An Ensemble-based Approach to Fast Classification of Multi-label Data Streams

Xiangnan Kong
Department of Computer Science
University of Illinois at Chicago
Chicago, IL, USA
Email: xkong4@uic.edu

Philip S. Yu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL, USA
Email: psyu@cs.uic.edu

***Abstract*—Network operators are continuously confronted with online events, such as online messages, blog updates, etc. Due to the huge volume of these events and the fast changes of the topics, it is critical to manage them promptly and effectively. There have been many softwares and algorithms developed to conduct automatic classification over these stream data. Conventional approaches focus on single-label scenarios, where each event can only be tagged with one label. However, in many stream data, each event can be tagged with more than one labels. Effective stream classification systems should be able to consider the unique properties of multi-label stream data, such as large data volumes, label correlations and concept drifts. To address these challenges, in this paper, we propose an efficient and effective method for multi-label stream classification based on an ensemble of fading random trees. The proposed model can efficiently process high-speed multi-label stream data with concept drifts. Empirical studies on real-world tasks demonstrate that our method can maintain a high accuracy in multi-label stream classification, while providing a very efficient solution to the task.**

***Index Terms*—Data stream, data mining, multi-label classification, random tree**

I. Introduction

Due to the recent advances in computer networks and data storage, network operators are continuously confronted with large amount of online events which are produced at a very high speed. Examples of such online events include text streams of online news, emails, twitter posts, credit card transactions, stock market data, network traffic records, *etc.* These data are called *stream data*. It is important to manage them promptly and effectively. For example, in email systems, researchers want to be able to automatically classify vast amount of incoming emails into different categories, like spams, personal emails, business emails, important emails, *etc.*; In bank systems, people are interested in monitoring the credit card transactions and classify all the incoming transactions in real-time. Although there have been many softwares and algorithms developed to conduct automatic classification over these online events, it remains a challenging task due to following factors: (1) vast amount of data arrive at high speed [9]. The whole data can seldom fit into the memory, while they need to be analyzed in real time. Therefore an efficient system for online events should be able to process data in one-pass and use only limited memory to deal with the potentially infinite data streams; (2) usually, the concepts within the events can evolve or drift over time [34]. For example, the behavior of the online users may change over time depending on various factors. Effective systems for online events should be able to adapt to the concept changes and revise itself accordingly. Motivated by these challenges, *stream classification* has received considerable attention in the last decade.

In the literature, stream classification problem has been extensively studied [9], [31], [17], [3], [4]. Conventional approaches focus on classifying stream data under single-label settings (binary classification or multi-class classification) [17], [31], [13], [18], [33], [3], which assume, explicitly or implicitly, that each online event can only be assigned with one class label. However, in many real-world applications, each online event can be associated with more than one label. For example, an online news article about 'Steve Jobs resigns as Apple CEO' can be annotated with labels like *IT*, *legend*, *company*, *etc.* The analysis and management of the online documents can be greatly improved if one can train a model to automatically tag each document with multiple labels in real time. This setting is also known as multi-label classification which aims at designing models to classify each instance into multiple categories. It has been shown useful in many real-world applications such as text categorization [24], [28] and bioinformatics [10].

Formally, the multi-label stream classification problem corresponds to training a model to associate each instance in a high-speed data stream with a *label set* in the space of all possible label sets, *i.e.* the power set of all labels. The label set space can be extremely large even with a small number of possible labels. Multi-label classification is particularly challenging in data stream scenarios. The reason is that, conventional multi-label classification approaches work under the batch settings which assume all the data are able to fit into the memory and the models can be trained with multiple passes through the entire dataset. But in stream data, the data continuously flood in with very high speed, which make it impossible to be stored in the memory. Stream classification algorithms should only take one-pass through the data stream to train the model, and the model should be able to responds to the stream data in real time.

COLLABORATECOM 2011, October 15-18, Orlando, United States

DOI 10.4108/collaboratecom.2011.31

Despite the value and significance, there is very limited research on multi-label data stream classification problem. Some of the existing solutions focus on extending single-label stream classification approaches to multi-label cases [2], without addressing some special challenges in multi-label data streams.

If we consider stream mining and multi-label classification as a whole, the major research challenges for multi-label stream classification are as follows:

Multiple labels: A multi-label data stream contains multiple label concepts, and different labels usually have correlations with each other. One type of label correlations that people care about in conventional multi-label learning research is the pairwise relationship between different labels. Although many previous works have explicitly studied this issue under batch settings, these approaches cannot be directly applied to stream data due to the one-pass constraint in stream classification. Another type of label correlations in multi-label stream data is the joint sparseness of different labels, *i.e.*, most of the instances can only have a small number of labels in their label sets. In other words, the cardinality of each instance's label set usually can neither be too large (close to the total number of possible labels) nor be too small (be zero). The joint sparseness among multiple labels should be explicitly considered during the label set prediction.

Stream data: Another challenge in multi-label stream classification lies in the huge amount data with high speed and concept drifts. In multi-label data streams, data continuously flood in with very high speed. An ideal model for these data should be able to process the data very efficiently in order to cope with the speed of data stream. What makes the problem even more interesting and challenging is that the concepts within the data streams can evolve or drift over time [34]. For example, the meaning of label concepts in the data stream may change over time due to various reasons. Effective models for data streams should be able to adapt to the concept changes and revise itself accordingly.

In this paper, we propose a fast multi-label stream classification approach, called Streaming Multi-lAbel Random Trees (SMART), to address the above issues. More specifically, we first propose a random-tree based algorithm to keep track of multiple labels in a data stream while considering the label correlations. The trees are built at the beginning of the stream with randomly selected testing variables and random cutting values on the tree nodes. Thus the tree building process is highly efficient while consuming constant memories. The proposed SMART approach is able to efficiently update tree node's statistics with the labeled stream data and classify the incoming testing data in real-time. We show that in order to effectively consider the pairwise label correlations and joint sparseness in multi-label stream classification, it is efficient and sufficient to collect some simple statistics over the labeled data in each tree node: (1) the estimated relevance of each label in each node. (2) the estimated cardinality of the label sets in each node. Then, multiple random trees are trained as an ensemble, and final outputs are averaged over all the trees to improve the generalization abilities of our model. The above processes are highly efficient, and can easily be implemented in parallel fashions, since the trees in the ensemble are independent from each other during both training and testing. Moreover, in order to handle concept drifts in the stream, we design a fading factor on each tree node to gradually reduce the influence of historical data on the statistics of the node. Empirical studies on real-world multi-label data streams demonstrate that the proposed method can maintain a high accuracy in multi-label stream classification, while providing a very efficient solution to the task.

The rest of the paper is organized as follows. We start by a brief review on related work of multi-label classification and data stream classification in Section II. Then Section III defines the problem of multi-label data stream classification, and then we propose a multi-label data stream classification method using a streaming random tree ensemble. Experimental results are reported in Section IV, and we conclude in Section V.

II. Related Work

Our work is related to both multi-label classification and stream classification techniques. We briefly discuss both of them.

Multi-label learning deals with the classification problem where each instance can belong to multiple different classes simultaneously [27], [25]. Conventional multi-label approaches are focused on offline settings. One well-know type of approaches is binary relevance (one-vs-all technique [6]), which transforms the multi-label problem into multiple binary classification problems, one for each label. ML-KNN[37] is one of the binary relevance methods, which extends the lazy learning algorithm, kNN, to a multi-label version. It employs label prior probabilities gained from each example's *k* nearest neighbors and use *maximum a posteriori* (MAP) principle to determine label set. Elisseeff and Weston [10] presented a kernel method RANK-SVM for multi-label classification, by minimizing a loss function named *ranking loss*. Extension of other traditional learning techniques have also been studied, such as probabilistic generative models [24], [32], decision trees [7], neural networks [36], maximal margin methods [16], [20], maximum entropy methods [15], [39] and ensemble methods[12], *etc.*

There are also some previous works on multi-label stream classification problem [2], [26], [35]. The first work [2] builds an ensemble of classifiers trained on successive data chunks. It adopts stacked binary relevance model to handle label correlations among multiple labels. Another work on multi-label stream classification [26] modified the single-label stream classification approach, Heoffding Tree [9] by extending with a mulit-label version of entropy, and on each leaf node a batch multi-label classifier is trained. The latest work on multi-label stream classification is [35], which focus on the class imbalance and concept drift problems. A balanced version of sliding window is used on each class label. Binary relevance model is adopted with kNN as the base learner.

TABLE I
IMPORTANT NOTATIONS

Symbol	Definition
$\mathcal{L}$	all candidate labels, $\mathcal{L} = \{l_1, \cdots, l_m\}$
$\boldsymbol{x}_i$	The i-th stream instance
Y_i	The label set of $\boldsymbol{x}_i$
$\boldsymbol{y}_i$	the binary vector for Y_i, $\boldsymbol{y}_i \in \{0,1\}^{\lvert\mathcal{L}\rvert}$
n_t	the number of random trees in an ensemble
λ	The fading factor
d	The height of the tree
$Node$	A node of the tree
$Node.\boldsymbol{c}$	the aggregated label relevance vector
$Node.n$	the aggregated number of instances
$Node.\theta$	the aggregated cardinality of label sets
$Node.t$	the time stamp of the latest update

Many works have also been proposed to single-label stream classification problem [2], [26]. There two set of solutions: single-model based and ensemble based. Single-model based approaches use new data to incrementally update their model so that the model can scale to large data volume [9]. These approaches need to incrementally modify the structure of the model with complex operations. Ensemble based approaches, on the other hand, partition the data stream into equal sized chunks, and train multiple base models on different chunks of data. Then all the models are combined for prediction. [33], [23].

We follow a similar strategy to design our classification model with many random tree ensemble methods in either single-label stream classification[1] or multi-label batch classification [38]. But we designed our random tree model to address the special properties in multi-label stream classification problem. Streaming Random Forest[1] builds streaming decision trees by extending Breiman's Random Forests[29], which is focused on single-label data stream classification problems. Random Decision Tree[11] has also be extended to multi-label batch classification problems in[38].

III. PROBLEM FORMULATION

A. Multi-label Stream Classification

We first introduce the notations that will be used throughout this paper. Let $\mathcal{X}$ denote the input feature space, $\mathcal{X} \subseteq \mathbb{R}^d$, and $\mathcal{L} = \{l_1, \cdots, l_m\}$ denote the set of all candidate labels. A multi-label data stream is denoted as $\{(\boldsymbol{x}_1, Y_1), \cdots, (\boldsymbol{x}_t, Y_t), \cdots\}$, where each data point $\boldsymbol{x}_i = (x_i^1, \cdots, x_i^d) \in \mathcal{X}$ is assigned with a set of labels $Y_i \subseteq \mathcal{L}$. The label sets can also be represented as label vectors $\{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_t, \cdots\}$, where $\boldsymbol{y}_i = \left(y_i^{(1)}, \cdots, y_i^{(|\mathcal{L}|)}\right) \in \{0,1\}^{|\mathcal{L}|}$. Here $y_i^j = 1$ iff $l_j \in Y_i$. Let $(\boldsymbol{x}_t, Y_t)$ be the current example in the data stream. The aim of multi-label stream classification is to train a classifier based on both current and historical multi-label examples $\{(\boldsymbol{x}_1, Y_1), \cdots, (\boldsymbol{x}_t, Y_t)\}$ in the stream. Then the model is used to predict the label sets of incoming data points in the stream, *i.e.,* automatically associate each incoming data point $\boldsymbol{x}_i$ with a label set $Y_i \subseteq \mathcal{L}$.

Multi-label stream classification is a non-trivial task due to the following problems:

(P1) How to effectively predict multiple labels for each stream instance, while considering label correlations and joint sparseness in the label set.

(P2) How to design an efficient model to cope with the high speed data streams while maintaining a high accuracy? Clearly, it is impractical to put the entire historical data into the memory, since the large-scale data may result in running out of memory. An ideal algorithm for multi-label stream classification should be very fast in training and testing (with one-pass over the data), and it should only consume a limited size of memory.

(P3) How to make the model be able to adapt to concept drifts in multi-label data streams? The potentially obsolete historical data may result in inaccurate predictions. An ideal model should be able to gradually reduce the influence of the historical data over time.

In the following sections, we will first introduce the framework for multi-label stream classification. Then, we will propose an effective and efficient approach based upon streaming random tree. Moreover, a fading process in our random tree model is introduced to cope with the concept drifts in multi-label data streams.

B. The Proposed Framework

We first address the problem (P1) discussed in Section III-A by defining the multi-label classification as a risk-minimization problem. Our target is to find an optimal model to predict multiple labels for each instance. Suppose a multi-label classification model $\boldsymbol{h}(\boldsymbol{x})$ is a mapping function $\mathcal{X} \to 2^{\mathcal{L}}$. The output of the model is a binary vector:

$$\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \cdots, h_m(\boldsymbol{x}))$$

Usually a multi-label classification model also outputs a real-valued vector, indicating the relevance of each label based upon a ranking function:

$$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x}))$$

In general, we want to find a model which minimizes a risk function $L(\cdot)$ over the joint distribution on $\mathcal{X} \times 2^{\mathcal{L}}$. The risk function should be able to consider the special properties in multi-label stream classification. Generally, in multi-label classification problems, we need to consider two special properties: (a) the pairwise label correlations; (b) joint sparseness on each label set.

For property (a), we only care about the pairwise relationship between each pair of labels. Thus, we use a famous loss function in multi-label learning, called *ranking loss* [10], as our risk function. Ranking loss evaluates the performance of a model's ranking function based upon the ground-truth label sets. It is calculated as the average fraction of incorrectly ordered label pairs:

$$RankLoss(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \frac{1}{|\boldsymbol{y}|(m-|\boldsymbol{y}|)} \sum_{y^{(i)}>y^{(j)}} I(f_i(\boldsymbol{x}) < f_j(\boldsymbol{x}))$$

Here $I(\pi)$ equals 1 if the condition π holds and 0 otherwise. $RankLoss \in [0, 1]$. The smaller the value, the better the performance. It has been prove by [8] that a ranking function that sorts the labels according to their probability of relevance, *i.e.* define the score function $\boldsymbol{f}(\boldsymbol{x})$ as $f_i(\boldsymbol{x}) = P_{\boldsymbol{x}}(y^{(i)} = 1)$, minimize the expected ranking loss. Thus for problem (a), we only need to estimate the probability of relevance for each label, $P_{\boldsymbol{x}}(y^{(i)} = 1)$. Then each label can be ranked based on the estimated relevance, which can minimize the expected ranking loss in multi-label classification. We will future discuss how to effectively estimate $P_{\boldsymbol{x}}(y^{(i)} = 1)$ for each label in the next subsection.

For property (b), we need to decide which subset of labels should be associated with instance $\boldsymbol{x}$, *i.e.* $\boldsymbol{h}(\boldsymbol{x})$. According to the above ranking function, an optimal label set prediction should include the top ranked labels based on $\boldsymbol{f}(\boldsymbol{x})$, where the only variable we need to estimate is the cardinality of the predicted label set. Thus the label set prediction problem becomes a conventional regression problem, where the loss function is:

$$L_c(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{y}) = (|\boldsymbol{y}| - |\boldsymbol{h}(\boldsymbol{x})|)^2$$

.

Thus, the proposed framework for multi-label stream classification is as follows: (1) *Ranking*: We first learn a ranking function which estimates the probability of relevance for each label ($f_i(\boldsymbol{x}) = P_{\boldsymbol{x}}(y^{(i)} = 1)$). (2)*Regression*: Then we learn an additional function to estimate the label set cardinality for each instance, denoted as $\theta(\boldsymbol{x})$. (3)*Label Set Prediction*: Then for each instance, we rank the labels according to the output of the ranking function, and predict the label set with the top ranked $\theta(\boldsymbol{x})$ labels. For example, suppose the ranked order of three labels is (l_3¿ l_1¿ l_2), and the estimated label set cardinality is 2. Then the predicted label set for $\boldsymbol{x}$ would be $\{l_1, l_3\}$.

We chose the above framework for the following reasons:

- The ranking + regression framework is simple yet very effective in handling multi-label classification problem. The ranking loss can be minimized, and joint sparseness of label sets can naturally be solved under this framework. The joint sparseness of different labels are explicitly considered within the regression step.
- The ranking + regression framework is highly efficient, which can cope with the real-time requirement of data stream scenarios. Both the relevance estimation and regression problem can be efficiently solved with our random tree model, as will be discussed in the next subsection.

C. Multi-label Random Tree

In this section, we address the problem (P2) discussed in Section III-A by proposing an efficient method to estimate the label relevance and label set cardinality simultaneously.

SMART Training(n_t, d, λ)

Input:
n_t: the number of random trees in the ensemble.
d: the height of the tree.
λ: the fading factor.

Initialization:
Build n_t multi-label random trees:$\{T_1, \cdots, T_{n_t}\}$;
  Build each random tree with height d;
  Randomly select testing variable and threshold value on each tree node;

Training on Data Stream:
while new instance $(\boldsymbol{x}, \boldsymbol{y})$ arrives at time t
  for each random tree T_i
    TreeUpdate$(T_i.root, \boldsymbol{x}, \boldsymbol{y}, t)$;
end

Output:
$\{T_i\}$ Streaming Multi-label Random Trees.

Fig. 1. Training Process of SMART

TreeUpdate$(Node, \boldsymbol{x}, \boldsymbol{y}, t)$

Input:
$Node$: a node of the random tree
  $\boldsymbol{x}$: a train instance in the data stream
  $\boldsymbol{y}$: label vector
Process:
Update the statistics of $Node$ with fading factor λ.
Collect information from the current instance:
  $Node.n$++;
  $Node.\theta = Node.\theta + 1^\top \boldsymbol{y}$;
  $Node.\boldsymbol{c} = Node.\boldsymbol{c} + \boldsymbol{y}$;
Forward the instance to child nodes:
  if $Node.Level = d$, stop;
  Decise which child node $\boldsymbol{x}$ will traverse
  Update the child node with **TreeUpdate**$(\boldsymbol{x}, \boldsymbol{y}, t)$;

Fig. 2. UpdateTree Process

Our approach starts from a mainstream idea for learning label relevance and regression that builds an ensemble of multiple random trees [1], [38]. As its name implies, this idea is about training a set of random trees where tree nodes are built by randomly selected testing variables and cutting values.

A couple of issues may arise if we directly use these models to multi-label stream data: (a) the random tree model should be able to handle multiple label concepts within the data stream, *i.e.* the model should be able to estimate relevances for all labels (ranking) and estimate the label set cardinalities (regression) simultaneously. (b) the random tree model should be able to handle the high-speed data stream, with concept drifts.

To deal with the above issues, we propose the following approach, called Streaming Multi-lAbel Random Trees (SMART). At the beginning of the data stream, we first train a set of binary random trees with height d. d is a parameter of our model corresponding to the tree sizes. In each tree

node, we randomly select a testing feature with a random splitting value within the valid range. Thus, with each tree node, the feature space $\mathcal{X}$ is partitioned into two subspaces. All the nodes in random trees are trained from the labeled instances in the stream only.

The multi-label random trees are built at the beginning of data stream and timely updated over the data stream as follow: The tree building process starts with all empty tree nodes. when training data points in the data stream arrive, each node keeps updating two types of statistics information for the multiple label data stream: (1) the probabilities of label relevances that traverse through that node, which is similar to conventional decision trees. (2) the estimated cardinality of the label sets in that node, which is similar to regression trees.

DEFINITION *1:* A node of multi-label random tree for a set of m candidate labels $\{l_1, \cdots, l_m\}$ is defined as follows: We collect a $(m+3)$ dimensional simple statistics over the data stream, *i.e.,* $(\boldsymbol{c}, n, \theta, t)$. Suppose there is a set of multi-label streaming points $\{(\boldsymbol{x}_1, Y_1), \cdots, (\boldsymbol{x}_t, Y_t)\}$ traverse to the node. The entries of each of tree node are defined as follow:

- For each class label l_i, the count of label l_i within all label sets of the streaming points, that traverse through the node, are maintained in the i-th entry of vector $\boldsymbol{c}$, *i.e.,* $\boldsymbol{c} = (c_1, \cdots, c_m)$ and $c_j = \sum_{i=1}^{t} y_i^{(j)}$.
- The number of data points that traverse through the node is maintained in n.
- The sum of the label sets' cardinalities is maintained in θ. *i.e.,* $\theta = \sum_{i=1}^{t} |Y_i|$.
- The latest time stamp for instances, that traverse through the node, is maintained in t.

The above simple statistics are maintained at each tree node, which are crucial to the success of our method. Each node maintains the summary information for the multi-label data points that traverse through the node. The $\boldsymbol{c}$, θ and n are all additive sum over different data points, thus they can be efficiently updated over data streams.

Training Process: Figure 1 shows the training process of our multi-label random tree models. The SMART method starts with all empty nodes, and zero values for all the statistics variables within the node. As streaming data points arrive, SMART keeps updating statistic information on each tree node. Figure 2 shows the process when new instances traverse to a tree node. The training process of SMART is highly efficient, with constant time costs for each instance, $O(n_t \times m \times d)$.

Multi-label Prediction: Figure 3 shows the classification process of our multi-label random tree model. When a testing instance arrives, it is forwarded to each of the random tree in the ensemble. Each tree node can output two types of information for the testing instance: (1) the estimated label relevances , $\boldsymbol{f}(\boldsymbol{x}) = \frac{Node.\boldsymbol{c}}{Node.n}$, and (2) the estimated label set cardinality, $\frac{Node.\theta}{Node.n}$. We only output with the last non-empty node that the instance traverses on each random tree.

SMART Classify$(\boldsymbol{x})$

Input:
$\{T_i\}$: Multi-label random trees.
$\boldsymbol{x}$: a test instance in the data stream.
Process:
1 **for** each random tree T_i
2 Forward instance $\boldsymbol{x}$ to each tree
$(\boldsymbol{p}_i, \theta_i) \leftarrow$ **TreePredict**$(\boldsymbol{x}, T_i.root)$;
3 **end**
4 Average the predictions of all random trees
$\boldsymbol{p} = \sum_i \boldsymbol{p}_i / n_t$ and $\theta = \sum_i \theta_i / n_t$;
5 Sort all the labels according to their scores $\boldsymbol{p}$ in descending order;
6 Let $\boldsymbol{y}$ be the set of first $\lfloor \theta + 0.5 \rfloor$ labels in sorted list;
7 **return** $\boldsymbol{y}$ and $\boldsymbol{p}$;
Output:
$\boldsymbol{y}$: the predicted label set;
$\boldsymbol{p}$: the predicted scores.

Fig. 3. Classification Process of SMART

TreePredict$(Node, \boldsymbol{x})$

Input:
$Node$: a node of the random tree
$\boldsymbol{x}$: a testing instance in the data stream
Process:
1 **if** $Node$ is leaf node
2 **return**: $\boldsymbol{p} = Node.\boldsymbol{c}/Node.n$ and $\theta = Node.\theta/Node.n$;
3 Decide which child node ($child$) will $\boldsymbol{x}$ traverse;
4 **if** $child.n == 0$, *i.e.,* child node is empty
5 **return**: $\boldsymbol{p} = Node.\boldsymbol{c}/Node.n$ and $\theta = Node.\theta/Node.n$;
6 **else**: forward $\boldsymbol{x}$ to the node $child$
7 **return**: **TreePredict**$(child, \boldsymbol{x})$;
Output:
$\boldsymbol{p}$ vector of label relevances;
θ estimated label set cardinality.

Fig. 4. The Prediction Process of a Single Random Tree

Then, the final outputs for $\boldsymbol{x}$ are computed by averaging over the ensemble of n_t random trees $\{T_1, \cdots, T_{n_t}\}$, as shown in Figure 3.

$$\boldsymbol{p} = \sum_{i=1}^{n_t} \boldsymbol{p}_i / n_t \text{ and } \theta = \sum_{i=1}^{n_t} \theta_i / n_t$$

where $\boldsymbol{p}_i$ denotes T_i's probability outputs for $\boldsymbol{x}$'s label relevances, and θ_i denotes T_i's estimated cardinality for $\boldsymbol{x}$'s label set.

Then the label set predication for $\boldsymbol{x}$ can be determined by the following steps: (1) Ranking all labels according to the final probability values for $\boldsymbol{x}$ in descending order (Line 5); (2) Predict the top ranked $\lfloor \theta + 0.5 \rfloor$ labels as the label set for $\boldsymbol{x}$ (Line 6).

D. Handling Concept Drifts

In the previous subsection, our multi-label random tree approach focused on static data streams without concept drifts. In this subsection, we address the problem (P3) discussed in Section III-A, to discuss how to deal with concept drifts in our proposed SMART approach.

To handle the concept drifts in the data streams, we introduce a fading factor in each tree node to gradually reduce the influences of historical data. We assume that each instance is weighted with by a *fading function*, $F(\cdot)$, based upon the time stamp t. Similar to [5], we define the fading function as follows: Suppose an instance $\boldsymbol{x}_i$ with time stamp t_i, and the current time is t. The weight of instance $\boldsymbol{x}_i$ at time t is $W_i(t) = 2^{-(t-t_i)/\lambda}$. When t increases over time, instance $\boldsymbol{x}_i$ gradually turns into historical data and its weight decreases exponentially with t. Here λ is a parameter of our model, called *fading factor*, indicating the speed of the fading effects. Generally, λ represents the period of time when the weights of the instancse will be reduced by half. The higher the value of λ, the slower the weight of each instance will decay.

Then we can use the fading function to maintain the weighted sum of version of statistics on each random tree node, so that the influence of the historical data can be gradually reduced. Suppose the current time is t, and the last updated time stamp for a tree node $Node$ is $Node.t$. When a new instance traverse to $Node$, we only need to update the statistics of $Node$ by multiplying each value with $2^{-(t-Node.t)/\lambda}$. This process is highly efficient, only takes constant time for each training data.

IV. Experiments

In this section, we conduct experiments to examine the performances of SMART in real-world multi-label data streams.

A. Experiment Setup

Data Collections: In order to evaluate the multi-label stream classification performances, we tested our algorithm on three real-world data sets as follows: (Summarized in Table II.)

- MediaMill: The first dataset is for video annotation task, which is part of the MediaMill challenge[1] for automated detection of semantic concepts within the video frame. This dataset contains (43907) video frames annotated with 101 label concepts. We use the setting of experiment 1 as described in [30], where each video frame is represented by 120 visual features.
- TMC2007: The second dataset is for text classification task, which is based on the competition organized by the text mining workshop of the 7th SIAM international conference on data mining[2]. The dataset contains (28,596) aviation safety reports, each annotated with a subset of the 22 problem types that appear during flights. Safety Reports are represented as instances using bag-of-words, and we remove the infrequent words that occur in less than 10% of the documents.
- RCV1-v2: The third dataset is a large-scale dataset for text classification task. It is based on the well known benchmark dataset for text classification, Reuters (RCV1) dataset. We use the topics full set[3] that contains (804,414) news articles. Each article is assigned into a subset of the 103 topics. An detailed description of the RCV1 dataset can be found in [21].

TABLE II
A summary of datasets used. $|\mathcal{D}|$ denotes the number of instances; $|\mathcal{L}|$ denotes the number of classes; $|\mathcal{X}|$ denotes the number of dimensions. 'lDens' denotes the label density of the dataset.

	Properties					
Data set	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{L}\rvert$	Avg $\lvert Y\rvert$	lDens	Domain
MediaMill	43,907	120	101	4.415	0.044	Video
TMC2007	28,596	204	22	2.158	0.098	Text
RCV1-v2	804,414	203	103	3.241	0.031	Text

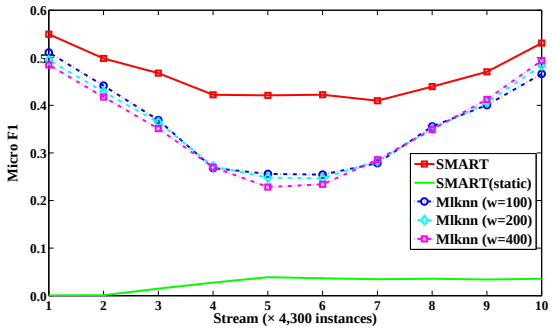
We generate each dataset into a stream of instances according to their original orders. To simulate concept drifts, we create a new multi-label data stream by shuffling the labels, *i.e.*, $l_1 \rightarrow l_2 \rightarrow \cdots \rightarrow l_{|\mathcal{L}|} \rightarrow l_1$ for all instances. Then the new data stream is mixed into the old stream: As the data stream continues, the two data streams are combined using percentages $(P_{\text{old}}, P_{\text{new}}) = (100\%, 0\%), (90\%, 10\%), (80\%, 20\%), \cdots, (0\%, 100\%)$.

Comparing Methods: We compare the performance of our approach against two baselines:

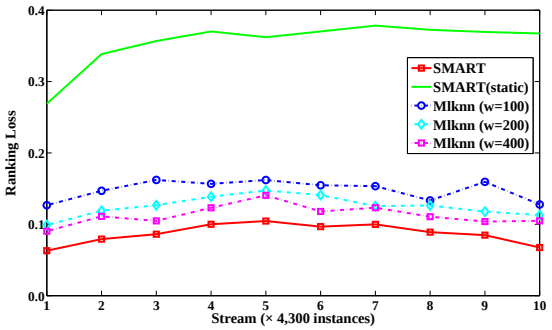
- Stream Multi-lAbel Random Tree (SMART): The proposed multi-label stream classification method. The default parameter settings for SMART are as follows: the tree height $d = 15$; the number of trees in the ensemble $n_t = 20$; fading factor $\lambda = 200$. The sensitivity of the parameters are discussed later in Section IV-C.
- SMART(static): Our stream multi-label random tree method without decay, *i.e.*, $\lambda = \infty$. All other parameters are set the same as SMART.
- ML-KNN[37]: A kNN based multi-label classification method, which is one of the state-of-the-art methods in offline settings. In order to fit into the stream classification problem, we adopted an approach, which is similar to sliding window approaches, *i.e.*, we train a ML-KNN classifier on the latest chunk of data, and use it to classify the next chunk of data. Since the data chunk size can affect the classification performance as well as the training time cost, here we test the performance of ML-KNN with different chunk sizes (sliding window sizes).

We used ML-KNN with the above setting for following reason: Due to the real-time constraint and concept drift in streaming environment, it is infeasible to use a large training

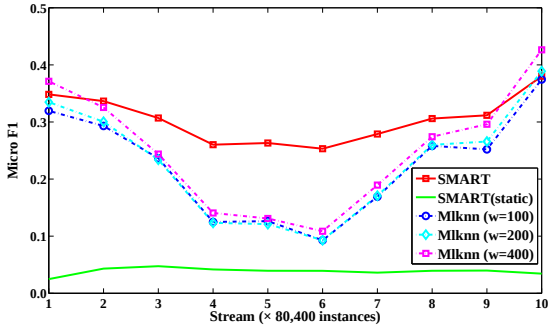
[1]http://www.science.uva.nl/research/mediamill/challenge/
[2]http://www.cs.utk.edu/tmw07/
[3]http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/multilabel.html



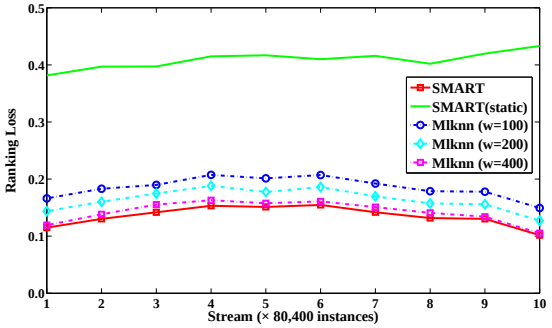
(a) Micro F1↑ on MediaMill dataset



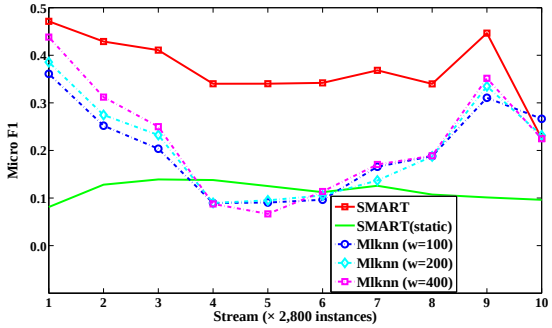
(b) Ranking Loss↓ on MediaMill dataset



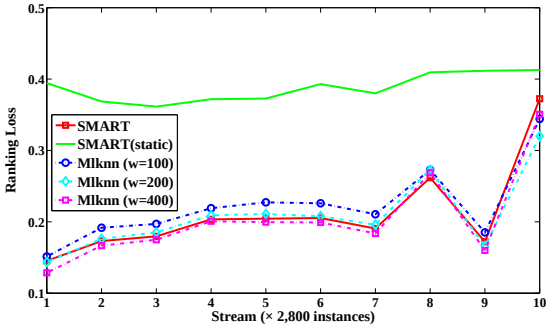
(c) Micro F1↑ on RCV1-v2 dataset



(d) Ranking Loss↓ on RCV1-v2 dataset



(e) Micro F1↑ on TMC2007 dataset



(f) Ranking Loss↓ on TMC2007 dataset

Fig. 5. Performances of multi-label data stream classification. Here "↓" means the smaller the value the better the performance and "↑" means the larger the value the better the performance.

TABLE III
PROCESSING RATE. HERE "↓" MEANS THE SMALLER THE VALUE THE BETTER THE PERFORMANCE AND "↑" MEANS THE LARGER THE VALUE THE BETTER THE PERFORMANCE.

Method	RCV1-v2		MediaMill		TMC2007	
	Rate ↑ (instance/sec)	Time ↓ (sec)	Rate ↑ (instance/sec)	Time ↓ (sec)	Rate ↑ (instance/sec)	Time ↓ (sec)
SMART	**6,484**	**124**	**5,000**	**8.6**	**14,973**	**1.87**
ML-KNN(w=100)	122	6,597	267	150	221	132
ML-KNN(w=200)	115	6,963	227	189	174	161
ML-KNN(w=400)	100	7,990	158	272	137	204

set to train the ML-KNN model. In many data stream classification problems, conventional classifiers trained on the latest data serve as perform better than using more historical data in an evolving data stream.

Evaluation Metrics: Multi-label classification require different evaluation metrics than conventional single-label classification problems. Here we adopt some metrics used in [10], [19], [22] to evaluate the multi-label classification performance in a data stream. Assume we have a multi-label data stream $\mathcal{D}$ containing $|\mathcal{D}|$ multi-label instances $(\boldsymbol{x}_i, Y_i)$, where $Y_i \subseteq \mathcal{L}$ ($i = 1, \cdots, |\mathcal{D}|$). Let $h(\boldsymbol{x}_i) \subseteq \mathcal{L}$ denote a multi-label classifier's predicted label set for $\boldsymbol{x}_i$ and $f(\boldsymbol{x}_i, k)$ denote the classifier's probability outputs for $\boldsymbol{x}_i$ on the k-th label (l_k). We have the following evaluation criteria:

a) Micro F1 [19], [22]: evaluates a classifier's label set prediction performance, which considers both micro average of Precision and Recall with equal importance.

$$MicroF1(h, \mathcal{D}) = \frac{2 \times \sum_{i=1}^{|\mathcal{D}|} |h(\boldsymbol{x}) \cap Y_i|}{\sum_{i=1}^{|\mathcal{D}|} |h(\boldsymbol{x})| + \sum_{i=1}^{|\mathcal{D}|} |Y_i|}$$

The larger the value, the better the performance.

b) Ranking Loss [10]: evaluates the performance of classifier's probability outputs or real-value outputs $f(\boldsymbol{x}_i, k)$. It is calculated as the average fraction of incorrectly ordered label pairs:

$$RankLoss(f, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \frac{1}{|Y_i||\overline{Y_i}|} loss(f, \boldsymbol{x}_i, Y_i)$$

Where the $\overline{Y_i}$ denotes the complementary set of Y_i in $\mathcal{L}$.

$$loss(f, \boldsymbol{x}_i, Y_i) = \sum_{k \in Y_i} \sum_{k' \in \overline{Y_i}} I(f(\boldsymbol{x}_i, k) \leq f(\boldsymbol{x}_i, k'))$$

and $I(\pi)$ equals 1 if the condition π holds and 0 otherwise. $RankLoss \in [0, 1]$. The smaller the value, the better the performance.

The above criteria evaluate the multi-label classification performances from two aspects: 1) Ranking Loss: the performance of the real-value outputs. 2) Micro F1: the performance of the label set predictions.

In addition to the classification performances, we also show the running time of each approach. All experiments are conducted on machines with 24 GB RAM and Intel Xeon^{TM}Quad-Core CPU of 2.27 GHz.

B. Multi-label Classification Performance

To evaluate the multi-label stream classification performances, a prequential setting similar to [14] is used, where we first divide the data stream into chunks of the same size (100 by default) and each chunk is first used for testing the model before being used as training data.

The results are summarized in Figure 5. We show the multi-label classification performances of the compared methods on three real-world datasets. To show the evaluation result of the compared methods, we report the average performance on every $|\mathcal{D}|/10$ instances.

TABLE IV
INFLUENCE OF # TREE, n_t, ON SMART (MEDIAMILL DATASET).

# tree	Evaluation Criteria		
	Micro F1 ↑	Ranking Loss ↓	Time (sec) ↓
5	0.457	0.094	2.7
10	0.463	0.087	4.9
15	0.465	0.087	6.9
20	0.463	0.086	8.6
25	**0.466**	0.085	10.9
30	0.465	**0.084**	12.8

TABLE V
INFLUENCE OF TREE HEIGHT, d, ON SMART (MEDIAMILL DATASET).

Height	Evaluation Criteria		
	Micro F1 ↑	Ranking Loss ↓	Time (sec) ↓
5	0.451	0.092	3.5
10	0.461	0.088	5.5
15	0.463	0.087	8.6
20	**0.468**	**0.085**	15.7

TABLE VI
INFLUENCE OF THE FADING FACTOR, λ, ON SMART (MEDIAMILL DATASET).

λ	Evaluation Criteria		
	Micro F1 ↑	Ranking Loss ↓	Time (sec) ↓
100	0.464	0.089	8.3
200	0.463	0.086	8.6
400	0.463	**0.082**	8.8
800	**0.467**	0.083	8.7
1600	0.465	0.084	9.6

Now, we first study the effectiveness of our method in handling concept-drifting multi-label data streams. Figure 5 indicates that, 1) Our proposed SMART method outperforms all the other baseline methods in almost all cases. This is because SMART can effectively handle the concept drifts within the stream data by reducing the influence of historical data. Thus it can effectively adapt to concept drifts and make use of both the latest data in the data stream and some of the historical data to train the random tree models. This setting is crucial for streaming environment with concept drifts. 2) The SMART(static) method, which never reduce the influence of historical data in the model, *i.e.* it uses all the historical data to train the model, can not adjust to the concept drifts in the data stream. These results support our first intuition, that SMART method with fading operations can effectively adapt to concept drifts in evolving data streams.

We further observe that SMART's performances are better than our second baseline ML-KNN, with different sliding window sizes. Although we can still enlarge the window size of ML-KNN to further improve the classification performances,

however, the training and testing time cost of ML-KNN will increase accordingly. Thus, we only compare with the above settings of ML-KNN which can process the high-speed data streams in real-time. The classification results support our second intuition thatSMART can effectively keep track of multiple label concepts within the data stream and can effectively make good multi-label predictions for new streaming data.

In Table III, we report the running time (training+testing) of the compared methods on all the datasets. As been shown, our SMART approach is much more efficient than other baselines and can process 5,000 - 15,000 data points in each second. This is because the our random tree approach can speed up the training process by randomly select testing variable and splitting values on each node. Our SMART approach is a very fast one-pass algorithm. It only collect simple statistics along the data stream, without storing any instance in the memory. Moreover, it is worth mentioning that our SMART approach takes constant memory cost while processing the data stream. This is crucial for data stream applications, since no additional memory will be requested after SMART starting the data processing.

C. Sensitivity of Parameters

We further study the influences of the parameters on our SMART method. We compare the classification performance and the running time of SMART approach with different settings on n_t (the number of trees in the ensemble), d (the height of each random tree) and λ (fading factor). The average performances over the whole data stream under each parameter settings are reported.

Table IV shows the performance of SMART with different number of trees in the ensemble. Generally, the classification performances will increase with more trees in the ensemble. This is desirable, as with all ensemble learning methods, the performance improves as the number of trees increases. However, after a certain point (10), there is no significant improvement. It is worth noting that, in our current implementation, we sequentially train each random tree in the ensemble, thus the running time increases when n_t becomes larger. However, our approach is very easy to be fit into parellel training and testing on multi-core systems, since each random tree can be trained and tested independently.

Table V shows the performance of SMART with different tree sizes. Generally, the larger each random tree is, the better the performance we can get. This is because the accuracy of label relevance estimation and regression will both be improved with larger trees. However, larger trees will consume much larger memory. Thus, in practice, we usually set the maximium height of random trees as $d \leq 20$.

Table VI shows the performance of SMART with different fading factors. The fading factor λ controls how fast SMART reduces the influence of historical data. The larger λ is, the slower the fading process is. We observe that our method is not quite sensitive to the parameter λ.

V. Conclusion

In this paper, we study the multi-label classification problem for evolving data streams. It is significantly more challenging than the conventional offline problems and single-label data stream classification problems because of the properties of data stream and multiple labels assigned to each instances. To address these challenges, we propose a one-pass multi-label data stream classification approach, named Streaming Multi-lAbel Random Trees (SMART), to efficiently update model structure and statistics on each tree node over the multi-label data stream. Then the trained SMART model can effectively and efficiently predict multiple labels for future data points. Empirical studies on real-world tasks show that our multi-label data stream classification approach is very effective and efficient for multi-label stream classification and can automatically adapt to concept drifts.

Acknowledgment

This work is supported in part by NSF through grants IIS-0905215, OISE-0968341, and DBI-0960443.

References

[1] H. Abdulsalam, D. Skillicorn, and P. Martin. Classifying evolving data streams using dynamic streaming random forests. In *DEXA*, pages 643–651, Turin, Italy, 2008.

[2] H. Abdulsalam, D. Skillicorn, and P. Martin. Classifying evolving data streams using dynamic streaming random forest. In *FSKD*, pages 275–279, 2009.

[3] C. Aggarwal, J. Han, J. Wang, and P. Yu. On demand classification of data streams. In *KDD*, pages 503–508, Seattle, WA, 2004.

[4] C. Aggarwal and P. Yu. LOCUST: An online analytical processing framework for high dimensional classification of data streams. In *ICDE*, pages 426–435, Cancun, Mexico, 2008.

[5] C. Agrawal, J. Han, J. Wang, and P. Yu. A framework for projected clustering of high dimensional data streams. In *VLDB*, 2004.

[6] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[7] F. D. Comité, R. Gilleron, and M. Tommasi. Learning multi-label altenating decision tree from texts and data. In *MLDM*, pages 35–49, Leipzig, Germany, 2003.

[8] K. Dembczynski, W. Cheng, and E. Hullermeier. Bayes-optimal multilabel classification via probabilistic classifier chains. In *ICML*, pages 279–286, Haifa, Israel, 2010.

[9] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD*, pages 71–80, Boston, MA, 2000.

[10] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *NIPS*, pages 681–687. 2002.

[11] W. Fan, H. Wang, P. S. Yu, and S. Ma. Is random model better? On its accuracy and efficiency. In *ICDM*, pages 51–58, Melbourne, FL, 2003.

[12] I. Vlahavas G. Tsoumakas. Random k-labelsets: An ensemble method for multilabel classification. In *ECML*, pages 406–417, Warsaw, Poland, 2007.

[13] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *KDD*, pages 523–528, Washington, DC, 2003.

[14] J. Gama, R. Sebastia, and P. P. Rodrigues. Issues in evaluation of stream learning algorithm. In *KDD*, pages 329–338, Paris, France, 2009.

[15] N. Ghamrawi and A. McCallum. Collective multi-label classification. In *CIKM*, pages 195–200, Bremen, Germany, 2005.

[16] S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *PAKDD*, pages 22–30, Sydney, Australia, 2004.

[17] G. Hulten, L. Spencer, and P. Domingos. Mining time changing data streams. In *KDD*, pages 97–106, San Francisco, CA, 2001.

[18] R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In *KDD*, pages 571–576, Washington, DC, 2003.

[19] F. Kang, R. Jin, and R. Sukthankar. Correlated label propagation with application to multi-label learning. In *CVPR*, pages 1719–1726, New York, NY, 2006.

[20] H. Kazawa, T. Izumitani, H. Taira, and E. Maeda. Maximal margin labeling for multi-topic text categorization. In *NIPS*, pages 649–656. 2005.

[21] D. D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[22] Y. Liu, R. Jin, and L. Yang. Semi-supervised multi-label learning by constrained non-negative matrix factorization. In *AAAI*, pages 421–426, Boston, MA, 2006.

[23] M. Masud, J. Gao, L. Khan, and J. Han. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *ICDM*, pages 929–934, Pisa, Italy, 2008.

[24] A. McCallum. Multi-label text classification with a mixture model trained by EM. In *AAAI'99 Workshop on Text Learning*, Orlando, FL, 1999.

[25] J. Read. *Scalable Multi-label Classification*. PhD thesis, University of Waikato, 2010.

[26] J. Read, A. Bifet, G. Holmes, and B. Pfahringer. Efficient multi-label classification for evolving data streams. Technical Report 04, University of Waikato, 2010.

[27] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. In *ECML*, pages 254–269, Bled, Slovenia, 2009.

[28] R. E. Schapire and Y. Singer. Boostexter: a boosting-based system for text categorization. *Machine Learning*, 39(2-3):135–168, 2000.

[29] R. E. Schapire and Y. Singer. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[30] C. Snoek, M. Worring, J. Gemert, J. Geusebroek, and A. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *ACM Multimedia*, pages 421–430, New York, NY, 2006.

[31] W. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *KDD*, pages 377–382, San Francisco, CA, 2001.

[32] N. Ueda and K. Saito. Parametric mixture models for multi-labeled text. In *NIPS*, pages 721–728. 2003.

[33] H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD*, pages 226–235, Washington, DC, 2003.

[34] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[35] E. Xioufis, M. Spiliopoulou, G. Tsoumakas, and I. Vlahavas. Dealing with concept drift and class imbalance in multi-label stream classification. In *IJCAI*, Barcelona, Spain, 2011.

[36] M.-L. Zhang and Z.-H. Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1479–1493, 2006.

[37] M.-L. Zhang and Z.-H. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

[38] X. Zhang, Q. Yuan, S. Zhao, W. Fan, W. Zheng, and D. Wang. Multi-label classification without multi-label cost. In *SDM*, pages 778–789, Columbus, OH, 2010.

[39] S. Zhu, X. Ji, W. Xu, and Y. Gong. Multi-labelled classification using maximum entropy method. In *SIGIR*, pages 274–281, Salvador, Brazil, 2005.