Tables 2 and 3, the algorithms that do not consider the cost explicitly, such as CC, ECC and MLkNN, are generally worse than the cost-sensitive algorithms. The results demonstrate the importance and effectiveness of properly considering the cost information in the algorithm.

5.2. Comparison with Tree-based Algorithms

In Tables 1, 2, 3 and 4, when comparing CFT with TT, CFT wins on 16 and ties on 8 of the 27 cases by t-test. The results justify the importance of bottom-up training of the tree model. When comparing UFT with CFT, CFT is better than UFT on 18 and ties on 5 out of 27 cases by t-test. The results demonstrate the effectiveness of focusing on key paths (nodes).



We further study UFT and CFT for varying M. We show the result of F1 score on `emotions`, while observing similar behaviors across other datasets and criteria. When number of paths increases, both the training and testing performance of CFT and UFT are improved. Moreover, CFT converges to a better F1 score than UFT as M increases, which explains its better performance during testing.

While CFT is usually better than UFT, on `medical` and `slash`, CFT loses to UFT in Tables 2 and 3. We study the reasons and find that the cause is overfitting. For instance, the training Rank loss of CFT on `medical` is 0.083, which is much smaller than the UFT result of 0.264. That result implies that CFT indeed optimizes the desired evaluation criteria during training, but the focus on key paths could suffer worse generalization in a few datasets. A preliminary study shows that a mixture of CFT and UFT is less prone to overfitting.

5.3. Comparison with PCC and CFT

For the Hamming loss, the Rank loss and the F1 score, the exact inference algorithms of PCC have been proposed (Dembczynski et al., 2010; 2011). From Tables 1, 2, 3 and 4, PCC and CFT are competitive to each other on the three criteria, having similar number of winning and losing cases.

To demonstrate the full ability of CFT, we consider two other criteria which there is no inference rule (yet) for PCC, including Accuracy[3] (Boutell et al., 2004; Tsoumakas et al., 2010), and a composite score from the F1 score and the Hamming loss in Table 5. The definitions of the criteria are **Accuracy** $= \frac{\|\mathbf{y} \cap \hat{\mathbf{y}}\|_1}{\|\mathbf{y} \cup \hat{\mathbf{y}}\|_1}$, and **Composite Score**$=$ F1 Score$-5\times$HAM Loss.

Here we use the approximate inference rules for PCC. For the Accuracy score, we couple PCC with the inference rule of the F1 score in view of the similarity in the formula. For the Composite score, which considers the F1 score and the Hamming loss concurrently, we run PCC with either the inference rule of the F1 score or the inference rule of the Hamming loss, and optimistically report the best one in Table 5.

Table 5 can be summarized as follows. Due to the similarity in the formula, CFT and PCC-F1 reach similar results for the Accuracy score. For the Composite score, which is similar to neither the F1 score nor the Hamming loss, PCC is much worse than CFT.

When K is small, PCC can use exhaustive search to enumerate 2^K possible $\hat{\mathbf{y}}$ and locate the Bayes optimal $\hat{\mathbf{y}}$. We further list the performance of this PCC-exhaust approach for `emotions`, `scene` and `yeast`, which are of no more than 14 labels.

Infer.	Acc.(↑)			Comp.(↑)		
	emo.	scene	yeast	emo.	scene	yeast
Apprx.	**0.534**	0.676	0.518	**-0.566**	0.150	-0.398
Exhau.	0.530	**0.709**	**0.535**	-0.570	**0.176**	**-0.383**

By the exhaustive inference, the performance of PCC is significantly improved in most cases. The good performance highlights the importance of exact and efficient inference rules for PCC. Nevertheless, if the desired evaluation criteria are complicated, it is non-trivial to design exact and efficient inference rules. When comparing PCC-exhaust with CFT, we see that CFT wins on 3 cases, ties on 1 case and loses on 2 cases. Thus, the efficient CFT is quite competitive with the inefficient PCC-exhaust in performance.

6. Conclusion

We tackle the general cost-sensitive multi-label classification problem without any specific subroutine for different evaluation criteria, which meets the demands in real-world applications. We proposed the condensed filter tree (CFT) algorithm by coupling several tools and ideas: the label powerset approach for reducing to cost-sensitive classification, the tree-based algorithms for cost-sensitive classification, the proper-ordering and K-classifier tricks that utilize the structural property of multi-label classification, and the theoretical bound to locate the key tree nodes (paths) for training. The resulting CFT is as efficient as the common label-wise decomposition approaches in training and prediction, with respect to the number of possible labels. Experimental results demonstrate that CFT is competitive with leading approaches for multi-label classification, and usually outperforms those approaches on the evaluation criteria that those approaches are not designed from.

CFT can currently handle evaluation criteria defined by a desired label vector and a predicted label vector. We can view CFT as the first step towards tackling more complicated evaluation criteria, which shall be an important future research direction.

7. Acknowledgement

We thank Profs. Yuh-Jye Lee, Chih-Jen Lin, Shou-De Lin, Chi-Jen Lu, Hung-Yi Lo, the anonymous reviewers, and the members of the NTU Computational Learning Lab for valuable suggestions. This work is mainly supported by National Science Council (NSC 101-2628-E-002-029-MY2) of Taiwan.

A. Proof of Theorem 2

Theorem 2. *Under the proper ordering and K-classifier tricks, for each $\mathbf{x}$ and the multi-label classifier h formed by chaining K binary classifiers $(h_1, ..., h_K)$ as in the predic-*

[3] α-Accuracy with $\alpha = 1$

tion procedure of Filter Tree, the regret $rg(h, \mathcal{P})$ is

$$rg(h,\mathcal{P}) \leq \sum_{\mathbf{t}\in\langle r,\mathbf{y}^*\rangle} [\![h_k(\mathbf{x},\mathbf{t}) \neq \mathbf{y}[k]]\!] rg\Big(h_k(\mathbf{x},\mathbf{t}), \mathsf{FT}_{\mathbf{t}}(\mathcal{P}, h_{k+1}, ..., h_K)\Big),$$

where k denotes the layer that $\mathbf{t}$ is on, and $\mathsf{FT}_{\mathbf{t}}(\mathcal{P}, h_{k+1}, ..., h_K)$ represents the procedure that generates weighted examples $(\mathbf{x}, b, w)$ to train the node at index $\mathbf{t}$ based on sampling $\mathbf{y}$ from $\mathcal{P}_{|\mathbf{x}}$ and considering the predictions of classifiers in the lower layers.

Proof. The proof is similar to the one in (Beygelzimer et al., 2008), which is based on defining the overall-regret of any subtree. The key change in our proof is to define the *path-regret* of any subtree to be the total regret of the nodes on the ideal path of the subtree. The induction step follows similarly from the proof in (Beygelzimer et al., 2008) by considering two cases: one for the ideal prediction to be in the left subtree and one for the ideal prediction to be in the right. Then an induction from layer K to the root proves the theorem.

For each node $\mathbf{t}$ on layer k, h_k makes a weighted binary classification decision of 0 or 1, which directs the prediction procedure to move to either the node $\mathbf{t}_0$ or $\mathbf{t}_1$. Without loss of generality, assume $h_k(\mathbf{x}, \mathbf{t}) = 1$. We denote $\hat{\mathbf{t}}$ as the prediction (leaf) on $\mathbf{x}$ when starting at node $\mathbf{t}$. For each leaf node $\tilde{\mathbf{y}}$, let $\bar{C}(\tilde{\mathbf{y}}) \equiv E_{\mathbf{y}\sim\mathcal{P}|\mathbf{x}} C(\mathbf{y}, \tilde{\mathbf{y}})$. Then the node regret $rg(\mathbf{t})$ is simply $\bar{C}(\hat{\mathbf{t}}_1) - \min_{i\in\{0,1\}} \bar{C}(\hat{\mathbf{t}}_i)$. Obviously, $rg(\mathbf{t}) \geq \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\hat{\mathbf{t}}_0)$ for all node $\mathbf{t}$.

In addition to the regret of nodes, we also define the regret of the subtree $T_{\mathbf{t}}$ rooted at node $\mathbf{t}$. The regret of the subtree $T_{\mathbf{t}}$ is as defined as the regret of the predicted path (vector) $\hat{\mathbf{t}}$ within the subtree $T_{\mathbf{t}}$, that is, $rg(T_{\mathbf{t}}) = \bar{C}(\hat{\mathbf{t}}) - \bar{C}(\mathbf{t}^*)$, where $\mathbf{t}^*$ denotes the optimal prediction (leaf node) in the subtree $T_{\mathbf{t}}$. By this definition, $rg(h, \mathcal{P})$ can be treated as $rg(T_r)$.

We now prove by induction from layer K to the root. The induction hypothesis is that

$$rg(T_{\mathbf{t}}) \leq \sum_{\mathbf{t}'\in\langle\mathbf{t},\mathbf{t}^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}'),$$

where k is the corresponding layer of each node $\mathbf{t}'$. The hypothesis states that the regret of the subtree is bounded by the sum of the regrets for the wrongly predicted nodes from $\mathbf{t}$ to the ideal prediction $\mathbf{t}^*$. The base case is the reduction tree with one single internal node $\mathbf{t}$ and two leaf nodes, which is a cost-sensitive binary classification with $rg(T_{\mathbf{t}}) = rg(\mathbf{t})$ trivially. If h_1 predicts correctly, then $rg(T_{\mathbf{t}}) = 0$. Otherwise $rg(T_{\mathbf{t}}) = rg(\mathbf{t})$. Then the induction hypothesis is satisfied.

For the inductive step, for node $\mathbf{t}$ on layer k, assume

$$R_0 \equiv rg(T_{\mathbf{t}_0}) \leq \sum_{\mathbf{t}'\in\langle\mathbf{t}_0,\mathbf{t}_0^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}'),$$

and

$$R_1 \equiv rg(T_{\mathbf{t}_1}) \leq \sum_{\mathbf{t}'\in\langle\mathbf{t}_1,\mathbf{t}_1^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}').$$

The optimal prediction $\mathbf{t}^*$ is either on the right subtree T_1 or the left subtree T_0. For the first case, it implies $\mathbf{t}^* = \mathbf{t}_1^*$ and $\mathbf{y}[k] = h_k(\mathbf{x}, \mathbf{t}) = 1$, then

$$\begin{aligned} rg(T_{\mathbf{t}}) &= \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\mathbf{t}^*) \\ &= \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\mathbf{t}_1^*) \\ &= R_1 \leq \sum_{t'\in\langle\mathbf{t}_1,\mathbf{t}_1^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}') \\ &= \sum_{\mathbf{t}'\in\langle\mathbf{t},\mathbf{t}^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}'). \end{aligned}$$

For the second case, it implies $\mathbf{t}^* = \mathbf{t}_0^*$ and $\mathbf{y}[k] \neq h_k(\mathbf{x}, \mathbf{t}) = 1$, then

$$\begin{aligned} rg(T_{\mathbf{t}}) &= \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\mathbf{t}^*) \\ &= \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\mathbf{t}_0^*) \\ &= \bar{C}(\hat{\mathbf{t}}_1) - \bar{C}(\hat{\mathbf{t}}_0) + \bar{C}(\hat{\mathbf{t}}_0) - \bar{C}(\mathbf{t}_0^*) \\ &\leq rg(\mathbf{t}) + R_0 \\ &\leq rg(\mathbf{t}) + \sum_{\mathbf{t}'\in\langle\mathbf{t}_0,\mathbf{t}_0^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}') \\ &= \sum_{\mathbf{t}'\in\langle\mathbf{t},\mathbf{t}^*\rangle} [\![h_k(\mathbf{x}, \mathbf{t}') \neq \mathbf{y}[k]]\!] rg(\mathbf{t}'). \end{aligned}$$

Then we complete the induction. □

Table 6. The properties of each dataset

Dataset	# Instances	# Labels (K)
CAL500	502	174
emotions	593	6
enron	1702	53
imdb	86290	28
medical	662	45
scene	2407	6
slash	3279	22
tmc	28596	22
yeast	2389	144

B. Datasets

Here we summarize the basic statistics of the used datasets in Table 6.

References

Beygelzimer, A., Dani, V., Hayes, T., Langford, J., and Zadrozny, B. Error limiting reductions between classification tasks. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

Beygelzimer, A., Langford, J., and Ravikumar, P. Error correcting tournaments, 2008. URL http://arxiv.org/abs/0902.3176.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. Learning multi-label scene classification. *Pattern Recognition*, 2004.

Dembczynski, K., Cheng, W., and Hüllermeier, E. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th International Conference on Machine learning*, 2010.

Dembczynski, K., Waegeman, W., Cheng, W., and Hüllermeier, E. An exact algorithm for f-measure maximization. In *Advances in Neural Information Processing Systems 24*. 2011.

Dembczynski, K., Kotlowski, W., and Hüllermeier, E. Consistent multilabel ranking through univariate losses. In *Proceedings of the 29th International Conference on Machine learning*, 2012a.

Dembczynski, K., Waegeman, W., and Hüllermeier, E. An analysis of chaining in multi-label classification. In *Proceedings of the 20th European Conference on Artificial Intelligence*, 2012b.

Domingos, P. Metacost: a general method for making classifiers cost-sensitive. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999.

Elisseeff, A. and Weston, J. A kernel method for multilabelled classification. In *Advances in Neural Information Processing Systems 14*, 2002.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 2008.

Kumar, A., Vembu, S., Menon, A. K., and Elkan, C. Beam search algorithms for multilabel learning. *Machine Learning*, 2013.

Lo, H.-Y., Wang, J.-C., Wang, H.-M., and Lin, S.-D. Cost-sensitive multi-label learning for audio tag annotation and retrieval. *IEEE Transactions on Multimedia*, 2011.

Mineiro, P. Cost sensitive multi label: an observation, 2011. URL http://www.machinedlearnings.com/2011/05/cost-sensitive-multi-label-observation.html.

Petterson, J. and Caetano, T. S. Reverse multi-label learning. In *Advances in Neural Information Processing Systems 23*. 2010.

Petterson, J. and Caetano, T. S. Submodular multi-label learning. In *Advances in Neural Information Processing Systems 24*. 2011.

Read, J. Meka: a multi-label extension to weka, 2012. URL http://meka.sourceforge.net.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. Classifier chains for multi-label classification. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, 2009.

Srivastava, A.N. and Zane-Ulman, B. Discovering recurring anomalies in text reports regarding complex space systems. In *IEEE Aerospace Conference*, 2005.

Tsoumakas, G. and Vlahavas, I. Random k-labelsets: an ensemble method for multilabel classification. In *Machine Learning: the European Conference on Machine Learning*. 2007.

Tsoumakas, G., Katakis, I., and Vlahavas, I. Mining multilabel data. In *Data Mining and Knowledge Discovery Handbook*. Springer US, 2010.

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., and Vlahavas, I. Mulan: a java library for multi-label learning. *Journal of Machine Learning Research*, 2011.

Tsoumakas, G., Zhang, M.-L., and Zhou, Z.-H. Introduction to the special issue on learning from multi-label data. *Journal of Machine Learning Research*, 2012.

Turnbull, D., Barrington, L., Torres, D. A., and Lanckriet, G. R. G. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech and Language Processing*, 2008.

Zhang, M.-L. and Zhou, Z.-H. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 2007.