An Efficient Algorithm for Instance-Based Learning on Data Streams

Jürgen Beringer[1] and Eyke Hüllermeier[2]

[1] Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg
beringer@iti.cs.uni-magdeburg.de
[2] Fachbereich Mathematik und Informatik
Philipps-Universität Marburg
eyke@mathematik.uni-marburg.de

Abstract. The processing of data streams in general and the mining of such streams in particular have recently attracted considerable attention in various research fields. A key problem in stream mining is to extend existing machine learning and data mining methods so as to meet the increased requirements imposed by the data stream scenario, including the ability to analyze incoming data in an online, incremental manner, to observe tight time and memory constraints, and to appropriately respond to changes of the data characteristics and underlying distributions, amongst others. This paper considers the problem of classification on data streams and develops an instance-based learning algorithm for that purpose. The experimental studies presented in the paper suggest that this algorithm has a number of desirable properties that are not, at least not as a whole, shared by currently existing alternatives. Notably, our method is very flexible and thus able to adapt to an evolving environment quickly, a point of utmost importance in the data stream context. At the same time, the algorithm is relatively robust and thus applicable to streams with different characteristics.

1 Introduction

In recent years, so-called *data streams* have attracted considerable attention in different fields of computer science. As the notion suggests, a data stream can roughly be thought of as an ordered sequence of data items, where the input arrives more or less continuously as time progresses [16]. There are various applications in which streams of this type are produced, such as network monitoring or telecommunication systems.

Apart from other issues such as data processing and querying, the problem of mining data streams has been studied in a number of recent publications (see e.g. [13] for an up-to-date overview). In this connection, different data mining problems have already been considered, such as clustering [1], classification [17], and frequent pattern mining [8]. In this paper, we are concerned with the classification problem. More specifically, we investigate the potential of the instance-based

approach to supervised learning within the context of data streams and propose an efficient instance-based learning algorithm.

The remainder of the paper is organized as follows: Section 2 provides some background information, both on data streams and on instance-based learning, and briefly reviews related work. Our approach to instance-based learning on data streams is introduced in section 3 and empirically evaluated in section 4. The paper concludes with a brief summary in section 5.

2 Background

2.1 Data Streams and Concept Change

The *data stream model* assumes that input data are not available for random access from disk or memory, such as relations in standard relational databases, but rather arrive in the form of one or more continuous data streams. The stream model differs from the standard relational model in various ways [4]: (i) The elements of a stream arrive incrementally in an "online" manner. That is, the stream is "active" in the sense that the incoming items trigger operations on the data rather than being sent on request. (ii) The order in which elements of a stream arrive are not under the control of the system. (iii) Data streams are potentially of unbounded size. (iv) Data stream elements that have been processed are either discarded or archived. They cannot be retrieved easily unless being stored in memory, which is typically small relative to the size of the stream (stored/condensed information about past data is often referred to as a *synopsis*). (v) Due to limited (memory) resources and strict time constraints, the computation of exact results will often not be possible. Therefore, the processing of stream data does commonly produce *approximate* results.

For the problem of mining data streams, the aforementioned characteristics have a number of important implications. First of all, in order to guarantee that results are always up-to-date, it is necessary to analyze the incoming data in an online manner, tolerating not more than a constant time delay. Since learning from scratch every time is generally excluded due to limited time and memory resources, corresponding learning algorithms must be *incremental*. Moreover, algorithms for learning on data streams must also be *adaptive*, i.e., they must be able to adapt to an evolving environment in which the data (stream) generating process may change over time. Thus, the handling of changing concepts is of utmost importance in mining data streams [5]. It has not only been considered in this context, however. In general, the literature distinguishes between different types of concept change over time [27]. The first type refers to a sudden, abrupt change of the underlying concept to be learned and is often called *concept shift*. Roughly speaking, in case of a concept shift, any knowledge about the old concept will typically become obsolete and the new concept has to be learned from scratch. The second type refers to a gradual evolution of the concept over time. In this scenario, old data might still be relevant, at least to some extent. Finally, one often speaks about *virtual* concept drift if not the concept itself changes

but the distribution of the underlying data generating process [29]. Note that in practice virtual and real concept drift can occur simultaneously.

Concept change can be handled in a direct or indirect way. In the indirect approach, the learning algorithm does not explicitly attempt to detect a concept drift. Instead, the use of outdated or irrelevant data is avoided from the outset. This is typically accomplished by considering only the most recent data while ignoring older observations, e.g., by sliding a window of fixed size over a data stream or by weighing the nearest neighbors of new observations, not only according to their distance but also according to their age. More generally, such strategies belong to the class of *instance selection* or *instance weighing* methods. To handle concept change in a more direct way, appropriate techniques for discovering the drift or shift are first of all required. Such techniques are typically based on statistical tests. Roughly speaking, the idea is to compare a certain statistic that refers to recently observed data with a corresponding statistic for older data, and to decide whether the difference between them is significant in a statistical sense. (A corresponding technique will be discussed in section 3 below.)

2.2 Instance-Based Learning

As opposed to model-based machine learning methods which induce a general model (theory) from the data and use that model for further reasoning, instance-based learning (IBL) algorithms simply store the data itself. They defer the processing of the data until a prediction (or some other type of query) is actually requested, a property which qualifies them as a *lazy* learning method [3, 2]. Predictions are then derived by combining the information provided by the stored examples.

Such a combination is typically accomplished by means of the *nearest neighbor* (NN) estimation principle [9]. Consider the following setting: Let $\mathcal{X}$ denote the instance space, where an instance corresponds to the description x of an object (usually though not necessarily in attribute–value form). $\mathcal{X}$ is endowed with a distance measure $\Delta(\cdot)$, i.e., $\Delta(x, x')$ is the distance between instances $x, x' \in \mathcal{X}$. $\mathcal{L}$ is a set of class labels, and $\langle x, \lambda_x \rangle \in \mathcal{X} \times \mathcal{L}$ is called a labeled instance, a case, or an example. In classification, which is the focus of this paper, $\mathcal{L}$ is a finite (usually small) set comprised of m classes $\{\lambda_1 \ldots \lambda_m\}$.

The current experience of the learning system is represented in terms of a set $\mathcal{D}$ of examples $\langle x_\imath, \lambda_{x_\imath} \rangle$, $1 \leq \imath \leq n = |\mathcal{D}|$. From a machine learning point of view, $\mathcal{D}$ plays the role of the *training set* of the learner. More precisely, since not all examples will necessarily be stored by an instance-based learner, $\mathcal{D}$ is only a subset of the training set. In case-based reasoning, it is also referred to as the *case base*; besides, in the context of data streams, $\mathcal{D}$ corresponds to the aforementioned *synopsis*.

Finally, suppose a novel instance $x_0 \in \mathcal{X}$ (a query) to be given, the class label λ_{x_0} of which is to be estimated. The NN principle prescribes to estimate this label by the label of the nearest (most similar) sample instance. The *k-nearest neighbor* (k-NN) approach is a slight generalization, which takes the

$k \geq 1$ nearest neighbors of x_0 into account. That is, an estimation $\lambda_{x_0}^{est}$ of λ_{x_0} is derived from the set $\mathcal{N}_k(x_0)$ of the k nearest neighbors of x_0, usually by means of a *majority vote*:

$$\lambda_{x_0}^{est} = \arg\max_{\lambda \in \mathcal{L}} \text{card}\{x \in \mathcal{N}_k(x_0) \mid \lambda_x = \lambda\}. \quad (1)$$

Regarding the suitability of IBL in the context of data streams, note that IBL algorithms are inherently incremental, since adaptation basically comes down to adding or removing observed cases. On the other hand, this training efficiency comes at the cost of high complexity at classification time, which involves retrieving the query's nearest neighbors. Consequently, IBL might be preferable (to model-based methods) in a data stream application if the number of incoming data is large compared with the number of queries to be answered, i.e., if model updating is the dominant factor.

2.3 Related Work

Data mining on streams is a topic of active research, and several adaptations of standard statistical and data analysis methods to data streams or related models have been developed recently [12]. Likewise, several online data mining methods have been devised (e.g. [10]), with a particular focus on unsupervised techniques like clustering. Supervised learning on streams, including classification, has received less attention so far, even though some approaches have already been developed.

A very early approach is the FLORA (Floating Rough Approximation) system [30]. The corresponding algorithm learns rule-based binary classifiers on a sliding window of fixed size. The FRANN (Floating Rough Approximation in Neural Networks) algorithm trains RBF networks on a sliding window of adaptive size [22]. The LWF (Locally Weighted Forgetting) algorithm of Salganicoff [25] is among the best adaptive learning algorithms. It is an instance-based learner that reduces the weights of the k nearest neighbors $x_1 \ldots x_k$ (in increasing order according to distance) of a new instance x_0 by the factor $\tau + (1-\tau)\Delta(x_i, x_0)^2/\Delta(x_k, x_0)^2$. An instance is completely removed if its weight falls below a threshold θ. To fix the size of the case base, the parameter k is adaptively defined by $k = \lceil \beta |\mathcal{D}| \rceil$ where $|\mathcal{D}|$ is the size of the current case base. As an obvious alternative to LWF, Salganicoff considers the TWF (Time Weighted Forgetting) algorithm that weights instances according to their age: at time point t, the example observed at time $t-k$ is weighted by w^k, where $w \in (0,1)$ is a constant. The Prediction Error Context Switching algorithm (PECS), also proposed in [25], does not delete but only deactivates instances. That is, removed instances are still stored in memory and might be reactivated later on. This strategy can avoid some disadvantages of LWF but entails storage requirements that disqualify PECS for the data stream context.

In the above approaches, the strategies for adapting the size of a sliding window, if any, are mostly of a heuristic nature. In [19], the authors propose to adapt the window size in such a way as to minimize the estimated generalization

error of the learner trained on that window. In [20], this approach is further generalized by allowing for the selection of arbitrary subsets of batches instead of only uninterrupted sequences. Despite the appealing idea of this approach to window (training set) adjustment, the successive testing of different window lengths is computationally expensive and therefore not immediately applicable in a data stream scenario with tight time constraints.

Recently, some efforts have been made to extend decision tree induction to the streaming scenario. In their CVFDT (Continuous Very Fast Decision Trees) algorithm, Hulton and Domingos learn and maintain decision trees on a sliding window of fixed size [17]. Another approach to adaptive learning is the use of ensemble techniques [21]. Here, the idea is to train multiple classifiers, often decision trees, on different blocks of data. To achieve adaptivity, the classifiers are weighted according to their (recent) performance or, even simpler, only the best classifier is selected to classify new instances. If concept drift occurs, outdated or poorly performing classifiers are replaced by new ones.

So-called *editing strategies* for nearest neighbor classification or, more generally, lazy learning have been studied for quite a while [24]. Even though these strategies are of course related to the problem of adaptive learning and handling concept change, they are not suitable for data stream applications, mainly for the following reasons: Firstly, they solely focus on the goal to maximize classification accuracy while disregarding other aspects like space complexity. Secondly, they are not flexible and efficient enough for online classification. In this connection, let us also mention the well-known IB3 algorithm [3], which is built upon IB1 and includes means to delete noisy and old instances that do no longer comply with the current concept. Even though IB3 is thus principally able to handle gradual concept drift, the adaptation is relatively slow [30, 25].

Finally, we note that there is also a bunch of work on time series data mining (e.g. [18]). However, even though time series data mining is of course related to stream data mining, one should not overlook important differences between these fields. Particularly, time series are still static objects that can be analyzed offline, whereas the focus in the context of data streams is on dynamic adaptation and online data mining.

3 Instance-Based Learning on Data Streams

This section introduces our approach to instance-based learning on data streams, referred to as IBL-DS. The learning scenario consists of a data stream that permanently produces examples, potentially with a very high arrival rate, and a second stream producing query instances to be classified. The key problem for our learning system is to maintain an implicit concept description in the form of a case base (memory). Before presenting details of IBL-DS in section 3.2, some general aspects and requirements of concept adaptation (case base maintenance) in a streaming context will be discussed in section 3.1.

3.1 Concept Adaptation

The simplest adaptive learners are those using sliding windows of fixed size. Unfortunately, by fixing the number of examples in advance, it is impossible to optimally adapt the size of the case base to the complexity of the concept to be learned, and to react to changes of this concept appropriately. Moreover, being restricted to selecting a subset of successive observations in the form of a window, it is impossible to disregard a portion of observations in the middle (e.g. outliers) while retaining preceding and succeeding blocks of data. To avoid both of the aforementioned drawbacks, non-window-based approaches are needed that do not only adapt the size of the training data but also have the liberty to select an arbitrary *subset* of examples from the data seen so far. Needless to say, such flexibility does not come for free. Apart from higher computational costs, additional problems such as avoiding an unlimited growth of the training set and, more generally, trading off accuracy against efficiency have to be solved.

Instance-based learning seems to be attractive in light of the above requirements, mainly because of its inherently incremental nature and the simplicity of model adaptation. In particular, since in IBL an example has only local influence, the update triggered by a new example can be restricted to a local region around that observation.

Regarding the updating (editing) of the case base in IBL, an example should in principle be retained if it improves the predictive performance (classification accuracy) of the classifier; otherwise, it should better be removed. Unfortunately, this criterion cannot be used directly, since the (future) usefulness of an example in this sense is simply not known. Instead, existing approaches fall back on suitable indicators of usefulness:

- Temporal relevance: According to this indicator, recent observations are considered as potentially more useful and, hence, are preferred to older examples.
- Spatial relevance: The relevance of an example can also depend on its position in the instance space. In IBL, examples can be redundant in the sense that they don't change the nearest neighbor classification of any query. More generally (and less stringently), one might consider a set of examples redundant if they are closely neighbored in the instance space and, hence, have a similar region of influence.
- Consistency: An example might be removed if it seems to be inconsistent with the current concept, e.g., if its own class label differs from those labels in its neighborhood.

Many algorithms use only one indicator, either temporal relevance (e.g. window-based approaches), spatial relevance (e.g. LWF), or consistency (e.g. IB3). A few methods also use a second indicator, e.g. the approach of Klinkenberg (temporal relevance and consistency), but only the window-based system FLORA4 uses all three aspects.

3.2 IBL-DS

In this section, we describe the main ideas of IBL-DS, our approach to IBL on data streams, that not only takes all of the aforementioned three indicators into account but also meets the efficiency requirements of the data stream setting.

IBL-DS optimizes the composition and size of the case base autonomously. On arrival of a new example $\langle x_0, \lambda_{x_0} \rangle$, this example is first added to the case base. Moreover, it is checked whether other examples might be removed, either since they have become redundant or since they are outliers (noisy data). To this end, a set C of examples within a neighborhood of x_0 are considered as candidates. This neighborhood is given by the k_{cand} nearest neighbors of x_0, and the candidate set C consists of the 50% oldest examples within that neighborhood. The 50% most recent examples are excluded from removal due to the difficulty to distinguish potentially noisy data from the beginning of a concept change. Even though unexpected observations will be made in both cases, noise and concept change, these observations should be removed only in the former but not in the latter case.

If the current class λ_{x_0} is the most frequent one within a larger test environment of size[3] $k_{test} = (k_{cand})^2 + k_{cand}$, those candidates in C are removed that have a different class label. Furthermore, to guarantee an upper bound on the size of the case base, the oldest element of the similarity environment is deleted, regardless of its class, whenever the upper bound would be exceeded by adding the new example.

Using this strategy, the algorithm is able to adapt to concept drift but will also have a high accuracy for non-drifting data streams. Still, these two situations –drifting and stable concept– are to some extent conflicting with regard to the size of the case base: If the concept to be learned is stable, classification accuracy will increase with the size of the case base. On the other hand, a large case base turns out to be disadvantageous in situations where concept drift occurs, and even more in the case of concept shift. In fact, the larger the case base is, the more outdated examples will have to be removed and, hence, the more sluggish the adaptation process will be.

For this reason, we try to detect an abrupt change of the concept using a statistical test as in [14, 15]. If a corresponding change has been detected, a large number of examples will be removed instantaneously from the case base. The test is performed as follows: We maintain the prediction error p and standard deviation $s = \sqrt{\frac{p(1-p)}{100}}$ for the last 100 training instances. Let p_{min} denote the smallest among these errors and s_{min} the associated standard deviation. A change is detected if the current value of p is significantly higher than p_{min}. Here, statistical significance is tested using a standard (one-sided) z-test, i.e., the condition to be tested is $p + s > p_{min} + z_\alpha s_{min}$, where α is the level of confidence (we use $\alpha = 0.999$).

[3] This choice of k_{test} aims at including in the test environment the similarity environments of all examples in the similarity environment of x_0; of course, it does not guarantee to do so.

Finally, in case a change has been detected, we try to estimate its extent in order to determine the number of examples that need to be removed. More specifically, we delete p_{dif} percent of the current examples, where p_{dif} is the difference between p_{min} and the classification error for the last 20 instances. Examples to be removed are chosen at random according to a distribution which is spatially uniform but temporally skewed (see below).

IBL-DS is implemented under the data mining library WEKA [31]. The data is stored in the M-tree data structure of XXL, a query processing library developed and maintained at the Informatics Institute of Marburg University [6]. Below, we describe the distance function employed by IBL-DS and the M-Tree [7] which allows for processing streams with both continuous and categorical attributes and, moreover, to perform nearest neighbor queries in an efficient way even for very large case memories.

As a distance function we use an updateable variant of SVDM which is a simplified version of the VDM distance measure [26] and was successfully used in the classification algorithm RISE [11]. Let an instance x be specified in terms of ℓ features $F_1 \dots F_\ell$, i.e., as a vector $x = (f_1 \dots f_\ell) \in D_1 \times \dots \times D_\ell$. Numerical features F_i with domain $D_i = \mathbb{R}$ are first normalized by the mapping $f_i \mapsto f_i/(max - min)$, where max and min denote, respectively, the largest and smallest value for F_i observed so far; these values are permanently updated. Then, $\delta_i(f_i, f_i')$ is defined by the distance between the normalized values of f_i and f_i'. For a discrete attribute F_j, the distance between two values f_j and f_j' is defined by the following measure:

$$\delta_i(f_j, f_j') = \sum_{k=1}^{m} \left\| P(\lambda_k \mid F_j = f_j) - P(\lambda_k \mid F_j = f_j') \right\| ,$$

where m is the number of classes and $P(\lambda \mid F = f)$ is the probability of the class λ given the value f for attribute F. Finally, the distance between two instances x and x' is given by the mean squared distance

$$\Delta(x, x') \;=\; \frac{1}{\ell} \sum_{i=1}^{\ell} \delta_i(f_i, f_i')^2$$

To delete instances in a spatially uniform but temporally skewed way, we exploit the properties of the M-Tree index structure [7]. In this tree, the leaves store instances that belong to a small sphere within the instance space. The inner nodes combine subnodes to bigger spheres and the root node represents the sphere that corresponds to the whole data set. Each node n consists of a center instance c_n and an associated radius r_n. Moreover, each node maintains a list of successors (subnodes) l_n. The number of instances or subnodes of a node is restricted to an interval $[minCapacity, maxCapacity]$. Our experience has shown that the interval $[6, 15]$ yields good performances. The stored examples correspond to the instance nodes of the tree (located directly under the leaf nodes), the radius of which is 0.

To delete data with preference to older instances, the number of items to be removed in a node is uniformly spread among the subnodes. In a leaf, only the oldest instances are removed. This way, we ensure that the spatial distribution of the deleted instances is uniform in the instance space. Regarding the temporal distribution, however, old instances are more likely to be removed than more recent examples.

Finally, in order to classify a new query instance x_0, we employ a simple majority voting procedure among the k nearest neighbors. As in standard IBL, the computationally most expensive step consists of finding the query's neighbors. In our implementation, this step is again supported by the aforementioned M-tree (more specifically, the nearest neighbors are computed in an iterative way using a (min-)heap H of nodes which is initialized with the root of the M-tree).

4 Empirical Evaluation

A convincing experimental validation of online learning algorithms is an intricate problem for several reasons. Firstly, the evaluation of algorithms in a streaming context is obviously more difficult than the evaluation for static data sets, mainly because simple, one-dimensional performance measures such as classification accuracy will now vary over time and, hence, turn into functions (of time) which are not immediately comparable. Besides, additional criteria become relevant, such as the handling of concept drift, many of which are rather vague and hard to quantify. Secondly, real-world and benchmark streaming data is currently not available in a form that is suitable for conducting systematic experiments.

Due to these reasons, we mainly used synthetic data for our experiments, which allows for conducting experiments in a *controlled* way. Besides, a further experimental study using real-world data is presented in section 4.3.

We compared IBL-DS with the following instance-based approaches: The simple sliding window approach with fixed window-sizes of 200, 400, 800, respectively (Win200, Win400, Win800); Local Weighed Forgetting with $\beta = 0.04$ and $\beta = 0.02$ (LWF04, LWF02); Time Weighed Forgetting with $w = 0.996$ and $w = 0.998$ (TWF996, TWF998).

For IBL-DS we used the parameters $k_{cand} = 5$ and a maximal size of 5,000 examples for the case memory. For nearest neighbor classification, the neighborhood size was set to $k = 5$ for all algorithms.[4] In order to show the flexibility of IBL-DS, we employed synthetic data with quite different characteristics (see below).

4.1 Performance Measures and Data Sets

The learning scenario we considered is a straightforward extension of supervised learning to the data stream setting: At each point of time a new instance

[4] Note that the primary purpose of our studies is to compare the algorithms under equal conditions. This is why we used a fixed neighborhood size instead of optimizing this parameter.

x_0 arrives (from the query stream) and its class label λ_{x_0} is predicted. After the prediction has been made, the correct label is provided by a teacher, the prediction is evaluated, and the case base is updated (i.e., the new example is submitted to the example stream).

All data streams were tested with 50,000 elements, using an initial training set of 100 examples and adding 5% random noise. We derived two types of classification rate: (i) The *streaming* classification rate measures the accuracy on the last 100 instances of the stream. Thus, it is a kind of real accuracy that refers to a certain section of the stream. (ii) The *absolute* classification rate aims at estimating the accuracy at a particular moment of time. To this end, 1,000 extra test instances are generated at random according to a uniform distribution, and the classification accuracy for this test set is derived by using the current case base; this is done for every 10 time points.

We conducted experiments with 8 different data streams (see table 1), some of which have already been used in the literature before: The streams GAUSS, SINE2, STAGGER and MIXED were used in [14], and the HYPERPLANE data (that we generated for the dimensions $d = 2$ and $d = 5$) was used in multiple experiments for data streams in [28]. Besides, we used the following data streams:

DISTRIB: Instances are uniformly distributed in the unit square $[0, 1] \times [0, 1]$. An instance (x, y) belongs to class 1 if $(x, y) \in [0, 0.5] \times [0, 0.5]$ or $(x, y) \in]0.5, 1] \times]0.5, 1]$, otherwise to class 0. Even though the concept remains fixed, the underlying distribution does change: the data is only generated in one quarter of the instance space, changing the quarter clockwise every 2,000 instances.

RANDOM: Instances are uniformly distributed in the unit square $[0, 1] \times [0, 1]$. An instance (x, y) belongs to class 1 with probability p, independently of x and y. Within an interval of 2,000 examples, the probability p increases linearly from 0 to 1, then decreases linearly from 1 to 0 during the next interval of the same length, and so on.

MEANS: The n classes are defined by n center points in $[0, 1]^d$. An instance belongs to the class of the nearest center. Each center moves with a fixed drift for each dimension. If it leaves the unit interval in one dimension, the drift for this dimension is inverted. We have made experiments with $n = 5$ and $d \in \{2, 5\}$. For each dimension, the drift is initialized by a random value in $[-(1/8)^{-3}, (1/8)^{-3}]$.

Table 1. Properties of data streams.

	attributes	classes	drift/shift
GAUSS	2 num	2	shift
SINE2	2 num	2	shift
DISTRIB	2 num	2	virtual shift
RANDOM	2 num	2	drift (distr.)
STAGGER	3 discr	2	shift
MIXED	2 num + 2 discr	2	shift
HYPER	2/5 num	2	drift
MEANS	2/5 num	5	drift

4.2 Results

The absolute and streaming classification rates are shown, respectively, in tables 2 and 3. For the data streams GAUSS, SINE2, RANDOM and MIXED, our method IBL-DS shows the best performance regardless of the type of measure; for DISTRIB it performs best in terms of the absolute classification rate. Even if IBL-DS is not the best method for the remaining streams (STAGGER, HYPER, and MEAN), its results are always competitive and close to optimal.

Apparently, IBL-DS performs comparatively well especially for the data streams with concept shift. Thus, our strategy for handling such situations, including a flexible size of the case base, seems to work in practice. In fact, some other methods do obviously have difficulties with abrupt changes of the concept, as suggested by their relatively poor classification rates. Note that concept shift does also occur in STAGGER. Here, however, only 12 different instances exist, so a small case base is always sufficient. In fact, it is not useful to store all examples that support the current concept; this only makes the model less flexible with regard to the next concept shift but does not lead to a higher accuracy.

Table 2. Absolute classification rates.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	**.843**	.805	.837	.834	.804	.734	.750	.693
SINE2	**.919**	.863	.898	.896	.868	.788	.838	.762
DISTRIB	**.948**	.913	.888	.504	.508	.514	.500	.505
RANDOM	**.723**	.706	.718	.712	.704	.674	.655	.625
STAGGER	.956	.806	.806	**.978**	.962	.917	.916	.908
MIXED	**.906**	.713	.707	.898	.870	.790	.840	.765
HYPER2	.969	**.970**	.965	.959	.965	.963	.923	.919
HYPER5	**.904**	.892	.896	.876	.886	.880	.839	.834
MEANS2	.944	**.950**	.935	.918	.939	.946	.913	.914
MEANS5	.809	**.828**	.796	.736	.778	.810	.735	.763

For the DISTRIB data, the extreme differences between absolute and streaming classification rate call for explanation. To understand these differences recall the special distribution of the training data: After a shift of this distribution, it takes 6,000 time steps (instances) until the next instance for the previous quarter will arrive. All window-based approaches will soon forget all the data of this quarter. Only the LWF algorithm stores all the data the whole time. IBL-DS will have the highest accuracies after training instances have been seen in all quarters (viz. after 6,000 instances). Before, LWF performs slightly better, since this algorithm does not delete as many of the 100 examples used for initialization.

Table 3. Streaming classification rates.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	**.807**	.772	.801	.798	.771	.707	.724	.668
SINE2	**.878**	.827	.858	.857	.833	.758	.804	.738
DISTRIB	.939	.943	**.945**	.943	.940	.937	.900	.899
RANDOM	**.724**	.706	.721	.714	.706	.673	.654	.621
STAGGER	.909	.770	.770	**.929**	.914	.875	.872	.866
MIXED	**.865**	.691	.685	.856	.831	.758	.804	.738
HYPER2	.921	**.922**	.918	.913	.918	.916	.879	.875
HYPER5	**.865**	.855	.858	.839	.849	.844	.807	.804
MEANS2	.908	**.914**	.899	.885	.903	.910	.881	.881
MEANS5	.780	**.800**	.770	.714	.751	.783	.710	.740

The simple window-based algorithm shows a very good performance for the HYPERPLANE and the MEANS data. Again, there is a simple explanation for this result: These two data streams have a small concept drift rate which does hardly change over time. Therefore, the optimal size of the case base will remain more or less constant as well. Since training data is furthermore uniformly distributed, using a window of fixed size is indeed a suitable strategy. Again, however, note that the classification rate of IBL-DS is not much worse.

4.3 Real World Data

So far, only synthetic data sets have been used. Even though synthetic data allows one to model special effects and, hence, is advantageous from this point of view, conducting experiments with real-word data is of course also desirable. As already mentioned above, however, real-world data streams are hard to obtain. To overcome this problem, at least to some extent, we decided to use (static) benchmark data sets from the UCI repository and to prepare them as data streams.

In this regard, note that any data set, provided it is not too small, can be considered as a stream, simply by imposing an arbitrary ordering of the data items. More specifically, however, since a data stream is by definition an open-ended sequence, an ordered data set can at best be considered as a *section* of a stream. Besides, concept drift will usually not be present in streams of that kind. To simulate concept drift we did the following, inspired by [23]: First, the data set is put into a random order. Then, the most relevant input feature is identified using the "Correlation-based Feature Subset Selection" method implemented in WEKA, and the data is sorted according to the value of that attribute. Finally, the attribute itself is deleted from the data, thereby becoming a "hidden factor". This way, a kind of concept shift is obtained in the case of discrete attributes, whereas numerical attributes will produce gradual concept drift.

To qualify as a (pseudo-)stream, a data set should first of all not be too small. Moreover, a useful data set will have a reasonable number of classes and moreover, should not contain features that are highly correlated with the attribute

used to simulate concept drift. These requirements are satisfied by only a few UCI data sets. For our studies, we selected the Balance, Car and Nursery data. Table 4 summarizes the main properties of these data sets.

Table 4. UCI data sets prepared as (pseudo-)streams.

data set	#instances	#classes	#attributes	selected attribute
Balance	625	3	4	right-distance (numeric)
Car	1728	4	6	safety(nominal)
Nursery	11025	5	8	health (nominal)

IBL-DS was employed in its default parameter setting. For LWF the parameters $\beta = .04$ and $\beta = .1$ are used, TWF is run with $weight = .99$ and $weight = .995$, and the fixed sliding window approach (Win) with $size = 50, 100, 200$. To show that concept drift does really occur, the standard instance based algorithm (IBL) that simply stores all instances is additionally applied.

The results are presented in table 5. As can be seen, IBL-DS again performs very well, even for these different types of data streams, without the need to change its parameters (apart from the case base size). Moreover, the standard instance-based algorithm clearly drops off and cannot compete, probably due to the simulated drift of the concept.

Table 5. Streaming classification rates for UCI data.

	Balance	Car	Nursery
IBL-DS	.805	.889	.900
LWF04	.782	.700	.899
LWF10	.846	.700	.842
Win50	.813	.819	.827
Win100	.815	.862	.856
Win200	.785	.888	.878
TWF99	.795	.889	.877
TWF995	.787	.887	.900
IBL	.693	.745	.839

5 Summary and Conclusions

We have presented an instance-based adaptive classification algorithm for learning on data streams. This algorithm, called IBL-DS, has a number of desirable

properties that are not, at least not as a whole, shared by existing alternative methods. Our experiments suggest that IBL-DS is very flexible and thus able to adapt to an evolving environment quickly, a point of utmost importance in the data stream context. In particular, two specially designed editing strategies are used in combination in order to successfully deal with both gradual concept drift and abrupt concept shift. Besides, IBL-DS is relatively robust and produces good results when being used in a default setting for its parameters.

The JAVA implementation of IBL-DS is available for experimental purposes and can be downloaded, along with a documentation, under the following address: wwwiti.cs.uni-magdeburg.de/iti_dke.

There are various directions for further research. For example, techniques for model (case base) maintenance and adaptation like the one proposed in [19] are quite interesting, since they are less heuristic than those currently employed in IBL-DS. Even though it is not immediately clear how such techniques can be used in a streaming application with tight time and resource constraints, investigating such approaches in more detail and trying to adapt them correspondingly seems worthwhile.

References

1. CC. Aggarwal, J. Han, J. Wang, and PS. Yu. A framework for clustering evolving data streams. In *Proc. VLDB, Int. Conf. on Very Large Data Bases*, Berlin, Germany, 2003.
2. D.W. Aha, editor. *Lazy Learning.* Kluwer Academic Publ., 1997.
3. D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, Madison, Wisconsin*, pages 1–16. ACM, 2002.
5. S. Ben-David, J. Gehrke, and D. Kifer. Detecting change in data streams. In *Proc. VLDB–04*, 2004.
6. J. Bercken, B. Blohsfeld, J. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL - a library approach to supporting effcient implementations of advanced database queries. In *Proceedings of the VLDB*, pages 39–48, 2001.
7. Paolo Ciaccia, Marco Patella, Fausto Rabitti, and Pavel Zezula. Indexing metric spaces with M-tree. In Proc. SEBD'97, pages 67–86, Verona, Italy, June 1997.
8. G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proc. 22nd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, pages 296–306. ACM Press, 2003.
9. B.V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, California, 1991.
10. Mayur Datar and S.Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Algorithms - ESA 2002*, pages 323–334. Springer, 2002.
11. P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168, 1996.
12. P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12, 2003.

13. MM. Gaber and A. Zaslavsky amd S. Krishnaswamy. Mining data streams: A review. *ACM SIGMOD Record*, 34(1), 2005.
14. Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Proc. SBIA–04*, pages 286–295, 2004.
15. Joao Gama, Pedro Medas, and Pedro Rodrigues. Learning decision trees from dynamic data streams. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 573–577, New York, NY, USA, 2005. ACM Press.
16. L. Golab and M. Tamer. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.
17. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM Press, 2001.
18. E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 102–111, Edmonton, Alberta, Canada, July 2002.
19. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proc. ICML, 17th Int. Conf. on Machine Learning*, pages 487–494, San Francisco, CA, 2000.
20. Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, 8(3):281–300, 2004.
21. J.Z. Kolter and M.A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. Technical Report CSTR-20030610-3, Department of Computer Science, Georgetown University, Washington, DC, June 2003.
22. M. Kubat and G. Widmer. Adapting to drift in continuous domains. *Lecture Notes in Computer Science*, 912:307ff., 1995.
23. YN. Law and C. Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In *Proc. PKDD–05, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Porto, 2005.
24. Elizabeth McKenna and Barry Smyth. Competence-guided editing methods for lazy learning. In *ECAI*, pages 60–64, 2000.
25. Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artif. Intell. Rev.*, 11(1-5):133–155, 1997.
26. C. Stanfil and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29:1213–1228, 1986.
27. Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, Department of Computer Science, Trinity College Dublin, Ireland, 2004.
28. Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235. ACM Press, 2003.
29. Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, volume 667, pages 227–243. Springer-Verlag, 1993.
30. Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, 23(1):69–101, 1996.
31. IH. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.