Mach Learn (2012) 88:243–272
DOI 10.1007/s10994-012-5279-6

Scalable and efficient multi-label classification for evolving data streams

Jesse Read · Albert Bifet · Geoff Holmes · Bernhard Pfahringer

Received: 19 October 2010 / Accepted: 20 January 2012 / Published online: 21 February 2012

Abstract Many challenging real world problems involve multi-label data streams. Efficient methods exist for multi-label classification in non-streaming scenarios. However, learning in evolving streaming scenarios is more challenging, as classifiers must be able to deal with huge numbers of examples and to adapt to change using limited time and memory while being ready to predict at any point.

This paper proposes a new experimental framework for learning and evaluating on multi-label data streams, and uses it to study the performance of various methods. From this study, we develop a multi-label Hoeffding tree with multi-label classifiers at the leaves. We show empirically that this method is well suited to this challenging task. Using our new framework, which allows us to generate realistic multi-label data streams with concept drift (as well as real data), we compare with a selection of baseline methods, as well as new learning methods from the literature, and show that our Hoeffding tree method achieves fast and more accurate performance.

Keywords Multi-label classification · Data streams classification

Editors: Grigorios Tsoumakas, Min-Ling Zhang, and Zhi-Hua Zhou.

J. Read currently at: Universidad Carlos III, Spain: E-mail: jesse@tsc.uc3m.es.
A. Bifet currently at: Yahoo! Research Barcelona, Catalonia: E-mail: abifet@yahoo-inc.com.

J. Read · A. Bifet (✉) · G. Holmes · B. Pfahringer
Computer Science Department, University of Waikato, Hamilton, New Zealand
e-mail: abifet@cs.waikato.ac.nz

J. Read
e-mail: jmr30@cs.waikato.ac.nz

G. Holmes
e-mail: geoff@cs.waikato.ac.nz

B. Pfahringer
e-mail: bernhard@cs.waikato.ac.nz

1 Introduction

The trend to larger data sources is clear, both in the real world and the academic literature. Nowadays, data is generated at an ever increasing rate from sensor applications, measurements in network monitoring and traffic management, log records or click-streams in web exploration, manufacturing processes, call-detail records, email, blogs, RSS feeds, social networks, and other sources. Real-time analysis of these data streams is becoming a key area of data mining research as the number of applications demanding such processing increases.

A *data stream* environment has different requirements from the traditional batch learning setting. The most significant are the following, as outlined in Bifet and Gavaldà (2009):

- process an example at a time, and inspect it only once (at most);
- be ready to predict at any point;
- data may be evolving over time; and
- expect an infinite stream, but process it under finite resources (time and memory).

In this work we specifically look at the task of *multi-label classification*; the generalisation of the traditional multi-class (single-label) task. In multi-label classification, instead of a single class-label, each example can be associated with *multiple* labels. Multi-label classification has seen considerable development in recent years, but so far most of this work has been carried out in the static context of *batch learning* where train-then-test scenarios or cross-fold validation evaluations are typical. Despite the fact that most of the example data stream applications above can be applied to multi-label contexts, very few authors have looked explicitly at the task in a data stream context. It is not yet known how well various multi-label approaches function in such a context, and there are no benchmarks to compare to.

The first truly instance-incremental learning approach in a multi-label data stream context considering concept drift was presented in Read et al. (2010). In this paper, we extend this work. We argue (and later provide empirical evidence to show) that instance-incremental learning approaches are the best approach for learning from data streams due to two main advantages: (1) they learn from each example as it arrives (and thus maintain a relevant learner), and (2) they do not discard information prematurely. Batch-incremental learning approaches often, therefore, fail to learn a complete concept within typical (or even generous) time and memory constraints, as they must phase out models built on earlier batches, as memory fills up. Moreover, they must wait to fill a batch with incoming examples before they can learn from them.

The main contributions of this paper are:

- a study of the application of existing multi-label methods to evolving data streams;
- the development of a framework for learning from and evaluating on multi-label data streams;
- modelling evolving synthetic multi-label data as part of this framework;
- a high-performance instance-incremental adaptive method which resulted from the aforementioned study; and
- an empirical demonstration that our method outperforms base-line methods as well as modern methods from the recent literature, thus creating a new set of benchmarks in predictive performance and time complexity.

In Sect. 2 we review related work, and in Sect. 5 we discuss how to deal with concept drift. In Sect. 3 we review our multi-label data stream framework. In Sect. 4 we detail

our method for generating synthetic multi-label streams. In Sect. 6 we discuss new methods for multi-label data stream classification as well as the adaptation of existing methods. Section 7 details a first comprehensive cross-method comparison. We summarise and draw conclusions in Sect. 8.

2 Related work

We will first look at existing multi-label approaches, and then discuss data stream specific applications.

A simple base-line method for multi-label classification is the *binary relevance* method (BR). BR transforms a multi-label problem into multiple binary problems, such that binary models can be employed to learn and predict the relevance of each label. BR has often been overlooked in the literature because it fails to take into account label correlations directly during the classification process (Godbole and Sarawagi 2004; Tsoumakas and Vlahavas 2007; Read et al. 2008), although there are several methods that overcome this limitation (e.g. Cheng and Hüllermeier 2009; Godbole and Sarawagi 2004; Read et al. 2009). BR can be applied directly to data streams by using incremental binary base models.

An alternative paradigm to BR is the *label combination* or *label powerset* method (LC). LC transforms a multi-label problem into a single-label (multi-class) problem by treating all label combinations as atomic labels, i.e. each labelset becomes a single class-label within a single-label problem. Thus, the set of single class-labels represents all distinct label subsets in the original multi-label representation. Disadvantages of LC include its worst-case computational complexity and tendency to over-fit the training data, although this problem has been largely overcome by newer methods (Tsoumakas and Vlahavas 2007; Read et al. 2008).

Another multi-label approach is *pairwise classification* (PW), where binary models are used for every possible *pair* of labels (Fürnkranz et al. 2008). PW performs well in some contexts, but the complexity in terms of models ($(L \times (L-1)/2)$ for L labels) means that this approach is usually intractable on larger problems.

Note that these are all *problem transformation* methods, wherein a multi-label problem is transformed into one or more single-label problems, and any off-the-shelf multi-class classifier (or binary classifier in the case of BR and PW) can be used. These methods are interesting generally due to their flexibility and general applicability. In fact, all methods in the literature use, mention or compare to at least one of them. Therefore, in this paper we will start by considering them as the basic methods to compare to, using them in combination with off-the-shelf incremental classifiers for data streams.

A number of single-label classifiers have also been adapted directly to multi-label classification, most notably decision trees (Clare and King 2001), k-nearest neighbor (Zhang and Zhou 2007) and boosting (Schapire and Singer 2000). We note that most *adaptation* methods have some connection with problem transformation.

Let us now look at existing data stream classifiers. A simple approach to the data stream problem is to train a batch-learning classifier on batches of new examples that replace classifiers built from earlier batches over time (Widmer and Kubat 1996; Qu et al. 2009). We call this approach *batch-incremental*. This procedure can be made to satisfy the above constraints, and can be used with any existing learning method. A first approach to multi-label data stream classification was reported in Qu et al. (2009). This method is batch-incremental, using K batches of S examples to train meta-BR (MBR); a well-known multi-label method described in Godbole and Sarawagi (2004), where the outputs of one BR are used in the

attribute space of a second BR (a form of stacking, employed to take into account label correlations). An additional weighting scheme supplements the standard MBR functionality. An MBR classifier is built on every new batch of S examples and replaces the oldest classifier. In this scheme the concept is represented by at most $K \times S$ examples. We implement and compare to MBR in our experiments, in this batch-incremental way, and show why *instance-incremental* methods are advantageous for learning in a data stream environment.

A *Hoeffding Tree* (Domingos and Hulten 2000) is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams by exploiting the fact that a small sample is often enough to choose an optimal splitting attribute. Using the *Hoeffding bound* one can show that the output of a Hoeffding Tree is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

In Law and Zaniolo (2005) an incremental and adaptive kNN algorithm is presented; incremental Naive Bayes has been used in Cesa-Bianchi et al. (2006); and perceptrons can also be used in incremental settings (Crammer and Singer 2003). Kirkby (2007) makes improvements to Hoeffding trees and introduces the MOA framework[1] for data stream generation and classification. Bifet et al. (2009) additionally considers *concept drift* in this framework, and presents several high-performing ensemble methods employing Hoeffding trees. Ensemble learning classifiers often have better accuracy and they are easier to scale and parallelize than single classifier methods. Oza and Russell (2001) developed online versions of bagging and boosting for data streams. They show how the process of sampling bootstrap replicates from training data can be simulated in a data stream context. So far Hoeffding trees and ensemble methods have proved very successful. In Bifet et al. (2009) new state-of-the-art bagging methods were presented: the most applicable, ADWIN Bagging using a change detector to decide when to discard under-performing ensemble members, is adapted here for use in a multi-label context.

Using Clare and King (2001)'s entropy adaption to C4.5, Hoeffding trees can be made multi-label, although to our knowledge this has not yet been investigated in the literature until now.

Recently, Kong and Yu (2011) presented an instance-incremental scheme for multi-label classification on data streams by employing an ensemble of random trees. This is a similar idea to the one we presented in Read et al. (2010), which also involved ensembles of random (Hoeffding) trees. A main difference is that, to deal with concept drift, they use simple 'fading functions' at the nodes of each tree to simply fade out the influence of past data, rather than using a drift-detection mechanism (as we do) to detect the point of change. Thus, their method assumes that relevance deteriorates directly with time. Furthermore, the results of this work do not show much impact: they only show some improvement over batch-incremental MLkNN (a well known but arguably outdated multi-label kNN-based classifier), which is an unusual choice for comparison since kNN methods are actually suitable to an instance-incremental context.

Also in recent work, Spyromitros-Xioufis et al. (2011) studied concept drift in multi-label data in a data stream context, and they introduce a sophisticated parameterised windowing mechanism for dealing with it, which they exemplify with an efficient instance-incremental multi-label kNN method. Like Kong and Yu (2011), this method does not explicitly try and detect drift, but rather just phases out older examples over time, which could be considered as a disadvantage when compared to the methods presented in this paper. However, unlike Kong and Yu (2011) this classification method realises the potential of kNN as an instance-incremental method, and we compare to it in our experiments.

[1] Massive Online Analysis (MOA): http://moa.cs.waikato.ac.nz/.

In a regression context, an arguably similar setting to multi-label learning is the multi-target problem, that is addressed in Appice and Džeroski (2007), and on data streams in Ikonomovska et al. (2011). However, we are not aware of any existing work dealing with this approach on multi-label data streams; the work done in this paper does not deal with any multi-label data sources, and thus is it not of specific relevance to this paper.

3 A framework for multi-label data stream mining

We use the following notation.

- $\mathcal{X} = \mathbb{R}^A$ is the input attribute space of A attributes
- $\mathbf{x} \in \mathcal{X}$ is an *instance*; $\mathbf{x} = [x_1, \ldots, x_A]$
- $\mathcal{L} = \{1, \ldots, L\}$ is a set of L possible labels
- $\mathbf{y} \in \{0, 1\}^L$ represents a *labelset*; $\mathbf{y} = [y_1, \ldots, y_L]$ where $y_j = 1 \iff$ the jth label is relevant (otherwise $y_j = 0$)
- $(\mathbf{x}_i, \mathbf{y}_i)$ is an *example*; instance and associated label relevances
- $y_j^{(i)}$ is the jth label relevance of the ith example; $x_a^{(i)}$ is the ath attribute of the ith instance
- $\sum \mathbf{y}$ is the number of relevant labels
- $z \equiv \phi_{\mathrm{LC}}(D) = \frac{1}{N} \sum_{i=1}^{N} \sum \mathbf{y}$ refers to *label cardinality*: the average number of labels for examples $1, \ldots, N$ in a dataset D

A classifier h makes a prediction $\hat{\mathbf{y}}_i = h(\mathbf{x}_i)$, which is evaluated against $\mathbf{y}_i$, then $(\mathbf{x}_i, \mathbf{y}_i)$ becomes a new training example.

Typically, a classifier in a real-world incremental context interleaves classification and training, where new examples are classified (i.e. labelled) automatically as they become available, and can be used for training as soon as their label assignments are confirmed or corrected. It may be a human or community of humans doing the checking (often termed a *folksonomy*), or checking might be automatic. For example, a robot learns multiple actions to complete a task, and each time it attempts the task, it can auto-evaluate its success.

Our complete framework supplies a way to generate different types of evolving multi-label data (Sect. 4), as well as a variety of methods for multi-label classification (Sect. 6). In this section we present an incremental multi-label evaluation component.

In multi-label classification, we must deal with the extra dimension of L binary relevance classifications per example, as opposed to one simple 0/1 classification accuracy measure.

The *Hamming-loss* metric is a *label-based* measure, where the binary relevance of each label is evaluated separately $L \times N$ binary evaluations in total for a window of N examples:

$$\text{Hamming-loss} = \frac{1}{LN} \sum_{i=1}^{N} \sum_{j=1}^{L} 1_{y_j^{(i)} = y_j^{(i)}}$$

A similar, increasingly popular measure is the *label-based F-measure* where the F_1 score (commonly used in the information-retrieval community) is calculated separately for each label, then averaged. For a window of N examples where $\mathtt{F}_1(\mathbf{v}, \hat{\mathbf{v}})$ gives the F_1 score for two binary vectors:

$$\text{F-measure} = \frac{1}{L} \sum_{j=1}^{L} \mathrm{F}_1([y_j^{(1)}, \ldots, y_j^{(N)}], [\hat{y}_j^{(1)}, \ldots, \hat{y}_j^{(N)}])$$

The 0/1-*loss* metric (also known as *exact match*), is an *example-based* measure that considers an example classified correctly if and only if *all* label relevances of that example are correct. For a window of N examples:

$$0/1\text{-loss} = \frac{1}{N}\sum_{i=1}^{N} 1_{\mathbf{y}_i = \hat{\mathbf{y}}_i}$$

These measures contrast well with each other. We also use the subset *accuracy* metric (defined in Godbole and Sarawagi 2004), which is a trade-off between strict label-based and strict example-based measures; a score ($\in [0, 1]$) is given for each example. For a window of N examples:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \frac{\sum_{j=1}^{L} y_j^{(i)} \wedge \hat{y}_j^{(i)}}{\sum_{j=1}^{L} y_j^{(i)} \vee \hat{y}_j^{(i)}}$$

As a single measure, accuracy works well: algorithms get a score from partially-correct labelset predictions, unlike 0/1-loss, but it is not purely label-based, as is Hamming-loss and the F-measure. All measures, as we have employed them, consider all labels $1, \ldots, L$, whether or not they appear in the current window. They must be predicted as zeros (true negatives) to be considered correct.

Given a test example, nearly all multi-label algorithms, including all ensemble methods, initially result in a confidence output vector (e.g. posterior probabilities) where the score for each label is $\in [0, 1]$, and require an extra process to bipartition the labelset into relevant and irrelevant labels to yield a multi-label prediction $\hat{\mathbf{y}} = \{0, 1\}^L$. This is typically a threshold function where $\hat{y}_j = 1$ iff the score for this jth label is greater than a threshold t. The importance of good threshold tuning is explained in Read (2010).

In Read et al. (2010) we adjusted the threshold instance-by-instance by a small amount depending on the predicted vs actual label cardinality. However, Spyromitros-Xioufis et al. (2011) have reported good results with batch-adjustments, which we have since adopted (except using the single-threshold method we reported in Read et al. 2011).

Initially we set $t = 0.5$ for the first window of w examples (let's call this window W_0). Subsequently, we use the threshold on window W_i which best approximated the true label cardinality of window W_{i-1}. For convenience, we use the same w as the evaluation window (see the following).

Finally, we also consider *log-loss* for evaluation, which analyses confidence outputs directly. This measure is ideal to make sure that results are not purely dependent on a particular threshold-calibration method. On a window of N examples:

$$\text{Log-Loss} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{j=1}^{L} \min(\text{log-loss}(\hat{y}_j^{(i)}, y_j^{(i)}), \ln(N))$$

where:

$$\text{log-loss}(\hat{y}, y) = -(\ln(\hat{y})y + \ln(1-\hat{y})(1-y))$$

All measures have been reviewed in Read (2010) and are commonly used in the literature (e.g. Tsoumakas and Katakis 2007; Tsoumakas and Vlahavas 2007; Read et al. 2009; Cheng et al. 2010). We use more of the methods for the later, more extensive and important experiments.

There are two basic evaluation procedures for data streams: holdout evaluation and prequential evaluation. Standard estimation procedures for small datasets, such as cross validation, do not apply due to the continuous and time-ordered nature of testing examples in streams.

In data stream mining, the most frequently used measure for evaluating predictive accuracy of a classifier is prequential accuracy. Gama et al. (2009) propose a forgetting mechanism for estimating holdout accuracy using prequential accuracy: either a sliding window of size w with the most recent observations can be used, or fading factors that weigh observations using a decay factor α. The output of the two mechanisms is very similar, as every window of size w_0 can be approximated by some decay factor α_0.

The MOA framework (Bifet et al. 2010) (software available at http://moa.cs.waikato.ac.nz/) provides an environment for implementing algorithms and running experiments in a data stream context. Synthetic data streams can be generated in addition to using real-world data sets, including concept drift, as it is presented in Bifet et al. (2009).

The MEKA framework (software available at http://meka.sourceforge.net) includes well-known methods for multi-label learning, which we updated for incremental classification. These methods can be called from MOA and, likewise, can use MOA classifiers.

For this work, we extend MOA to multi-label functionality for using real-world datasets, generating synthetic data, and developing new algorithms for classifying this data, although we currently still use MEKA-implementations for many algorithms. The software used for the experiments in this paper is available at http://meka.sourceforge.net/ml.html.

In the next section we detail our synthetic-data generator, and—in the section thereafter—our novel methods for classification.

4 Generating synthetic multi-label data

Despite the ubiquitous presence of multi-label data streams in the real world (such as the examples listed in the Introduction), assimilating and storing them on a large scale with both labels and time-order intact has so far proved largely impractical. Furthermore, in-depth domain knowledge may be necessary to determine and pinpoint changes to the concepts represented by the data, making drift-analysis difficult. This fact provides strong motivation for generating synthetic data.

There are several existing methods for generating synthetic multi-label data, for example Park and Fürnkranz (2008), Qu et al. (2009), but most are task-specific and not sufficient for general use. Of these, only Qu et al. (2009) introduces concept shift, by changing generation parameters, but is far too simple, involving only two labels, to be considered as a good approximation of real-world data. Overall, prior methods produce data which contains very few attributes and labels, as few as two to three, and are therefore not a generally good real-world approximation, even though they can be useful for analysing or highlighting particular characteristics of certain algorithms. Our framework contributes a general method which can simulate dependencies between labels as found in real data, as well as any number and type of attributes in the attribute space, and their relationships to the label space. Furthermore, it can realistically simulate how these dependencies and relationships evolve over time.

4.1 Generating multi-label data streams

It is acknowledged in the literature that dependencies exist between labels in multi-label data (Cheng et al. 2010; Read et al. 2008; Tsoumakas and Vlahavas 2007; Dembczyński et al.

Fig. 1 Pairwise unconditional dependence on the *IMDB* dataset, measured with mutual information: $I_{Y_j;Y_k} = \sum_{y_j \in \{0,1\}} \sum_{y_k \in \{0,1\}} p(y_j, y_k) \times \log \frac{p(y_j,y_k)}{p_1(y_j)p_2(y_k)}$. *Shades* represent information i.e. dependencies between labels; the *darker the shade* the more information

2010). We can look at these dependencies between labels from a probabilistic point of view, as did Cheng et al. (2010) for classification, where two types of dependency in multi-labelled data were identified: *unconditional dependence*, a global dependence between labels, and *conditional dependence* where dependencies are conditioned on a specific instance. The generator method we present is able to incorporate both these types of dependencies in the multi-label data it produces.

4.1.1 Unconditional dependence

Unconditional dependence refers to the idea that certain labels are likely or unlikely to occur together. For example, in the *IMDB* dataset (see Sect. 7.2), labels `Romance` and `Comedy` often occur together, whereas `Adult` and `Family` are mutually exclusive. Unconditional dependence can be measured with, for example, Pearson's correlation coefficient or mutual information. Figure 1 shows pairwise unconditional dependence in the *IMDB* dataset.

To synthesise unconditional dependence, the following parameters need to be specified:

- z the average number of labels per example (*label cardinality*); $z \in [0, L]$
- u the 'amount' of dependence among the labels; $u \in [0, 1]$

First we randomly generate a prior probability mass function (pmf) $\boldsymbol{\pi}$, where $\sum_{j=1}^{L} \pi_j = z$ and π_j gives the prior probability of the jth label; i.e. $\pi_j := P(Y_j = 1)$.

From $\boldsymbol{\pi}$ we can generate a conditional distribution: $\boldsymbol{\theta}$ over all label pairs. First $\theta_{jk} := P(Y_j = 1 | Y_k = 1)$ for all $L \leq j > k \leq 1$, where u of these values are set randomly to be $\in [\min(\pi_j, \pi_k), \max(0.0, (\pi_j + \pi_k - 1.0))]$ (label dependence) and the rest are set to $\theta_{jk} \approx \pi_j$ (independence). We then simply apply Bayes rule so that $\theta_{kj} = (\theta_{jk} \cdot \pi_k)/\pi_j$. Finally, for convenience, we set $\boldsymbol{\pi}$ to the diagonal of $\boldsymbol{\theta}$ ($\theta_{jj} \equiv \pi_j$) so matrix $\boldsymbol{\theta}$ is our only parameter. We model label dependence as the joint distribution, calculated as follows, noting that we can get any $P(Y_j = y_j | Y_k = y_k)$ with $\boldsymbol{\theta}$ using probability laws:

$$\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{y}) = p(y_1) \prod_{j=2}^{L} p(y_j | y_{j-1}) \tag{1}$$

Real-world multi-label data varies considerably in the number (L) and distribution ($\boldsymbol{\pi}$) of labels. After analysing the distribution of $\mathbf{y}$s in real-world data, we find that generating $\boldsymbol{\pi}$ from uniform random numbers, before normalising it by z, provides a realistic skew; and setting $u = zL/((L(L-1))/2)$ gives a realistic number of significant label dependencies. Figure 3 shows evidence for this. We emphasise that we do not try to mimic any particular

real-world dataset, and define 'realistic' as being comfortably within the range of real-world data that we have looked at. In this paper we do not consider extreme cases, but $\boldsymbol{\pi}$ and $\boldsymbol{\theta}$ can easily be tweaked differently if this is desired.

Algorithm 1 provides pseudo code for the generation of labelsets (generating a $\mathbf{y}$ with $\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{y})$—see (1)). Also see our source code for our generator, available at http://moa.waikato.ac.nz/details/classification/multi-label/.

Algorithm 1 Generating a labelset vector $\mathbf{y}$ (i.e. sampling from $\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{y})$ from (1)); where $i \sim \mathbf{p}$ returns i with probability p_i, and 0 with probability $1.0 - \sum \mathbf{p}$ (indicating that no new label is relevant).

- $\mathbf{y} \leftarrow [0_1, \ldots, 0_L]$ // create an empty labelset vector
- **while**(TRUE):
  1. $\mathbf{p} = [0_1, \ldots, 0_L]$
  2. **for** $j \in \{1, \ldots, L | y_j = 0\}$:
     (a) $\mathbf{y}' \leftarrow \mathbf{y}$
     (b) $y'_j \leftarrow 1$
     (c) $p_j \leftarrow \mathbf{p}_{\boldsymbol{\theta}}(\mathbf{y}')$
  3. $i \sim \mathbf{p}$
  4. **break if** $i = 0$
  5. $y_i \leftarrow 1$
- **return y**

4.1.2 *Conditional dependence*

Conditional dependence reflects how likely or unlikely labels are to occur together given the attribute values of a specific instance. For example in the *20 Newsgroups* dataset (see Sect. 7.2) the word attribute 'arms' is linked strongly to the presence of *both* labels `politics.guns` *and* `misc.religion` together, but not either label individually as strongly. In other words, these two labels are likely to occur together if an instance contains that particular word. As in single-label classification, an attribute value may identify only a single label, for example the word attribute 'linux' which pertains strongly to the label `Linux` in the *Slashdot* dataset.

Conditional dependence can be represented as $\mathbf{p}(\mathbf{y}|\mathbf{x})$, although since we have $\mathbf{p}(\mathbf{y})$ from (1), we can instead look at generating $\mathbf{x}$ from $\mathbf{y}$, since $\mathbf{p}(\mathbf{y}|\mathbf{x}) = \mathbf{p}(\mathbf{x}|\mathbf{y})\mathbf{p}(\mathbf{y})$. This is where we use a binary generator g to get $\mathbf{p}_{g;\zeta}(\mathbf{y}|\mathbf{x})$, where ζ is a map described as follows.

In ζ, all A attributes are mapped to the A most likely occurring label combinations (which we can get from sampling $\mathbf{p}(\mathbf{y})$). As long as $A > L$ (as is the case in all multi-label datasets we are aware of), this usually allows for attributes identifying single labels, as well as attributes identifying label combinations. The mapping ζ defines these relationships: $\zeta[a] \rightarrow \mathbf{y}_a$ where $\mathbf{y}_a$ is the ath most likely combination of labels. Algorithm 2 outlines the creation of an instance $\mathbf{x}$ given a label combination $\mathbf{y}$. Figure 2 illustrates the mapping function with an example.

Algorithm 2 Generating an instance $\mathbf{x}$ given a labelset $\mathbf{y}$ (i.e. sampling from $\mathbf{p}_{g,\zeta}(\mathbf{x}|\mathbf{y})$ with mapping ζ and binary generator g), where $\mathbf{y}_1 \subseteq \mathbf{y}_2$ if $\sum(\mathbf{y}_1 \wedge \mathbf{y}_2) \leq \sum \mathbf{y}_1$, and $x^{(a)}$ is the ath attribute of $\mathbf{x}$.

1. $\mathbf{x} \leftarrow [0_1, \ldots, 0_A]$ // create an empty instance vector
2. $\mathbf{x_{+1}} \leftarrow g.\text{gen}_+()$ // generate a positive binary example
3. $\mathbf{x_{-1}} \leftarrow g.\text{gen}_-()$ // generate a negative binary example
4. **for** $a = 1, \ldots, A$:
   - **if** $(\exists q : \zeta[a] \subseteq \mathbf{y}_q)$ **then** $x^{(a)} \leftarrow x^{(a)}_{+1}$
   - **else** $x^{(a)} \leftarrow x^{(a)}_{-1}$
5. **return x**

attribute	$a =$	$\mathcal{X}[1]$	$\mathcal{X}[2]$	$\mathcal{X}[3]$	$\mathcal{X}[4]$	$\mathcal{X}[5]$
mapping	$\zeta =$	{1}	{2}	{3}	{2, 3}	{4}

label(s)	instance (attribute space)					
	$\mathbf{x_{+1}} =$	0.9	0.8	0.2	0.9	−0.1
	$\mathbf{x_{-1}} =$	0.1	0.7	−0.1	0.8	0.2
$y = \{1, 3\}$		+	−	+	−	−
	$\mathbf{x_{ML}} =$	0.9	0.7	0.2	0.8	0.2

Fig. 2 Combining two binary instances ($\mathbf{x_{+1}}$ and $\mathbf{x_{-1}}$) to form a multi-label instance ($\mathbf{x}_{\text{ML}}$ associated with $\mathbf{y} = [1, 0, 1, 0, 0]$) using mapping ζ. According to ζ and $\mathbf{y}$, attribute values 1 and 3 are taken from $\mathbf{x_{+1}}$, and the rest from $\mathbf{x_{-1}}$ to form $\mathbf{x}_{\text{ML}}$

4.1.3 The synthetic generation process

We have described the essential components for generating realistic multi-label data (Algorithms 1 and 2). Algorithm 3 outlines the generation process for a multi-label example using these components.

Algorithm 3 Overall process for generating a multi-label example (i.e. sampling a $(\mathbf{x}, \mathbf{y})$) parameterised by $\boldsymbol{\theta}, g, \zeta$; using Algorithms 1 and 2.

1. $\mathbf{y} \sim \mathbf{p}_{\boldsymbol{\theta}}(\mathbf{y})$ // see Algorithm 1
2. $\mathbf{x} \sim \mathbf{p}_{g,\zeta}(\mathbf{y}|\mathbf{x})$ // see Algorithm 2
3. **return** $(\mathbf{x}, \mathbf{y})$

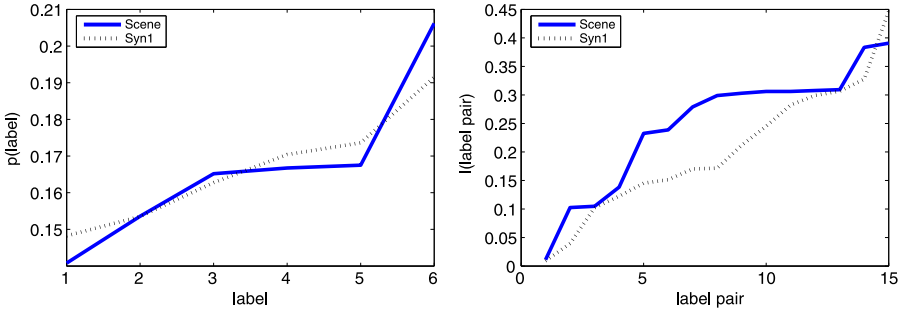
4.2 Analysis and discussion

Let us show that the synthetic data generated by our framework is realistic. Table 1 displays statistics of two synthetic multi-label datasets generated by our framework. *Syn1* was generated using a binary *Random Tree Generator* and *Syn2* by a binary *Random RBF Generator* (refer to Sect. 7.2 and Bifet et al. 2009 for details on these generators). The distribution of

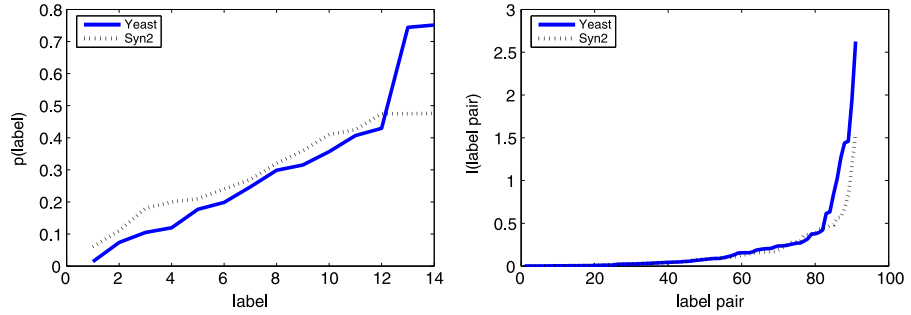
Table 1 A sample of real and synthetic datasets and their associated statistics

	N	L	A	$\phi_{\mathrm{LC}}(\mathcal{D})$	Type
Scene	2407	6	294n	1.07	media
Yeast	2417	14	103n	4.24	biology
Medical	978	45	1449b	1.25	medical text
Enron	1702	53	1001b	3.38	e-mail
Syn1	2400	6	30b	1.16	Tree-based synthetic
Syn2	2400	14	80n	4.33	RBF-based synthetic

n and b indicate numeric and binary attributes, $\phi_{\mathrm{LC}}(\mathcal{D})$ is the label cardinality



(a) Label priors (left) and pairwise information gain (right) for *Syn1* and *Scene* datasets.



(b) Label priors (left) and pairwise information gain (right) for *Syn2* and *Yeast* datasets.

Fig. 3 The label prior probabilities $p(Y_j)$ for $j = 1, \ldots, L$ (*top*) and pairwise information gain $I(Y_j, Y_k)$ for $1 \leq j < k \leq L$ (*bottom*) for real (*solid lines*) and synthetic datasets (*dashed lines*), ordered by value

prior and conditional-pair distributions for both datasets are displayed in Fig. 3. We see that the distributions of prior and conditional probabilities are similar in real and synthetic data.

Let us emphasise that in trying to generate realistic data in general we do not wish to mimic a particular dataset. This is why in Fig. 3 we ordered label frequencies and label-pair information gain by frequencies rather than by index. We have created *Syn1* and *Syn2* with similar dimensions to aid comparison, not as a form of mimicry. Our goal is: given a real and synthetic dataset, it should not be clear which is the synthetic data.

Table 2 displays the performance of the binary relevance `BR` and label combination `LC` problem transformation methods on both synthetic datasets, as well as a selection of real datasets (*Scene*, *Yeast*, *Enron*, and *Medical*; see e.g. Tsoumakas and Katakis 2007;

Table 2 The performance of the label-based BR and example-based LC problem transformation methods on the real and synthetic datasets from Table 1, under three evaluation measures

	Syn1		Syn2		Scene	
	BR	LC	BR	LC	BR	LC
Exact match	0.609	0.755	0.058	0.069	0.458	0.545
Accuracy	0.696	0.782	0.533	0.519	0.546	0.603
Hamming loss	0.102	0.075	0.203	0.207	0.156	0.139

	Yeast		Enron		Medical	
	BR	LC	BR	LC	BR	LC
Exact match	0.108	0.186	0.002	0.071	0.515	0.569
Accuracy	0.408	0.464	0.266	0.313	0.662	0.663
Hamming loss	0.283	0.243	0.078	0.068	0.016	0.016

Read 2010 for details), for comparison. These two methods were chosen for their contrast: BR models labels individually and does not explicitly model any dependencies between them, whereas LC models labelsets and thus the dependencies between labels. As the base classifier we use Naive Bayes (an incremental classifier). Predictive performance is measured under the evaluation measures described in 3.

The variety in the results of Table 2 is expected due to the different dimensions and sources of each dataset, but several trends are common throughout: accuracy and exact match is generally higher under LC; BR achieves either higher or comparable Hamming loss to LC; exact match is relatively poorer on data with irregular labelling and a skewed label distribution (*Enron*, *Syn2*); and the evaluation statistics all fit within a range—about 0.25 to 0.70 in terms of accuracy. Based on this varied analysis, the synthetic data is indistinguishable from the real-world datasets.

It is preferable to use real data whenever it is available, and as interest in mining multi-label data streams continues to rise, more public collections are likely to become available. However, our framework can provide a range of data by using any single-label generation scheme, and analysis indicates that this data is very similar to real-world data and therefore able to serve for general analysis and evaluation of incremental multi-label algorithms.

4.3 Adding concept drift

A general framework for modelling concept drift in streaming data was presented in Bifet et al. (2009). This framework uses a sigmoid function (Fig. 4) to define the probability that every new instance of a stream belongs to the new concept after the drift. If a and b are two data stream concepts, then $c = a \oplus_{t_0}^{W} b$ is the data stream joining the two streams a and b, where t_0 is the center of change and W is the number of instances needed to change from one concept to the other.

We can use the same approach in our multi-label stream generator, where a and b are simply multi-label streams. If these streams are generated with the same random seed and with the same parameters, they will be identical. We add parameter v: the portion of values changed (randomly to different degrees) in the dependency matrix $\boldsymbol{\theta}$ after generation. This way we can control subtle changes in label dependencies. For stronger changes in dependencies, we can use different random seeds and/or other parameters; both for the label generation, and the underlying single-label generator. This method can be illustrated with a

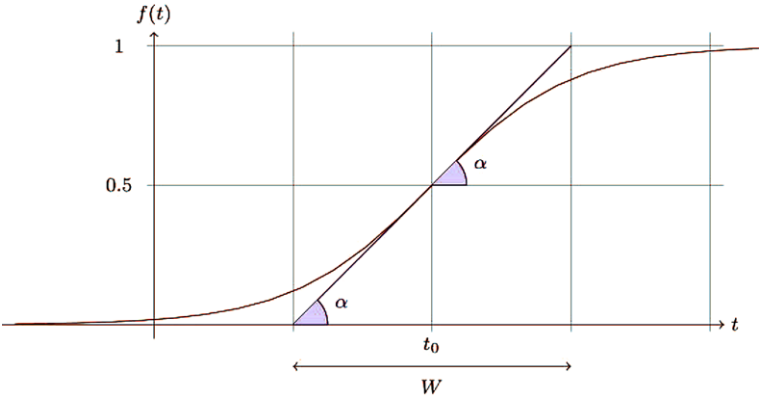


Fig. 4 A sigmoid function $f(t) = 1/(1 + e^{-s(t-t_0)})$, where $s = 4 \tan \alpha = 4/W$

simple example; consider stream a, of 3 labels, where $\pi = [0.2, 0.8, 0.4]$, and:

$$\boldsymbol{\theta}_a = \begin{bmatrix} 0.2 & 0.0 & 0.0 \\ 0.0 & 0.8 & 0.2 \\ 0.0 & 0.3 & 0.4 \end{bmatrix}$$

The multi-label data coming from stream a will have very strong dependencies in the form of Y_1 being mutually exclusive. Now, let us consider a second stream b where the concept is changed randomly (this would be parameter $v = \frac{1}{3}$ on stream b i.e. $1/3$ of dependencies are changed):

$$\boldsymbol{\theta}_b = \begin{bmatrix} 0.2 & 0.0 & 0.5 \\ 0.0 & 0.8 & 0.4 \\ 1.0 & 0.3 & 0.4 \end{bmatrix}$$

In stream b, the dependencies relating to Y_3 have changed: rather than being mutually exclusive, it now occurs whenever label Y_3 also occurs. However, the relationships between labels Y_1 and Y_2 have not changed (this pair is still mutually exclusive). Stream $c = a \oplus_{t_0}^{W} b$ will represent a shift from concept a to concept b.

This example is small and rather extreme, but it illustrates how we can make only different parts of a concept change over time and to a different degree, if at all. By introducing several changes to the concept (d, e, etc.) we model different changes to different labels at different points in time, and we can specify different types and magnitudes of change; affecting both unconditional and conditional dependencies (the latter naturally involves the attribute space). Because $\mathbf{p}(\mathbf{y})$ changes, the mapping ζ also changes; thus affecting conditional dependencies.

To also change the actual underlying concepts in the data, rather than just the labelling scheme, the parameters of the underlying single-label generator are made different between streams. For example, if we are using the Random Tree Generator (see Sect. 7.2), we may change the depth.

In other words, with this generator framework, we have control over the way drift may occur; allowing us to simulate real-world drift in different ways.

5 Dealing with concept drift

Usually the assumption that the data examples are generated from a static distribution does not hold for data streams. Classifiers must therefore adapt to changes in a stream of examples, by incorporating new information and phasing out information when it is no longer relevant. For this, it is important to detect when and where concept drift occurs—and possibly also the extent of it, and to do this in real time.

We have already mentioned "batch-incremental" methods, which adapt naturally to concept drift by constantly rebuilding classifiers from scratch on new batches of examples (and thus, discarding models built on old potentially-outdated batches). The problem is that, when no drift actually occurs, relevant information is discarded and, with it, possible predictive power. Instance-incremental methods need only to rebuild models when drift occurs—however this makes drift-detection crucial.

Detecting concept drift can, in most cases, be independent of the task of classification. A common strategy is to maintain windows of examples, and to signal a change where two windows differ substantially (for example in terms of *error-rate*). For multi-label learning, we use the "accuracy" measure which we describe and justify in Sect. 3. A modern example of this is `ADWIN` (Bifet and Gavaldà 2007) (`AD`aptive `WIN`dowing); which maintains an adaptive variably-sized window for detecting concept drift, under the time and memory constraints of the data stream setting. When `ADWIN` detects change, i.e. a new concept, models built on the previous concept can be discarded.

`ADWIN` is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, inherent to all algorithms dealing with random processes.

Also important for our purposes, `ADWIN` does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item.

More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: one, that change is reliably declared whenever the window shrinks; and two, that at any time the average over the existing window can be reliably taken as an estimate of the current average in the stream (barring a very small or very recent change that is still not statistically visible). A formal and quantitative statement of these two points (in the form of a theorem) appears in Bifet and Gavaldà (2007).

6 Multi-label classification methods

In this section, we show how we can use problem transformation methods to build classifiers for multi-label data streams, and we present new methods based on using incremental decision tree methods for data streams.

6.1 Using problem transformation methods in data streams

An important advantage of problem transformation is the ability to use any off-the-shelf single-label base classifier to suit requirements. In this section we discuss using incremental base classifiers to meet the requirements of data streams. The methods which we outline

here have been previously discussed and cited in many works, for example, Tsoumakas and Katakis (2007), Read (2010) both give detailed overviews. In this discussion we focus mainly on their possible applications to the data stream context.

6.1.1 Binary relevance (BR)-based methods in data stream settings

Binary Relevance (BR)-based methods are composed of binary classifiers; one for each label. It is straightforward to apply BR to a data stream context by using incremental binary base classifiers.

Some of the advantages of BR methods are low time complexity and the ability to run in parallel (in most cases), as well as a resistance to labelset overfitting (since this method learns on a per-label basis). These advantages are also particularly beneficial when learning from data streams.

BR is regarded as achieving poor predictive accuracy, although this can be overcome. For example, in Read et al. (2009) we introduced ensembles of BR (EBR), and also the concept of *classifier-chains* (CC), both of which improve considerably over BR in terms of predictive performance.

A problem for BR methods in data streams is that class-label imbalance may become exacerbated by large numbers of training examples. Countermeasures to this are possible, for example by using per-label thresholding methods or classifier weightings as in Ráez et al. (2004).

6.1.2 Label combination (LC)-based methods in data stream settings

The *label combination* or *label power-set* method (LC) transforms a multi-label problem into a single-label (multi-class) problem by treating all labelsets (i.e. label *combinations*) as atomic labels: each unique labelset in the training data is treated as a single label in a single-label multi-class problem.

Although LC is able to model label correlations in the training data and therefore often obtains good predictive performance, its worst-case computational complexity (involving 2^L classes in the transformed multi-class problem) is a major disadvantage.

A particular challenge for LC-based methods in a data stream context is that the label space expands over timedue to the emergence of new label combinations.

It is possible to adapt probabilistic models to account for the emergence of new labelset combinations over time, however probabilistic models may not necessarily be the best performing ones. We instead implement a general 'buffering' strategy, where label combinations are learned from an initial number of examples and these are considered sufficient to represent the distribution. During the buffering phase, another model can be employed to adhere to the 'ready to predict at any point' requirement.

Unfortunately, basic LC must discard any new training examples that have a labelset combination that is not one of the class labels. The pruned sets (PS) method from Read et al. (2008) is an LC method much better suited to this incremental learning context: it drastically reduces the number of class-labels in the transformation by pruning all examples with infrequent labelsets (usually a minority). It then additionally subsamples the infrequent labelsets for frequent ones so as to reintroduce examples into the data without reintroducing new labelset combinations. PS thus retains (and often improves upon) the predictive power of LC, while being up to an order of magnitude faster. Most importantly, its subsampling strategy means that far fewer labelsets are discarded than would be the case under LC. In practice, due to the typical sparsity of labels, hardly any (if any) labelsets are discarded

under PS because there almost always exists the possibility to subsample a combination with a single label.

We find that the first 1000 examples are an adequate representation of the labelset distribution of the initial concept. PS stores the first unique labelsets of these 1000 examples with their frequency of occurrence (not necessarily the examples themselves), then builds a PS model, and continues to model these labelsets incrementally until the end of the stream or until concept drift is detected (see Sect. 5), at which point PS is reinitialised and buffers again. Values like 500 and 5000 worked similarly as effectively. Higher values let PS learn a more precise labelset distribution, but leave fewer instances to train on with that distribution, and vice versa for lower values. The range 500–5000 appears to allow a good trade off—we leave more precise analysis for future work. To avoid unusual cases (such as the potential worst case of 2^{1000} class labels) we limit the number of unique combinations to the 30 most frequent. On realistic data, we found that this safe-guard parameter has little or no negative effect on predictive performance but can reduce running time significantly. In an *ensemble* (EPS) any negative effects of buffering are further mitigated by reducing overfitting on the original label space.

We emphasise that PS in this setup conforms to the requirements of a data stream context. Storing labelset *combinations* with their frequencies does not entail storing *examples*, and can be done very efficiently. We use an incremental majority-labelset classifier up until the 1000th example and thus are ready to predict at any point (using BR during buffering would be another possibility). Reinitialising a model when concept drift is detected is a standard technique; and is not a limitation of PS.

6.1.3 Pairwise (PW)-based methods in data stream settings

A binary *pairwise* classification approach (PW) can also be applied to multi-label classification as a problem transformation method, where a binary model is trained for each *pair* of labels, thus modelling a decision boundary between them. Although PW performs well in several domains and can be adapted naturally to the data stream setting by using incremental base classifiers, as in Fürnkranz et al. (2008), creating decision boundaries between labels, instead of modelling their co-occurrence raises doubts and, importantly, faces quadratic complexity in terms of the number of labels and, for this reason, is usually intractable for many data stream problems.

6.2 Multi-label Hoeffding trees (HT)

The Hoeffding tree is the state-of-the-art classifier for single-label data streams, and performs prediction by choosing the majority class at each leaf. Predictive accuracy can be increased by adding naive Bayes models at the leaves of the trees. Here, we extend the Hoeffding Tree to deal with multi-label data: a *Multi-label Hoeffding Tree*.

Hoeffding trees exploit the fact that a small sample can often suffice to choose a splitting attribute. This idea is supported by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the goodness of an attribute). More precisely, the Hoeffding bound states that with probability $1-\delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

A theoretically appealing feature of Hoeffding trees not shared by other incremental decision tree learners is that it has sound guarantees of performance (Domingos and Hulten 2000). Using the Hoeffding bound one can show that its output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

Information gain is a criterion used in leaf nodes to decide if it is worth splitting. Information gain for attribute A in a splitting node is the difference between the entropy of the training examples S at the node and the weighted sum of the entropy of the subsets S_v caused by partitioning on the values v of that attribute A.

$$\text{Info Gain}(S, A) = \text{entropy}(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \text{entropy}(S_v)$$

Hoeffding trees expect that each example belongs to just one class. Entropy is used in C4.5 decision trees and single-label Hoeffding trees for a set of examples S with L classes and probability $P(c)$ for each class c as:

$$\text{entropy}_{SL}(S) = -\sum_{i=1}^{L} p(c_i) \log(p(c_i))$$

Clare and King (2001) adapted C4.5 to multi-label data classification by modifying the entropy calculation:

$$\begin{aligned}
\text{entropy}_{\text{ML}}(S) &= -\sum_{i=1}^{L} (p(c_i) \log(p(c_i)) + (1 - p(c_i)) \log(1 - p(c_i))) \\
&= -\sum_{i=1}^{L} p(c_i) \log(p(c_i)) - \sum_{i=1}^{L} (1 - p(c_i)) \log(1 - p(c_i)) \\
&= \text{entropy}_{SL}(S) - \sum_{i=1}^{L} (1 - p(c_i)) \log(1 - p(c_i))
\end{aligned}$$

We use their strategy to construct a Hoeffding Tree. A majority-*labelset* classifier (the multi-label version of majority-class) is used as the default classifier on the leaves of a Multi-label Hoeffding Tree. However, we may use any multi-label classifier at the leaves. Of all the multi-label classifiers that scale well with the size of the data we prefer to use the Pruned Sets method (PS) (Read et al. 2008). Reasons for choosing this method over competing ones are presented in the next section.

6.3 Multilabel Hoeffding trees with PS at the leaves (HT_{PS})

Given the rich availability and strengths of existing methods for multi-label learning which can be applied to data streams (as we have shown in Sect. 6), it worth investigating their efficacy ahead of the invention of new incremental algorithms. In this section we detail taking the state of the art in single-label data stream learning (Hoeffding Trees) and combining it with a suitable and effective multi-label method (PS). We demonstrate the suitability of this selection further in Sect. 7.

In Sect. 6.1.2, we discussed how to use LC-based (e.g. PS) methods in data stream settings using a buffering strategy, and that PS is much more suitable than LC in this respect. In

the leaves of a Hoeffding Tree, however, PS may be reset without affecting the core model structure at all, thus creating an even more attractive solution: $\mathtt{HT}_{PS}$.

PS is employed at each leaf of a Hoeffding Tree as we described in Sect. 6.1.2. Each new leaf is initialised with a majority labelset classifier. After a number of examples (we allow for 1000) fall into the leaf, a PS classifier is established to model the combinations observed over these examples, and then proceeds to build this model incrementally with new instances. On smaller datasets, the majority labelset classifier may be in use in many leaves during much of the evaluation. However, we chose 1000 with large data streams in mind.

We emphasise that $\mathtt{HT}_{PS}$, like HT and PS (see Sect. 6.1.2), meets all the requirements of a data stream learner: it is instance-incremental, does not need to store examples (only a constant number of distinct labelsets are stored at each leaf), and is ready to predict at any point. As described in the following, we wrap it in a bagged ensemble with a change detector to allow adaption to concept drift in an evolving stream.

6.3.1 Ensemble methods

A popular way to achieve superior predictive performance, scalability and parallelization is to use ensemble learning methods, combining several models under a voting scheme to form a final prediction.

When used in an ensemble scheme, any incremental method can adapt to concept drift. We consider ADWIN Bagging (Bifet et al. 2009), which is the online bagging method of Oza and Russell (2001) with the addition of the ADWIN algorithm as a change detector.

When a change is detected, the worst performing classifier of the ensemble of classifiers is replaced with a new classifier. Algorithm 4 shows its pseudocode.

Algorithm 4 ADWIN Bagging for M models

1: Initialize base models h_m for all $m \in \{1, 2, \ldots, M\}$
2: **for all** training examples **do**
3: **for** $m = 1, 2, \ldots, M$ **do**
4: Set $w = Poisson(1)$
5: Update h_m with the current example with weight w
6: **if** ADWIN detects change in accuracy of one of the classifiers **then**
7: Replace classifier with lower accuracy with a new one
8: **anytime output:**
9: **return** hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

Using this strategy, all the methods we have discussed and presented can adapt to concept drift in evolving data streams, which may include the emergence of new label dependencies.

7 Experimental evaluation

We perform two evaluations: a first comparing several basic problem transformation methods in a data stream environment using Naive Bayes as a base classifier, and a second comparing newer and more competitive methods with our presented method, using the Hoeffding Tree as a base classifier for problem transformation methods.

The first evaluation shows that PS is a suitable method to apply on the leaves of the Hoeffding trees, and we use them in the ensemble methods of the second evaluation as our presented method.

7.1 Evaluation measures and methodology

We used evaluation measures and the prequential methodology discussed in Sect. 3, using a window $w = \frac{N}{20}$ (i.e. the full stream is divided into 20 windows). We display the average of each particular measure across the data windows, and *cumulative* running time (training plus classification time) in seconds.

The Nemenyi test (Demšar 2006) is used for computing significance: it is an appropriate test for comparing multiple algorithms over multiple datasets, being based on the average ranks of the algorithms across all datasets. We use the symbol $\succ$ to indicate statistically significant improvement according to this test. We use a p-value of 0.05. We also supply the algorithms' per-dataset rankings in parenthesis.

7.2 Datasets

Table 3 provides statistics of our data sources. We chose large multi-label real-world datasets. *TMC2007* contains instances of aviation safety reports labelled with problems being described by these reports. We use the version of this dataset specified in Tsoumakas and Vlahavas (2007). *IMDB* comes from the Internet Movie DataBase; data, where movie plot text summaries are labelled with relevant genres. *MediaMill* originates from the 2005 NIST TRECVID challenge dataset and contains video data annotated with various concepts. The *Ohsumed* collection is a subset consisting of peer-reviewed medical articles, labelled with disease categories. *Slashdot* consists of article blurbs with subject categories, mined from http://slashdot.org. *20NG* is the classic *20 newsgroups* collection with around 20,000 articles sourced from 20 newsgroups. We also include labelled emails from the *Enron* dataset; a small collection by data stream standards, but is time-ordered and a superficial analysis shows that it evolves over time. We treat these datasets as a stream by processing them in the order that they were collected. At least in the case of *Enron* and *20 newsgroups*, this represents time order. Read (2010) provides a summary and sources for all these datasets.

For the synthetic data we use two types of generation schemes:

1. *The Random Tree Generator (RTG)* generator proposed by Domingos and Hulten (2000) produces concepts that in theory should favour decision tree learners. It constructs a decision tree by choosing attributes at random to split, and assigning a random class label to each leaf. Once the tree is built, new examples are generated by assigning uniformly distributed random values to attributes which then determine the class label via the tree. We use a tree depth of 5.
2. *The Radial Basis Function (RBF)* generator was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. This generator effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Drift is introduced by moving the centroids with constant speed initialized by a drift parameter. We use two central points for this data.

All generation schemes used by our framework are initialised as binary generators. All non-generator specific parameters are inferred from Table 3. Three concept drifts of varying type, magnitude, and extent occur in *SynT-drift* and *SynG-drift*. For N generated examples, the drifts are centred over examples $1/N$, $2/N$, and $3/N$, extending over $N/1000$,

Table 3 A sample of seven real and four synthetic multi-label datasets and their associated statistics

	N	L	A	$\phi_{\mathrm{LC}}(D)$
TMC2007	28596	22	500b	2.2
MediaMill	43907	101	120n	4.4
20NG	19300	20	1001b	1.1
IMDB	120919	28	1001b	2.0
Slashdot	3782	22	1079b	1.2
Enron	1702	53	1001b	3.4
Ohsumed	13929	23	1002n	1.7
SynG($g = \mathtt{RBF}$)	1E5	25	80n	2.8
SynT($g = \mathtt{RTG}$)	1E6	8	30b	1.6
SynG-drift($g = \mathtt{RBF}$)	1E5	25	80n	$1.5 \rightarrow 3.5$
SynT-drift($g = \mathtt{RTG}$)	1E6	8	30b	$1.8 \rightarrow 3.0$

n indicates numeric attributes, and b binary, $\phi_{\mathrm{LC}}(\mathcal{D})$ is the label cardinality

Table 4 Multi-label data stream-capable methods listed with an abbreviation, associated parameters, and relevant citations and in-paper references

Key	Algorithm	Parameters	Citation / Reference
BR	Binary Relevance		Tsoumakas and Katakis (2007), Sect. 6.1
CC	Classifier Chains (BR-based)		Read et al. (2011), Sect. 6.1
PS	Pruned Sets (LC-based)	$p = 1, n = 1$	Read et al. (2008), Sect. 6.1
HT	Multi-label Hoeffding Trees		Clare and King (2001), Read et al. (2010), Sect. 6.2
E_aBR	Ensembles of BR	$m = 10$	Read et al. (2011), Sect. 6.3.1
E_aPS	Ensembles of PS	$m = 10$	Read et al. (2008), Sect. 6.3.1
$\mathrm{E}_a\mathrm{HT}_{PS}$	Ensembles of HT_{PS}	$m = 10$	Sects. 6.3, 6.3.1
MBR	Meta-BR	$I = 8$, $S = 1000$	Qu et al. (2009), Godbole and Sarawagi (2004), Sect. 2
MWC	Multiple Windows Classifier	$w = 1000$, $p = 400$, $n = 600$	Spyromitros-Xioufis et al. (2011), Sect. 2

$N/100$, and $N/10$ examples, respectively. In the first drift, only 10% of label dependencies change. In the second drift, the underlying concept changes and more labels are associated on average with each example (a higher label cardinality). In the third drift, 20% of label dependencies change. Such types and magnitudes of drift can be found in real world data. Table 3 provides the basic attributes of the data.

7.3 Methods

Table 4 lists the methods we use in our experimental evaluation. This table includes the key as used in the results tables, the corresponding algorithm, the parameters used, and relevant citations to the original method as well as references within this paper.

We use the subscript $_a$ to denote ADWIN monitoring for concept drift adaption. Therefore BR is an incremental binary classification scheme, and E_aBR is an ensemble of BR with an ADWIN monitor to detect concept drift. Likewise, $\mathrm{E}_a\mathrm{HT}_{PS}$ is a bagged ADWIN-monitored ensemble of our HT_{PS} method (multi-label Hoeffding trees with PS at the leaves).

Table 5 Basic incremental multi-label methods: accuracy

Dataset	BR	CC	HT	PS
20NG	0.276 (2)	0.224 (3)	0.050 (4)	0.588 (1)
Enron	0.144 (2)	0.142 (3)	0.134 (4)	0.241 (1)
IMDB	0.139 (4)	0.170 (2)	0.210 (1)	0.146 (3)
MediaMill	0.119 (3)	0.080 (4)	0.301 (1)	0.297 (2)
Ohsumed	0.297 (2)	0.292 (3)	0.125 (4)	0.372 (1)
Slashdot	0.054 (3)	0.054 (3)	0.145 (2)	0.200 (1)
SynG	0.516 (1)	0.502 (3)	0.054 (4)	0.509 (2)
SynT	0.178 (2)	0.076 (4)	0.411 (1)	0.174 (3)
TMC2007	0.415 (3)	0.446 (2)	0.171 (4)	0.562 (1)
avg. rank	2.44	3.00	2.78	1.67
avg. value	0.24	0.22	0.18	0.34

Nemenyi significance: PS≻CC

Note that in Qu et al. (2009) the MBR method involved additional strategies to fine tune performance, such as weights to deal with concept drift. This strategy involved nearest-neighbor searching which, as will be shown and explained in the following, would be too costly to add given the already relatively slow performance. The parameters we use for MBR are among those discussed in Qu et al. (2009). We found that other variations (such as $S = 500$) made negligible difference in relative performance.

For PS, we use parameters $p = 1$ and $n = 1$; values recommended in Read et al. (2008), and we use the 1000-combination buffering scheme described in Sect. 6.1.2. For MWC's kNN implementation, we set $k = 11$, and used the Jaccard coefficient for distance calculation, as recommended in Spyromitros-Xioufis et al. (2011).

In our experiments, we consider two base classifiers: Naive Bayes for the first set of experiments, and Hoeffding trees for the second set. For Hoeffding trees, we use the Hoeffding tree parameters suggested in Domingos and Hulten (2000). For MBR (in the second set of experiments) we use the WEKA (Hall et al. 2009) implementation of J48[2] as a suitable base-classifier for comparing decision tree classifiers (MBR was not presented with incremental classifiers).

7.4 Comparing problem transformation methods

In the first of two experimental settings we compared the following methods (all of which were described in Sects. 6.1 and 6.2): BR (the standard baseline), and BR-based CC, LC-based PS (similar results to LC but without the exponential complexity) and multilabel HT. The results, given in Tables 5, 6, 7, 8, help support the decision to create a Hoeffding tree classifier with PS at the leaves (introduced in Sect. 6.3).

PS performs best overall. Clearly 1000 examples are sufficient for learning the model in these datasets.

The multi-label Hoeffding trees (HT) generally perform poorly, except on the larger datasets (like *IMDB* and *SynT*), where this method suddenly performs very well. An analysis quickly reveals the cause: Hoeffding trees are rather conservative learners and need

[2]A Java implementation of C4.5 decision tree algorithm release 8.

Table 6 Basic incremental multi-label methods: exact match

Dataset	BR	CC	HT	PS
20NG	0.185 (2)	0.040 (4)	0.050 (3)	0.577 (1)
Enron	0.006 (4)	0.007 (3)	0.058 (2)	0.086 (1)
IMDB	0.031 (2)	0.014 (4)	0.108 (1)	0.027 (3)
MediaMill	0.008 (3)	0.000 (4)	0.050 (1)	0.017 (2)
Ohsumed	0.115 (2)	0.054 (4)	0.083 (3)	0.212 (1)
Slashdot	0.000 (3)	0.000 (3)	0.137 (1)	0.113 (2)
TMC2007	0.149 (2)	0.123 (3)	0.087 (4)	0.298 (1)
SynG	0.250 (3)	0.271 (2)	0.022 (4)	0.290 (1)
SynT	0.039 (4)	0.042 (3)	0.244 (1)	0.099 (2)
avg. rank	2.78	3.33	2.22	1.56
avg. value	0.09	0.06	0.09	0.19

Nemenyi significance: PS≻CC

Table 7 Basic incremental multi-label methods: Hamming loss (displayed as 1.0−loss)

Dataset	BR	CC	HT	PS
20NG	0.933 (2)	0.807 (4)	0.904 (3)	0.958 (1)
Enron	0.524 (3)	0.503 (4)	0.926 (2)	0.934 (1)
IMDB	0.884 (2)	0.834 (4)	0.918 (1)	0.875 (3)
MediaMill	0.897 (3)	0.634 (4)	0.958 (1)	0.947 (2)
Ohsumed	0.913 (2)	0.866 (4)	0.900 (3)	0.926 (1)
Slashdot	0.055 (3)	0.054 (4)	0.915 (1)	0.912 (2)
TMC2007	0.907 (2)	0.871 (4)	0.884 (3)	0.935 (1)
SynG	0.956 (2)	0.960 (1)	0.911 (4)	0.955 (3)
SynT	0.631 (4)	0.699 (2)	0.767 (1)	0.649 (3)
avg. rank	2.56	3.44	2.11	1.89
avg. value	0.74	0.69	0.90	0.90

Nemenyi significance: PS≻CC

a considerable number of examples to identify the best split point for a node. As predictive performance is averaged over evaluation windows, initial low performance can have an overall negative impact on figures.

As expected, Hoeffding trees excel on *SynT* which is generated based on tree models. It also performs relatively poorly compared to other methods on *SynG* which is generated from Gaussian models.

Generally, when HT can grow properly, the power of Hoeffding trees can be observed. For any serious application of data streams with a very large number of examples, Hoeffding tree methods are desirable over Naive Bayes, as also discovered by other researchers (e.g. Kirkby 2007).

The fact that we expect ever increasing performance from Hoeffding trees with more data does not mean that PS is unsuitable at the leaves. On the contrary, it tells us that on large datasets (and certain types of data), trees are a better base-classifier option for problem transformation methods. This was part of the reasoning behind our construction of $\mathtt{HT}_{PS}$ (Sect. 6.3), evaluated in the next section.

Unexpectedly, CC is worse than the baseline BR in several cases. This is almost certainly due to the change in base classifier: CC was introduced in Read et al. (2011) with Support

Table 8 Basic incremental multi-label methods: running time (s)

Dataset	BR	CC	HT	PS
20NG	449 (2)	476 (1)	5 (4)	233 (3)
Enron	129 (1)	111 (2)	10 (4)	11 (3)
IMDB	5315 (2)	2027 (3)	180 (4)	16243 (1)
MediaMill	1069 (1)	861 (2)	71 (4)	361 (3)
Ohsumed	438 (2)	457 (1)	15 (4)	409 (3)
Slashdot	116 (1)	107 (2)	2 (4)	35 (3)
TMC2007	58 (3)	82 (1)	4 (4)	69 (2)
SynG	244 (2)	196 (3)	9 (4)	481 (1)
SynT	41 (2)	42 (1)	18 (4)	39 (3)
avg. rank	1.78	1.78	4.00	2.44
avg. value	873.22	484.33	34.89	1986.78

Nemenyi significance: HT≻{BR, CC, PS}

Vector Machines (SVMs) as a base classifier. Unfortunately SVMs are not a realistic option for instance-incremental learning; and Naive Bayes is clearly a poor choice in terms of predictive performance. Also discussed in Read (2010) is that CC may tend to overfit and succumb to class imbalance (along with BR) with large numbers of training instances.

BR-based methods are clearly not suitable for learning from the *MediaMill* and *Slashdot* datasets. HT and PS perform much better in these cases.

There is a lot of variation in the results. Methods can differ in performance by 20 percentage points or more. This would be surprising in a more established area of machine learning, but in the relatively unexplored area of multi-label data streams, there are many unexplored or only partially explored variables, including: the multi-label aspect and the concept-drift detection aspect, thresholding, single-label learner selection, and algorithm parameters. Furthermore, on such large datasets, algorithm advantages and disadvantages on different kinds of data become more apparent.

7.5 Comparing $\texttt{EaHT}_{PS}$ based methods to state of the art

In this second evaluation we ran experiments comparing our new Hoeffding tree based method (with PS at the leaves: $\texttt{EaHT}_{PS}$, as introduced in Sect. 6.3) with baseline methods HTa and BRa ('upgraded' to use an ADWIN monitor for adapting to concept drift), competitive ensemble methods (which we have also adapted to use ADWIN) and MBR and MWC from the literature. Full results are given in Tables 9–13. For measures denoted with ↑, lower values are better. Plotting all measures over time for each dataset would be result in too many plots, but we provide Figs. 5 and 6 for interests' sake. Figure 7 provides details on how often our ADWIN methods detect change for the drifting synthetic datasets.

Our novel method $\texttt{EaHT}_{PS}$ performs overall best under most measures of predictive performance, with the exception of log-loss, and also under exact-match where HTa does better. Performance is strongest relative to other methods on the *20NG* and *OHSU* datasets.

MBR performs averagely overall, although does well on the synthetic datasets under accuracy, and the smaller real datasets (*ENRN*, *SDOT*) under log-loss. It is noteworthy that it does not often improve on the baseline BRa as reported in the paper in which it was introduced, which could initially be seen as contradictory—but it is not: BRa is using ADWIN to detect change. If there is no change, BRa continues learning, whereas MBR, as a batch-incremental approach, continuously phases out old information even when it belongs to the current concept. This result is our strongest case for using instance-incremental methods.

Table 9 Adaptive incremental multi-label methods: accuracy

Dataset	BRa	EaBR	EaHT$_{PS}$	EaPS	HTa	MBR	MWC
20NG	0.075 (4)	0.349 (3)	0.542 (1)	0.062 (7)	0.073 (6)	0.075 (5)	0.379 (2)
ENRN	0.313 (3)	0.318 (2)	0.157 (4)	0.127 (5)	0.127 (5)	0.109 (7)	0.388 (1)
IMDB	0.209 (2)	0.226 (1)	0.134 (6)	0.200 (3)	0.195 (4)	0.160 (5)	0.077 (7)
MILL	0.361 (1)	0.361 (2)	0.290 (4)	0.320 (3)	0.265 (5)	0.198 (6)	0.073 (7)
OHSU	0.172 (4)	0.293 (2)	0.358 (1)	0.119 (6)	0.119 (6)	0.132 (5)	0.236 (3)
SDOT	0.149 (4)	0.176 (3)	0.237 (2)	0.136 (6)	0.137 (5)	0.129 (7)	0.346 (1)
SYNG	0.124 (3)	0.418 (2)	0.467 (1)	0.066 (5)	0.051 (6)	0.099 (4)	0.025 (7)
SYNG-drift	0.251 (2)	0.272 (1)	0.223 (3)	0.168 (5)	0.178 (4)	0.153 (6)	0.094 (7)
SYNT	0.176 (4)	0.178 (3)	0.267 (1)	0.160 (7)	0.160 (6)	0.181 (2)	0.166 (5)
SYNT-drift	0.196 (3)	0.195 (4)	0.221 (1)	0.184 (5)	0.164 (6)	0.199 (2)	0.159 (7)
TMC7	0.292 (4)	0.419 (2)	0.526 (1)	0.165 (7)	0.170 (6)	0.229 (5)	0.337 (3)
avg. rank	3.09	2.27	2.27	5.36	5.36	4.91	4.55
avg. value	0.21	0.29	0.31	0.16	0.15	0.15	0.21

Nemenyi significance: EaBR $\succ$ {EaPS, HTa, MBR}; EaHT$_{PS}$ $\succ$ {EaPS, HTa, MBR}

Table 10 Adaptive incremental multi-label methods: exact match

Dataset	BRa	EaBR	EaHT$_{PS}$	EaPS	HTa	MBR	MWC
20NG	0.046 (6)	0.258 (3)	0.507 (1)	0.039 (7)	0.073 (4)	0.050 (5)	0.277 (2)
ENRN	0.022 (6)	0.022 (5)	0.052 (4)	0.055 (2)	0.055 (2)	0.004 (7)	0.063 (1)
IMDB	0.050 (4)	0.050 (3)	0.026 (6)	0.096 (2)	0.102 (1)	0.027 (5)	0.012 (7)
MILL	0.033 (3)	0.012 (6)	0.027 (4)	0.036 (2)	0.076 (1)	0.004 (7)	0.013 (5)
OHSU	0.039 (6)	0.097 (2)	0.172 (1)	0.079 (4)	0.079 (4)	0.030 (7)	0.081 (3)
SDOT	0.089 (6)	0.103 (5)	0.143 (2)	0.125 (4)	0.130 (3)	0.060 (7)	0.231 (1)
SYNG	0.017 (4)	0.168 (2)	0.210 (1)	0.006 (6)	0.021 (3)	0.008 (5)	0.000 (7)
SYNG-drift	0.055 (1)	0.053 (2)	0.035 (4)	0.020 (5)	0.050 (3)	0.014 (7)	0.016 (6)
SYNT	0.036 (6)	0.036 (5)	0.068 (3)	0.098 (2)	0.099 (1)	0.034 (7)	0.038 (4)
SYNT-drift	0.018 (5)	0.015 (6)	0.026 (3)	0.030 (2)	0.046 (1)	0.020 (4)	0.014 (7)
TMC7	0.035 (7)	0.135 (2)	0.266 (1)	0.079 (5)	0.091 (4)	0.049 (6)	0.101 (3)
avg. rank	4.91	3.73	2.73	3.73	2.45	6.09	4.18
avg. value	0.04	0.09	0.14	0.06	0.07	0.03	0.08

Nemenyi significance: {EaHT$_{PS}$, HTa} $\succ$ MBR

If fully implemented according to Qu et al. (2009) we would expect higher performance from MBR. However, such an implementation would not be feasible for large data, as we explained in 7.3. Already from these results we see MBR is significantly slower than most other methods.

MWC performs outstandingly on *ENRN* and *SDOT*, often doubling the performance of some of the other methods. This shows the effectiveness of the positive/negative-example balancing method used for the BR-type implementation of this paper. However, on the larger datasets, especially the very large synthetic datasets performance is not as strong (except

Table 11 Adaptive incremental multi-label methods: log loss↑

Dataset	BRa	EaBR	EaHT$_{PS}$	EaPS	HTa	MBR	MWC
20NG	3.7 (4)	2.7 (2)	2.9 (3)	8.7 (6)	12.2 (7)	4.0 (5)	2.6 (1)
ENRN	8.6 (3)	8.4 (2)	15.6 (5)	16.5 (6)	16.5 (6)	5.3 (1)	9.1 (4)
IMDB	5.8 (2)	5.6 (1)	13.1 (5)	18.4 (6)	19.0 (7)	6.3 (3)	7.1 (4)
MILL	10.0 (2)	9.6 (1)	19.8 (5)	22.2 (6)	23.8 (7)	13.2 (3)	17.9 (4)
OHSU	5.0 (3)	4.2 (1)	6.3 (5)	14.3 (6)	14.3 (6)	5.4 (4)	5.0 (2)
SDOT	3.8 (4)	3.7 (3)	4.5 (5)	9.2 (6)	9.4 (7)	2.8 (1)	3.1 (2)
SYNG	5.9 (2)	3.7 (1)	6.9 (3)	14.1 (5)	19.8 (7)	17.5 (6)	7.2 (4)
SYNG-drift	5.4 (2)	5.3 (1)	11.8 (5)	13.4 (6)	19.1 (7)	8.2(3)	10.5 (4)
SYNT	3.3 (2)	3.3 (1)	5.4 (5)	16.6 (6)	17.1 (7)	3.3 (3)	3.6 (4)
SYNT-drift	4.4 (2)	4.4 (3)	16.9 (5)	21.7 (6)	23.1 (7)	4.4 (1)	5.2 (4)
TMC7	5.2 (2)	4.1 (1)	6.2 (4)	17.1 (6)	17.5 (7)	7.8 (5)	5.3 (3)
avg. rank	2.55	1.55	4.55	5.91	6.82	3.18	3.27
avg. value	5.5	5.0	9.9	15.7	17.4	7.1	7.0

Nemenyi significance: EaBR $\succ$ {EaHT$_{PS}$, EaPS, HTa}; {BRa, MBR, MWC}$\succ$ {EaPS, HTa}

Table 12 Adaptive incremental multi-label methods: label-based F-measure

Dataset	BRa	EaBR	EaHT$_{PS}$	EaPS	HTa	MBR	MWC
20NG	0.054 (5)	0.393 (3)	0.481 (1)	0.016 (6)	0.012 (7)	0.100 (4)	0.428 (2)
ENRN	0.041 (3)	0.051 (2)	0.026 (5)	0.008 (6)	0.008 (6)	0.040 (4)	0.101 (1)
IMDB	0.060 (4)	0.063 (3)	0.087 (2)	0.022 (6)	0.021 (7)	0.044 (5)	0.089 (1)
MILL	0.050 (1)	0.038 (3)	0.041 (2)	0.035 (4)	0.034 (5)	0.028 (7)	0.032 (6)
OHSU	0.100 (4)	0.295 (1)	0.261 (2)	0.013 (6)	0.013 (6)	0.083 (5)	0.244 (3)
SDOT	0.056 (5)	0.091 (3)	0.174 (2)	0.012 (6)	0.011 (7)	0.079 (4)	0.241 (1)
SYNG	0.094 (3)	0.423 (1)	0.401 (2)	0.014 (6)	0.005 (7)	0.080 (4)	0.050 (5)
SYNG-drift	0.182 (3)	0.244 (1)	0.186 (2)	0.058 (6)	0.050 (7)	0.106 (5)	0.143 (4)
SYNT	0.176 (3)	0.161 (4)	0.333 (1)	0.066 (7)	0.075 (6)	0.132 (5)	0.244 (2)
SYNT-drift	0.186 (3)	0.160 (5)	0.247 (1)	0.147 (6)	0.103 (7)	0.163 (4)	0.242 (2)
TMC7	0.098 (5)	0.275 (3)	0.385 (1)	0.023 (7)	0.026 (6)	0.114 (4)	0.382 (2)
avg. rank	3.55	2.64	1.91	6.00	6.45	4.64	2.64
avg. value	0.10	0.20	0.24	0.04	0.03	0.09	0.20

Nemenyi significance: {BRa, EaBR}$\succ$HTa; {EaBR, MWC}$\succ$ {EaPS, HTa}; EaHT$_{PS}$ $\succ$ {EaPS, HTa, MBR}

under F-measure). In many cases, this could be seen as a kNN vs decision tree result. We have already mentioned that Hoeffding Trees grow slowly and are thus not ideal on smaller datasets. It is likely that results would differ to some extent with different parameter configurations. However, in an evaluation of this size, as we have done with Hoeffding Trees, MBR, and PS, we must simply use a single combination, as recommended in the literature. That our method performs well on default parameters (of which it has few) is an advantage. We note that MWC is one of the fastest methods, only slower than the two baseline methods.

Table 13 Adaptive incremental multi-label methods: running time↑ (s)

Dataset	BRa	EaBR	EaHT$_{PS}$	EaPS	HTa	MBR	MWC
20NG	36 (2)	270 (5)	1818 (6)	56 (3)	5 (1)	2872 (7)	168 (4)
ENRN	14 (4)	91 (6)	37 (5)	4 (1)	5 (2)	290 (7)	6 (3)
IMDB	545 (2)	2923 (4)	9874 (6)	8130 (5)	101 (1)	29058 (7)	1288 (3)
MILL	923 (3)	6128 (6)	1362 (5)	195 (2)	26 (1)	18126 (7)	1118 (4)
OHSU	43 (2)	277 (4)	1616 (6)	294 (5)	9 (1)	2644 (7)	106 (3)
SDOT	7 (2)	61 (5)	234 (6)	14 (4)	1 (1)	402 (7)	9 (3)
SYNG	90 (2)	525 (5)	419 (4)	295 (3)	7 (1)	2019 (7)	789 (6)
SYNG-drift	90 (2)	764 (4)	689 (3)	1102 (6)	10 (1)	2716 (7)	1084 (5)
SYNT	2 (2)	21 (4)	56 (6)	10 (3)	2 (1)	35 (5)	159 (7)
SYNT-drift	62 (3)	375 (4)	34 (2)	628 (5)	14 (1)	687 (6)	1869 (7)
TMC7	19 (2)	209 (4)	294 (6)	80 (3)	3 (1)	1132 (7)	229 (5)
avg. rank	2.36	4.64	5.00	3.64	1.09	6.73	4.55
avg. value	167	1059	1494	983	17	5453	620

Nemenyi significance: BRa$\succ${EaHT$_{PS}$, MBR}; EaPS$\succ${MBR, EaBR, EaHT$_{PS}$, EaPS, MBR, MWC}

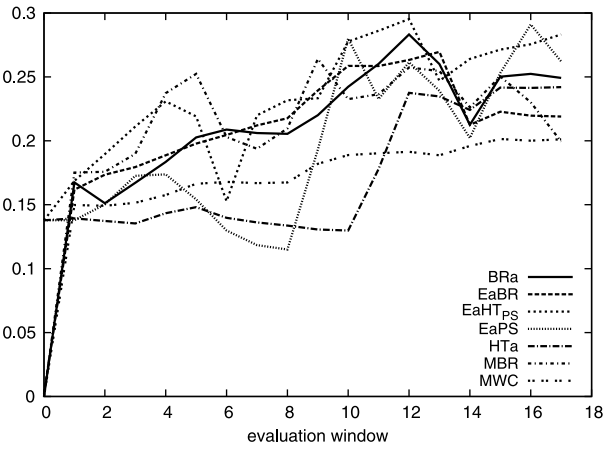


Fig. 5 Accuracy on the *SynT-drift* dataset over time

EaPS often does relatively poorly, particularly under F-measure evaluation and log-loss. The 1000-combination buffering strategy is still somewhat problematic, and has to be re-buffered each time there is a change. This is precisely why we developed EaHT$_{PS}$.

On the other hand, we are surprised at the high performance of BRa-methods; particularly EaBR—which is particularly strong overall. ADWIN detects drift, causing the binary classifiers to be reset, instead of succumbing to class-imbalance issues as they often do otherwise on large data (discussed in Read et al. 2011). Naturally, these results do not extend completely to the exact-match measure, since BR does not explicitly model labelsets in any way.

Occasionally the ADWIN($_a$) models do not perform as well as the standard methods (if we compare between the results of both sets of experiments). This can be expected, since ADWIN occasionally detects change when there is none; or when it is only temporary. The parameters for ADWIN in a multi-label setting, such as the datum it monitors for change,

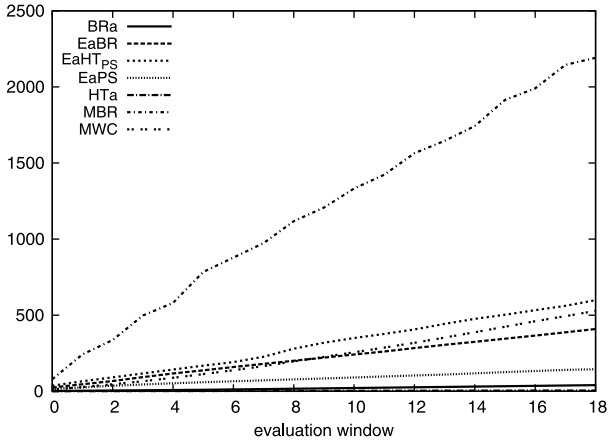


Fig. 6 Cumulative running time (s) on the *TMC2007* dataset over time

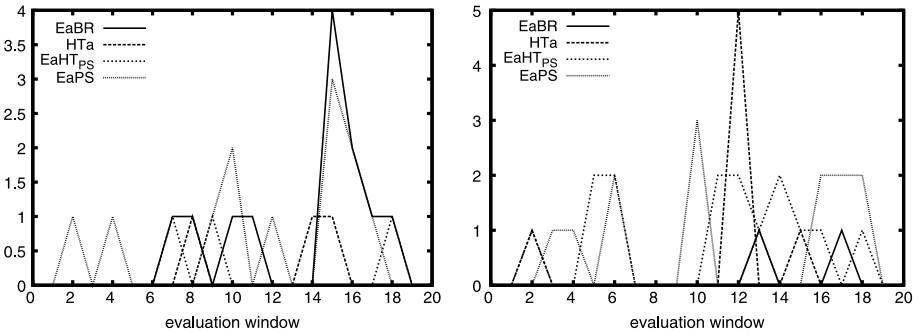


Fig. 7 Change-detections on the *SynG-drift* (*left*) and *SynT-drift* datasets over time (*right*). Change is centred over windows 5, 10, and 15, with each shift being more abrupt

are an interesting area for future work (beyond the scope of this method-focussed paper). We see in Fig. 5 that MWC's drift-detection method results in much less fluctuation in predictive performance.

Although not perfect, ADWIN performs its task well as a change detector. For example, looking at the detections under EaHT$_{PS}$ in Fig. 7, it is clear that ADWIN has detected each concept change. The picture is similar for the other methods using ADWIN (recall that ADWIN monitors the accuracy metric and thus does not perform equally across all methods). We also see that more changes are signalled in a shorter period of time for the latter two drifts (which are more abrupt). EaPS's ADWIN signals change more often—indicating that learning label-combinations found in the first 1000 instances may not be enough in this context. BR's ADWIN does not find the first concept-drift because it isn't one: the first drift involves a partial change in the label dependencies, but BR learns labels independently and thus its accuracy isn't affected. Comparing to MBR's performance, we can say that methods perform better trying to detect points of change with ADWIN (and risk a possible false alarm) than simply assuming that there is always change (as MBR does as a batch-incremental method, by continuously phasing out old data).

Overall, our new EaHT$_{PS}$ method achieves high predictive performance, and can deal well with evolving data. Time performance is sufficient for a data stream context: even in the worst cases processing many examples per second. Its ranking is similar to MWC and EaBR,

but much faster than the batch-incremental MBR, which takes many hours to complete on *IMDB*; it is over three times slower, in fact.

Generally, the best methods for each dataset perform approximately as well as batch-methods in cross fold evaluation (when we compare to papers Read et al. 2008, 2009 etc.). Although the base classifiers are different, as well as the evaluation methodology, we can nevertheless see that, given enough training examples, incremental streaming methods are able to get close to the performance obtained by batch-training methods. This is an important justification for incremental methods.

8 Conclusions and future work

We studied methods for multi-label data stream classification, and presented a novel method using Hoeffding trees with multi-label Pruned Sets classifiers at the leaves. This approach inherits the high performance of incremental decision trees, and the predictive power of an efficient multi-label method. We used an ADWIN-bagging ensemble framework to allow adaption to concept drift in evolving data. Unlike batch-incremental methods, all our methods learn and make predictions in real time and update their models at every example—only restarting models when drift is detected.

Our experimental framework is the first of its kind involving a variety of instance-incremental multi-label stream classifiers, able to generate many different types of evolving synthetic multi-label data, and use several multi-label evaluation measures.

We carried out a multi-dimensional evaluation including running time on a range of real and synthetic data sources of up to one million training instances, including cases of concept drift. Results were conclusive. Our method performed efficiently in terms of time and memory and also convincingly achieved the overall highest predictive performance. Compared to other methods in the literature, we build a strong case for using instance-incremental methods, as opposed to batch-incremental methods.

In the future we plan to look more closely at the drift-detection mechanism in the multi-label context and compare with competing drift-detection methods, as well as implement non-tree based methods and experiment with even larger data streams.

Acknowledgements The authors would like to thank Eleftherios Spyromitros-Xioufis and coauthors for kindly providing their MWC software implementation.

References

Appice, A., & Džeroski, S. (2007). Stepwise induction of multi-target model trees. In *Proceedings of the 18th European conference on machine learning, ECML '07* (pp. 502–509). Berlin: Springer.

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the seventh SIAM international conference on data mining*, April 26–28, 2007, Minneapolis, Minnesota, USA (pp. 443–448). Philadelphia: SIAM.

Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Lecture notes in computer science: Vol. 5772. Advances in intelligent data analysis VIII, 8th international symposium on intelligent data analysis, IDA 2009*, Lyon, France, August 31–September 2, 2009 (pp. 249–260). Berlin: Springer.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., & Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, Paris, France, June 28–July 1, 2009 (pp. 139–148). New York: ACM.

Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., & Seidl, T. (2010). MOA: Massive online analysis, a framework for stream classification and clustering. *Journal of Machine Learning Research*, *11*, 44–50.

Cesa-Bianchi, N., Gentile, C., & Zaniboni, L. (2006). Hierarchical classification: Combining Bayes with SVM. In *ICML '06: Proceedings of the 23rd international conference on machine learning* (pp. 177–184). New York: ACM Press.

Cheng, W., & Hüllermeier, E. (2009). Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, *76*(2–3), 211–225.

Cheng, W., Dembczynski, K., & Hüllermeier, E. (2010). Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, June 21–24, 2010, Haifa, Israel (pp. 279–286). Madison: Omnipress.

Clare, A., & King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In *Lecture notes in computer science: Vol. 2168. Principles of data mining and knowledge discovery, 5th European conference, PKDD 2001*, Freiburg, Germany, September 3–5, 2001 (pp. 42–53). Berlin: Springer.

Crammer, K., & Singer, Y. (2003). A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, *3*, 1025–1058.

Dembczyński, K., Waegeman, W., Cheng, W., & Hüllermeier, E. (2010). On label dependence in multi-label classification. In *Workshop proceedings of learning from multi-label data*, Haifa, Israel (pp. 5–12).

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 71–80). New York: ACM.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., & Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning*, *73*(2), 133–153.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, Paris, France, June 28–July 1, 2009 (pp. 329–338). New York: ACM.

Godbole, S., & Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In *Lecture notes in computer science: Vol. 3056. Advances in knowledge discovery and data mining, 8th Pacific-Asia conference, PAKDD 2004*, Sydney, Australia, May 26–28, 2004 (pp. 22–30). Berlin: Springer.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Explorations*, *11*(1), 10–18.

Ikonomovska, E., Gama, J., & Dzeroski, S. (2011). Incremental multi-target model trees for data streams. In *Proceedings of the 2011 ACM symposium on applied computing, SAC'11* (pp. 988–993). New York: ACM.

Kirkby, R. (2007). *Improving Hoeffding trees*. Ph.D. thesis, University of Waikato.

Kong, X., & Yu, P. S. (2011). An ensemble-based approach to fast classification of multi-label data streams. In *Proceedings of the 7th IEEE international conference on collaborative computing: networking, application and worksharing, collaborate Com'11*.

Law, Y. N., & Zaniolo, C. (2005). An adaptive nearest neighbor classification algorithm for data streams. In *Lecture notes in computer science: Vol. 3721. Knowledge discovery in databases: PKDD 2005, 9th European conference on principles and practice of knowledge discovery in databases*, Porto, Portugal, October 3–7, 2005 (pp. 108–120). Berlin: Springer.

Oza, N. C., & Russell, S. J. (2001). Online bagging and boosting. In *Artificial intelligence and statistics* (pp. 105–112). San Mateo: Morgan Kaufmann.

Park, S. H., & Fürnkranz, J. (2008) *Multi-label classification with label constraints* (Tech. rep.). Knowledge Engineering Group, TU Darmstadt.

Qu, W., Zhang, Y., Zhu, J., & Qiu, Q. (2009). Mining multi-label concept-drifting data streams using dynamic classifier ensemble. In *Lecture notes in computer science: Vol. 5828. Advances in machine learning, first Asian conference on machine learning, ACML 2009*, Nanjing, China, November 2–4, 2009 (pp. 308–321). Berlin: Springer.

Ráez, A. M., López, L. A. U., & Steinberger, R. (2004). Adaptive selection of base classifiers in one-against-all learning for large multi-labeled collections. In *Lecture notes in computer science: Vol. 3230. Advances in natural language processing, 4th international conference, EsTAL 2004*, Alicante, Spain, October 20–22, 2004 (pp. 1–12). Berlin: Springer.

Read, J. (2010). *Scalable multi-label classification*. Ph.D. thesis, University of Waikato.

Read, J., Pfahringer, B., & Holmes, G. (2008). Multi-label classification using ensembles of pruned sets. In *Proceedings of the 8th IEEE international conference on data mining (ICDM 2008)*, December 15–19, 2008, Pisa, Italy (pp. 995–1000). Washington: IEEE Computer Society.

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2009). Classifier chains for multi-label classification. In *Lecture notes in computer science: Vol. 5782. Machine learning and knowledge discovery in databases, European conference, ECML PKDD 2009, Proceedings, Part II*, Bled, Slovenia, September 7–11, 2009 (pp. 254–269). Berlin: Springer.

Read, J., Bifet, A., Holmes, G., & Pfahringer, B. (2010). *Efficient multi-label classification for evolving data streams* (Tech. rep.). University of Waikato, Hamilton, New Zealand. Working Paper 2010/04.

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*.

Schapire, R. E., & Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine Learning*, *39*(2/3), 135–168.

Spyromitros-Xioufis, E., Spiliopoulou, M., Tsoumakas, G., & Vlahavas, I. (2011). Dealing with concept drift and class imbalance in multi-label stream classification. In *IJCAI* (pp. 1583–1588).

Tsoumakas, G., & Katakis, I. (2007). Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, *3*(3), 1–13.

Tsoumakas, G., & Vlahavas, I. (2007). Random *k*-labelsets: An ensemble method for multilabel classification. In *Lecture notes in computer science: Vol. 4701. Machine learning: ECML 2007, 18th European conference on machine learning, Proceedings*, Warsaw, Poland, September 17–21, 2007 (pp. 406–417). Berlin: Springer.

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

Zhang, M. L., & Zhou, Z. H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, *40*(7), 2038–2048.