Batch-Incremental versus Instance-Incremental Learning in Dynamic and Evolving Data

Jesse Read[2], Albert Bifet[1], Bernhard Pfahringer[1], and Geoff Holmes[1]

[1] University of Waikato
Hamilton, New Zealand
{abifet,bernhard,geoff}@cs.waikato.ac.nz
[2] Universidad Carlos III
Madrid, Spain
jesse@tsc.uc3m.es

Abstract. Many real world problems involve the challenging context of data streams, where classifiers must be incremental: able to learn from a theoretically-infinite stream of examples using limited time and memory, while being able to predict at any point. Two approaches dominate the literature: batch-incremental methods that gather examples in batches to train models; and instance-incremental methods that learn from each example as it arrives. Typically, papers in the literature choose one of these approaches, but provide insufficient evidence or references to justify their choice. We provide a first in-depth analysis comparing both approaches, including how they adapt to concept drift, and an extensive empirical study to compare several different versions of each approach. Our results reveal the respective advantages and disadvantages of the methods, which we discuss in detail.

Keywords: data streams, incremental, dynamic, evolving, on-line

1 Introduction

The trend towards dynamic data sources is clear, both in the real world and the academic literature. Modern data sources are not only dynamic but generated at high speed in real time. Such contexts can be found in sensor applications, measurements in network monitoring and traffic management, log records or click-streams in web exploration, manufacturing processes, call-detail records, email, blogs, news feeds, and social networks. 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 [1]:

- 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).

It is important to note that in this study we do not tackle two important aspects of this setting. First, we do not consider changes to the input distribution in terms of the addition, deletion or updating of attributes. In a sensor network, for example, a new sensor could be added to or deleted from an existing network, or a new sensor could replace an old one and produce a different range of values. Second, we assume that all instances *can* be labelled. Labels could, in practice, come at a cost. Stream-based methods to tackle these problems are being developed but are beyond the scope of this study.

The approaches to data-stream classification in the literature can generally be considered as being of one of two types: *batch-incremental*, or *instance incremental*.

In the batch-incremental approach, a traditional batch-learning method is trained on batches of the data: every w new examples form a batch, and when that batch is complete, it is given to a learner to train on. The main disadvantages of these methods are that they:

- require a parameter w specifying the batch-size;
- are forced to delete trained models to make room for new ones; and
- cannot learn from the most recent examples until a new batch is full.

Having to delete trained models may affect these methods' ability to learn a complete concept, and not being able to learn from new examples immediately may affect their ability to respond to a new concept.

Instance-incremental methods are truly incremental in the sense that they learn from each training example as it arrives. This category includes lazy learners (like k-Nearest Neighbour, e.g., [2, 3]) and incremental learners such as Naive Bayes [4] and Hoeffding Trees [5] that can essentially learn indefinitely. Due to their incremental nature, instance-incremental methods are often chosen over batch-incremental methods, but they also have disadvantages; most notably:

- are fewer in number than batch methods (thus a smaller selection of appropriate methods); and often
- only learn a concept correctly from a huge number of examples.

For example, Hoeffding Trees grow very slowly with respect to the number of instances they observe. Lazy methods such as k-Nearest Neighbour learn more quickly in this respect from far fewer examples, but these methods are limited to a relatively small internal buffer of instances, that they must search through and add to for every new example in the stream. Thus, `kNN` methods must discard information over time like batch-incremental methods do (although only one instance at a time in this case).

Since data streams can be susceptible to concept drift, response to concept drift is an important issue; although detecting drift is a more important issue for instance-incremental methods like Hoeffding Trees. k-Nearest Neighbour and batch-incremental methods adapt to some extent automatically, as they are forced to phase out part of their model over time due to resource limitations, but Hoeffding Trees will simply learn a new concept 'on top' of an old concept

and, therefore, often perform better when used with a scheme with an explicit concept-change detector [6].

In the following section we review batch-incremental and instance-incremental methods. Although both kinds of methods have seen improvement over the years, so far no exhaustive comparison study has been published. Instance-incremental papers consistently mention the benefits of learning incrementally, whereas batch-incremental papers consistently mention that batch-learning is adequate for data streams.

In this paper we address this important lack of knowledge. We investigate the relative importance of the advantages and disadvantages of both approaches with a large empirical evaluation involving a variety of concept-drifting data streams and several of the best batch-incremental and instance-incremental approaches from the literature. We present these results as evidence indicating which approaches are better in which circumstances; thus providing crucial information for researchers deciding on an approach to use. It should be noted that such studies have been crucial to the development of classification algorithms in the non-incremental batch setting, both in terms of improving the quality of datasets used for evaluation [7] and for the assessment of overall classifier performance [8].

2 Prior Work

Naive Bayes [4] is a widely known instance-incremental learner; it simply updates internal counters with each new instance and uses these counters to assign a class in a probabilistic fashion to a new item in the stream. Stochastic Gradient Descent [9] is another incremental algorithm which forms the base for many neural network methods.

Naive Bayes provides a baseline for instance-incremental classification, but in terms of performance, has already been superseded by *Hoeffding Trees* [5]: 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.

Bagging and Boosting ensemble methods can be adapted to the stream setting and do improve the accuracy of base classifier methods. Oza and Russell [10, 11] proposed *Online Bagging* which gives each example a weight according to a Poisson distribution. This method has been shown to work well with Hoeffding Trees [12]. A more recent version of Bagging is presented in [13] which obtains better accuracy albeit at the cost of additional use of computational resources.

Batch-incremental methods cannot learn instance-by-instance, but must create models from batches, and at some point remove them as memory fills up. The size of the batch must be chosen to provide a balance between best model accuracy (large batches) and best response to new instances (smaller batches). In this context, it makes sense to use an ensemble, where several models are created from relatively small batches and their predictive power is combined under a voting scheme, such as in [14] and the Accuracy Weighted Ensemble (AWE) [15], where typically, when the maximum number of models is reached (the ensemble

Table 1: The methods we consider. Ensemble iterations are specified by parameter n. Note that Accuracy Weighted Ensemble (`AWE-`) can be used with any classifier; Leveraging Bagging (`LB-`) can be used with any incremental classifier.

Key	Classifier	Parameters
`NB`	Naive Bayes	
`SGD`	Stochastic Gradient Descent	
`HT`	Hoeffding Tree	
`LB-HT`	Leveraging Bagging / `HT`	$n = 10$
`kNN`	k Nearest Neighbour	$w = 1000, k = 10$
`LB-kNN`	Leveraging Bagging / `kNN`	$n = 10$
`AWE-SMO`	AWE of Support Vector Machines	$w = 500, n = 10$
`AWE-J48`	AWE of C4.5 Decision Trees	$w = 500, n = 10$
`AWE-LR`	AWE of Logistic Regression	$w = 500, n = 10$

size) the oldest model is reset with a model built from the newest batch. In AWE, ensemble members are additionally weighted by their classification performance.

The k-Nearest Neighbour algorithm, which assigns the most common class of the k most similar examples, has been used in data streams in [2]. This algorithm is naturally suited to this setting because of its instance-incremental nature. Improved searching [3] and instance-compressing techniques [16] have been shown to improve its capacity considerably.

Reacting to concept drift is a fundamental part of learning from data streams. `kNN` and batch-incremental methods inherently phase out data (and thus, old concepts) but instance-incremental models such as Hoeffding Trees need an explicit change detection, or else they will learn a new concept 'on top of' and old concept. `ADWIN` [6] keeps a variable-length window of recently seen items (such as the current classification performance) and signals change when the concept within this window changes, and thus can be coupled with any method that requires explicit change detection, as has been done successfully with Hoeffding Trees in [12].

3 Experimental Setup and Methodology

We compare the performance of the batch-incremental methods (using the recent Accuracy Weighted Ensemble method of [15]) employed with powerful batch classifiers (Support Vector Machines, C4.5 Decision Trees, Logistic Regression) with instance-incremental methods (Naive Bayes, Hoeffding Tree ensembles, Stochastic Gradient Descent, and k Nearest Neighbour variations). These methods, and their parameters, are displayed in Table 1. All methods have been implemented in Java extending the MOA framework for data streams [18]. Any parameters not shown in the table are the default ones set in this framework.

We use the experimental framework for concept drift presented in [1]: considering data streams as data generated from pure distributions, we can model

a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after drift. This framework defines the probability that a new instance of the stream belongs to the new concept after the drift based on the sigmoid function.

3.1 Data

In our experiments we use a range of both real and synthetic data sources.

Synthetic data has several advantages–it is easier to reproduce and there is little cost in terms of storage and transmission. We use the data generators most commonly found in the literature.

SEA Concepts Generator An artificial dataset, introduced in [19], which contains abrupt concept drift. It is generated using three attributes. All attributes have values between 0 and 10. The dataset is divided into four concepts by using different thresholds θ; such that: $f_1 + f_2 \leq \theta$ where f_1 and f_2 are the first two attributes, for $\theta = 9, 8, 7$ and 9.5.

Rotating Hyperplane The orientation and position of a hyperplane in d-dimensional space is changed to produce concept drift; see [20].

Random RBF Generator Using a fixed number of centroids of random position, standard deviation, class label and weight. Drift is introduced by moving the centroids with constant speed.

LED Generator The goal is to predict the digit displayed on a seven-segment LED display, where each attribute has a 10inverted; LED comprises 24 binary attributes, 17 of which are irrelevant; see [21].

We consider three of the largest datasets from the UCI repository [22]:

Forest Covertype Contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service data. It contains $581,012$ instances and 54 attributes. It has been used in, for example, [23, 10].

Poker-Hand $1,000,000$ instances represent all possible poker hands. Each card in a hand is described by two attributes: suit and rank. Thus there are 10 attributes describing each hand. The class indicates the value of a hand. We sorted by rank and suit and removed duplicates.

Electricity Contains $45,312$ instances describing electricity demand. A class label identifies the change of the price relative to a moving average of the last 24 hours. It was described by [24] and analysed also in [17].

Since these real datasets are relatively small compared to the synthetic datasets we consider, and because we do not know when drift occurs (or, indeed, if there is any drift) we simulate concept drift, joining the three datasets, merging attributes, and supposing that each dataset corresponds to a different concept, as described in [13].

Text data is also an important source of data streams in the real-world. We consider the following two sources:

20 Newsgroups [25] is a dataset commonly used in cluster analysis. It has 19300 entries, each represented with 1000 binary attributes (word presence/absence), corresponding to at least one of 20 newsgroups. We convert this dataset into 20 binary classification problems, one for each newsgroup, and append them all into one large binary-class dataset. Thus we model a stream (similarly to CovPokElec) of 386,000 records with 19 shifts in concept.

IMDB A dataset used in multi-label learning [26]. It contains 120919 textual movie plot summaries of 1000 binary class attributes and 0/1-associations to genres. We use the *drama* genre, which is the most frequently occurring.

3.2 Methodology

The experiments were performed on 2.66 GHz Core 2 Duo E6750 machines with 4 GB of memory. We used the Interleaved Test-Then-Train evaluation methodology: every example was used for testing the model before using it to train. From the synthetic concepts we generated 1 million examples with the following parameters:

- RBF(x,v): RandomRBF of 5 classes with x centroids moving at speed v.
- HYP(x,v): Hyperplane of 2 classes with x attributes changing at speed v.
- SEA(v): SEA dataset, with length of change v.
- LED(v): LED dataset, with length of change v.

The Nemenyi test [27] 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 a p-value of 0.05. Under the Nemenyi test, $\{x\} \succ \{z\}$ indicates that algorithm x is statistically significantly more likely to be more favourable than z.

In [28] the use of RAM-Hours is introduced as an evaluation measure of the resources used by streaming algorithms. Every GB of RAM deployed for 1 hour equals one RAM-Hour.

3.3 Parameter Selection

As discussed in Section 1, both lazy-learning and batch-incremental methods require a *window* parameter, which determines the number of examples used by their model (the only difference being that lazy methods learn from this window incrementally rather than as a batch). We conduct an analysis on the effects of different window-size parameters for `kNN` and `AWE-` methods. Table 2 displays results for a variety of window/batch sizes, which provides justification for our choice of parameters in the following section: $w = 1000$ and $w = 500$ for `kNN` and `AWE-`, respectively. Although $w = 5000$ results in slightly better accuracy for `kNN`, the computation cost is clearly potentially prohibitive. The setting of $w = 500$ appears to work well for all batch methods both with respect to classification and time and memory performance. However, the optimal w is different for each stream. For example as we see in Table 3, `AWE-J48` has no clear optimal value for all datasets.

Table 2: Finding the best window size for kNN, and AWE-.

(a) Average accuracy

	$-w$ 100	$-w$ 500	$-w$ 1000	$-w$ 5000
kNN	66.32	80.24	82.33	**82.63**
AWE-J48	70.72	**77.36**	76.90	73.76
AWE-LR	68.77	**69.62**	67.83	65.56
AWE-SMO	67.13	**70.77**	70.07	67.67

(b) Total Time (seconds)

	$-w$ 100	$-w$ 500	$-w$ 1000	$-w$ 5000
kNN	2,180	9,993	18,349	71,540
AWE-J48	3,809	6,883	10,865	28,429
AWE-LR	9,659	66,757	10,247	10,112
AWE-SMO	13,860	5,800	6,414	39,298

(c) RAM-Hours

	$-w$ 100	$-w$ 500	$-w$ 1000	$-w$ 5000
kNN	0.13	1.11	2.98	41.27
AWE-J48	1.96	8.49	21.81	221.66
AWE-LR	12.65	48.07	22.47	67.52
AWE-SMO	3.19	4.12	9.36	255.96

Table 3: Finding the best window size for AWE-J48.

	$-w$ 100	$-w$ 500	$-w$ 1000	$-w$ 5000
20 Newsgroups	94.30	94.74	**95.06**	94.60
IMDB	**55.09**	53.59	53.54	54.33
CovType	55.79	**87.82**	85.58	76.05
Electricity	**78.47**	75.27	74.37	65.10
Poker	76.06	77.89	**79.32**	75.98
CovPokElec	68.03	**81.60**	81.45	74.32
LED(50000)	70.60	71.99	**72.03**	71.37
SEA(50)	84.95	88.03	88.56	**88.68**
SEA(50000)	84.63	87.71	88.16	**88.43**
HYP(10,0.0001)	66.69	71.58	73.41	**78.63**
HYP(10,0.001)	70.95	75.79	77.69	**79.94**
RBF(0,0)	69.42	83.01	84.96	**87.38**
RBF(50,0.0001)	69.12	**79.30**	77.05	60.75
RBF(10,0.0001)	68.49	81.79	**82.78**	80.79
RBF(50,0.001)	**53.78**	50.95	38.55	24.50
RBF(10,0.001)	65.18	76.76	77.92	**79.36**
Average	70.72	**77.36**	76.90	73.76

4 Results: Batch-incremental versus Instance-Incremental

Table 4 displays the final accuracy and resource use (time and RAM-hours) of methods and their parameters (see Table 1). Accuracy is measured as the final percentage of examples correctly classified over the test/train inter-leaved evaluation.

Naive Bayes (NB) uses very few resources, but its accuracy is poor compared to all other methods; with a few exceptions: HYP(10,0.0001), the SEA streams, and to some extent on Electricity and IMDB. Such low average Naive Bayes accuracy indicates that the concepts to be learned are reasonably hard problems. A similar scenario is observed for Stochastic Gradient Descent (SGD) which is only seriously competitive on the 20 Newsgroups dataset.

As claimed in the literature, Hoeffding Trees (HT) are generally a better option than NB for instance-incremental methods. Table 4 provides an important comparison between Hoeffding trees and non-incremental decision trees in a batch setting (AWE-J48). It shows that, although the computational cost of

Table 4: Comparison of all methods.

(a) Accuracy

	NB	kNN	HT	AWE-J48	LB-HT	SGD	AWE-LR	AWE-SMO	LB-kNN
20 Newsgroups	68.13	94.86	94.30	94.74	94.38	94.86	88.43	95.56	DNF
IMDB	60.42	60.82	63.51	53.59	61.76	63.79	53.96	54.52	62.44
CovType	60.52	92.22	80.31	87.82	88.61	60.70	84.50	84.24	92.39
Electricity	73.36	78.38	79.20	75.27	88.77	57.58	70.55	68.56	80.78
Poker	59.55	69.35	76.07	77.89	94.97	68.92	60.90	60.38	70.34
CovPokElec	24.24	78.41	79.34	81.60	92.41	68.06	70.07	69.77	79.09
LED(50000)	54.02	63.20	68.65	71.99	73.15	11.84	73.03	72.80	69.77
SEA(50)	85.37	86.80	86.42	88.03	88.24	85.41	89.44	89.57	88.00
SEA(50000)	85.38	86.55	86.42	87.71	88.80	85.21	89.01	89.15	87.74
HYP(10,0.0001)	91.25	83.29	89.04	71.58	88.06	79.54	93.73	93.41	87.10
HYP(10,0.001)	70.91	83.33	78.77	75.79	84.85	71.10	91.75	92.02	86.91
RBF(0,0)	51.21	88.99	83.25	83.01	89.70	16.63	46.91	50.52	90.59
RBF(50,0.0001)	30.99	89.36	45.49	79.30	76.70	16.63	54.89	57.85	90.49
RBF(10,0.0001)	52.10	89.30	79.24	81.79	85.54	16.63	50.96	52.80	90.73
RBF(50,0.001)	29.14	84.03	32.29	50.95	55.72	16.63	46.48	50.42	82.10
RBF(10,0.001)	51.96	88.34	76.39	76.76	81.82	16.63	49.37	50.74	88.93
Average	59.29	82.33	74.92	77.36	83.34	51.89	69.62	70.77	83.16

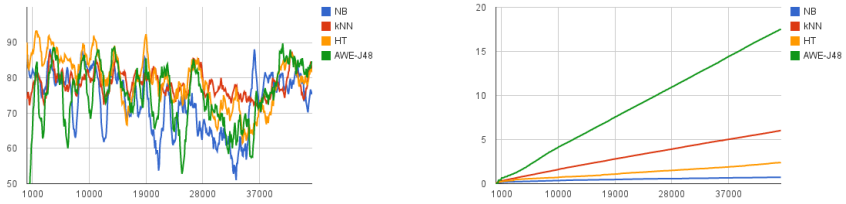
Nemenyi significance: kNN≻NB; kNN≻SGD; LB-HT≻NB; LB-HT≻SGD; LB-kNN≻NB; LB-kNN≻SGD;

(b) Time (seconds)

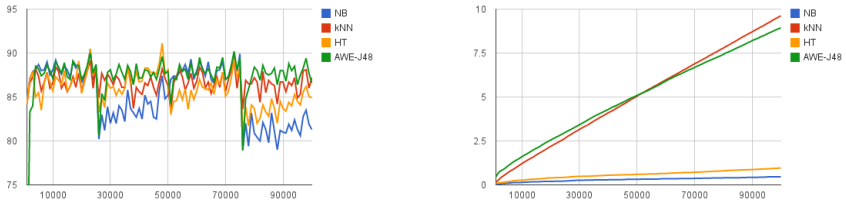
	NB	kNN	HT	AWE-J48	LB-HT	SGD	AWE-LR	AWE-SMO	LB-kNN
20 Newsgroups	93.48	11,544.68	177.34	3,448.89	4,996.64	5.45	3,187.23	293.93	DNF
IMDB	27.85	2,761.45	49.54	1,855.76	1,563.07	1.49	1,240.25	224.85	63,784.28
CovType	18.52	266.18	20.06	91.72	247.59	5.43	823.42	257.51	6,708.11
Electricity	0.64	7.05	1.15	4.64	8.72	0.32	4.32	9.73	163.61
Poker	8.88	177.82	9.26	81.80	127.59	2.29	381.05	322.51	3,454.15
CovPokElec	47.72	1,447.92	46.77	284.39	1,032.43	8.60	2,006.50	899.75	30,329.08
LED(50000)	9.14	447.36	15.90	135.10	189.13	2.04	57,466.83	1,662.00	10,816.07
SEA(50)	2.90	107.82	4.63	63.46	94.44	1.30	56.04	91.96	3,138.38
SEA(50000)	3.01	112.84	4.80	60.83	94.62	1.41	56.18	95.59	3,125.99
HYP(10,0.0001)	4.33	222.72	8.46	103.50	221.79	1.45	191.67	164.04	6,492.46
HYP(10,0.001)	4.34	222.26	9.70	103.62	224.57	1.46	193.61	159.64	6,379.68
RBF(0,0)	8.20	206.97	13.90	136.90	203.74	2.38	227.62	357.36	6,279.06
RBF(50,0.0001)	8.24	200.41	14.92	128.60	236.72	2.41	234.64	332.03	7,494.13
RBF(10,0.0001)	7.31	202.51	13.00	132.31	200.82	1.70	231.30	311.82	5,523.36
RBF(50,0.001)	8.34	218.12	14.16	120.13	234.15	2.41	225.13	303.85	6,765.54
RBF(10,0.001)	7.40	202.84	12.94	131.49	201.36	1.68	231.23	313.33	5,858.01
Total	260.28	18,348.95	416.53	6,883.14	9,877.38	41.82	66,757.02	5,799.90	166,311.91

(c) RAM-Hours (MB)

	NB	kNN	HT	AWE-J48	LB-HT	SGD	AWE-LR	AWE-SMO	LB-kNN
20 Newsgroups	0.01	2.18	3.61	4.94	340.77	0.00	10.24	0.63	DNF
IMDB	0.00	0.41	0.38	2.63	20.39	0.00	3.76	0.44	33.95
CovType	0.00	0.04	0.01	0.08	0.05	0.00	0.75	0.25	4.21
Electricity	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04
Poker	0.00	0.01	0.00	0.03	0.20	0.00	0.13	0.12	0.88
CovPokElec	0.00	0.25	0.11	0.33	1.95	0.00	2.31	1.12	20.94
LED(50000)	0.00	0.03	0.01	0.10	0.49	0.00	30.38	0.95	4.47
SEA(50)	0.00	0.00	0.00	0.01	1.95	0.00	0.01	0.02	0.65
SEA(50000)	0.00	0.00	0.00	0.01	0.89	0.00	0.01	0.02	0.65
HYP(10,0.0001)	0.00	0.01	0.00	0.04	13.30	0.00	0.06	0.05	1.77
HYP(10,0.001)	0.00	0.01	0.01	0.04	1.54	0.00	0.06	0.05	1.75
RBF(0,0)	0.00	0.01	0.00	0.06	3.01	0.00	0.07	0.11	1.69
RBF(50,0.0001)	0.00	0.01	0.00	0.05	0.20	0.00	0.07	0.10	2.01
RBF(10,0.0001)	0.00	0.01	0.00	0.06	3.31	0.00	0.07	0.09	1.50
RBF(50,0.001)	0.00	0.01	0.00	0.05	0.02	0.00	0.07	0.09	1.81
RBF(10,0.001)	0.00	0.01	0.00	0.06	3.08	0.00	0.07	0.09	1.59
Total	0.02	2.98	4.15	8.49	391.16	0.00	48.07	4.12	77.90
Total without 20 Newsgroups	0.01	0.80	0.54	3.55	50.39	0.00	37.83	3.49	77.90



(a) A selection of methods on Electricity dataset, accuracy (left) time (right)



(b) A selection of methods on SEA dataset, accuracy (left) time (right)

Fig. 1: Classification accuracy (left) and running time (right) over time for methods on a selection of datasets.

`AWE-J48` is often up to 10 times greater than Hoeffding trees (see also Figure 1), it often improves on them–for example on the RBF(50,0.0001) and CovType streams. On the other hand, on IMDB and Electricity, this trend is reversed; `HT` is superior. These two real-world datasets do not contain any obvious abrupt concept drift, thus supporting the claim that batch approaches automatically deal to some extent with concept drift. Figure 1b clearly shows the importance of adapting or changing models when there is concept drift.

Under a modern adaptive bagging scheme (`LB-HT`) (which resets models when drift is detected) Hoeffding Trees are powerful, albeit – in many cases – at an increased computational cost, particularly with regard to RAM-Hours.

The lazy `kNN` method performs very well across all data sources, and even as a standalone method it is one of the highest-performing methods overall. This is particularly surprising since `kNN`'s model is based on an internal buffer of 1000 instances; `kNN` models a concept well with a relatively small number of examples. We also note that in our experiments `AWE` is based only upon $n \times w = 5000$ instances; a small number relative to the size of the streams. The strengths of `kNN` are even apparent on datasets without drift, but it competes best on the most evolving data since it models new examples as soon as they arrive in the stream. Only `LB-HT` can compete seriously with the `kNN` methods, but the

difference in accuracy is insignificant. It is true that the time costs of kNN can be quite high, and that LB-kNN is one of the most expensive methods to run, but this can be mitigated somewhat by different search techniques, as explained in [3]. kNN is particularly robust: it obtains very good results across a wide range of data sources.

As expected, AWE gives different performance depending on its base classifier. This illustrates a key advantage of batch-methods: any existing classifier can be used. Under all the SEA and HYP streams and the 20 Newsgroups text dataset AWE-SMO obtains the best accuracy of all methods. On the other hand; its accuracy is very poor compared to several other methods on the RBF streams and the Electricity and Poker datasets; contributing significantly to its overall average performance. With the exception of the HYP streams, Logistic Regression (AWE-LR) does not make much of an impact in classification accuracy, and clearly runs very slowly on many datasets.

A summary of the most important observations are:

- storing a model from recent examples can be as effective as learning incrementally and keeping statistics from hundreds of thousands of examples,
- the best batch size is dependent on the data stream in consideration; and
- certain batch methods excel on certain problems, but lazy learners provide similar or better classification performance using less resources.

5 Conclusions

We investigated a variety of methods from two distinct branches of the data-stream literature: *instance-incremental* and *batch-incremental* approaches to classification. Our extensive and varied empirical evaluation of real and synthetic data sources of up to 1 million training instances with diffrent types and magnitudes of concept drift provide us with enough evidence to draw some important novel conclusions: instance-incremental methods perform similarly to their equivalent batch-learning implementation while using fewer resources. An explicit drift-detection and adaption mechanism is essential for any learner which does not automatically discard old information. We found lazy methods perform exceptionally well when using just a buffer of the 1000 most recent instances, even compared to powerful incremental methods.

References

1. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: KDD. (2009) 139–148
2. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. Intelligent Data Analysis **11**(6) (2007) 627–650
3. Zhang, P., Gao, B.J., Zhu, X., Guo, L.: Enabling fast lazy learning for data streams. In: ICDM. (2011) 932–941

4. John, G.H., Langley, P.: Estimating continuous distributions in bayesian classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, Morgan Kaufmann (1995) 338–345
5. Domingos, P., Hulten, G.: Mining high-speed data streams. In: KDD. (2000) 71–80
6. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: SDM. (2007)
7. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **11** (1993) 63–91
8. Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. In: ICML. (2006) 161–168
9. Bottou, L.: Online algorithms and stochastic approximations. Online Learning and Neural Networks (1998)
10. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: KDD. (2001) 359–364
11. Oza, N., Russell, S.: Online bagging and boosting. In: Artificial Intelligence and Statistics 2001, Morgan Kaufmann (2001) 105–112
12. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: IDA. (2009) 249–260
13. Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: ECML/PKDD (1). (2010) 135–150
14. Qu, W., Zhang, Y., Zhu, J., Qiu, Q.: Mining multi-label concept-drifting data streams using dynamic classifier ensemble. In: ACML. (2009) 308–321
15. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: KDD '03, New York, NY, USA, ACM (2003) 226–235
16. Spyromitros-Xioufis, E., Spiliopoulou, M., Tsoumakas, G., Vlahavas, I.: Dealing with concept drift and class imbalance in multi-label stream classification. In: IJCAI. (2011) 1583–1588
17. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In: SBIA. (2004) 286–295
18. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. Journal of Machine Learning Research (JMLR) (2010)
19. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: KDD. (2001) 377–382
20. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: KDD. (2001) 97–106
21. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
22. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
23. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: KDD. (2003) 523–528
24. Harries, M.: Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales (1999)
25. Lang, K.: The 20 newsgroups dataset. "`http://people.csail.mit.edu/jrennie/20Newsgroups/`" (2008)
26. Read, J., Bifet, A., Holmes, G., Pfahringer, B.: Scalable and efficient multi-label classification for evolving data streams. Machine Learning (2012) 1–30
27. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research **7** (2006) 1–30
28. Bifet, A., Holmes, G., Pfahringer, B., Frank, E.: Fast perceptron decision tree learning from evolving data streams. In: PAKDD. (2010)