2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Efficient Class Incremental Learning for Multi-label Classification of Evolving Data Streams

Zhongwei Shi
School of Computer Science and Engineering
Guilin University of Electronic Technology
Guilin, China
e-mail: jk_szw@163.com

Yimin Wen*
Guangxi Key Laboratory of Trusted Software
Guilin University of Electronic Technology
Guilin, China
e-mail: ymwen2004@aliyun.com

Yun Xue
School of Municipal and Surveying Engineering
Hunan City University
Changsha, China
e-mail: yunxue1209@163.com

Guoyong Cai
Guangxi Key Laboratory of Trusted Software
Guilin University of Electronic Technology
Guilin, China
e-mail: ccgycai@guet.edu.cn

***Abstract*—Multi-label stream classification has not been fully explored for the unique properties of large data volumes, real-time, label dependencies, etc. Some methods try to take into account label dependencies, but they only focus on the existing frequent label combinations, leading to worse performance for multi-label classification. To deal with these problems, this paper proposes an algorithm which dynamically recognizes some new frequent label combinations and updates the trained classifier by class incremental learning strategy. Experimental results over both real-world and synthetic datasets demonstrate its better predictive performance.**

Keywords—class incremental learning; concept drift; evolving data streams; multi-label classification

I. INTRODUCTION

In the real life, many applications, such as sensor networks, traffic management, social networks, email systems, twitter posts, blogs, etc., generate tremendous amount of data, which is characterized by real-time, high speed, large data volumes, etc. This type of data is called *data stream*. At present, stream classification has become an important research field in data mining community and made great research achievements. Conventional approaches focus on classifying data streams under single-label scenarios where each sample can only be assigned to a single label.

However, in many emerging applications, each sample can be assigned to more than one label i.e. the sample is assigned to a label combination. For example, in the task of online news page classification, a news article about "Alibaba will go public" can be marked with labels like IT, economy, company, etc., which makes it necessary to design an online multi-label classification model to classify such pages into multiple topics.

On the other hand, the target concepts within data streams are often not stable but change over time, which is known as *concept drift* [1]. These changes often make the learning model built on old data inconsistent with the new data. Then it's necessary to exactly and rapidly detect concept drift and update the current learning model in real time [2]. In addition, for the method of detecting concept drift based on the predictive performance of learning model, the better the performance is, the more efficient the detecting is.

Multi-label evolving data streams inherit the properties of multi-label and concept drift, such as large data volumes, high speed, label dependencies and changes within target concepts, which make it more unique. For these properties, Read et al. used *Pruned Set* (*PS*) [3] method to train one base classifier at the leaf nodes of *Multi-label Hoeffding Tree* (*EaHTps* [4]) to deal with the multi-label evolving data streams. Since the whole data can seldom fit into the limit memory, the base classifier is trained from the samples with frequent label combinations in the buffer that is initialized with a number of firstly arrived samples. By pruning away some infrequent label combinations that bring about the over-fitting and complexity problems, *EaHTps* has achieved good performance.

However, the size of the buffer needs to be set in advance. The buffer being small can't represent the whole distribution of the data streams, but being large will take too long time to build the base classifier, even though the buffer may include more frequent label combinations. What's more, the set of frequent label combinations is fixed and doesn't change over time for training the base classifier. This leads to that some potential frequent label combinations aren't stored for class incremental learning, making worse the predictive performance of the learning model. The above factors make it become a challenge to efficiently deal with these frequent label combinations to get good predictive performance for handing multi-label evolving data streams.

In the paper, we particularly analyses the reason why some frequent label combinations can't be selected by the buffer

* To whome correspondence should be addressed.

scheme from the perspective of statistics and experimental results. In order to deal with the challenge, we propose a class incremental learning approach that dynamically recognizes some new frequent label combinations and updates the set of label combinations in real time, which makes the learning model more accurate. Last but not least, multiple groups of experiments with different size of buffer are taken to prove the necessity of class incremental learning.

The rest of the paper is organized as follows. In Section II, some related work is displayed. Section III analyzes the source of the challenge. Section IV presents how to recognize new frequent label combinations and train the class incremental learning model. Experiments and results are showed in section V. The last section concludes the paper.

II. Related Work

The current approaches for multi-label stream classification can be sorted to *Binary Relevance* and *Label Combination* [5], [6]. *Binary Relevance* transforms a multi-label classification problem into multiple binary problems, one for each label, but *Label Combination* treats each label combination as an atomic class to form a single-label problem.

Qu et al. [7] proposed an ensemble strategy to handle the multi-label stream classification. It assumed that the multi-label data streams arrived in chunk with fixed size. Then a group of classifiers were trained: each for a single-label data chunk that was transformed from a multi-label data chunk by the *Binary Relevance* method. Based on this, an ensemble of classifiers was trained on several successive multi-label data chunks. Their following work [8] adopted the *Stacked Binary Relevance* [9] method to learn from each data chunk, which took the label dependencies into account and performed well than their previous method. Similarly, Xioufis et al. [10] also followed the *Binary Relevance* method to transform a multi-label problem into multiple binary problems. To deal with class imbalance and multiple concept drift, they employed two windows with fixed ratio for each label, one for positive and the other for negative samples.

Read et al. [11] proposed a *Multi-label Hoeffding Tree* that was an extension of *Hoeffding Tree* [12]. They used the multi-label information gain to filter the multi-label data streams to different leaf nodes where a multi-label classifier was built by *PS* [3] method based on *Label Combination*. Their subsequent work, named as *EaHTps* [4], presented a method that was a bagged ADWIN-monitored ensemble of *Multi-label Hoeffding Tree* classifiers. ADWIN [13] was used to detect concept drift through monitoring the change of predictive performance. Comparing with the state of art methods including *Multiple Windows Classifier* (*MWC*) [10] and *Meta-BR* (*MBR*) [8], they drew a conclusion that the proposed method achieved the overall highest predictive performance.

In the method of *EaHTps*, there're two stages in the process of training a multi-label classifier at leaf nodes. At first, at the stage of initializing, a number of samples are buffered. Before the buffer is full, a simple *majority-label-set* classifier (the multi-label version of majority-class) needs to be employed to make multi-label classification. While the buffer is full, all the label combinations of the buffered samples are pruned according to their occurring number by *PS* method. The multi-label classifier is initialized by those samples with frequent label combinations whose occurring number are larger than a pre-set threshold.

Then, at the stage of updating, if the label combination of a newly arrived sample is frequent, then the sample will be used to update the built multi-label classifier directly. Otherwise, the strategy of sub-sampling will take effect that the label combination of the sample is replaced with a similar frequent label combination. After that, the sample with altered label combination is used to update the built multi-label classifier to reduce information loss.

From the above process, it's more possible that those label combinations that don't occur frequently in the buffer are considered to be infrequent all the way. But in fact, they're not all infrequent and some potential frequent label combinations should have been recognized and used to update the built multi-label classifier.

III. Motivation Analysis

Before presenting the entire process of the proposed algorithm, let us particularly analyze why some frequent label combinations aren't selected for training the multi-label classifier. We assume: there is a multi-label data stream that contains n (a finite number) samples and each sample is assigned to a label combination. The $m(c)$ denotes the occurring number of a label combination c in the whole data stream.

Then, in the process of randomly sampling b (namely the size of buffer in *EaHTps*) samples from the whole data stream, the probability of the label combination c not being selected can be represented as follows:

$$P(c) = (1-\frac{m(c)}{n})\times(1-\frac{m(c)}{n-1})\times\cdots\times(1-\frac{m(c)}{n-b+1}) = \prod_{i=0}^{b-1}(1-\frac{m(c)}{n-i}) \quad (1)$$

Because n is much larger than b, the process of sampling b samples is approximately equivalent to a simple random sampling with replacement, then

$$P(c) = \prod_{i=0}^{b-1}(1-\frac{m(c)}{n-i}) \approx (1-\frac{m(c)}{n})^b \quad (2)$$

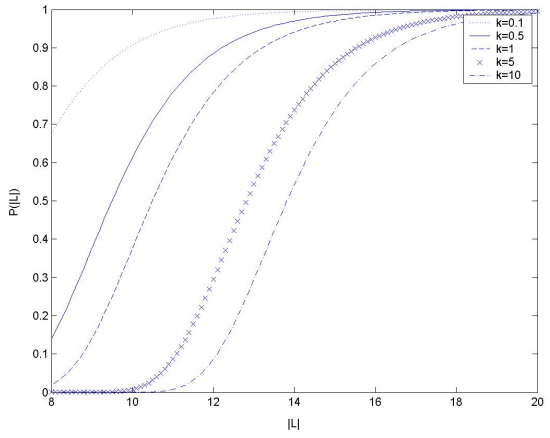

Fig. 1. The curves of $P(|L|)$

A frequent label combination is the one whose occurring number isn't less than k (k>0) times of an average value.

$$m(c) \geq k \times \frac{n}{2^{|L|}-1} \quad (3)$$

In (3), $|L|$ represents the size of label set and $2^{|L|}$-1 is the total number of all label combinations. Then $\frac{n}{2^{|L|}-1}$ denotes the average occurring number of all label combinations.

Thus, for the frequent label combination c, the probability of it not being selected is bigger than $P(|L|)$:

$$P(c) \approx (1-\frac{m(c)}{n})^b \geq P(|L|) \quad (4)$$

$$P(|L|) = (1-\frac{k}{2^{|L|}-1})^b \quad (5)$$

According to (5), what influence the probability of c not being selected are the size of buffer (b) and the size of label set ($|L|$). In the following, we discuss how they take effect by experimental results. *EaHTps* is run 100 times on synthetic datasets and the average results are showed in Table II, III and IV. The specific parameters of the algorithm and dataset are showed in Section V.

At first, when b is set as 1000 [4], [11], the curves of $P(|L|)$ are showed in Fig 1. From Fig 1, it can be seen clearly that the probability $P(|L|)$ is becoming bigger along with the increasing size of label set. Actually, many real-world datasets always involve large size of label set like in Table I.

From Table II, it can be observed that the larger value of $|L|$ always lead to more frequent label combinations that aren't selected. This result is in favor of the observation from Fig 1.

Table III and IV illustrate the results over three datasets for different size of buffer. In these two tables, for each dataset, *EaHTps* [4] performs variously for different size of buffer. Over the dataset *RTG12*, *EaHTps* performs best when the size of buffer is set as 2000 for subset accuracy measure or 1000 for exact match measure and over the dataset *RTG15*, *EaHTps* performs best when the size of buffer is set as 5000. Over the dataset *RTG8*, *EaHTps* performs best when the size of buffer is set as 500. So the larger size of buffer doesn't definitely lead to better performance.

In a word, some frequent label combinations aren't stored for training in the buffer, which influences the predictive performance of *EaHTps*. However, it can't be efficiently improved by changing the size of buffer.

TABLE I. STATISTICS OF REAL-WORLD DATASETS. $|L|$ DENOTES THE SIZE OF LABEL SET; $|X|$ DENOTES THE NUMBER OF ATTRIBUTES; $|D|$ DENOTES THE NUMBER OF SAMPLES; AVG$|Y|$ DENOTES THE AVERAGE NUMBER OF LABELS PER SAMPLE

Dataset	Properties			
	$\|L\|$	$\|X\|$	$\|D\|$	$Avg\|Y\|$
MediaMill	101	120	43907	4.4
TMC2007	22	500	28596	2.2
20NG	20	1001	19300	1.1
Slashdot	22	1079	3782	1.2
Enron	53	1001	1702	3.38

TABLE II. THE NUMBER OF FREQUENT LABEL COMBINATIONS OF NOT BEING SELECTED ON RTG

$\|L\|$	The Size of Buffer				
	200	*500*	*1000*	*2000*	*5000*
8	10	3	5	2	1
12	38	46	57	40	29
15	50	66	103	76	46

TABLE III. SUBSET ACCURACY ON RTG

$\|L\|$	The Size of Buffer				
	200	*500*	*1000*	*2000*	*5000*
8	0.552 ±0.073	0.560 ±0.074	0.558 ±0.074	0.559 ±0.072	0.557 ±0.073
12	0.336 ±0.080	0.346 ±0.085	0.347 ±0.086	0.348 ±0.090	0.330 ±0.079
15	0.203 ±0.053	0.227 ±0.064	0.232 ±0.058	0.242 ±0.071	0.243 ±0.067

TABLE IV. EXACT MATCH ON RTG

$\|L\|$	The Size of Buffer				
	200	*500*	*1000*	*2000*	*5000*
8	0.319 ±0.058	0.333 ±0.066	0.327 ±0.065	0.328 ±0.070	0.324 ±0.067
12	0.165 ±0.054	0.171 ±0.056	0.178 ±0.058	0.177 ±0.060	0.164 ±0.052
15	0.071 ±0.032	0.093 ±0.04	0.091 ±0.039	0.094 ±0.043	0.094 ±0.041

IV. CLASS INCREMENTAL LEARNING

In order to address the above issue, the strategy of class incremental learning is introduced. The learning process consists of two stages. At the first stage, a number of samples with frequent label combinations are collected to initialize the learning model by *PS* method. The set of the frequent label combinations is also saved for the next stage.

At the second stage, for each subsequent sample, if its label combination isn't in the set of frequent label combinations, the label combination will be saved and its occurring number is also updated in real time. These saved values are reset at the interval of the same number of samples compared with the first stage. Once the occurring number of one label combination is larger than that of any existing frequent label combination, then the corresponding samples will be used to update the learning model by the class incremental learning strategy [14]. The label combination is also used to update the set of frequent label combinations of the learning model. Otherwise, if the label combination of the newly arrived sample is in the set of frequent label combinations, it will be used to update the learning model by the instance incremental learning strategy [14].

The implementation of the proposed algorithms are showed in the Fig.2 and Fig.3. Table V summarizes these notations used in the proposed algorithms.

In Algorithm 1, at first, b samples are collected to initialize the learning model by *PS* method (line 3 - line 13). In the process of learning, the function PruneLC is called to prune the infrequent label combinations by the pre-set parameter p

(line 7). The *min_N* saves the minimum occurring number in *N* (line 8) and *C* saves the frequent label combinations (line 9).

TABLE V. NOTATIONS

Notation	Description
(x_i, Y_i)	The i^{th} sample in the data stream
$Y_i = \{y_1, ..., y_{\|L\|}\}$	The label combination of a sample
b	The size of buffer
$S = \{(x_1, Y_1), ..., (x_b, Y_b)\}$	The initial set of samples
p	A pre-set parameter to prune the label combinations
$LC = \{lc_1, ... , lc_i\}$	lc_i is the i^{th} label combination
$N = \{ n_1, ... , n_i \}$	n_i is the occurring number of lc_i
C	The set of frequent label combinations
min_N	The minimum occurring number of all frequent label combinations

Algorithm 1:TrainModel(x_i, Y_i)

Input: (x_i, Y_i), p
Output: The learning model

```
1   counter1 = 0
2   counter2 = 0
3   if  counter1 < b  then
4       S ← S ∪ {(x_i, Y_i)}
5       counter1 = counter1 + 1
6       if  counter1 >= b  then
7           (LC, N) ← PruneLC(S, p)
8           min_N ← Min(n_i ∈ N)
9           C ← LC
10          Model ← InitializeModel(S, C)
11          LC ← Φ && N ← Φ
12      end if
13  else
14      lc = Φ
15      if  counter2 < b  then
16          c ← (x_i, Y_i)
17          if  c ∉ C  then
18              lc ← UpdateLC(c, min_N, LC, N )
19          end if
20          if  lc != Φ  then
21              ClassIncremental(lc, (x_i, Y_i ), Model)
22              C ← C ∪ lc
23          else
24              InstanceIncremental((x_i, Y_i), Model)
25          end if
26          counter2 = counter2 + 1
27      end if
28      if  counter2 >= b  then
29          LC ← Φ && N ← Φ
30      end if
31  end if
```

Fig. 2. The outline of the proposed algorithm

Algorithm 2: UpdateLC(c, *min_N, LC, N*)

Input: c, *min_N, LC, N*
Output: The frequent label combination lc_i

```
1   lc_i ← c
2   if  lc_i ∉ LC  then
3       n_i ← 1
4       LC ← LC ∪ lc_i  && N ← N ∪ n_i
5   else
6       n_i ← n_i+1
7   end if
8   if  n_i > min_N  then
9       return lc_i
10  else
11      return Φ
12  end if
```

Fig. 3. The process of recognizing the frequent label combinations

After the learning model is initialized, for each newly arrived sample (x_i, Y_i), if its label combination is not in C (line 17), the function UpdateLC will be called to estimate whether the label combination is frequent or not according to its occurring number. In Algorithm 2, when a new label combination arrives, LC and N are updated (line 2 – line 7). If the occurring number of a label combination is larger than *min_N*, it will be considered to be frequent (line 9).

If the label combination is considered to be frequent(line 20) by Algorithm 2, (x_i, Y_i) will be used to update the learning model by class incremental strategy (line 21). In order to meet the need of class incremental learning, we alter Naïve Bayes by adding the prior probability of the new class and the new conditional probability for each attribute and updating the prior probabilities of the existed classes and the previous conditional probability for each attribute. On the other hand, this new frequent label combination is used to update C (line 22).

Otherwise, (x_i, Y_i) will be used to update the learning model by the strategy of instance incremental (line 24). In the end, the values of N and LC are reset at the interval of b samples.

V. EXPERIMENTS AND RESULTS

In this section, the proposed algorithm is evaluated on both real-world and synthetic multi-label datasets. All algorithms are implemented in Java with help of MOA [15] software package.

A. Data Collection

a) Real-world datasets: Table I provides the particular statistics of five real-world datasets, which can be found in [4], [6], [10], [11], and [16]. *MediaMill* contains video annotation data annotated with 101 labels. *TMC2007* consists of the aviation safety reports labeled with 22 types. *20NG* originates from 20 newsgroups around 20,000 articles data. *Slashdot* contains the article blurbs which have 22 different subject categories. *Enron* is the email data with 53 categories.

b) Synthetic datasets: The multi-label synthetic datasets are generated by the multi-label stream generation framework

[17], which needs a base generator. There're two types of base generator: *Random Tree Generator* (*RTG*) [12] and *Radial Basis Function* (*RBF*) [11].

RTG: Random tree that has five values for each nominal attribute and two values for each label, and its depth is five. *RTG8* denotes that the random tree has eight labels and ten nominal attributes. *RTG12* has twelve labels and sixteen attributes. *RTG15* has fifteen labels and twenty attributes.

RBF refers to fifty centers and two values for each label. *RBF8*, *RBF12* and *RBF15* respectively have the same number of labels and attributes like *RTG8*, *RTG10* and *RTG15*.

For each multi-label data stream generated by *RTG* or *RBF*, the parameter z (approximate average number of labels per sample) is experimented with: z = 1.5 and $|L|$ = 8, z = 1.8 and $|L|$ = 12, and z = 2.0 and $|L|$ = 15.

To simulate the concept drift, $|z|$ and *ld* (label dependencies) are altered simultaneously or alone [4], [11]. The *g* represents the base generator in the following.

Drift-RTG8 (*g* = *RTG8*) contains the change of $|z|$: 1.8, 3.0, 2.5, and 4.5.

Drift-RTG15 (*g* = *RTG15*) contains the change of $|z|$ and *ld*: 1.8 and 0%, 1.8 and 10%, 3.0 and 0%, and 3.0 and 20%.

Drift-RBF8 (*g* = *RBF8*) contains the change of $|z|$ and *ld*: 1.5 and 0%, 1.5 and 10%, 3.5 and 0%, and 3.5 and 20%.

Drift-RBF15 (*g* = *RBF15*) contains the change of $|z|$: 1.5, 3.5, 2.0, and 4.5.

For the evolving multi-label data streams with T samples, the three concept drifts take place at the position of T/4, 2T/4, and 3T/4. The value of T is set to 1000000.

B. Evaluation Measures

For multi-label classification problem, new measures are needed since the simple accuracy metric of single-label tends to be overly harsh. The adopted measures are common and used in [4], [5], [6], and [11]. Assume that a multi-label data stream D has $|D|$ (a finite number) samples (x_i, Y_i), where $Y_i = (y_1, ..., y_{|L|})$ represents the actual label set of the sample x_i, $\hat{Y}_i = (\hat{y}_1, ..., \hat{y}_{|L|})$ represents the predicted label set of the sample x_i and y_i^j denotes the j^{th} label of the sample x_i.

Exact Match: A sample is thought to be classified correctly only if its predicated label set is fully equal to the actual label set. I is an indicator function that its value equals to 1, if the condition is correct, otherwise 0.

$$ExactMatch = \frac{1}{|D|}\sum_{i=1}^{|D|} I(Y_i = \hat{Y}_i) \tag{6}$$

Subset Accuracy [4], [5], [6], [11]: It gives a score ([0, 1]) calculated from the actual and predicted label set for each sample.

$$SubsetAccuracy = \frac{1}{|D|}\sum_{i=1}^{|D|} \frac{\sum_{j=1}^{|L|} y_{(i)}^j \wedge \hat{y}_{(i)}^j}{\sum_{j=1}^{|L|} y_{(i)}^j \vee \hat{y}_{(i)}^j} \tag{7}$$

Hamming-accuracy [4], [5], [6]: The metric analyses the binary relevance of each label for all samples.

$$Hamming-accuracy = 1 - \frac{1}{|L||D|}\sum_{i=1}^{|D|}\sum_{j=1}^{|L|} I(y_{(i)}^j = \hat{y}_{(i)}^j) \tag{8}$$

F_1-macro [4], [11]: It's a macro-averaged F_1 over all labels and F_1 is the harmonic mean between Precision and Recall [9].

$$F_1 - macro = \frac{1}{|L|}\sum_{j=1}^{|L|} F_1[(y_{(1)}^j, ..., y_{(|D|)}^j), (\hat{y}_{(1)}^j, ..., \hat{y}_{(|D|)}^j)] \tag{9}$$

For evaluation methodology, we employ the most popular *prequential* strategy [18] where each example is used for testing before it's used for training.

C. Experiment Design

In order to test and verify the reasonability and availability of the proposed algorithm, we take two groups of experiments: one for verification of the analysis in Section III and the significance of the *EaHTps* method; the other for evaluation of the proposed algorithm.

a) The first group: In Section III, *EaHTps* [4] is run over the synthetic datasets to support the analysis of some frequent label combinations not being selected. In this section, the experimental results are included when *EaHTps* is compared with *MBR* [8] and *MWC* [10]. The parameter p is set as 1.

b) The second group: We take experiments to compare the proposed method with *EaHTps* on the synthetic and real-world multi-label datasets. The size of buffer is set as 1000 for the synthetic datasets, but the size of buffer is set as 200 for the real-world datasets because the number of samples is small for them.

For convenience, the proposed approach is called *EaHTcl,* because it selects more new frequent label combinations to update the learning model by class incremental learning strategy. p = 1 is set the same in these two algorithms.

D. Experimental Results and Discussion

The final experimental results on the synthetic datasets are averages of 100 runs, which are carried out over one million *prequential* evaluation.

TABLE VI. SUBSET ACCURACY ON REAL-WORLD DATASETS

Methods	Dataset				
	MediaMill	***TMC2007***	***20NG***	***Slashdot***	***Enron***
MBR	0.147	0.151	0.046	0.054	0.049
MWC	0.065	0.133	0.162	0.116	0.090
EaHTps	0.284	0.511	0.219	0.087	0.095

TABLE VII. EXACT MATCH ON REAL-WORLD DATASETS

Methods	Dataset				
	MediaMill	***TMC2007***	***20NG***	***Slashdot***	***Enron***
MBR	0.002	0.032	0.038	0.018	0.004
MWC	0.006	0.049	0.057	0.072	0.068
EaHTps	0.010	0.229	0.070	0.055	0.073

TABLE VIII. MEASURES ON SYNTHETIC DATASETS

Dataset	Subset Accuracy		Exact Match		Hamming-accuracy		F_1-macro	
	EaHTps	***EaHTcl***	***EaHTps***	***EaHTcl***	***EaHTps***	***EaHTcl***	***EaHTps***	***EaHTcl***
Drift-RTG8	0.558±0.034	0.602±0.040	0.175±0.033	0.221±0.054	0.169±0.022	0.189±0.023	0.561±0.043	0.610±0.044
Drift-RTG15	0.361±0.033	0.395±0.044	0.069±0.019	0.092±0.029	0.170±0.017	0.188±0.019	0.368±0.050	0.422±0.052
Drift-RBF8	0.473±0.036	0.519±0.034	0.129±0.023	0.169±0.040	0.218±0.015	0.228±0.024	0.486±0.043	0.535±0.040
Drift-RBF15	0.272±0.039	0.296±0.037	0.028±0.009	0.036±0.014	0.238±0.018	0.248±0.024	0.336±0.056	0.383±0.043

TABLE IX. MEASURES ON REAL-WORLD DATASETS

Dataset	Subset Accuracy		Exact Match		Hamming-accuracy		F_1-macro	
	EaHTps	***EaHTcl***	***EaHTps***	***EaHTcl***	***EaHTps***	***EaHTcl***	***EaHTps***	***EaHTcl***
MediaMill	0.284	0.301	0.010	0.013	0.051	0.058	0.035	0.043
TMC2007	0.511	0.545	0.229	0.254	0.067	0.072	0.248	0.274
20NG	0.219	0.244	0.070	0.109	0.099	0.104	0.217	0.269
Slashdot	0.087	0.096	0.055	0.059	0.098	0.095	0.061	0.070
Enron	0.095	0.103	0.073	0.079	0.090	0.091	0.007	0.021

Table VI and VII display the comparative results over the real-world datasets when *EaHTps* is compared with other baseline methods.

From Table VI and VII, *EaHTps* makes the overall best performance on the total real-world datasets except *Slashdot*. In addition, *EaHTps* has evident advantage over other baseline methods on the *MediaMill* and *TMC2007* datasets. These all demonstrate the significance of *EaHTps*.

Table VIII and IX show the experimental results of comparing the proposed algorithm with *EaHTps*. Specifically, Table VIII shows the results over the synthetic datasets, while Table IX shows the results over the real-world datasets

From Table VIII and IX, it can be seen that *EaHTcl* performs well over *EaHTps* under almost all measures on synthetic and real-world datasets except the case of hamming-accuracy on *Slashdot*.

The reason why the proposed algorithm outperforms *EaHTps* lies in that: When a new frequent label combination is recognized, the learning model will be updated by the sample with this new frequent label combination, and then the updated learning model can efficiently make multi-label classification before concept drift takes place. However, *EaHTps* uses *PS* method to alter the new frequent label combination and keeps the set of frequent label combinations the same before concept drift takes place.

VI. CONCLUSION

In this paper we have analyzed why some frequent label combinations aren't selected for training the multi-label classifier and proposed an algorithm which dynamically recognizes some new frequent label combinations and updates the trained classifier by class incremental learning strategy. The experimental results show that the proposed algorithm is promising, which indicates that the proposed algorithm is effective in classifying multi-label evolving data streams.

Future work involve 1) try a new method to detect concept drift from the aspect of label dependencies that doesn't occur in single-label data streams scenarios 2) consider how to handle the class imbalance brought about by new class to efficiently deal with multi-label evolving data streams.

ACKNOWLEDGMENT

This work was supported in part by Guangxi Key Laboratory of Trusted Software (KX201311), and the National Natural Science Foundation of China (61363029).

REFERENCES

[1] G. Widmer and M. Kubat. "Learning in the presence of concept drift and hidden contexts," Machine learning, vol. 23(1), pp. 69-10, 1996.

[2] A. Tsymbal. "The problem of concept drift: definitions and related work," Technical Report TCD-CS. Trinity College Dublin, 2004.

[3] J. Read, B. Pfahringer, and G. Holmes. "Multi-label classification using ensembles of pruned sets," IEEE International Conference on Data Mining, pp. 995-1000, Pisa, Italy, 2008.

[4] J. Read, A. Bifet, G. Holmes, and B. Pfahringer. "Scalable and efficient multi-label classification for evolving data streams," Machine learning, vol. 88(1-2), pp. 243-272, 2012.

[5] G. Tsoumakas and I. Katakis. "Multi-label classification: An overview," International Journal of Data Warehousing and Mining, vol. 3, pp. 1-13, 2007.

[6] G. Tsoumakas, I. Katakis and I.Vlahavas. "Mining multi-label data," Data mining and knowledge discovery handbook, O.Maimon and L. Rokach, Eds. Springer, 2010, pp. 667- 685.

[7] W. Qu, Y. Zhang, J. P. Zhu and Y. Wang. "Mining Multi-label Concept-Drifting Streams Using Ensemble Classifiers," IEEE Sixth International Conference Fuzzy Systems and Knowledge Discovery, vol. 5, pp. 275-279, Tianjin, China, 2009.

[8] W. Qu, Y. Zhang, J. P. Zhu and Q. Qiu. “Mining multi-label concept-drifting data streams using dynamic classifier ensemble,” Proceedings of the 1st Asian Conference on Machine Learning, vol. 5828, pp. 308-321, Nanjing, China, 2009.

[9] S. Godbole and S. Sarawagi. “Discriminative methods for multi-labeled classification,” Proceedings of the 8th Pacific-Asia Conference on Knowledge Dsicovery and Data Mining, pp. 22-30, 2004

[10] E. Xioufis, M. Spiliopoulou, G. Tsoumakas, and I. Vlahavas. “Dealing with concept drift and class imbalance in multi-label stream classification,” Proceedings of the Twenty-Second international joint conference on Artificial Intelligence, Barcelona, Spain , vol. 2, pp.1583-1588, 2011.

[11] J. Read, A. Bifet, G. Holmes, and E. Frank. “Efficient multi-label classification for evolving data streams,” Technical Report 04. University of Waikato, 2010.

[12] P. Domingos and G. Hulten. “Mining high-speed data streams,” Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 71-80, 2000.

[13] A. Bifet and R. Gavalda. “Learning from Time-Changing Data with Adaptive Windowing,” SIAM International Conference on Data Mining, vol. 7, pp. 443-448, 2007.

[14] Z. H. Zhou and Z. Q. Chen. “Hybrid decision tree,” Knowledge-based systems, vol. 15(8), pp. 515-528, 2002.

[15] A. Bifet, G. Holmes, R. Kirkby and B. Pfahringer. “Moa: Massive online analysis,” Journal of Machine Learning Research, vol. 99, pp. 1601-1604, 2010.

[16] J. Read. “Scalable multi-label classification,” Ph D Thesis, University of Waikato, 2010.

[17] J. Read, B. Pfahringer, and G. Holmes. “Generating synthetic multi-label data streams,” ECML/PKKD 2009 Workshop on Learning from Multi-label Data. pp. 69-84, 2009.

[18] J. Gama, R. Sebastião and P. P. Rodrigues. “Issues in evaluation of stream learning algorithms,” Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 329-338, 2009.