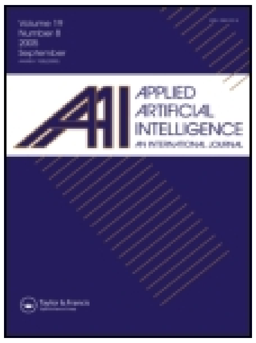

An Incremental Decision Tree for Mining Multilabel Data

Peipei Li, Xindong Wu, Xuegang Hu & Hao Wang

To cite this article: Peipei Li, Xindong Wu, Xuegang Hu & Hao Wang (2015) An Incremental Decision Tree for Mining Multilabel Data, Applied Artificial Intelligence, 29:10, 992-1014, DOI: 10.1080/08839514.2015.1097154

To link to this article: http://dx.doi.org/10.1080/08839514.2015.1097154

Published online: 23 Nov 2015.

Applied Artificial Intelligence, 29:992–1014, 2015
Copyright © 2015 Taylor & Francis Group, LLC
ISSN: 0883-9514 print/1087-6545 online
DOI: 10.1080/08839514.2015.1097154

AN INCREMENTAL DECISION TREE FOR MINING MULTILABEL DATA

Peipei Li[1], Xindong Wu[2], Xuegang Hu[1], and Hao Wang[1]
[1]*Department of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China*
[2]*Department of Computer Science, University of Vermont, Burlington, Vermont*

□ *Mining with multilabel data is a popular topic in data mining. When performing classification on multilabel data, existing methods using traditional classifiers, such as support vector machines (SVMs), k-nearest neighbor (k-NN), and decision trees, have relatively poor accuracy and efficiency. Motivated by this, we present a new algorithm adaptation method, namely, a decision tree–based method for multilabel classification in domains with large-scale data sets called decision tree for multi-label classification (DTML). We build an incremental decision tree to reduce the learning time and divide the training data and adopt the k-NN classifier at leaves to improve the classification accuracy. Extensive studies show that our algorithm can efficiently learn from multilabel data while maintaining good performance on example-based evaluation metrics compared to nine state-of-the-art multilabel classification methods. Thus, we draw a conclusion that we provide an efficient and effective incremental algorithm adaptation method for multilabel classification especially in domains with large-scale multilabel data.*

INTRODUCTION

Multilabel classification is a challenging and popular topic in many applications, such as text categorization and gene function classification. For example, we can classify a newspaper article concerning David Beckham into two categories of entertainment and sports, at least. It is hence a challenge for the traditional single-label classification, because it always assumes that predefined data categories are mutually exclusive and each data point can belong to exactly one category.

To handle the multilabel classification issue, an intuitive approach is problem transformation (PT; Tsoumakas, Katakis, and Vlahavas 2010), in which a multilabel problem is transformed into one or more single-label problems. The advantage of this method is that a multilabel classifier can

Address correspondence to Hao Wang, Hefei University of Technology, Department of Computer Science and Information Engineering, Tunxi Road #293, Hefei City, Anhui Province, China, 230009. E-mail: jsjxwangh@hfut.edu.cn

be readily built using many state-of-the-art binary classifiers off the shelf. However, the computational complexity of PT methods is in direct proportion to the total number of class labels (denoted as $|L|$), and the expressive power of such a system can be weak; for example, the systems of RAKEL (a random *k*-label sets algorithm; Tsoumakas and Vlahavas 2007; Tsoumakas, Katakis, and Vlahavas 2011) and EPS (Read, Pfahringer, and Holmes 2008; an efficient ensemble of pruned sets method). Contrary to the PT approach, many algorithm adaptation (AA) methods extended from specific learning algorithms have been designed to handle multilabel data directly, such as multilabel support vector machines (SVMs; J. H. Xu 2012), multilabel decision trees (Clare and King 2001), and multilabel lazy learning algorithms (M. L. Zhang and Zhou 2007; Cheng and Hüllermeier 2009). However, the computational complexity of AA methods is in direct proportion to $|L|$ and N (the size of training data) or their quadratic function. They will present a low efficiency when meeting large-scale multilabel data. In addition, because the selected single-label classifiers are meta-algorithms, the capabilities of these methods are also inferior in their efficiency. For example, the SVM classifier requires additional time overhead when handling nonlinear multilabel data with discrete attributes.

Motivated by this, we propose an efficient and effective algorithm adaptation method based on an incremental decision tree with the *k*-nearest neighbor (*k*-NN) classifier for multilabel classification, called decision tree for multi-label classification (DTML). Our contributions are as follows:

- We incrementally build a decision tree learning from multilabel data. It is conducive to improve the training efficiency compared to the existing methods using traditional classification models.
- We introduce the *k*-NN classifier at leaves using only subsets of training data. It is beneficial to improve the test efficiency while maintaining the classification accuracy in the classification.
- Extensive studies demonstrate that our algorithm can achieve better performance on the evaluation metrics and train much faster than several well-known PT methods and AA methods especially in domains with large-scale data sets.

The rest of this article is organized as follows. Section "Related Work" reviews the related work. Our DTML algorithm is described in detail in section "Our DTML Algorithm." Section "Experiments" provides the experimental study and section "Conclusions" is the summary.

RELATED WORK

To handle the multilabel classification issue, existing efforts mainly come from two thoughts, namely, PT and AA. Details are as follows.

PT Methods

In this subsection, we review three PT methods. First, the most well-known PT method is the binary relevance (BR) method (Tsoumakas, Katakis, and Vlahavas 2010). It is a "one-vs.-rest" paradigm, which treats every label independent of all other labels. The representative works include a BR-based *k*-NN algorithm (called BR*k*NN; Spyromitros, Tsoumakas, and Vlahavas 2008) and an efficient BR-based chaining method called an ensemble framework for classifier chains (ECC; (Read et al. 2009). These BR-based methods can reduce the complexity in the learning space, but they cannot perform very well on the evaluation metrics.

The second paradigm is the label powerset (LP) method. It considers each unique set of labels that exits in a multilabel training set as one of the classes. The representative works include a RAKEL algorithm (Tsoumakas and Vlahavas 2007; Tsoumakas, Katakis, and Vlahavas 2011) and an EPS (Read, Pfahringer, and Holmes 2008). However, these methods suffer when many label combinations are associated with very few examples. In addition, because the computational cost of LP methods is relevant to $O(K^2)$ $(K \leq |L|)$ and the complexity of metaclassifiers, it will be too high if the magnitude of class labels is large.

The third paradigm is the pair-wise classification. It is a "one-vs.-one" paradigm where one classifier is associated with each pair of labels. A famous work is an extended algorithm of label ranking (Füurnkranz et al. 2008) called multi-label classification via calibrated label ranking (MLCLR). It incorporates the calibrated scenario and substantially extends the expressive power of existing approaches to label ranking. However, this approach will also face the challenge of complexity when the number of labels is large.

In addition, we should mention the efficient random decision tree ensembling algorithms based on PT methods (BR-RDT and LP-RDT, collectively called multi-label classification based on random decision tree ([MLRDT]; X. T. Zhang et al. 2010) for multilabel classification. The main difference between random decision trees (RDTs) and other classical decision tree methods, such as C4.5, is that RDT constructs decision tree randomly and need not using any information of labels. That nature makes RDT fast and the computational cost independent of labels.

Algorithm Adaptation Methods

We summarize the algorithm adaptation method corresponding to the basic classifiers as follows.

- Methods using decision trees and boosting: The AA method using decision trees (Clare and King 2001) was first proposed in 2001. Authors modified the expression for entropy of C4.5 to allow a tree to predict a label vector. The first system, BoosTexter, for multilabel classification and ranking (Schapire and Singer 2000) was proposed in 2000. It supplies AdaBoost.MH and AdaBoost.MR, both of which belong to a multilabel adaptation of the well-known AdaBoost paradigm. However, this family of methods has been recognized as having high computational complexity (Petrovskiy 2006).

- Methods using SVMs: Representative works include rank-SVM (Elisseeff and Weston 2002) using the kernel SVM as a ranking function, an extended algorithm called calibrated rank-SVM (A. Jiang, Wang, and Zhu 2008), and a zero-label based rank-SVM method (J. H. Xu 2012). However, because the computational complexity of rank-SVM is in direct proportion to $\mathrm{O}\left(N^2\,|L|\right)$, it will be too high if the magnitude of a data set is large.
- Methods using *k*-NN: A state-of-the-art lazy multilabel method ML*k*NN (M. L. Zhang and Zhou 2007) was proposed in 2007. It adapts the *k*-NN algorithm for multilabel classification using Bayesian relevance. It has been noted that ML*k*NN outperforms rank-SVM (Elisseeff and Weston 2002), AdaBoost.MH (Comité, Gilleron, and Tommasi 2003), and BoosTexter (Schapire and Singer 2000). In 2009, an extended work on lazy classification called instance-based learning and logistic regression for multi-label data (IBLR) (Cheng and Hüllermeier 2009) was introduced. It combines both instance-based learning and logistic regression to classify multilabel data. In 2012, a fuzzy similarity measure and *k*-NN-based method was proposed for multilabel text classification (J.-Y. Jiang, Tsai, and Lee 2012). These new extended algorithms enable improving the classification performance while maintaining the time efficiency compared to ML*l*NN. However, because the computational complexity of *k*-NN is in direct proportion to $O(k\mathrm{N}^2)$, it also will be too high if the magnitude of a data set is large.
- Methods using neural networks: In 2006, a neural network adaptation algorithm for multilabel data called back-propogation for multi-label learning ([BP-MLL]; M. L. Zhang and Zhou 2006) was proposed. It adapts the error function of a back-propagation algorithm to account for multiple labels in the learning process. In 2009, adaptive resonance theory neural networks (Sapozhnikova 2009) were developed to handle multilabel classification. In 2012, a competitive neural network–based algorithm called multi-label hierarchical classification using a competitive neural network ([MHC-CNN]; Borges and Nievola 2012) was proposed for multilabel hierarchical classification. This family of methods performs better on the classification performance but presents a higher computational complexity due to the complexity of metamodels.
- Other: A novel algorithm for effective and efficient multilabel classification in domains with many labels, called hierarchy of multi-label

classifiers ([HOMER]; Tsoumakas, Katakis, and Vlahavas 2008), was proposed in 2008. It constructs a hierarchy of multilabel classifiers, each dealing with a much smaller set of labels and a more balanced example distribution. This leads to improved predictive performance along with linear training and logarithmic testing complexities with respect to $|L|$. An efficient max-margin multilabel classification algorithm called M3L (Hariharan et al. 2010; Hariharan, Vishwanathan, and Varma 2012) was presented and was further applied into the zero-shot learning scenario in 2012. Concerning the prediction on all labels as well as the rankings of only relevant labels, an efficient PRObability-LOSS-based algorithm called ProSVM (M. Xu, Li, and Zhou 2013) was proposed in 2013. The authors improved its efficiency with an upper approximation that reduces the number of constraints to $\mathrm{O}(|L|)$. The computational cost in the aforementioned algorithms decreases to the complexity of metaclassifier $+\ \mathrm{O}(|L|)$ or $\mathrm{O}(N\ |L|)$, but it is still a challenge in the computational cost when handling large-scale multilabel data with larger values of $|L|$ and N.

OUR DTML ALGORITHM

We first formalize the problem of multilabel classification in this section. Let E denote the domain of an instance space and $L = \{\lambda_1, \cdots, \lambda_{|L|}\}$ denote the set of labels. Suppose a multilabel training data set $D = \{(e_1, Y_{e_1}), \cdots, (e_N, Y_{e_N})\}$, whose instances are drawn identically and independently from an unknown distribution. Each instance, $e \in E$ is associated with a label set $Y_e \subseteq L$. The goal of multilabel classification is to train a classifier $f : E \rightarrow 2^{|L|}$ that maps a feature vector to a set of labels, while optimizing some specific evaluation metrics. Our DTML algorithm presented in this section aims to handle this problem efficiently and effectively. Algorithm 1 shows the processing flow of DTML. First, it employs a "divide and conquer" strategy by recursively partitioning the training set and generates an incremental decision tree to reduce the learning time in Steps 1–18. Second, it implements the test using the kNN classifier at leaves in Steps 19–22. Details of techniques are as follows.

Construction of a Decision Tree

Our DTML algorithm is built on an incremental decision tree, which follows the strategy of split-feature selection that discrete attributes should not be chosen again, whereas numerical ones can be chosen multiple times in a decision path. In DTML, a training instance-e first traverses the decision tree from the root to an available growing node, evaluating the appropriate attribute at each node and following the branch corresponding to the attribute's value in e. In addition, relevant statistics are stored at the current node (Step 3). Contrary to the split test using all training instances in traditional decision trees, it is evaluated only using n_{min} instances accumulated at the current node in DTML (Steps 4–5). To deal with multilabel decision trees, we adopt the same estimation method of information gain for multilabel data in Clare and King (2001). Suppose $G(\cdot)$ is the estimator function of information gain (IG), then, correspondingly, the IG of a discrete attribute a_i could be expressed in Eq. (1). It specifies the difference between the entropy of training instances S accumulated at the node and the weighted sum of the entropy of the subsets s_v caused by partitioning on the value v of that attribute a_i, where the size of S is n_{min}.

$$G(S, a_i) = I(S) - \sum_{v=1}^{|v_{a_i}|} \frac{|S_v|}{|S|} I(S_v) \tag{1}$$

subject to

$$\begin{aligned} I(S) &= I\left(s_1, \cdots, s_{|v_{a_i}|}\right) \\ &= -\sum_{i=1}^{|L|} \left[p(\lambda_i) \cdot log_2 p(\lambda_i) + \left(1 - p(\lambda_i)\right) \cdot log_2 \left(1 - p(\lambda_i)\right)\right] \end{aligned}.$$

Algorithm 1 DTML

Input: Training set: D; Test set: TE; Label set: L; Minimum count of split instances: n_{min}; Check period: CP; Memory size: MS; Maximum height of tree: maxH; Attribute set: A.

Output: Vector of predicted class labels;

```
1. Generate the root of a decision tree T;
2. for each training instance $e$ in the training data set D do
3. Sort the training instance e into an available node and store the
   information;
4. if the count of instances at the current node of $curNode \geq n_{min}$ and the
   depth of the current node $\leq$ the maximum height maxH then
5. Install the split test for a split attribute $a_i \in A$ at curNode;
6. if the selected split feature is the cut-point of a numerical attribute $a_i$
   then
7. Generate two children nodes;
8. end if
9. if it is a discrete attribute $a_i$ then
10. Generate children nodes by the number of attribute values in $a_i$;
```

```
11. end if
12. if the count of training instances observed in the current tree T % $CP == 0$ then
13. if the memory consumption in the current tree T > MS then
14. Install the pruning in the current decision tree T;
15. end if
16. end if
17. end if
18. end for
19. for each test instance e in the test data set TE do
20. Traverse the tree from the root to leaves;
21. Predict the label set $Y(e)$ of instance $e$ in the $k$-nearest neighbor algorithm;
22. end for
```

$$I(s_v) = -\sum_{i=1}^{|L|} \left[p_{s_v}(\lambda_i) \cdot log_2 p_{s_v}(\lambda_i) + \left(1 - p_{s_v}(\lambda_i)\right) \cdot log_2\left(1 - p_{s_v}(\lambda_i)\right)\right],$$

where $\left|v_{a_i}\right|$ indicates the number of different attribute values for a_i, and $p(\lambda_i)$ and $p_{s_v}(\lambda_i)$ specify the probability of the ith class label in S and s_v, respectively. Regarding the solution to cut-points of numerical attributes, our estimation method is given as follows. Suppose a_i is a numerical attribute at the current growing node. In our algorithm, sequential attribute values of a_i will be discretized into $\min(10, \left|v_{a_i}\right|)$ intervals. Hence, the IG of the xth cut-point in a_i (denoted as $v_{a_i}^x$) could be estimated in Eq. (2).

$$G\left(v_{a_i}^x\right) = p\left(v_{a_i} < v_{a_i}^x\right) \cdot low\left(v_{a_i}^x\right) + p\left(v_{a_i} \geq v_{a_i}^x\right) \cdot high\left(v_{a_i}^x\right) \tag{2}$$

subject to

$$low\left(v_{a_i}^x\right) = -\sum_{k=1}^{|L|} \left[p\left(K = k \middle| v_{a_i} < v_{a_i}^x\right) \cdot log_2\left(p\left(K = k \middle| v_{a_i} < v_{a_i}^x\right)\right) + p\left(K \neq k \middle| v_{a_i} < v_{a_i}^x\right) \cdot log_2\left(p\left(K \neq k \middle| v_{a_i} < v_{a_i}^x\right)\right)\right]$$

$$high\left(v_{a_i}^x\right) = -\sum_{k=1}^{|L|} \left[p\left(K = k \middle| v_{a_i} \geq v_{a_i}^x\right) \cdot log_2\left(p\left(K = k \middle| v_{a_i} \geq v_{a_i}^x\right)\right) + p\left(K \neq k \middle| v_{a_i} \geq v_{a_i}^x\right) \cdot log_2\left(p\left(K \neq k \middle| v_{a_i} \geq v_{a_i}^x\right)\right)\right],$$

where $p\left(v_{a_i} < v_{a_i}^x\right)$ refers to the ratio of the number of instances whose attribute values of a_i are lower than $v_{a_i}^x$ divided be the total number of instances at the current node, and $low\left(v_{a_i}^x\right)$ specifies the information expectation of values of $v_{a_i} < v_{a_i}^x$. This is similar to the definition on $p\left(v_{a_i} \geq v_{a_i}^x\right)$ and $high\left(v_{a_i}^x\right)$, respectively. According to the values of IG, we select a candidate split attribute or cut-point with the highest IG as the final split feature. The current growing node is correspondingly replaced with the decision node and children nodes regarding the attribute type (Steps 6–11).

Pruning

In the growing of a decision tree mentioned above, we take two measures in DTML to avoid the space overflow below. First, a threshold is specified to limit the maximum height of the decision tree (namely, maxH) as shown in Step 4. That is, if the depth of a decision path in the current tree is up to maxH, this path will not grow continuously. Second, a pruning mechanism is adopted. As shown in Steps 12–16, if the checking threshold CP and the memory limit MS are met, the pruning is installed from bottom to top. First, stop all undergoing splits of growing nodes and change them into leaves and then remove the storage space of information in the nodes corresponding to the evaluation metric of accuracy. Second, cut off several subtrees from bottom to top with the roots whose values of accuracy are lower than 50%.

Prediction in *k*-NN

In our algorithm, we utilize the *k*-NN classifier to implement the prediction on the test data set. More precisely, each test instance e first traverses from the root to an available leaf corresponding to the attribute values. Second, we select *k*-nearest neighbors of e by the Euclidean distance[1] and count the label distributions in *k*-nearest neighbors, denoted as $c(\lambda_i)$, namely, the number of instances with the λ_i label $(1 \leq i \leq |L|)$. Last, we use the majority voting with a threshold r $(0 < r < 1)$ to label the test instance e. That is, if the value of $c(\lambda_i)$ is no less than the value of r multiplying k neighboring instances at the current leaf, we will assign the class label λ_i to e. Thus, we can obtain a label set of Y_e for e.

Illustration

Let us consider the following example to show the process of our algorithm. As shown in Table 1, we randomly select 14 objects from the Emotions

[1]It is also suitable for data sets with discrete attributes, because the selected data sets with discrete attributes have binary values; that is, 0 and 1.

TABLE 1 Fourteen objects from emotions

Index	$\mathrm{attr}_{1\sim58}$	attr_{59}	attr_{60}	attr_{61}	$\mathrm{attr}_{62\sim72}$	class_1	class_2	class_3	class_4	class_5	class_6
1	...	0.136020	...	0.100517	...	0	1	1	0	0	0
2	...	0.059457	...	0.067684	...	1	0	0	0	0	1
3	...	0.090650	...	0.100852	...	0	1	0	0	0	1
4	...	0.167114	...	0.112815	...	0	0	1	0	0	0
5	...	0.329188	...	0.265049	...	0	0	0	1	0	0
6	...	0.123337	...	0.082887	...	0	1	1	0	0	0
7	...	0.078375	...	0.078946	...	1	1	0	0	0	0
8	...	0.105881	...	0.081464	...	0	0	0	0	0	1
9	...	0.068104	...	0.060114	...	1	1	0	0	0	0
10	...	0.159680	...	0.185677	...	0	0	1	1	1	0
11	...	0.104672	...	0.109043	...	0	1	1	0	0	0
12	...	0.122011	...	0.075501	...	0	0	1	0	0	0
13	...	0.056388	...	0.043043	...	1	0	0	0	0	0
14	...	0.146425	...	0.094339	...	0	0	1	0	1	0

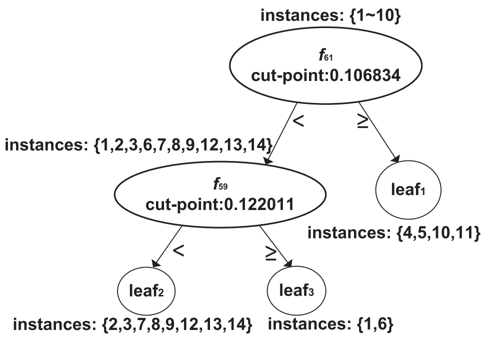

FIGURE 1 A decision tree grows over 14 instances.

database, which contains 72 continuous dimensions of attributes and six class labels. Due to the space limit, we only list two attributes involved in the split test here. According to Algorithm 1, each example arrives one by one at the root of the current decision tree. When the statistical count of instances at the root is up to n_{min} (e.g., 10), we install the split test and we can get the IG of all cut points using Eq. (2). We specify the attribute value of attribute f_{61} with the maximum IG as the split point. Correspondingly, we generate two child nodes regarding the cut point. In addition, we divide the previous 10 instances collected at the current node into child nodes regarding the split point. In a similar way, the decision tree grows up continuously until all training instances arrive. Figure 1 illustrates the final decision tree generated over 14 instances in Table 1.

Classification Risk Analysis of DTML

In this subsection, we will analyze the performance of DTML at a leaf. More specifically, suppose that the decision tree generated in DTML is a complete binary tree and the height of this tree is l. In addition, suppose that there are m instances arrived at the current tree and the probability that each training instance reaches a leaf is equal. Thus, the mean number of instances at a leaf amounts to $ln = m/2^{l-1}(\geq 1)$, denoted as $D' = \{(e_j, Y_j)\,|1 \leq j \leq ln\}$. We select k-nearest neighbors from the data set D' according to the Euclidean distance. Therefore, we can formalize these selected instances whose label set contains $\lambda_i(\in L)$ as follows:

$$c(\lambda_i) = \sum_{j=1}^{ln} \left|c_{e_j}(\lambda_i)\right| \quad s.t., \quad \left|c_{e_j}(\lambda_i)\right| = \begin{cases} 1 & \lambda_i \in Y_j \\ 0 & otherwise \end{cases}. \tag{3}$$

Further, we could define the classification risk (denoted as *PE*) in DTML for any test instance (e_x, Y_x) arrived at the current leaf in Eq. (4).[2]

$$PE = 1 - P(Y_x|e_x) \quad s.t., \quad P(Y_x|e_x) = \prod_{i=1}^{\lambda_i \in Y_x} P(\lambda_i), \tag{4}$$

where $P(Y_x|e_x)$ indicates the probability that e_x is predicted correctly in DTML and $P(\lambda_i)$ indicates the probability that the label λ_i belongs to Y_x. According to the aforementioned description, we can get the value of $P(\lambda_i)$ in Eq. (5).

$$P(\lambda_i) = P\left(\frac{c(\lambda_i)}{k} \geq r\right) \leq P\left(\frac{c(\lambda_i)}{k} > 0\right) \equiv P(c(\lambda_i) \geq 1), \tag{5}$$

where $P(c(\lambda_i) \geq 1)$ indicates the probability that there is at least an instance containing the label λ_i in the k-nearest neighbors of (e_x, Y_x) from the data set of D'. Suppose that the probability that each instance has a label belonging to the label set L is equal; thus, we can get the infimum value of $P(c(\lambda_i) \geq 1)$ in Eq. (6).

$$P(\lambda_i) = 1/(k \cdot |L|). \tag{6}$$

In the above analysis, we can rewrite Eq. (4) in Eq. (7):

$$\begin{aligned} PE &= 1 - \prod_{i=1}^{\lambda_i \in Y_x} P(\lambda_i) \geq 1 - \prod_{i=1}^{\lambda_i \in Y_x} P(c(\lambda_i) \geq 1) = 1 - 1/(k \cdot |L|)^{|Y_x|} \\ &= 1 - 1/(\sigma \cdot ln \cdot |L|)^{|Y_x|} \end{aligned}, \tag{7}$$

where the value of k is relevant to the value of ln, denoted as $k = \sigma\, ln\,(0 < \sigma \leq 1)$. According to Eq. (7), if we aim to reduce the classification risk in DTML, it is better to reduce the value of ln. However, we know that the higher the height of the decision tree, the lower the value of ln. Thus, we conclude that a higher height of the decision tree is conducive to improve the classification performance in DTML. This conclusion is built on the training data when there are no irrelevant or noisy variables, but as complexity increases the model is bound to overfit and the test performance will start degrading (Hastie, Tibshirani, and Friedman 2008). In our algorithm, it occurs when the height of the decision tree is larger than half the dimension count of the database. It is also validated in the experimental

[2]To simplify the problem, our model is built on the assumption that class labels are independent of each other.

results in section "Relationship between the Height of the Decision Tree and Classification Performance." Thus, to avoid overfitting in our algorithm, we limit the value of maxH no more than half of the attribute dimensions of databases.

Computation Complexity Analysis

The computational cost of DTML consists of the training cost and the testing cost as discussed next.

Training Complexity

Training complexity mainly comes from the construction cost of a decision tree in DTML, which is in direct proportion to $O(VdN\log_2(\mathrm{N}))$, where V is the maximum number of values per attribute, d is the number of attribute dimensions, and N is the size of training data. Regarding the computational cost of MLRDT (X. T. Zhang et al. 2010), the least complexity is in direct proportion to $O(mN\log_2(N))$, where m is the number of random decision trees. Due to $m \ll Vd$, our algorithm is slower than MLRDT. This is because the time cost of the split test in our algorithm is directly proportional to $O(Vd)$, whereas there is no time cost in the split test in MLRDT. However, comparing the computational cost of a traditional decision tree used in the aforementioned multilabel classification methods (denoted as C4.5new), it is the same as that of C4.5, which is in direct proportion to $\mathrm{O}\left(\left(\sum_{i=1}^{d}\left(\left|v_{a_i}\right|-1\right)\right)^2 N\right)$. In a common case, due to $Vdlog_2(N) \ll \left(\sum_{i=1}^{d}\left(\left|v_{a_i}\right|-1\right)\right)^2$, the training complexity in DTML is hence much lower than that of C4.5new.

Test Complexity

The test computation complexity on a test instance in DTML is directly proportional to $\mathrm{O}\left(kN'^2\right)$, where N' specifies the average number of instances at leaves. Compared to other methods using simple label distribution statistics, such as the MLRDT method (X. T. Zhang et al. 2010), it is in direct proportion to $O(mq)$, where q refers to the average depth of the branches that the test instance belongs to. As $mq \ll kN'2$ in general, the time complexity of our algorithm is hence higher. However, compared to other k-NN-based algorithms, such as BRkNN and MLkNN (denoted as k-NNnew), it is directly proportional to $\mathrm{O(kN^2)}$. Apparently, we could conclude that the test computation complexity in DTML is less that in k-NNnew because of $N \gg N'$. The above conclusions will be validated in the experiments.

EXPERIMENTS

This section is devoted to validating the performance of our DTML algorithm with regard to effectiveness and efficiency. Before presenting the experimental results, we first give some information about data sets and baseline algorithms included in the study, as well as the criteria used for evaluation in sections "Data Sets and Baseline Algorithms" and "Evaluation Measures," respectively. Further, we discuss the relationship between the height of the decision tree and the performance in section "Relationship between the Height of the Decision Tree and the Classification Performance." Finally, we analyze the performance of DTML compared to nine state-of-the-art PT methods and algorithm adaptation methods on both of effectiveness and efficiency.

Data Sets and Baseline Algorithms

In our experiments, we select seven benchmark multilabel databases containing numerical or discrete attributes only from different applications domains and a simulated database containing numerical and discrete attributes.[3] Details of these data sets are listed in Table 2, where label cardinality is the average number of labels in a database and label density is the average number of labels in a database divided by $|L|$.

Regarding the baseline algorithms, we select several popular PT methods using the original base classifiers and the decision trees (the WEKA implementation of J48 in its default setting) as the base classifier respectively. In addition, we select several multidata classification methods using *k*-NN and three efficient multilabel classification algorithms—HOMER(Tsoumakas, Katakis, and Vlahavas 2008), MLRDT(X. T. Zhang et al. 2010), and M3L(Hariharan, Vishwanathan, and Varma 2012))[4]—as the baseline methods. Table 3 summarizes the details of all baseline methods. All parameters involved in these algorithms follow the optimal values defined in the original references; for example, the optimal value of k is 10 for ML*k*NN. However, regarding the parameters of CP, MS, and n_{min} in the training of DTML, they are specified by users. In this article, we specify these parameters below, namely, $CP = 100$k (every 100 k training instances), $MS = 1.4$ G (the value is limited by the memory configuration of used computers), $n_{min} = 200$, $k = 100$, and ratio $= 0.5$. All experiments are

[3]The simulated database simply connects a database with numerical attributes only and the other one with discrete attributes only in parallel. Both databases are generated in the multilabeled data generators from MOA (Bifet et al. 2010).

[4]We only give experimental results of hamLoss and OneError and the time overhead for M3L to compare our algorithm. This is because the source code of M3L (Hariharan, Vishwanathan, and Varma 2012) does not provide results on other estimation metrics.

TABLE 2 Descriptions of data sets

		Instances		Attributes				
Data set	Domain	Train	Test	Discrete	Numerical	Labels	Label cardinality	Label density
Emotions	Music	391	202	0	72	6	1.869	0.311
Yeast	Biology	1,500	917	0	103	14	4.237	0.303
Scene	Multimedia	1,211	1,196	0	294	6	1.074	0.179
Mediamill	Multimedia	30,993	12,914	0	120	101	4.376	0.043
SLASH-DOT-F	Text	3,782	3,782	1,079	0	22	1.180	0.208
IMDB-ECC-F	Movie	95,424	95,424	1,001	0	28	1.920	0.036
IMDB-F	Movie	120,919	120,919	1,001	0	28	2.000	0.037
Simulated-Data	Synthetic data generated in massive online analysis (MOA) (Bifet et al. 2010)	1,000	500	100	100	16	5.262	0.329

IMBD = Internet movie database.

TABLE 3 Description of baseline algorithms

Baseline algorithm	Base classifier	Type	Source
BR*k*NN (Spyromitros, Tsoumakas, and Vlahavas 2008)	*k*-NN	BR	http://mulan.sourceforge.net/
RAKEL (Tsoumakas, Katakis, and Vlahavas 2011)	SMO/J48	LP	
EPS (Read, Pfahringer, and Holmes 2008)	SMO/J48	LP	
MLCLR (Füurnkranz et al. 2008)	Linear perceptron/J48	Pair-wise	
ML*k*NN (M. L. Zhang and Zhou 2007)	*k*-NN	AA	
IBLR (Cheng and Hüllermeier 2009)	Instance-based learning and logistic regression	AA	
HOMER (Tsoumakas, Katakis, and Vlahavas 2008)	SMO	AA	
M3L (Hariharan, Vishwanathan, and Varma 2012)	Linear kernel function	AA	http://research.microsoft.com/enus/um/people/manik/code/m31/
MLRDT (X. T. Zhang et al. 2010)	Random decision trees	BR or LP	http://www.dice4dm.com/

SMO = Sequential minimal optimization.

conducted on a P4, 3.00-GHz PC with 2 G main memory, running Windows XP Professional.

Evaluation Measures

In this subsection, we give several popular example-based evaluation metrics (Tsoumakas, Katakis, and Vlahavas 2010) used in our experiments. For the definitions of these measures, let D be a multilabel evaluation data set, consisting of N multilabel instances (e_i, Y_i), where $Y_i \subseteq L$ is the set of true labels and $L = \{\lambda_j : j = 1, \cdots, |L|\}$ is the set of all labels. Given an instance e_i, the set of labels predicted by a multilabel classification method is denoted as $Z_i = f(e_i)$, where f is a multilabel classifier. Example-based measures[5] including hamLoss, accuracy, precision, and recall are as follows:

$$hamLoss(F, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \Delta Z_i|}{|L|} \quad accuracy(f, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$
$$precision(f, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{|Z_i|} \quad recall(f, D) = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{|Y_i|},$$

[5]We do not give the evaluation measure $F1$ here, because it can be estimated in precision and recall.

where Δ corresponds to the XOR operation in boolean logic. According to these definitions, we know that the smaller value of hamLoss and the greater values of accuracy, precision, and recall indicate better performance. In addition, we formulate a general evaluation measure regarding all example-based measures, called $tradeOff_{EM}$.

$$tradeOff_{EM} = \frac{hamLoss}{accuracy \times precison \times recall + \Delta}$$

This definition shows that a smaller value of $tradeOff_{EM}$ indicates the better overall performance on example-based measures. In this definition, the smoothing value of Δ is added to avoid dividing zero. Δ is set to a very small value; for example, 10^{-10}.

Relationship between the Height of the Decision Tree and the Classification Performance

Figure 2 shows the relationship between the height of the decision tree and the classification performance in our algorithm. We only report the trade-off values on the example-based measures varying with the height of the decision tree. A similar conclusion was obtained from experimental results on different data sets. Thus, to avoid redundancy, we only give experimental studies conducted on two representative databases, namely, Yeast with numerical attributes and IMDB-F with discrete attributes. In the observation, we can see that as the height of the decision tree increases, the

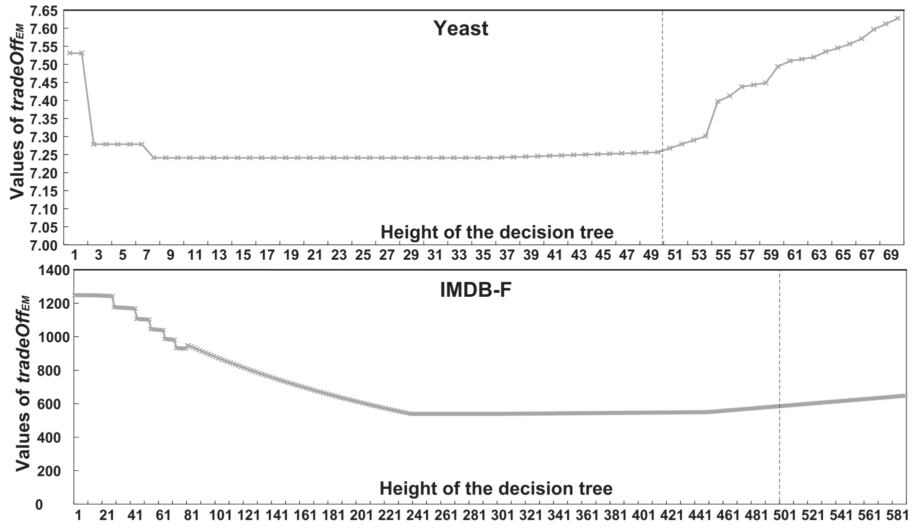

FIGURE 2 Prediction results varying with the height of the decision tree.

overall performance of $tradeOff_{EM}$ becomes better until it reaches half of the attribute dimensions (marked in the dotted line). This validates the conclusion mentioned in section "Classification Risk Analysis of DTML" that as the height of the decision tree increases, the classification performance will be improved, and then it will begin overfitting and start degrading. Therefore, in our algorithm, the value of maxH is specified by half of the attribute dimensions.

Performance Evaluation

In this subsection, we evaluate our algorithm against nine state-of-the-art algorithms with regard to effectiveness and efficiency. In our experiments, if algorithms cannot accomplish the calculation in 24 h on a data set, their experimental results are marked with "—" (In this case, we think that these algorithms are not comparable to our algorithm). If there is an exception in the running (i.e., out of memory), the experimental results are marked with "/".

Table 4 reports the effectiveness between DTML and nine baseline algorithms conducted on five small-sized data sets and three large-scale data sets with a larger number of labels or a large number of instances.

Considering the performance on five small-sized data sets, firstly, DTML performs best on hamLoss. This is because DTML adopts the threshold constraint in the *k*-NN classifier, which avoids labeling blindly. Secondly, considering the evaluation measures of accuracy, precision, and recall, DTML sometimes performs worse than the PT methods based on *k*-NN (such as BR*k*NN), sequential minimal optimization (SMO) (such as RAKEL and EPS), and perceptron (such as MLCLR) and the AA methods based on SMO (such as HOMER), *k*-NN (such as ML*k*NN), and logistic regression (such as IBLR). The reasons are respectively analyzed below. The classifiers of SMO, perceptron, *k*-NN, and logistic regression are more suitable for handling numerical attributes compared to the decision tree with informed split tests in our algorithm. Thus, in general, these classifier-based methods are superior to our algorithm on the data sets of Emotions, Yeast, and Scene with pure numerical attributes, whereas they are inferior to our algorithm on SLASHDOT-F and SimulatedData with discrete attributes. However, considering the decision tree–based PT methods, our algorithm performs better than EPS, RAKEL, and MLCLR on most data sets except MLRDT. This is because, compared to RAKEL and MLCLR based on the decision tree, our algorithm introduces the *k*-NN classifier at leaves of the decision tree instead of a single decision tree used in these methods. Compared to the MLRDT algorithm, our algorithm does not present advantages on the evaluation measures of accuracy, precision, and recall. This is because MLRDT is an ensemble model of random decision trees and the ensemble model usually

TABLE 4 Estimations on all data sets

	Algorithm												
		RAKEL		EPS		MLCLR							
Measures	BR*k*NN	SMO	J48	SMO	J48	Perceptron	DTML	ML*k*NN	J48	IBLR	HOMER	M3L	MLRDT
On five small sized data sets													
Emotions													
hamLoss	0.197	0.207	0.223	0.206	0.214	0.212	0.263	0.059	0.209	0.189	0.276	0.385	0.272
accuracy	0.510	0.554	0.503	0.552	0.537	0.505	0.434	0.351	0.506	0.555	0.416	N/A	0.467
precision	N/A	N/A	N/A	0.648	N/A	N/A	N/A	0.512	N/A	N/A	N/A	N/A	0.589
recall	0.570	0.658	0.629	0.681	0.629	0.577	0.600	0.512	0.573	0.637	0.496	N/A	0.604
tradeOf f_{EM}	N/A	N/A	N/A	0.846	N/A	N/A	N/A	0.641	N/A	N/A	N/A	N/A	1.637
Yeast													
hamLoss	0.203	0.198	0.233	0.203	0.212	0.199	0.226	0.073	0.198	0.199	0.266	0.187	0.229
accuracy	0.482	0.529	0.478	0.546	0.485	0.506	0.468	0.505	0.505	0.506	0.368	N/A	0.496
precision	N/A	0.694	N/A	N/A	N/A	0.710	N/A	0.679	0.677	N/A	N/A	N/A	0.629
recall	0.541	0.627	0.609	0.637	0.576	0.587	0.597	0.718	0.575	0.581	0.453	N/A	0.633
tradeOf f_{EM}	N/A	0.860	N/A	N/A	N/A	0.944	N/A	0.314	0.950	N/A	N/A	N/A	1.160
Scene													
hamLoss	0.108	0.100	0.115	0.093	0.101	0.118	0.141	0.028	0.095	0.091	0.159	0.178	0.140
accuracy	0.542	0.656	0.577	0.727	0.542	0.570	0.503	0.314	0.661	0.647	0.534	N/A	0.589
precision	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.331	N/A	N/A	N/A	N/A	0.622
recall	0.542	0.695	0.622	0.739	0.542	0.649	0.632	0.332	0.691	0.655	0.551	N/A	0.589
tradeOf f_{EM}	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.811	N/A	N/A	N/A	N/A	0.649
SLASHDOT-F													
hamLoss	0.083	0.004	0.031	0.051	0.013	0.004	0.014	0.005	0.048	0.045	0.053	0.004	0.003
accuracy	0.153	N/A	0.497	0.483	N/A	N/A	N/A	0.548	0.117	0.202	0.450	N/A	0.717
precision	N/A	N/A	N/A	0.532	N/A	N/A	N/A	0.611	N/A	N/A	N/A	N/A	0.738
recall	0.153	1.000	0.511	0.489	1.000	N/A	1.000	0.611	0.117	0.208	0.543	N/A	0.718
tradeOf f_{EM}	N/A	N/A	N/A	0.406	N/A	N/A	N/A	0.024	N/A	N/A	N/A	N/A	0.008

(*Continued*)

TABLE 4 (*Continued*)

	Algorithm												
		RAKEL		EPS		MLCLR							
Measures	BR*k*NN	SMO	J48	SMO	J48	Perceptron	DTML	ML*k*NN	J48	IBLR	HOMER	M3L	MLRDT
Simulated Data													
hamLoss	0.302	0.249	0.227	0.267	0.275	0.241	0.222	0.049	0.250	0.252	0.273	0.231	0.261
accuracy	0.434	0.489	0.519	0.434	0.416	0.484	0.419	0.503	0.381	0.389	0.478	N/A	0.454
precision	0.559	0.628	0.660	0.604	0.589	N/A	N/A	0.749	0.700	0.683	0.596	N/A	0.607
recall	0.608	0.646	0.681	0.563	0.536	0.617	0.644	0.750	0.423	0.446	0.682	N/A	0.603
tradeOf f_{EM}	2.044	1.253	0.975	1.805	2.091	N/A	N/A	0.208	2.219	2.128	1.404	N/A	1.569
On three large sized data sets													
Mediamill													
hamLoss	0.032	—	0.034	—	—	—	0.031	0.006	0.031	0.035	0.055	0.308	0.038
accuracy	N/A	—	N/A	—	—	—	N/A	0.351	N/A	N/A	N/A	N/A	0.414
precision	N/A	—	N/A	—	—	—	N/A	0.682	N/A	N/A	N/A	N/A	0.558
recall	0.437	—	0.467	—	—	—	0.407	0.691	0.424	0.391	0.418	N/A	0.596
tradeOf f_{EM}	N/A	—	N/A	—	—	—	N/A	0.036	N/A	N/A	N/A	N/A	0.276
IMDB-ECC-F													
hamLoss	0.072	—	—	—	—	—	—	0.058	0.069	0.069	0.112	—	/
accuracy	0.159	—	—	—	—	—	—	0.108	0.002	0.002	0.192	—	/
precision	N/A	—	—	—	—	—	—	0.113	N/A	N/A	N/A	—	/
recall	0.159	—	—	—	—	—	—	0.141	0.002	0.002	0.317	—	/
tradeOf f_{EM}	N/A	—	—	—	—	—	—	33.706	N/A	N/A	N/A	—	/
IMDB-F													
hamLoss	0.070	—	—	—	—	—	—	0.055	0.071	0.071	0.116	—	/
accuracy	0.040	—	—	—	—	—	—	0.147	0.002	0.002	0.204	—	/
precision	N/A	—	—	—	—	—	—	0.147	N/A	N/A	N/A	—	/
recall	0.040	—	—	—	—	—	—	0.226	0.002	0.002	0.339	—	/
tradeOf f_{EM}	N/A	—	—	—	—	—	—	11.262	N/A	N/A	N/A	—	/

– = no result in 24 hours; / = out of memory.

outperforms the single model. However, regarding the overall performance on tradeoff, DTML wins three times and MLRDT wins twice, and they can beat other baseline algorithms.

Considering the performance on three large-sized data sets, we can observe the obvious advantages of our DTML algorithm. DTML beats all baseline algorithms considering the overall performance on tradeoff. The reason is that the baseline algorithms are batch learning algorithms, and they cannot handle the large-scale data sets as well as our incremental algorithm due to the heavier space overheads.

On the other hand, we consider the time overheads in DTML compared to nine baseline algorithms. From the experimental results as shown in Table 5, we found the following. DTML does not always perform better on the test time overhead compared to the decision tree–based PT methods (such as RAKEL, EPS, and MLCLR), because our algorithm uses the *k*-NN classifier to classify the test instance after it traverses the decision tree from the root to a leaf. It probably requires more time compared to the majority class method used in RAKEL and MLCLR based on the decision tree. However, it is clearly superior in the training time overhead. Thus, our algorithm still has an advantage for the total time overhead.

In addition, this advantage is very obvious compared to RAKEL and MLCLR based on the classifiers of SMO and linear perceptrons. This is because our decision tree model built incrementally leads to a light time overhead compared to those models built in batch.

DTML is always inferior to the *k*-NN-based PT method (BR*k*NN) in the training time overhead but it is still superior in the total time overhead. This is because the training time complexity in BR*k*NN is $O(1)$, whereas the test time complexity is $\mathrm{O}(kN^2)$, which is much more than DTML. In a similar way, DTML performs more efficiently compared to the *k*-NN-based AA method (ML*k*NN) and the IBLR method based on the instance-based learning and logistic regression, because the number of instances at a leaf N' is much lower than the value of N.

Compared to other efficient multilabel classification methods, DTML performs faster than HOMER and M3L. This is because the basic classifiers in HOMER and M3L need to learn in batch, whereas our model can learn incrementally. However, DTML performs a little slower than MLRDT; this is because MLRDT adopts a random selection strategy in the split tests instead of the informed split tests in DTML. This is beneficial in reducing the computation overhead.

In sum, our DTML algorithm is the second fastest algorithm compared to all baseline algorithms. It preforms a little slower than MLRDT but it is more suitable for handling large-scale data sets due to the lighter space

TABLE 5 Overheads of training and test time on all data sets

	Training + Test Time (s)												
		RAKEL		EPS		MLCLR							
Dat a set	BR*k*NN	SMO	J48	SMO	J48	Perceptron	J48	DTML	ML*k*NN	IBLR	HOMER	M3 L	MLRDT
Emotions	1 + 1	20 + 1	5 + 1	15 + 1	14 + 1	3 + 1	3 + 1	1 + 1	1 + 1	2 + 1	1 + 1	8 + 1	1 + 1
Yeast	1 + 11	176 + 3	130 + 1	314 + 3	275 + 1	72 + 3	69 + 1	5 + 1	17 + 12	18 + 10	6 + 3	3 + 1	1 + 1
Scene	1 + 28	93 + 3	63 + 2	273 + 15	238 + 3	30 + 3	27 + 1	10 + 4	21 + 22	22 + 22	4 + 4	3 + 3	1 + 1
SLASH-DOT-F	1 + 17	15450 + 45	11530 + 20	22753 + 30	20316 + 7	3883 + 78	3498 + 8	16 + 10	16 + 16	25 + 17	85 + 77	50 + 57	9 + 5
simulatedData	1 + 23	318 + 2	182 + 1	352 + 2	314 + 1	216 + 2	153 + 1	1 + 1	13 + 7	115 + 10	81 + 1	15 + 1	1 + 1
Mediamill	1 + 799	—	24466 + 50	—	—	—	23160 + 91	74 + 115	1731 + 715	3162 + 321	507 + 63	5629 + 260	17 + 32
IMDB-ECC-F	1 + 23988	—	—	—	—	—	—	297 + 428	22610 + 22712	20020 + 19314	7155 + 5859	—	/
IMDB-F	1 + 31115	—	—	—	—	—	—	395 + 921	37230 + 36312	32180 + 29849	9562 + 7934	—	/

– = no result in 24 hours; / = out of memory.

overheads. In addition, DTML performs best regarding the overall performance on the example-based measures, which wins six times on the value of *tradeOf* f_{EM} from eight data sets. Therefore, we conclude that our DTML algorithm is an efficient and effective algorithm, especially in domains with large-scale data sets.

CONCLUSIONS

In this article, we have proposed a new algorithm adaptation method for multilabel classification, called DTML. It utilizes an incremental decision tree to reduce the training time and adopts the *k*-NN classifier at leaves to maintain the effectiveness. Experiments show that our algorithm is more effective on the example-based evaluation measures compared to nine state-of-the-art algorithms for multilabel classification. In addition, it is a fast algorithm and is more suitable for handling large-scale data sets. In future work, we will focus on how to handle sparse multilabel data and multilabel data streams with concept drifts.

REFERENCES

Bifet, A., G. Holmes, R. Kirkby, and B. Pfahringer. 2010. MOA: Massive online analysis. *Machine Learning Research* 11:1601–604.

Borges, H. B., and J. C. Nievola. 2012. Multi-label hierarchical classification using a competitive neural network for protein function prediction. Paper presented at International Joint Conference on Neural Networks, Brisbane, Australia, June 10–15.

Cheng, W., and E. Hüllermeier. 2009. Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning* 76:211–25.

Clare, A., and R. D. King. 2001. Knowledge discovery in multi-label phenotype data. Paper presented at Fifth European Conference on Principles of Data Mining and Knowledge Discovery, Freiburg, Germany, September 3–7.

Comité, F. D., R. Gilleron, and M. Tommasi. 2003. Learning multi-label altenating decision tree from texts and data. Paper presented at Machine Learning and Data Mining in Pattern Recognition, Leipzig, Germany, July 5–7.

Elisseeff, A., and J. Weston. 2002. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems*, ed. T. G. Dietterich, S. Becker, and Z. Ghahramani, 681–87. Cambridge, MA: MIT Press.

Füurnkranz, J., E. Hüllermeier, M. E. Loza, and K. Brinker. 2008. Multilabel classification via calibrated label ranking. *Machine Learning* 73:133–53. doi:10.1007/s10994-008-5064-8.

Hariharan, B., L. Z. Manor, S. V. N. Vishwanathan, and M. Varma. 2010. Large scale max-margin multilabel classification with priors. Paper presented at International Conference on Machine Learning, Haifa, Israel, June 21–24.

Hariharan, B., S. V. N. Vishwanathan, and M. Varma. 2012. Efficient max-margin multi-label classification with applications to zero-shot learning. *Machine Learning* 88:127–55. doi:10.1007/s10994-012-5291-x.

Hastie, T., R. Tibshirani, and J. Friedman. 2008. *The elements of statistical learning: Data mining, inference, and prediction.* New York, NY: Springer-Verlag.

Jiang, A., C. Wang, and Y. Zhu. 2008. Calibrated rank-SVM for multi-label image categorization. Paper presented at International Joint Conference on Neural Networks, Hong Kong, China, June 1–6.

Jiang, J.-Y., S.-C. Tsai, and S.-J. Lee. 2012. FS*k*NN: Multi-label text categorization based on fuzzy similarity and *k* nearest neighbors. *Expert Systems with Applications* 39:2813–21. doi:10.1016/j.eswa.2011.08.141.

Petrovskiy, M. 2006. Paired comparisons method for solving multi-label learning problem. Paper presented at International Conference on Hybrid Intelligent Systems, Auckland, New Zealand, December 13–15.

Read, J., B. Pfahringer, and G. Holmes. 2008. Multi-label classification using ensembles of pruned sets. Paper presented at IEEE International Conference on Data Mining, Pisa, Italy, December 15–19.

Read, J., B. Pfahringer, G. Holmes, and E. Frank. 2009. Classifier chains for multi-label classification. Paper presented at European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Bled, Slovenia, September 7–11.

Sapozhnikova, E. P. 2009. Multi-label classification with art neural networks Paper presented at Second International Workshop on Knowledge Discovery and Data Mining, Moscow, Russia, January 23–25.

Schapire, R. E., and Y. Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning* 39:135–68. doi:10.1023/A:1007649029923.

Spyromitros, E., G. Tsoumakas, and I. Vlahavas. 2008. An empirical study of lazy multilabel classification algorithms. Paper presented at Fifth Hellenic Conference on AI, Syros, Greece, October 2–4.

Tsoumakas, G., I. Katakis, and I. Vlahavas. 2008. Effective and efficient multilabel classification in domains with large number of labels. Paper presented at ECML/PKDD Workshop on Mining Multidimensional Data, Antwerp, Belgium, September 15–19.

Tsoumakas, G., I. Katakis, and I. Vlahavas. 2010. Mining multi-label data. In *Data mining and knowledge discovery handbook*, ed. O. Maimon, and L. Rokach, 667–85. New York, NY: Springer.

Tsoumakas, G., I. Katakis, and I. Vlahavas. 2011. Random *k*-label sets for multi-label classification. *IEEE Transactions on Knowledge and Data Engineering* 23:1079–89. doi:10.1109/TKDE.2010.164.

Tsoumakas, G., and I. Vlahavas. 2007. Random *k*-label sets: An ensemble method for multilabel classification. Paper presented at European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Warsaw , Poland, September 17–21.

Xu, J. H. 2012. An efficient multi-label support vector machine with a zero label. *Expert Systems and Applications* 39:4796–804.

Xu, M., Y. Li, and Z. Zhou. 2013. Multi-label learning with pro loss. Paper presented at Twenty-Seventh AAAI Conference on Artificial Intelligence, Bellevue, Washington, July 14–18.

Zhang, M. L., and Z. H. Zhou. 2006. Multi-label neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering* 18:1338–51.

Zhang, M. L., and Z. H. Zhou. 2007. ML-*k*NN: A lazy learning approach to multi-label learning. *Pattern Recognition* 40:2038–48.

Zhang, X. T., Q. Yuan, S. W. Zhao, W. Fan, W. T. Zheng, and Z. Wang. 2010. Multi-label classification without the multi-label cost. Paper presented at SIAM Conference on Data Mining, Columbus, Ohio, April 29–May 1.