Learning Tree Structure of Label Dependency for Multi-label Learning*

Bin Fu[1], Zhihai Wang[1], Rong Pan[2], Guandong Xu[3], and Peter Dolog[2]

[1] School of Computer and Information Technology,
Beijing Jiaotong University, Beijing 100044, China
{09112072,zhhwang}@bjtu.edu.cn

[2] Department of Computer Science, Aalborg University, Denmark
{rpan,dolog}@cs.aau.dk

[3] School of Engineering & Science, Victoria University, Australia
Guandong.Xu@vu.edu.au

Abstract. There always exists some kind of label dependency in multi-label data. Learning and utilizing those dependencies could improve the learning performance further. Therefore, an approach for multi-label learning is proposed in this paper, which quantifies the dependencies of pairwise labels firstly, and then builds a tree structure of the labels to describe them. Thus the approach could find out potential strong label dependencies and produce more generalized dependent relationships. The experimental results have validated that compared with other state-of-the-art algorithms, the method is not only a competitive alternative, but also has shown better performance after ensemble learning especially.

Keywords: classification; multi-label instance; multi-label learning; label dependency.

1 Introduction

Classification is to predict possible labels on unlabeled instance given a set of labeled training instances. Traditionally, it is assumed that each instance is associated with only one label. However, an instance often has multiple labels simultaneously in practice [1,2]. For example, a report about *religion* could also be viewed as a *politics* report. Classification for this kind of instance is called multi-label learning. Nowadays, multi-label learning is receiving more and more concerns, and becoming an important topic.

Various methods have been developed for multi-label learning, and these methods mainly fall into two categories [2]: (1) algorithm adaptation, which extends traditional single-label models so that they can deal with multi-label instances directly. Several adapted traditional models include Bayesian method,

* This research has been partially supported by the Fundamental Research Funds for the Central Universities, China (2011YJS223), National Natural Science Fund of China (60673089, 60973011), and EU FP7 ICT project M-Eco: Medical Ecosystem Personalized Event-based Surveillance (No. 247829).

AdaBoost, decision tree, associative rules, k-NN, etc. [3,4,5,6,7]. (2) problem transformation, which converts a multi-label problem into one or several single-label problems. Thus traditional single-label classifiers can be used directly without modification. Recently, many methods have been proposed to learn label dependency as a way of increasing learning performance [1,2,8,9,10,11,12]. However, most of them do not give an explicit description of label dependency. For example, classifier chain, a model proposed recently [9], links the labels into a chain randomly and assumes that each label is dependent on all its preceding labels in the chain. However, each label may be independent with its preceding labels while dependent on its following labels since they are linked randomly. Moreover, more complex dependency such as a tree or DAG-like hierarchical structure of labels often exists in practice, thus more appropriate models are needed to describe them.

Hence, we propose one kind of novel method for aforementioned issues. We quantify the dependencies of pairwise labels firstly, building a complete undirected graph that takes the labels as the set of vertices and the dependent values as edges. A tree is then derived to depict the dependency explicitly, so the unrelated labels can be removed for each label and the dependency model is generalized into a tree model. Furthermore, we also use ensemble technique to build multiple trees to capture the dependency more accurately. The experimental results would show our proposed method is competitive and could further enhance learning performance on most of datasets.

The remainder of this paper is organized as follows: We review the related works in section 2. A formal definition of multi-label learning is given in section 3. In section 4, we describe and analyze our proposed methods in detail. Section 5 is devoted to the experiment design and result analysis. The last section concludes this paper and gives some potential issues with further research.

2 Related Work

Many methods have been proposed to cope with multi-label learning by exploiting label's dependencies. According to the order of dependency be learned, these methods mainly fall into following categories.

(1) No label dependency is learned. Basic BR (Binary Relevance) method decomposed one multi-label problem into multiple independent binary classification problems, one for each label [2]. Boutell et al. used BR for scene classification [13]. Zhang et al. proposed ML-KNN, a lazy method based on BR [7]. Tsoumakas et al. proposed HOMER to deal with a large number of labels [14].

(2) Learning the dependencies of pairwise labels. Hullermeier et al. proposed the RPC method that learned the pairwise preferences and then ranked the labels [15]. Furnkranz et al. extended the RPC by introducing a virtual label [16]. Madjarov et al. proposed a two stage architecture to reduce the computational complexity of the pair-wise methods [17].

(3) Learning the dependencies within multiple labels. Basic LP (Label Powerset) method treated the whole set of labels as a new single label and learned dependencies within all them [2]. Tsoumakas et al. proposed the RAkEL$_d$

method that divided the label set into disjoint subsets of size k randomly [18]. Stacking-base method was proposed to aggregate binary predictions to form meta-instances [19]. Read proposed the PS, which decomposed the instance's labels until a threshold was met [8]. Read et al. proposed the CC (Classifier Chain) algorithm to link the labels into a chain randomly [9]. Dembcynski et al. proposed PCC, a probabilistic framework that solved the multi-label classification in terms of risk minimization [10].

A number of models have also been used to depict the labels dependencies explicitly, which include multi-dimensional Bayesian network, conditional dependency networks, conditional random fields [11,20,21,22,23]. Dembczynski et al. formally explained the difference between the conditional dependency and unconditional dependency [24]. Similar with these methods, our proposed method uses the tree to learn the label dependency explicitly. the difference is that we simply ignore the feature set in the process of constructing a tree, whereas others [11] build the models conditioned on the feature set.

3 The Concept of Multi-label Learning

Let X be the instance space, and $L = (l_1, l_2, \dots, l_m)$ be a set of labels. Given a training instance set $D = \{(x_1, C_1), (x_2, C_2), \dots, (x_n, C_n)\}$, where $x_k \in X$ is an instance, and $C_k \subset L$ is a subset of L denoting x_k's true labels, the target of multi-label learning is to build a classifier: $f : X \rightarrow 2^L$, that is a mapping from the instance space to a set of label subset, where 2^L is the power set of L. C_k can also be represented by a Boolean vector $(b_{k1}, b_{k2}, \dots, b_{km})$, where $b_{kj} = 1$ indicates label l_j is x_k's true label ($l_j \in C_k$), while $b_{kj} = 0$ indicates the opposite.

Let $x \in X$ be an unlabeled instance, $y = (y_1, y_2, \dots, y_m)$ be its Boolean vector of predicted labels, we can also make prediction by calculating the joint conditional probability$P(y|x)$. For each label l_k, let $Parent(l_k)$ denote the set of labels that label l_k is dependent on, $Parent(y_k)$ is the corresponding Boolean counterpart. Hence $P(y|x)$ can be transformed as Eq.(1).

$$P(y|x) = \prod_{k=1}^{m} P(y_k|parent(y_k), x) \tag{1}$$

where y_k denotes the kth label. Hence we can get the label vector's posterior probability by calculating each label's posterior probability respectively, so the transformation of Eq.(1) is a kind of problem transformation. A key issue is how to exactly find the set of dependent labels for each label in order to calculate the posterior probability more accurately.

4 Learning a Tree Structure of Labels

As mentioned above, to eliminate weak dependencies in CC model, and fit the real data more accurately, we propose a new algorithm named as LDTS (Learning dependency from Tree Structure of Labels) in this section.

LDTS firstly measures the dependency for each pairwise labels l_i and l_j, notated as $dependency(l_i, l_j)$, thus an undirected complete graph $G(L, E)$ is constructed, where the label set L denotes the vertices, and $E = \{dependency(l_i, l_j) : l_i \in L, l_j \in L\}$ denotes the edges. To determine the dependent labels for each label, a maximum spanning tree is then derived using Prim algorithm, and each label is assumed to be dependent on its ancestor labels. A dataset is then created for each label and their dependent labels are added into the feature set, so we could utilize these dependency since the classifiers is trained based on the new feature set. The whole training process is outlined in Algorithm 1.

Algorithm 1. The process of training LDTS classifier

```
Input:
    The training dataset: $D = \{(x_1, C_1), (x_2, C_2), \ldots, (x_n, C_n)\}$;
    The algorithm for training base classifier: $B$.
Output:
    Classifiers: $(f_1, f_2, \ldots, f_m)$.
 1: for each pair of labels $(l_i, l_j)$, measure their dependency: $dependency(l_i, l_j)$
 2: create a undirected full graph $G = (L, E)$
 3: use the $Prim$ algorithm to derive a maximum spanning tree: $T$
 4: for label $l_1, l_2, ..., l_m$, get the set consists of its ancestor labels: $Parent(l_i)$
 5: for $i = 0$ to $m$ do
 6:   let the $D_i = D$
 7:   for $j = 0$ to $m$ do
 8:     if $l_j \notin Parent(l_i)$ then
 9:       delete this label from $D_i$
10:     end if
11:   end for
12:   for the $D_i$, set $l_i$ as its only label, train the classifier $f_i$
13: end for
14: return the $m$ classifiers: $(f_1, f_2, \ldots, f_m)$
```

At step 1, mutual information is used to compute the dependencies of pairwise labels. Its definition is shown as follows [25].

$$H(X, Y) = \sum_{x,y} P(x, y) \log \frac{P(x, y)}{P(x)P(y)} \tag{2}$$

where X and Y are two variables, x and y are their all possible values.

The labels is organized using a tree for two purposes. Firstly, the properties of maximum spanning tree ensure that each label is more dependent on its ancestor labels than other labels, since they have greater mutual information value. Hence it could eliminate weak dependency further by assuming each label is only dependent on its ancestor labels. When generating the graph and tree of labels, we simply assume that label dependency is independent with the feature set and only consider the mutual influence among the labels, this is one kind of the unconditional dependency described in [24]. Secondly, various kinds of dependencies including tree hierarchy and DAG of dependency may exist within labels. Therefore, we expect the performance could be improved, especially on the datasets in which the labels are organized into a tree indeed.

It is also should be noted that the main purpose of our method is to find more accurate label dependency, and it does not impose a hierarchy on labels since labels have the same parent or in different paths could exist simultaneously. This is different from the hierarchical classification that impose a strict label hierarchy. Although the randomness in not eliminated fully, we do reduce it to only select the strong dependency randomly. One possible issue is that a full graph of labels needs to be learned with the computational complexity $O(n^2)$, the efficiency needs further improvement to cater for a large number of labels.

When classifying an unlabeled instance x, each label l_i can not be predicted until its dependent labels are all predicted. Hence the labels should be predicted from the root of the tree and then its children recursively until all the leaves are reached. The detailed process is depicted in Algorithm 2. For each label, the labels dependencies are considered since its prediction is based on the feature set and the predictions of its dependent labels.

Algorithm 2. The process of classification using LDTS classifier

```
Input:
    A unlabeled instance $x$ that needs to be classified.
Output:
    The prediction $Y = (y_1, y_2, \dots, y_m)$.
 1: set the vector to be empty, $Y \leftarrow ()$
 2: set the root label as the current label need to be predicted: $t$
 3: predict current label $t$ for $x$ using corresponding classifier $f_t$
 4: add $f_t(x)$ into $Y$, $Y \leftarrow Y \bigcup f_t(x)$
 5: use the result $f_t(x)$ to update the $x$, $x = (x, f_t(x))$
 6: find all the children labels of $t$
 7: repeat
 8:    for each children labels $c_i$ of $t$ do
 9:       repeat the step 2-6
10:    end for
11: until all the labels are predicted
12: return the predicted vector $Y$
```

When generating the directed tree in LDTS, the root node is selected randomly. However, selecting a different label will result in a different tree and thus generating different dependent labels for each label. Another issue is the label dependency could not be utilized fully, since the dependency of pairwise labels l_i, l_j calculated here is mutual and useful equally to each other. One possibility is that a label may also depend on its children labels, but the directed tree does not allow for this situation. To address such issues, the ensemble learning is used to generate multiple LDTS classifiers iteratively. In each iteration, the classifier is trained on a sampling of the original dataset, and the root label is reselected randomly. Hence each iteration will get a different label tree and combining them will reduce the influence of the root's randomness and take full advantage of the label dependency. We call this extended method ELDTS(Ensemble of LDTS).

The detail process is depicted in Algorithm 3. Given an unlabeled instance x, all predictions of these classifiers will be aggregated into a final result by voting simply.

Algorithm 3. The process of training ELDTS classifier

```
Input:
    The training dataset: $D = \{(x_1, C_1), (x_1, C_1), \ldots, (x_n, C_n)\}$;
    The algorithm for training base classifier: $B$;
    The number of iteration: $n$.
Output:
    An ensemble of LDTS classifiers $F = (f_1, f_2, \ldots, f_m)$.
1: set the $F$ to be empty: $F = ()$
2: for $i = 0$ to $m$ do
3:     generate a new dataset $D_i$ by sampling on the original dataset with replacement

4:     select the root label $r_i$ randomly;
5:     train a LDTS classifier $t_i$ using B, based on the dataset $D_i$ and root label $r_i$
6:     add $f_i$ into $F$
7: end for
8: return the ensemble of classifiers: $F$
```

All above are the description and analysis of our proposed algorithms. Comparison with other state-of-the-art algorithms and further analysis will be given in the following section.

5 Experiment Design and Analysis

5.1 The Description of Datasets

We take several datasets from multiple domains for the experiments, and table 1 depicts them in detail.

Table 1. Description of the datasets used in experiments

Dataset	Domain	Instances	Attributes	Labels	LC	LD	DLS
emotions	music	593	72	6	1.869	0.311	27
enron	text	1702	1001	53	3.378	0.064	753
medical	text	978	1449	45	1.245	0.028	94
scene	image	2407	294	6	1.074	0.179	15
yeast	biology	2417	103	14	4.237	0.303	198

Several statistics as follows have been used to characterize these datasets.

(1) Label cardinality: LC $= \frac{1}{n}\sum_{i=1}^{n}|C_i|$. It calculates the average number of labels for each instance, where $|C_i|$ is the number of true labels of the ith instance.

(2) Label density: LD $= \frac{1}{n}\sum_{i=1}^{n}\left|\frac{C_i}{m}\right|$. It is calculated by dividing the label cardinality by m, the size of original label set.

(3) Distinct label sets: $\text{DLS}(D) = |\{C|\exists(x, C) \in D\}|$. It counts the number of distinct label sets that appear in the dataset.

Seen from table 1, these datasets cover many domains including text categorization, scene classification, emotion analysis, biology etc.. It should be noted that there are no label hierarchies in these datasets and we use them to examine whether our methods could find more strong label dependency and gain better performance. More detail description can be found on the official website of Mulan[1].

5.2 Evaluation Criteria

In order to evaluate the algorithm performance, the criteria should be specified. Let $D = \{(x_1, C_1), (x_2, C_2), \ldots, (x_n, C_n)\}$ be a dataset, where x_i is the ith instance, and $C_i \subset L$ is its true labels. Given a classifier f and an instance x_i, Y_i denotes the predicted labels for x_i, while $rank(x_i)$ or $rank_i$ denotes the predicted rank of labels, and $rank(x_i, l)$ denotes the label l's position in the rank. All the criteria we used are as follows.

(1) Hamming loss: It is proposed by Schapire and Singer [4].

$$\text{H-Loss}(f, D) = \frac{1}{n}\sum_{i=1}^{n}\frac{Y_i \bigoplus C_i}{m} \tag{3}$$

The operator $\bigoplus$ calculates the symmetric difference of two sets, which is the number of misclassified labels for an instance.

(2) Accuracy: It calculates the ratio between the intersection and union of the predicted set of labels and the true set of labels for the instances on average.

$$\text{Accuracy}(f, D) = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{Y_i \bigcap C_i}{Y_i \bigcup C_i}\right| \tag{4}$$

(3) One-error: It calculates how many times that top-ranked label is not a true label of the instance.

$$\text{One-Error} = \frac{1}{n}\sum_{i=1}^{n}\delta(\arg\min_{l \in L} rank(x_i, l)) \tag{5}$$

where $\delta(x) = 1$ if l is a true label of the instance, otherwise $\delta(x) = 0$.

(4) Ranking loss: It expresses the number of times when the irrelevant labels are ranked before the true labels.

$$\text{R-Loss} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\overline{C_i}|\,|C_i|}\left|\{(l_a, l_b) : rank(x_i, l_a) > rank(x_i, l_b), (l_a, l_b) \in C_i \times \overline{C_i}\}\right| \tag{6}$$

[1] http://mulan.sourceforge.net/

(5) Average precision: This measurement calculates the average fraction of labels ranked above a particular label $l \in C_i$, which are all also in C_i.

$$\text{AvePrec} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\overline{C_i}|}\sum_{l \in C_i}\frac{|\{l' \in C_i : rank(x_i, l') \leq rank(x_i, l)\}|}{rank(x_i, l)} \tag{7}$$

These criteria evaluate the different aspects of these methods. While Hamming loss and accuracy do not consider the relation between different predictions, the other 3 criteria take such a relation into considerations, since they are based on the ranking of the probabilities predicted for all labels. Because our methods are intended to get a more accurate probability for each label by finding more strong labels dependencies, thus for each label, it should be predicted more accurately and the true labels should be given greater possibilities. Therefore, we expect that our method could gain better performance under Hamming loss and ranking loss, since Hamming loss examines the predictions of all labels independently and ranking loss focuses on whether the true labels are given greater probabilities than other labels. For other 3 criteria, our method may be effective, but they are not what our method optimize for.

5.3 Algorithms and Settings

The algorithms used for comparison are listed in table 2 with their abbreviations respectively. To examine the effect of label dependency, BR algorithm is used as a baseline since it does not consider the label dependency, then we compare our proposed LDTS and ELDTS with CC and ECC methods to see their effectiveness after eliminating weak dependencies. RAkEL$_d$ and RAkEL are also used for comparison as other ways of learning label dependency.

The experiments are divided into two parts, according to the two purposes mentioned in section 4. One part is on the five aforementioned datasets without label hierarchy to see whether our method can find more strong dependency and thus gain better performance, the other part is on the dataset rcv1v2, a dataset in which there exists a tree hierarchy of labels, to see its performance when a tree structure is learned. Since only one tree exists in rcv1v2, we do not use the ensemble method on it.

Table 2. The algorithms used for comparison

Compared with LDTS	Compared with ELDTS
BR: Binary relevance method	EBR: Ensemble of BR
CC: Classifier chain	ECC: Ensemble of CC
RAkEL$_d$: Random disjoint k label subsets	RAkEL: Random k label subsets

All algorithms are implemented on the Mulan framework [26], an open platform for multi-label learning. The parameter values are chosen as those used in the paper [9]. For the RAkEL, we set the $k = \frac{m}{2}$. For the RAkEL$_d$, we set the

$k = 3$. For the ensemble algorithms, the number of iterations is 10, and for each iteration, 67% of the original dataset is sampled with replacement to form the training dataset. SMO, a support vector machine classifier implemented in Weka [27], is used as the base classifier. All algorithms are executed 5 times using 10-fold cross validation on all datasets expect rcv1v2 with different random seeds 1, 3, 5, 7, 11, respectively, and the final results are the averaged values. For the rcv1v2, only 100 attributes are kept, and 10-fold cross validation is used only one time since it has a huge amount of instances and attributes.

5.4 Experimental Results and Analysis

Based on above setup, we get the final results and the following tables display them in detail. The bold result indicates the best one, and result with the black dot "•" indicates our proposed algorithm is better than itself indicated algorithm.

Table 3. The hamming loss of each algorithm on the datasets

Dataset	BM	CC	LDTS	EBM	ECC	ELDTS	RA*k*EL
emotions	**0.1939**	0.2159•	0.2038	0.1947•	0.2108•	**0.1932**	0.2283•
enron	**0.0601**	0.0606•	0.0602	0.0540•	0.0536•	**0.0535**	0.0541•
medical	0.0101•	**0.0098**	0.0099	0.0098•	0.0096•	**0.0095**	0.0102•
scene	0.1046	**0.1037**	0.1056	0.1020•	0.0997•	**0.0968**	0.1047•
yeast	**0.1990**	0.2115•	0.2060	**0.1991**	0.2106•	0.2034	0.2371•

Table 4. The accuracy of each algorithm on the datasets

Dataset	BM	CC	LDTS	EBM	ECC	ELDTS	RA*k*EL
emotions	0.5199•	0.5336•	**0.5727**	0.5226•	0.5451•	**0.5808**	0.5119•
enron	0.4058•	0.4083•	**0.4085**	0.4370•	0.4110•	**0.4396**	0.4063•
medical	0.7580•	**0.7750**	0.7670	0.7655•	**0.7799**	0.7759	0.7686•
scene	0.5999•	**0.6949**	0.6496	0.6137•	**0.7020**	0.6783	0.6281•
yeast	0.5003•	0.4879•	**0.5073**	0.5031•	0.4929•	**0.5249**	0.4692•

As shown from table 3 to table 7, our proposed LDTS method performs better on the majority of datasets evaluated by the criteria. It is superior to CC on 3 datasets under the Hamming loss, accuracy, one-error, and ranking loss, but inferior under other metrics. LDTS algorithm does not improve all the time or the improvement is not significant. The possible reason is that although LDTS algorithm could learn the dependency further, it still ignore lots of useful dependency since it only considers unidirectional dependency of pairwise labels, especially when the labels are dependent mutually. We expect that ensemble learning that combines different trees can further utilize the label dependency , since the dependent direction between pairwise labels is changed in a different tree by choosing a different root.

To validate above assumption, we also use ensemble learning on these algorithms and compare them each other. Also shown from table 3 to table 7, our proposed ELDTS has a substantial improvement after the employing ensemble learning. Under all 5 criteria, ELDTS is superior to ECC on most datasets. These results show that through learning multiple label trees by ensemble learning, the influence of the root label's randomness could be mitigated, and the label dependencies are learned more effectively. Although ECC algorithm also changed the order of labels, it could not make sure that only the strong dependency is considered each time, since it's totally random when determining the dependent relationship within labels.

It can be observed that the proposed algorithms do not perform well on two datasets *medical* and *scene*. Seen from the table 1, these two datasets have very small label cardinality, which means there tend to be less label dependency in them and overemphasis on label dependency may not be preferable. Thus the algorithms we propose are more suitable for the datasets that there are indeed strong label dependency in them.

The results gotten on rcv1v2, a dataset with tree structure of labels, are also given in table 8. We can clearly see that our proposed LDTS method is superior under Hamming loss and ranking loss, the criteria it optimize for. Therefore, it has been proven that our method is more effective when there is complex dependency within labels.

Table 5. The one-error of each algorithm on the datasets

Dataset	BM	CC	LDTS	EBM	ECC	ELDTS	RA*k*EL
emotions	**0.2989**	0.3700●	0.3103	**0.2534**	0.3181●	0.2563	0.3096●
enron	0.4912●	0.4938●	**0.4897**	0.3078●	0.3071●	**0.3054**	0.3210●
medical	0.2029●	**0.1847**	0.1932	0.1407●	0.1415●	**0.1397**	0.1634●
scene	0.3389●	**0.2793**	0.3132	0.2553●	0.2609●	**0.2485**	0.2777●
yeast	**0.2557**	0.2559●	0.2557	0.2557●	0.2583●	**0.2551**	0.2895●

Table 6. The rank loss of each algorithm on the datasets

Dataset	BM	CC	LDTS	EBM	ECC	ELDTS	RA*k*EL
emotions	0.2778●	0.2876●	**0.2334**	0.2169●	0.2317●	**0.1743**	0.1964●
enron	**0.2915**	0.2929●	0.2925	0.1648●	0.1640●	**0.1622**	0.1784●
medical	0.0952●	**0.0926**	0.0932	0.0537●	**0.0516**	0.0518	0.0598●
scene	0.1718●	**0.1576**	0.1664	0.1162●	0.1115●	**0.0945**	0.1110●
yeast	0.3188●	0.3351●	**0.3181**	0.2739●	0.2771●	**0.2273**	0.2300●

Table 7. The average precision of each algorithm on the datasets

Dataset	BM	CC	LDTS	EBM	ECC	ELDTS	RA*k*EL
emotions	0.7384●	0.7159●	**0.7634**	0.7814●	0.7573●	**0.8040**	0.7734●
enron	0.4682●	**0.4702**	0.4693	0.6284●	0.6287●	**0.6314**	0.6011●
medical	0.7977●	**0.8115**	0.8066	0.8672●	**0.8710**	0.8707	0.8524●
scene	0.7752●	**0.8048**	0.7881	0.8353●	0.8351●	**0.8479**	0.8285●
yeast	0.6697●	0.6611●	**0.6728**	0.7003●	0.6975●	**0.7253**	0.7048●

Table 8. The performance of algorithms on dataset rcv1v2

criterion	BM	CC	LDTS	RAkEL$_d$
H-Loss	**0.0235**	0.0294●	**0.0235**	0.0237●
Accuracy	0.1810●	**0.2529**	0.2237	0.1783●
One-Error	0.7638●	**0.7065**	0.7120	0.9538●
R-Loss	0.3560●	0.3508●	**0.3381**	0.4590●
AvePrec	0.2537●	**0.3085**	0.2940	0.1001●

6 Conclusion

In this paper, one kind of novel approaches are proposed to exploit the label dependency. Specifically, the dependency degree of pairwise labels is calculated firstly and then a tree is build to represent the dependency structure of labels. The methods assume that the dependencies only exist between each label and its ancestor labels, resulting in reducing the influence of weak dependency. At the same time, they also generalize the label dependency into a tree model. Furthermore, we utilize ensemble learning to learn and aggregate multiple label trees to reflect the labels dependencies fully. The experimental results show that the algorithms we proposed perform better, especially after boosted by the ensemble learning.

One potential problem is that using mutual information to measure the dependency will give equal values to both of the labels, which assumes that the dependency for pairwise labels is mutual and equal for each other. However, the label dependency could be directed possibly and this assumption is often violated in reality. Hence how to measure the directed label dependency should be one of the next directions. Additionally, how to generalize the tree structure of labels further to graph or forest structure is another issue in the future work.

References

1. Cheng, W., Hullermeier, E.: Combining Instance-Based Learning and Logistic Regression for Multilabel Classification. Machine Learning 76(2-3), 211–225 (2009)
2. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining Multi-label Data. In: Oded, M., Lior, R. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 667–685. Springer, New York (2010)
3. McCallum, A.K.: Multi-label Text Classification with a Mixture Model Trained by EM. In: Proceedings of AAAI 1999 Workshop on Text Learning (1999)
4. Schapire, R.E., Singer, Y.: Boostexter: a Boosting-Based System for Text Categorization. Machine Learning 39(2-3), 135–168 (2000)
5. Clare, A.J., King, R.D.: Knowledge Discovery in Multi-label Phenotype Data. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS (LNAI), vol. 2168, pp. 42–53. Springer, Heidelberg (2001)
6. Thabtah, F.A., Cowling, P., Peng, Y.: MMAC: a New Multi-class, Multi-label Associative Classification Approach. In: Proceedings of the 4th International Conference on Data Mining, pp. 217–224 (2004)
7. Zhang, M., Zhou, Z.: ML-KNN: A Lazy Learning Approach to Multi-label Learning. Pattern Recognition 7(40), 2038–2048 (2007)
8. Read, J.: Multi-label Classification using Ensembles of Pruned Sets. In: Proceedings of the IEEE International Conference on Data Mining, pp. 995–1000. IEEE Computer Society, Washington, DC (2008)

9. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier Chains for Multi-label Classification. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part II. LNCS, vol. 5782, pp. 254–269. Springer, Heidelberg (2009)
10. Dembczynski, K., Cheng, W., Hullermeier, E.: Bayes Optimal Multilabel Classification via Probabilistic Classifier Chains. In: Proceedings of the 27th International Conference on Machine Learning, pp. 279–286. Omnipress (2010)
11. Zhang, M., Zhang, K.: Multi-label Learning by Exploiting Label Dependency. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 999–1000. ACM Press, Washington, DC (2010)
12. Zhang, Y., Zhou, Z.: Multi-label Dimensionality Reduction via Dependence Maximization. ACM Transactions on Knowledge Discovery from Data 4(3), 1–21 (2010)
13. Boutell, M.R., Luo, J., Shen, X.: Learning Multi-label Scene Classification. Pattern Recognition 37(9), 1757–1771 (2004)
14. Tsoumakas, G., Katakis, I., Vlahavas, I.: Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In: Proceedings of ECML/PKDD 2008 Workshop on Mining Multidimensional Data, pp. 30–44 (2008)
15. Hullermeier, E., Furnkranz, J., Cheng, W.: Label Ranking by Learning Pairwise Preferences. Artificial Intelligence 172(16-17), 1897–1916 (2008)
16. Furnkranz, J., Hullermeier, E., Mencia, E.L.: Multilabel Classification via Calibrated Label Ranking. Machine Learning 2(73), 133–153 (2008)
17. Madjarov, G., Gjorgjevikj, D., Dzeroski, S.: Two Stage Architecture for Multi-label learning. Pattern Recognition 45(3), 1019–1034 (2011)
18. Tsoumakas, G., Katakis, I., Vlahavas, I.: Random k-labelsets for Multi-label Classification. IEEE Transactions On Knowledge and Data Engineering 23(7), 1079–1089 (2011)
19. Tsoumakas, G., Dimou, A., Spyromitros, E.: Correlation-Based Pruning of Stacked Binary Relevance Models for Multi-Label Learning. In: Proceeding of ECML/PKDD 2009 Workshop on Learning from Multi-Label Data, Bled, Slovenia, pp. 101–116 (2009)
20. Gaag, L., Waal, P.: Multi-dimensional Bayesian Network Classifiers. In: Third European Workshop on Probabilistic Graphical Models, pp. 107–114 (2006)
21. Bielza, C., Li, G., Larranage, P.: Multi-dimensional Classification with Bayesian Networks. International Journal of Approximate Reasoning 52(6), 705–727 (2011)
22. Guo, Y., Gu, S.: Multi-label Classification using Conditional Dependency Networks. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, pp. 1300–1305 (2011)
23. Ghamrawi, N., McCallum, A.K.: Collective Multi-label Classification. In: Proceedings of the 2005 ACM Conference on Information and Knowledge Management, pp. 195–200 (2005)
24. Dembczynski, K., Waegeman, W., Cheng, W.: On Label Dependence in Multi-label Classification. In: Proceedings of the 2nd International Workshop on Learning From Multi-label Data, pp. 5–12 (2010)
25. Chow, C.K., Liu, C.N.: Approximating Discrete Probability Distributions with Dependency Trees. IEEE Transactions on Information Theory 14(3), 462–467 (1968)
26. Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., Vlahavas, I.: Mulan: A Java Library for Multi-Label Learning. Journal of Machine Learning Research 12, 2411–2414 (2011)
27. Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, San Francisco (2000)