Efficient Instance-Based Learning on Data Streams*

Jürgen Beringer

Department of Computer Science

Magdeburg University, Germany

beringer@iti.cs.uni-magdeburg.de

Eyke Hüllermeier

Department of Mathematics and Computer Science

Marburg University, Germany

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,

*Draft of a paper to appear in "Intelligent Data Analysis".

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.

Keywords: data streams, classification, instance-based learning, concept drift.

1 Introduction

In recent years, so-called *data streams* have attracted considerable attention in different fields of computer science, such as database systems, data mining, and distributed systems. 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 [31, 29, 13]. There are various applications in which streams of this type are produced, such as network monitoring, telecommunication systems, customer click streams, stock markets, or any type of multi-sensor system.

A data stream system may constantly produce huge amounts of data. To illustrate, imagine a multi-sensor system with 10,000 sensors, each of which sends a measurement once per second. Regarding aspects of data storage, management and processing, the continuous arrival of data items in multiple, rapid, time-varying, and potentially unbounded streams raises new challenges and research problems. Indeed, it is usually not feasible to simply store the arriving data in a traditional database management system in order to perform operations on that data later on. Rather, stream data must generally be processed in an online manner in order to guarantee that results are up-to-date and that queries can be answered with small time delay. The development of corresponding *stream processing systems* is a topic of active research [8].

Apart from data processing issues, the problem of mining data streams has been studied in a number of recent publications (see e.g. [1, 25] for up-to-date overviews).

In this connection, different data mining problems have already been considered, such as clustering [2], classification [22, 23, 24, 21, 34, 42], and frequent pattern mining [11]. 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. Related work is subsequently reviewed in section 3. Our approach to instance-based learning on data streams is introduced in section 4 and empirically evaluated in section 5. The paper concludes with a brief summary in section 6.

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 the following ways [5]:

(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 is 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 they are 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 [10].

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*. That is, an update of the current model or data mining results, necessitated by newly observed data, must not refer to old observations.

Apart from being incremental, 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 [6]. It has not only been considered in this context, however. In general, the literature distinguishes between different types of concept change over time [51]. 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.[1] In this scenario, old data might still be relevant, at least to some extent. Finally, one often speaks about *virtual* concept drift if only the distribution of the underlying data generating process changes while the concept itself remains unchanged [54]. To guarantee optimal predictive performance, an adaptation might also be necessary in such cases. 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,

[1] Admittedly, this distinction between shift and drift is rather vague. Yet, it will be sufficient for our purposes.

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 4 below.)

2.2 Instance-Based Learning

The term instance-based learning (IBL) stands for a family of machine learning algorithms, including well-known variants such as memory-based learning, exemplar-based learning and case-based learning [49, 47, 38]. As the term suggests, in instance-based algorithms special importance is attached to the concept of an *instance* [4]. An instance or exemplar can be thought of as a single experience, such as a pattern (along with its classification) in pattern recognition or a problem (along with a solution) in case-based reasoning.

As opposed to model-based machine learning methods which induce a general model (theory) from the data and use that model for further reasoning, 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]. 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 [14]. 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 $\{\lambda_1 \dots \lambda_m\}$.

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*.[2]

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*:

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

2.3 IBL in a Data Stream Context

Recall the aforementioned key requirements for learning and data mining algorithms on data streams: Above all, such algorithms must be incremental, highly adaptive, and they must be able to deal with concepts that may change over time. Is lazy, instance-based learning preferable to eager, model-based learning under these conditions? Unfortunately, this question cannot be answered unequivocally.

[2]We will use these terms interchangeably throughout the paper.

Obviously, IBL algorithms are inherently incremental, since adaptation basically comes down to adding or removing observed cases. Thus, incremental learning and model adaptation is simple and cheap in the case of IBL. As opposed to this, incremental learning is much more difficult to realize for most model-based approaches. Even though incremental versions do exist for a number of well-known learning methods, such as decision tree induction [52], the incremental update of a model is often quite complex and in many cases assumes the storage of a considerable amount of additional information.

The training efficiency of lazy learners does not come for free, however. Compared with model-based approaches, IBL has higher computational costs when it comes to answering new queries. In fact, the latter requires finding the k nearest neighbors of the query, and even though this retrieval step can be supported by efficient data and indexing structures, it remains costly in comparison with deriving a model-based prediction.

Consequently, IBL might be preferable 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. On the other hand, if queries must be answered frequently and under tight time constraints, whereas a need for updating the model due to newly observed examples rarely occurs, a model-based method might be the better choice. Thus, assuming an architecture as shown in Fig. 1, including an *example stream* that permanently produces new cases in the form of labeled instances and a *query stream* of unlabeled instances, the question is which of these streams has a higher rate.[3]

Regarding the handling of concept drift, a definite answer cannot be given either. Appropriately reacting to concept drift requires, apart from its discovery, flexible updating and adaptation strategies. In instance-based learning, model adaptation basically comes down to editing the case base, that is, adding new and/or deleting

[3]For our classification problem, it is in principle irrelevant whether examples and queries are produced, respectively, by a single stream or several parallel ones. Moreover, note that the example and query stream are not necessarily independent: correctly classified queries might be submitted as input to the example stream.

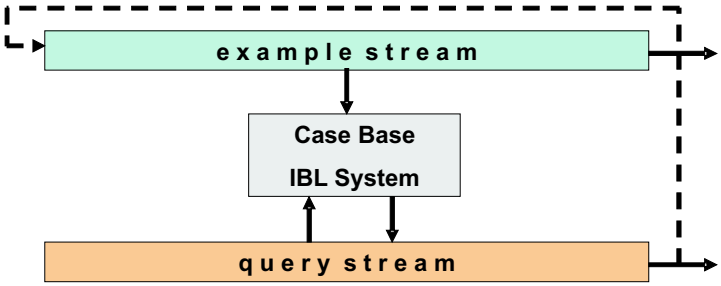

Figure 1: Architecture of the streaming classification scenario.

old examples. Whether or not this can be done more efficiently than adapting an other type of model, such as a classification tree or a neural network, does of course strongly depend on the particular model at hand. In any case, maintaining an implicit concept description by storing observations, as it is done by IBL, facilitates "forgetting" examples that seem to be outdated. In fact, such examples can simply be removed, while retracting the influence of outdated examples is usually more difficult in model-based approaches. In a neural network, for example, a new observation causes an update of the network weights, and this influence on the network cannot simply be cancelled later on.

3 Related Work

Data mining on streams [20] 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 (e.g. [15, 57]). Likewise, several online data mining methods have been devised (e.g. [16, 45, 11, 30, 26]), with a particular focus on unsupervised techniques like clustering [32, 19, 44]. 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 [55]. The corresponding algorithm learns rule-based binary classifiers on a sliding window of fixed size. It dynamically maintains three types of rules, namely rules that only

cover positive examples, rules that only cover negative examples, and rules of mixed type. The basic system has been improved in various enhancements. FLORA2 is able to adapt the window size. FLORA3 further improves adaptability by the ability to reactivate outdated rules. Finally, FLORA4 includes improved strategies for the handling of noise.

The FRANN (Floating Rough Approximation in Neural Networks) algorithm trains RBF networks on a sliding window of adaptive size [40]. An update of the model is initiated on the arrival of a fixed number of N new examples. The basis functions of the network are then derived from a subset of examples from the current sliding window; the corresponding selection process is guided by a hill climbing algorithm. The window size is adapted on the basis of the misclassification rate on the last M examples.

The LWF (Locally Weighted Forgetting) algorithm of Salganicoff [46] is one of 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 [46], 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 [36], 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. To this end, they divide the data into batches and successively (that is, for $k = 1, 2 \ldots$) test windows consisting of batches $t-k, t-k+1 \ldots t$. On each of these windows, they train a classifier (in this case a support vector machine) and estimate its predictive accuracy (by approximating the leave-one-out error). The window/model combination that yields the highest accuracy is finally selected. In [37], 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, Hulten and Domingos learn and maintain decision trees on a sliding window of fixed size [21, 34]. To achieve adaptability, they maintain statistics about the occurrence of attribute combinations and class labels as well as a list of alternative subtrees at inner nodes of the tree. If the estimated quality of an alternative subtree becomes significantly better than the quality of the current one, the two subtrees are exchanged. Due to the need to maintain statistics for attribute combinations, the algorithm can only handle discrete attributes with relatively small domains. This disadvantage is overcome by Ultra Fast Forest of Trees (UFFT), an extension of CVFDT that allows for handling numerical attributes [28] and also makes use of an improved splitting criterion for inner nodes. Moreover, each node of the UFFT decision tree maintains an appropriate statistic in order to detect concept drift. If drift is discovered, the whole subtree is forgotten and a new one is learned from scratch.

Another approach to adaptive learning is the use of ensemble techniques [53, 50, 39]. 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 [43]. 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 [4], 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 [55, 46].

Finally, we note that there is also a lot of work on time series data mining (e.g. [35]). However, even though time series data mining is of course related to stream data mining, one should not overlook some important differences between these fields. Particularly, time series are still static objects of fixed size that can, at least in principle, be analyzed offline, whereas the focus in the context of data streams is on dynamic adaptation and online data mining.

4 Instance-Based Learning on Data Streams

This section introduces our approach to instance-based learning on data streams, referred to as IBL-DS. As shown in Fig. 1, 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).[4]

Before presenting details of IBL-DS in sections 4.2–4.4, some general aspects and requirements of concept adaptation (case base maintenance) in a streaming context will be discussed in section 4.1.

4.1 Concept Adaptation

The simplest adaptive learners are those using sliding windows of fixed size. Since the update is very simple, these learners are also very fast. On the other hand, the assumption that the data which is currently relevant forms a fixed-sized window, i.e., that it consists of a *fixed* number of *consecutive* observations, is quite restrictive. In fact, 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

[4]Note that this problem is quite different from the task to maintain a synopsis which can be considered as a representative sample in a statistical sense, another problem that has recently received some attention in the data streams field [12]: A case base which is well suited for classification is not necessarily a representative sample.

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.[5] 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. This is the case, for example, if a concept drift only affects a part of the instance space. Besides, a more or less uniform coverage of the instance space is usually desirable, especially for local learning methods. 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. In other words, a new example in a region of the instance space already occupied by many other examples is considered less relevant than a new example in a sparsely covered region.

- Consistency: An example should 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

[5]Of course, this maxim disregards other criteria, such as the complexity of the method.

relevance and consistency), but only the window-based system FLORA4 uses all three aspects.

4.2 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 examples within that neighborhood. The 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 of the k_{cand} youngest examples in a larger test environment of size[6] $k_{test} = (k_{cand})^2 + k_{cand}$, those candidates in C are removed that have a different class label and do not belong to k_{cand} youngest examples in the larger test environment. 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

[6]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.

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 [27, 28]. 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 $\alpha = 0.999$).

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; the latter serves as an estimation of the current classification error.[7] Examples to be removed are chosen at random according to a distribution which is spatially uniform but temporally skewed (see section 4.3.2 for details). The complete updating algorithm is presented in Fig. 1. Regarding the above-mentioned indicators of usefulness, note that the aspect of temporal relevance is grabbed in lines 8 and 15–18, the aspect of spatial relevance in lines 13–14, and the aspect of consistency in lines 2–10 and 15–16.

[7]Note that, if this error, p, is estimated from the last k instances, the variance of this estimation is $\approx p(1-p)/k$. Moreover, the estimate is unbiased, provided that the error remained constant during the last k time steps. The value $k = 20$ provides a good trade-off between bias and precision.

Algorithm 1 IBL-DS-Update

Input: case base $\mathcal{D}$, example $e = \langle x_0, \lambda_{x_0} \rangle$
Output: updated case base $\mathcal{D}$

c = class estimate for x_0 derived from $\mathcal{D}$
compare c and λ_{x_0} and update statistics for the last 100 examples (error p and standard deviation s)
if $p + s < p_{min} + s_{min}$ **then**
  $p_{min} = p, s_{min} = s$
else if $p + s > p_{min} + z_\alpha s_{min}$ **then**
  p_{diff} = (error of the last 20 training data)$-p_{first}$
  if $p_{diff} > 0.2$ **then**
    delete $\min\{|\mathcal{D}|p_{diff}, |\mathcal{D}| - k_{test}\}$ cases in $\mathcal{D}$ (spatially uniform, temporally skewed)
    reset p_{min}, s_{min}
  end if
end if
repeat
  $S = \{e\} \cup k_{cand}$ nearest neighbors of e in $\mathcal{D}$
  $T = \{e\} \cup k_{test}$ nearest neighbors of e in $\mathcal{D}$
  if c is most frequent class among k_{cand} youngest instances of T **then**
    $\mathcal{D} = \mathcal{D} \setminus \{x \in S | x$ do not belong to c and to k_{cand} youngest instances of $T\}$
  else if $|\mathcal{D}|$ = maxSize **then**
    $\mathcal{D} = \mathcal{D} \setminus \{$oldest instance in $S\}$
  end if
until no further change of $\mathcal{D}$

4.3 Technical Details

IBL-DS is implemented under the data mining library WEKA [56]. This library contains various algorithms for supervised and unsupervised learning and additional tools for data pre- and post-processing. To realize learning in an adaptive way, the UpdateableClassifier interface of WEKA is used, which extends the Classifier Interface by the *updateClassifier(Instance inst)* method. 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 [7]. Below, we describe the distance function employed by IBL-DS and the M-Tree [9] 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.

4.3.1 Distance function

As a distance function we use an incremental variant of SVDM which is a simplified version of the VDM distance measure [49] and was successfully used in the classification algorithm RISE [17, 18]. 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.[8] Then, $\delta_i(f_i, f_i')$ is defined by the Euclidean 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:

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

[8] To make the transformation more robust toward outliers, it makes sense to replace max and min by appropriate percentiles of the empirical distribution.

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

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

4.3.2 Index structure M-Tree

To delete instances in a spatially uniform but temporally skewed way, we exploit the properties of the M-Tree index structure [9]. 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.

4.4 Classification of Queries

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. This heap is initialized with the root of the M-tree. The heap value of a node n, which is to be minimized, is given by the distance between n and the query instance x_0, which in turn is defined as $\max\{0, \|x_0 - c_n\| - r_n\}$. The next nearest neighbor is then obtained by calling the procedure shown in Algorithm 2.

Algorithm 2 getNextNeighbor(heap H)

```
1: while H is not empty do
2:    n = POP(H)
3:    if n is an instance node then
4:       return next neighbor c_n
5:    else
6:       H.insertAll(l_n)
7:    end if
8: end while
9: return: no further data
```

5 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. As an important advantage of synthetic data let us note that it allows for conducting experiments in a *controlled* way and, hence, to answer specific questions concerning the performance of a method and its behavior under particular conditions. Corre-

sponding experiments are presented in the following subsections. Besides, a further experimental study using real-world data is presented in section 5.7.

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). Note that, as opposed to IBL-DS, the parameters have a direct influence on the size of the case base for the above algorithms. The choice of these parameters has been optimized for the streams used in the experiments, where our first goal is to show that IBL-DS is competitive even under conditions which are optimal for the alternative algorithms. By changing the characteristics of the streams we will then show that IBL-DS is more stable in the sense of still producing good results for the same parameter configuration, whereas the performance of the other algorithms strongly deteriorates if the parameters are set in a suboptimal way.

As an aside, we note that a comparison with the well-known CVFDT approach and the FLORA-system is not reasonable. Apart from the fact that we focus on instance-based learning, these algorithms are not suitable for the settings considered in our experiments. CVFDT has been developed for extremely large data sets ("massive data") and, due to the use of Hoeffding trees, cannot handle data streams with a high drift rate. FLORA assumes discrete attributes and produces reasonable results only for the STAGGER stream (the only data that was used for evaluating the original FLORA-system). Even after discretizing the numerical attributes, FLORA's results were significantly inferior to those of the remaining algorithms.

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. 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, and $k = 5$ seems to be a reasonable choice. (For smaller values of k, there is a high sensibility toward noisy

data, whereas higher values decrease flexibility and necessitate a larger case base.) In order to show the flexibility of IBL-DS, we employed synthetic data with quite different characteristics (see section 5.2).

5.1 Performance Measures

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 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). We note that, even though this scenario is suitable for our purpose, it is probably not very realistic. In fact, in applications it will usually be difficult to realize a "fully supervised" process as outlined above. Instead, the correct label might be provided only for a subset of instances, perhaps on request and with a certain time delay.

All data streams were tested with 20,000 elements, using an initial training set of 100 examples and adding 5% random noise.[9] Each test were repeated 200 times. We derived two types of classification rate:[10] (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.

In addition, the following measures are monitored, again in steps of 10 time points: (i) The size of the case base. (ii) A *concept drift rate* as an indicator of the extent of

[9] That is, with a probability of .05, the correct label of an example is randomly exchanged with one of the other labels.

[10] These two accuracy measures will not differ very much unless the stream undergoes a concept drift.

concept drift. To this end, the labels of 10,000 uniformly distributed examples are compared with the labels 10 time points before, and the rate of change is computed by the fraction of examples that have changed their label. (This corresponds to the measure used in [33].)

5.2 Data Sets

We used 8 different data streams: GAUSS, SINE2, DISTRIB, RANDOM, STAGGER, MIXED, HYPERPLANE, MEANS. The STAGGER stream was presented in [48]. The streams GAUSS, SINE2, STAGGER and MIXED were used in [40, 27]. The HYPERPLANE data stream was used in multiple experiments for data streams [21, 53]. Table 1 gives a summary of some properties of these data streams.

GAUSS: This is a binary classification problem in $\mathbb{R} \times \mathbb{R}$ with two normally distributed classes, one with standard deviation 1 around $(0, 0)$ and the second with standard deviation 4 around $(2, 0)$. Abrupt concept shift is simulated by reversing the class membership every 2,000 instances.

SINE2: Instances are uniformly distributed in the unit square $[0, 1] \times [0, 1]$. Two classes are separated by the decision boundary $y = 0.5 + 0.3 \cdot \sin(3\pi x)$. Again, the classification is reversed every 2,000 instances.

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.

STAGGER: Instances are described by three symbolic attributes: size (small, medium, large), color (red, green), and shape (circular, non-circular). In the first context, instances satisfying (size = small) $\wedge$ (color = red) are classified as positive, all others as negative. In the second context, the concept description is (color = green) $\vee$ (shape = circular). In the third context, the description is (size = medium) $\vee$ large). The concept switches every 2,000 examples.

MIXED: The data are described by two boolean attributes v, w and two numerical attributes x, y with domain $[0, 1]$. The concept to be learned is a threshold concept: an example belongs to the concept if is satisfies at least two of the following three conditions: $v =$ `true`, $w =$ `true`, $y < 0.5 + 0.3 \cdot sin(3\pi x)$. The classification is reversed every 2,000 instances.

HYPERPLANE: The instance space $[0, 1]^d$ is separated into two classes by a moving hyperplane which is defined by the equation $\sum_{i=1}^{d} w_i \cdot x_i = w_0$. On arrival of a new instance, the weights w_i, $i = 0, 1 \ldots d$, are changed as follows: $w_i \leftarrow w_i + (1/4)^{-3} \cdot d \cdot f_i$, where $f_i \in \{-1, 1\}$ is randomly reselected every 2,000 instances. To ensure that both classes have equal size, the constant w_0 is always recomputed by $w_0 = 0.5 \sum_{i=1}^{d} w_i$. The w_i are initialized at random according to a uniform distribution on $[0, 1]$. We have conducted experiments for the dimensions $d = 2$ and $d = 5$.

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 $[-(1/8)^3, (1/8)^3]$.

5.3 Classification Rate

	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	2/5/15 num	5	drift

Table 1: Properties of data streams.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
Gauss	**.840**	.806	.838	.780	.755	.700	.758	.708
	(.0017)	(.0017)	(.0016)	(.0018)	(.0017)	(.0016)	(.0017)	(.0016)
Sine2	**.926**	.868	.901	.869	.842	.772	.843	.780
	(.0017)	(.0014)	(.0012)	(.0018)	(.0018)	(.0017)	(.0017)	(.0017)
Distrib	**.906**	.894	.877	.506	.511	.521	.512	.521
	(.0079)	(.0050)	(.0059)	(.0044)	(.0059)	(.0072)	(.0056)	(.0070)
Random	.708	.706	**.718**	.663	.655	.625	.658	.629
	(.0019)	(.0017)	(.0013)	(.0016)	(.0016)	(.0018)	(.0016)	(.0018)
Mixed	**.922**	.863	.895	.872	.845	.775	.846	.783
	(.0019)	(.0016)	(.0033)	(.0017)	(.0016)	(.0016)	(.0016)	(.0016)
Stagger	**.971**	.509	.509	.910	.888	.864	.897	.876
	(.0027)	(.0010)	(.0011)	(.0039)	(.0064)	(.0085)	(.0048)	(.0077)
Hyper2	.957	**.974**	.970	.924	.927	.924	.930	.927
	(.0056)	(.0092)	(.0049)	(.0042)	(.0079)	(.0149)	(.0077)	(.0145)
Hyper5	.865	.883	**.892**	.828	.835	.825	.836	.829
	(.0048)	(.0208)	(.0110)	(.0057)	(.0112)	(.0216)	(.0111)	(.0214)
Means2	.934	**.946**	.931	.894	.909	.910	.910	.912
	(.0044)	(.0043)	(.0047)	(.0042)	(.0041)	(.0062)	(.0041)	(.0060)
Means5	.794	**.830**	.799	.703	.737	.764	.738	.766
	(.0077)	(.0060)	(.0071)	(.0070)	(.0063)	(.0059)	(.0064)	(.0059)

Table 2: Absolute classification rates.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
Gauss	**.804**	.774	.803	.751	.728	.679	.731	.686
	(.0033)	(.0031)	(.0029)	(.0038)	(.0036)	(.0036)	(.0034)	(.0034)
Sine2	**.884**	.833	.862	.833	.809	.747	.810	.753
	(.0025)	(.0022)	(.0022)	(.0031)	(.0029)	(.0028)	(.0029)	(.0028)
Distrib	.931	.943	**.945**	.902	.901	.899	.903	.902
	(.0023)	(.0017)	(.0017)	(.0031)	(.0030)	(.0033)	(.0031)	(.0033)
Random	.709	.706	**.718**	.663	.654	.625	.658	.629
	(.0038)	(.0033)	(.0035)	(.0040)	(.0036)	(.0036)	(.0036)	(.0035)
Mixed	**.878**	.826	.855	.834	.809	.746	.810	.753
	(.0030)	(.0023)	(.0039)	(.0030)	(.0030)	(.0028)	(.0029)	(.0028)
Stagger	**.925**	.507	.507	.870	.850	.829	.858	.840
	(.0031)	(.0032)	(.0033)	(.0042)	(.0064)	(.0079)	(.0050)	(.0073)
Hyper2	.912	**.927**	.924	.882	.885	.882	.887	.885
	(.0057)	(.0086)	(.0051)	(.0049)	(.0078)	(.0138)	(.0076)	(.0135)
Hyper5	.828	.845	**.853**	.795	.801	.792	.803	.796
	(.0049)	(.0189)	(.0102)	(.0059)	(.0103)	(.0196)	(.0101)	(.0194)
Means2	.900	**.911**	.896	.862	.876	.877	.877	.879
	(.0050)	(.0046)	(.0051)	(.0048)	(.0049)	(.0069)	(.0050)	(.0067)
Means5	.766	**.800**	.771	.679	.712	.738	.712	.739
	(.0080)	(.0062)	(.0074)	(.0075)	(.0069)	(.0065)	(.0069)	(.0065)

Table 3: Streaming classification rates.

The absolute and streaming classification rates are shown, respectively, in tables 2 and 3. For the data streams Gauss, Sine2, Stagger 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 (Random, 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 well 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.

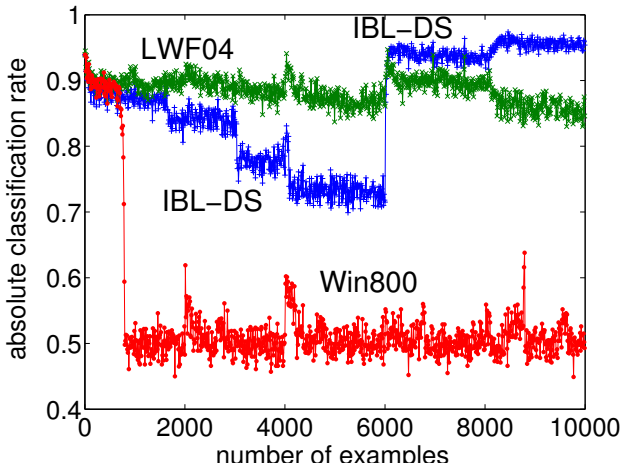

Figure 2: Absolute classification rate for the first 10,000 instances of the DISTRIB data stream.

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 (see figure 2).

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.

For comparison, table 4 shows the classification performance for a drift rate which is 40 times as high as the original one (technically, this was achieved by using only every 40-th example for training). Apart from DISTRIB, where no real concept drift exists,

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	**.609**	.515	.555	.502	.502	.502	.504	.504
SINE2	**.521**	.493	.488	.502	.502	.502	.503	.502
DISTRIB	.938	**.965**	.950	.909	.921	.929	.922	.932
RANDOM	**.568**	.501	.504	.500	.500	.500	.501	.501
STAGGER	**.697**	.477	.477	.567	.530	.568	.532	.604
MIXED	**.539**	.508	.513	.502	.502	.502	.504	.504
HYPER2	**.848**	.774	.812	.757	.716	.681	.720	.689
HYPER5	**.692**	.639	.668	.630	.599	.581	.603	.587
MEANS2	**.777**	.553	.683	.660	.507	.368	.518	.395
MEANS5	**.628**	.478	.600	.609	.505	.371	.514	.396

Table 4: Absolute classification rates for faster concept drift.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
absolute	.784	.835	.823	.722	.738	.753	.740	.755
rate	(.0092)	(.0068)	(.0074)	(.0138)	(.0127)	(.0117)	(.0126)	(.0116)
absolute rate	.687	.615	.674	.694	.651	.582	.656	.597
for faster drift	(.0096)	(.0157)	(.0147)	(.0056)	(.0126)	(.0186)	(.0127)	(.0182)

Table 5: Absolute classification rates for 15-dimensional MEANS data with two classes.

and STAGGER, with only 12 different instances and a high overlap of the different concepts to be learned, IBL-DS clearly outperforms the other algorithms on all data streams. The main reason for the poor performance of the other algorithms is the fact that the corresponding parameters, and therefore the size of the case bases, are not optimized for the high drift rate.

Since the above experiments were restricted to data streams of relatively low dimension, we conduced another experiment in which we used a 15-dimensional version of the MEANS data (with two classes). The results of this experiment are shown in table 5. As can be seen, with respect to the absolute classification rate, IBL-DS outperforms the Win and TWF approaches but is not as accurate as LWF. This changes, however, in the case of a (40 times) faster drift rate. Again, this is due to the flexibility of IBL-DS, which allows it to adapt to the drift rate in a very effective way.

5.4 Size of the Case Base

As already mentioned above, the strategies of IBL-DS for handling concept shift seem to work very well. To illustrate, consider figure 3 that shows the development of IBL-DS's absolute classification rate and the size of the case base between the 10,000th and 20,000th training example for the SINE2 data. In the second subfigure, the concept shift around the 12,000th training instance is shown. After each concept switch, the size of the case base decreases quickly, and a good classification rate is recovered after hardly more than 60 instances.

The case of gradual concept drift can be more intricate, however, as illustrated in figure 4 for the Means2 data: After a while, IBL-DS detects a small drift and deletes some data. According to our strategy, this data will be distributed uniformly across the complete input space, even though the concept drift may concern only a small part of this space. Consequently, many examples are then removed that are actually not involved in the drift. Nevertheless, a good performance of IBL-DS can still be guaranteed thanks to its second updating strategy. In fact, even though the adaptation induced by this strategy is much slower, each outdated example will sooner or later be deleted.

The only stream for which IBL-DS's case base reaches the maximum size of 5,000 is the DISTRIB data. Recall that the shift in this stream does not concern the concept itself but rather the distribution in the input space. Consequently, all examples except the noisy ones could in principle be memorized. In fact, the simple strategy of removing the oldest examples would not perform very well, because it involves the danger of completely loosing the examples of one quarter. In this case, our strategy of removing examples at random pays off, since it ensures that the deletion process does not focus on a small subregion of the data space only. As can be seen, both strategies for removing examples have their strengths and advantages, and in practice a combination of the two is likely to perform better than each of them alone, at least on average.

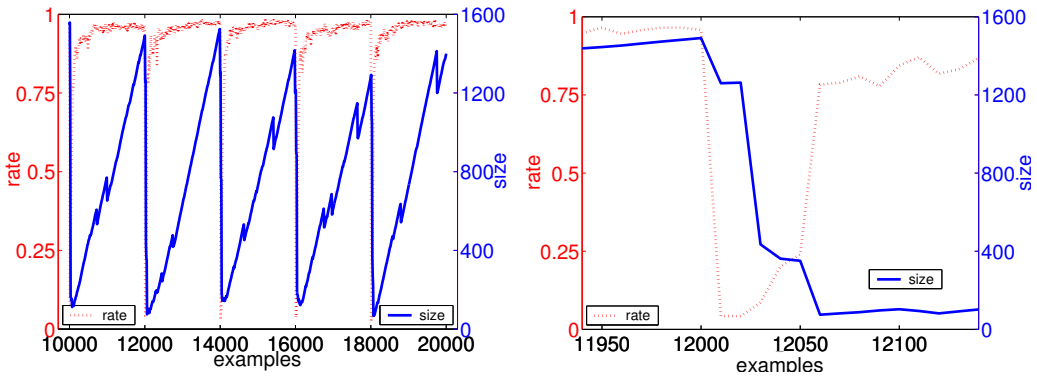

Figure 3: Absolute classification rate and size of case base for the SINE2 data.

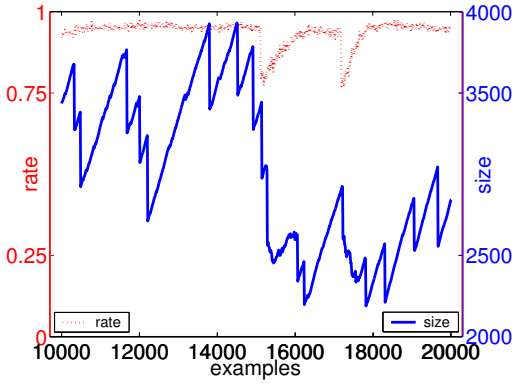

Figure 4: Absolute classification rate and size of case base for the Means2 data.

5.5 Time for Updating and Classification

In this section, we compare the running times for updating and classification. Tables 6 and 7 show, respectively, the average times needed for 1,000 updates of the case base and 1,000 classifications of new query instances. As expected, the more complex updating strategy used by IBL-DS comes along with slightly higher running times. Yet, being able to conduct 1,000 updates in less than 5 seconds, IBL-DS can still handle streams with very high arrival rates.

5.6 Parametrization and Robustness

Most learning methods can be parameterized in one way or the other, i.e., their behavior can be influenced by means of some user-defined parameters. On the one

	IBL-DS	LWF02	Win800	TWF998
Gauss	.56	.17	.033	.035
Sine2	.36	.12	.023	.024
Distrib	.49	.12	.027	.028
Random	.33	.12	.023	.024
Stagger	.63	.52	.050	.075
Mixed	.70	.30	.032	.066
Hyper2	.50	.12	.023	.025
Hyper5	4.3	.78	.048	.055
Means2	.47	.12	.023	.025
Means5	3.2	.76	.048	.054

Table 6: Running times (in seconds) for 1,000 updates.

	IBL-DS	LWF02	Win800	TWF998
Gauss	.17	.13	.13	.14
Sine2	.090	.087	.063	.067
Distrib	.13	.087	.079	.081
Random	.078	.088	.067	.066
Stagger	.16	.19	.078	.080
Mixed	.19	.23	.072	.24
Hyper2	.13	.087	.063	.067
Hyper5	1.3	.61	.37	.40
Means2	.12	.089	.066	.067
Means5	.92	.59	.37	.40

Table 7: Running times (in seconds) for 1,000 classifications.

hand, this offers the opportunity to adapt the method to a particular application in an optimal way, by "tuning" the parameters appropriately. On the other hand, having too many degrees of freedom might also be considered as a disadvantage, at least if a suboptimal specification of the parameters will badly deteriorate the algorithm's performance. In this regard, the robustness of the learning method toward parameter settings is a desirable property. Roughly speaking, this means that a slight variation of the parameter setting does not change the performance of the method drastically. Usually, this also implies that, even though a method might be optimized by appropriately tuning its parameters, it still performs well when being applied in a "default setting". The objective of this section is to show that IBL-DS is robust in this sense.

During the above experiments, all parameters of IBL-DS have been held fixed, i.e., a default setting has been used. In order to provide evidence for the robustness of IBL-DS toward small variations of the parameters, we have rerun the experiments with modified values for the following parameters: the size of the test environment (k_{test}); the number of young instances (k_{cand}); the threshold for detection of concept shift, referring to a change of the classification rate (Δp); the significance level of the t-test (α); the length of the history used to estimate the current classification rate (h); the number of examples used to estimate the new classification rate after recognition of a concept shift (u). The default values used for these parameters are summarized in table 8.

In each experiment, exactly one parameter has been modified while the default values have been used for the remaining ones. The absolute classification rates thus obtained are shown in tables 9 and 10, together with the result for the default setting. As can be seen, the classification rates are almost the same. Since the results for the streaming classification rate are very similar, we didn't present them here.

parameter	value
k_{test}	50
k_{cand}	5
Δp	0.2
$1 - \alpha$	0.999
h	100
u	20

Table 8: Default parameters in IBL-DS.

	default	$k_{test} = 30$	$k_{test} = 70$	$k_{cand} = 3$	$k_{cand} = 10$	$\alpha = .01$	$\alpha = .005$
Gauss	.840	.839	.836	.764	.837	.840	.840
Sine2	.926	.927	.921	.863	.926	.924	.924
Distrib	.906	.904	.902	.882	.910	.905	.906
Random	.708	.703	.711	.709	.704	.708	.708
Mixed	.922	.926	.912	.800	.920	.919	.919
Stagger	.971	.963	.962	.969	.944	.971	.971
Hyper2	.957	.950	.956	.959	.947	.955	.955
Hyper5	.865	.865	.868	.861	.860	.867	.867
Means2	.934	.937	.940	.752	.926	.939	.939
Means5	.794	.793	.794	.609	.794	.795	.795

Table 9: Absolute classification rates for varied parameter configurations.

5.7 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

	default	$\Delta p = .1$	$\Delta p = .4$	h=50	h=150	u=10	u=30
Gauss	.840	.840	.832	.840	.835	.840	.840
Sine2	.926	.924	.917	.925	.919	.925	.923
Distrib	.906	.883	.906	.905	.905	.906	.906
Random	.708	.710	.702	.709	.711	.708	.709
Mixed	.922	.920	.914	.922	.916	.919	.919
Stagger	.971	.967	.964	.960	.961	.959	.969
Hyper2	.957	.956	.954	.955	.954	.955	.955
Hyper5	.865	.875	.866	.873	.866	.867	.867
Means2	.934	.940	.939	.940	.939	.940	.939
Means5	.794	.772	.794	.758	.795	.794	.894

Table 10: Absolute classification rates for varied parameter configurations.

mentioned above, however, real-world data streams are hard to obtain, at least for the *classification setting* considered in this paper. To overcome this problem, at least to some extent, we generated "pseudo-streams" from real data in two different ways.

5.7.1 First Experimental Study

In a first experimental study, we used (static) benchmark data sets from the UCI repository and prepared them as data streams. In fact, 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 [41]: 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" or, say, a "hidden context". This way, a kind of concept shift is obtained in the case of discrete attributes, whereas numerical attributes will produce gradual concept drift. For example, if the hidden attribute is discrete and changes its value, it means that the observable data is generated in a new context. In fact, since the hidden attribute is relevant in the sense of having an influence on the output, the dependency between the observable inputs and the output will change.

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 11 summarizes the main properties of these data sets.

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)

Table 11: UCI data sets prepared as (pseudo-)streams.

	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

Table 12: Streaming classification rates for UCI data.

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 12. 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.

5.7.2 Second Experimental Study

In a second experiment, we generated (pseudo-)streams of data by recording the stock rates of seven car manufacturers (Volkswagen (VW), BMW, Daimler-Chrysler, Porsche, Peugeot, Toyota, General Motors) from 1.1.1990 to 30.9.2006. For every day, one data item was defined in terms of the relative change (in percent) of the

class	decreasing	unchanged	increasing
percent	37.3	23.8	38.9

Table 13: Class distribution in stock prediction experiment.

method	classification rate
IBL-DS	0.563
LWF02	0.569
LWF04	0.559
Win200	0.519
Win400	0.528
Win800	0.529
TWF996	0.520
TWF998	0.520
k-NN	0.539

Table 14: Classification rates for stock prediction experiment.

stock in comparison with the day before; thus, a total number of 4,088 examples was obtained. Based on these examples, we considered the following classification problem: Given the change of the other six stocks, decide whether the VW-stock is increasing (change $> 0.5\%$), unchanged ($-0.5\% \leq$ change $\leq 0.5\%$), or decreasing (change $< -0.5\%$); table 13 shows the distribution of the class attribute.

The classification results are given in table 14. To prove the existence of concept drift, we also included the standard k-NN classifier with $k = 5$ as a baseline. As can be seen, this method can indeed be improved, but not by those approaches using a simple time window. As a possible explanation for the poor performance of these approaches, note that the distribution of the (input) data is quite skewed and far from uniform; considerable changes of the stock rates occur rather rarely, but in case they do, there is usually a high influence on the other stocks. The results are much better for those approaches using a local (spatial) optimization of the case base. In any case, IBL-DS performs rather well and can also compete with LWF. Interestingly, the average size of the case base was 860 for LWF but only 377 for IBL-DS.

	LWF	TWF	Win	IBL-DS
flexible adaptation of case base size	−	−	−	+
limited case base size	±	+	+	+
control of processing time	−	−	−	+
case selection depends on temporal relevance	+	+	+	+
case selection depends on spatial relevance	+	−	−	+
case selection depends on consistency	−	−	−	+
fast adaptation to abrupt concept shift	−	−	−	+
good adaptation to gradual concept drift	+	+	+	+

Table 15: Qualitative comparison of learning algorithms.

5.8 Summary of Evaluation

As already mentioned above, the evaluation of learning algorithms for data streams is a delicate issue. Firstly, a good algorithm should have several desirable properties, some of which might even be conflicting: Concept shift should be recognized quickly and appropriate measures should be taken to react to such changes. Likewise, the model should be able to gradually adapted in the case of concept drift. To meet the strong requirements regarding processing time, the size of the model (case base) should be limited. In this connection, it seems reasonable to select or delete examples on the basis of several indicators like temporal relevance, spatial relevance, and consistency (even though these are of course not requirements per se). Secondly, some of the above criteria are hard to quantify. Table 15 therefore provides a qualitative evaluation of learning algorithms according to the above criteria, summarizing our experiences from the empirical studies.

6 Conclusions and Future Work

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 [36] are quite interesting, since they are less heuristic than those currently employed in IBL-DS. As mentioned previously, however, it is not immediately clear how such techniques can be used in a streaming application with tight time and resource constraints. Nevertheless, investigating such approaches in more detail and trying to adapt them correspondingly seems worthwhile. A second point concerns combining instance-based and model-based learning. We already mentioned that IBL and model-based learning have different advantages and disadvantages, and that their suitability for classification on data streams will strongly depend on characteristics of the streaming application. An interesting idea is to look at hybrid methods, such as the RISE algorithm [18], in order to combine the advantages of both types of approaches.

References

[1] CC. Aggarwal (editor). Data Streams: Models and Algorithms. Springer-Verlag, Berlin, 2006.

[2] 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.

[3] D.W. Aha, editor. *Lazy Learning.* Kluwer Academic Publ., 1997.

[4] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16. ACM, 2002.

[6] S. Ben-David, J. Gehrke, and D. Kifer. Detecting change in data streams. In *Proc. VLDB–04*, 2004.

[7] 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.

[8] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Proceedings CIDR–03: First Biennial Conference on Innovative Database Systems*, Asilomar, CA, 2003.

[9] Paolo Ciaccia, Marco Patella, Fausto Rabitti, and Pavel Zezula. Indexing metric spaces with M-tree. In Matteo Cristiani and Letizia Tanca, editors, *Atti del Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD'97)*, pages 67–86, Verona, Italy, June 1997.

[10] J. Considine, F. Li, G. Kollios, and JW. Byers. Approximate aggregation techniques for sensor databases. In *ICDE–04: 20th IEEE Int'l Conference on Data Engineering*, 2004.

[11] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 296–306. ACM Press, 2003.

[12] M. Dash and W. Ng. Efficient Reservoir Sampling for Transactional Data Streams. *Sixth IEEE International Conference on Data Mining (ICDM) - Workshops*, pages 662-666, Hong Kong, 2006.

[13] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 40–51. ACM Press, 2003.

[14] B.V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, California, 1991.

[15] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 2002)*, 2002.

[16] Mayur Datar and S.Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Algorithms - ESA 2002*, pages 323–334. Springer, 2002.

[17] P. Domingos. Rule induction and instance-based learning: A unified approach. In C.S. Mellish, editor, *Proceedings* IJCAI-95*, 14th International Joint Conference on Artificial Intelligence*, pages 1226–1232, Montreal, 1995. Morgan Kaufmann.

[18] P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168, 1996.

[19] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 106–113, Williamstown, MA, 2001.

[20] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12, 2003.

[21] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM Press, 2000.

[22] F. Ferrer-Troyano, JS. Aguilar-Ruiz, and JC. Riquelme. Data streams classification by incremental rule learning with parameterized generalization. In *Proceedings of the 2006 ACM Symposium on Applied computing*, pages 657–661, Dijon, France, 2006.

[23] F. Ferrer-Troyano, JS. Aguilar-Ruiz, and JC. Riquelme. Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005 ACM Symposium on Applied computing*, pages 568–572, Santa Fe, New Mexico, USA, 2005.

[24] F. Ferrer-Troyano, JS. Aguilar-Ruiz, and JC. Riquelme. Discovering decision rules from numerical data streams. In *Proceedings of the 2004 ACM Symposium on Applied computing*, pages 649–653, Nicosia, Cyprus, 2004.

[25] MM. Gaber and A. Zaslavsky amd S. Krishnaswamy. Mining data streams: A review. *ACM SIGMOD Record*, 34(1), 2005.

[26] MM. Gaber, S. Krishnaswamy, and A. Zaslavsky. Cost-efficient mining techniques for data streams. In *Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 109–114. Australian Computer Society, Inc., 2004.

[27] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Proc. SBIA–04*, pages 286–295, 2004.

[28] 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.

[29] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: you only get one look. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 635–635. ACM Press, 2002.

[30] C. Giannella, J. Han, J. Peia, X. Yan, and PS. Yu. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*. AAAI/MIT, 2003.

[31] L. Golab and M. Tamer. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[32] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[33] DP. Helmhold and PM. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14:27–45, 1994.

[34] Geoff Hulten, Laurie Spencer, and Pedro 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.

[35] 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.

[36] 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.

[37] 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.

[38] J.L. Kolodner. *Case-based Reasoning*. Morgan Kaufmann, San Mateo, 1993.

[39] 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.

[40] M. Kubat and G. Widmer. Adapting to drift in continuous domains. *Lecture Notes in Computer Science*, 912:307ff., 1995.

[41] 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.

[42] MA. Maloof and RS. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154:95–126, 2004.

[43] Elizabeth McKenna and Barry Smyth. Competence-guided editing methods for lazy learning. In *ECAI*, pages 60–64, 2000.

[44] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering, 2002.

[45] S. Papadimitriou, C. Faloutsos, and A. Brockwell. Adaptive, hands-off stream mining. In *29th International Conference on Very Large Data Bases*, 2003.

[46] Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artif. Intell. Rev.*, 11(1-5):133–155, 1997.

[47] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276, 1991.

[48] Jeffrey C. Schlimmer and Jr. Richard H. Granger. Incremental learning from noisy data. *Mach. Learn.*, 1(3):317–354, 1986.

[49] C. Stanfil and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29:1213–1228, 1986.

[50] K.O. Stanley. Learning concept drift with a committee of decision trees. Technical Report AI-TR-03-302, Department of Computer Science, University of Texas at Austin, USA, 2003.

[51] 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.

[52] P.E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[53] 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.

[54] 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.

[55] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, 23(1):69–101, 1996.

[56] IH. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.

[57] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proc. 28th VLDB Conference*, pages 358–369, Hong Kong, China, 2002.