IBLStreams: A System for Instance-Based Classification and Regression on Data Streams

Ammar Shaker and Eyke Hüllermeier
Department of Mathematics and Computer Science
Philipps-Universität Marburg
D-35032 Marburg, Germany

This is a draft version of paper to be published in the "Evolving Systems" journal.

Abstract

This paper presents an approach to learning on data streams called IBLStreams. More specifically, we introduce the main methodological concepts underlying this approach and discuss its implementation under the MOA software framework. IBLStreams is an instance-based algorithm that can be applied to classification and regression problems. In comparison to model-based methods for learning on data streams, it is conceptually simple. Moreover, as an algorithm for learning in dynamically evolving environments, it has a number of desirable properties that are not, at least not as a whole, shared by currently existing alternatives. Our experimental validation provides evidence for its flexibility and ability to adapt to changes of the environment quickly, a point of utmost importance in the data stream context. At the same time, IBLStreams turns out to be competitive to state-of-the-art methods in terms of prediction accuracy. Moreover, due to its robustness, it is applicable to streams with different characteristics.

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

1 Introduction

The idea of adaptive learning in dynamical environments has recently received increasing attention in different research communities, including the database and data mining community, in which it has been addressed under the notion of "learning from data streams" [17, 18], and the computational intelligence community, in which the notion of "evolving fuzzy systems" has been coined [5, 25, 4, 26]. Despite small differences regarding the basic assumptions and the technical setting, the emphasis of goals and performance criteria, and the focus on specific types of applications, the key motivation of these and related fields is the idea of a system that learns incrementally, and maybe even in real-time, on a continuous stream of data, and which is able to properly adapt itself to changes of environmental conditions or properties of the data-generating process. Systems with these properties have been developed for different machine learning and data mining problems, such as clustering [1], classification [21], and frequent pattern mining [10].

Domingos and Hulten [15] list a number of properties that an ideal stream mining system should exhibit, and suggest corresponding design decisions: the system uses only a limited amount of memory; the time to process a single record is short and ideally constant; the data is volatile and a single data record accessed only once; the model produced in an incremental way is equivalent to the model that would have been obtained through common batch learning (on all data records so far); the learning algorithm should react to concept drift (i.e., any change of the underlying data-generating process) in a proper way and maintain a model that always reflects the current concept.

Given the existence of a number of sophisticated and partly quite complicated methods for learning on data streams, it is surprising that one of the simplest approaches to machine learning, namely the instance-based (case-based) learning paradigm, has only received very little attention so far—all the more since the nearest neighbor estimation principle, which constitutes the core of this paradigm, is a standard method in machine learning, pattern recognition, and related fields.

In this paper, we elaborate on the potential of the instance-based approach to supervised learning within the context of data streams and propose an efficient instance-based learning algorithm for two important performance tasks, namely classification and regression. To this end, we build on our previous work [6], in which a basic version of our approach to classification was introduced. Besides, the problem of regression has already been addressed in [31], but only in a rather simple way (based on computing weighted averages). For the sake of

self-containedness, parts of these papers will be reproduced here.

The remainder of the paper is organized as follows: The next section recalls the basic ideas of instance-based learning, along with a short discussion of its possible advantages and disadvantages in a streaming context. Our approach to instance-based learning on data streams, IBLStreams, is introduced in Section 3. In Section 4, we provide some information about the MOA (Massive Online Analysis) framework for mining data streams, under which IBLStreams is implemented. Experimental results are presented in Section 5, prior to concluding the paper in Section 6.

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 [32, 30, 24]. As the term suggests, in instance-based algorithms special importance is attached to the concept of an *instance* [3]. 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 [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 [11]. 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 equipped with a distance measure $\Delta(\cdot)$, i.e., $\Delta(x, x')$ is the distance between instances $x, x' \in \mathcal{X}$. $\mathcal{Y}$ is the output space and $\langle x, y\rangle \in \mathcal{X} \times \mathcal{Y}$ is called a labeled instance, a case, or an example. In classification, $\mathcal{Y}$ is a finite (usually small) set comprised of m classes $\{\lambda_1, \ldots, \lambda_m\}$, whereas $\mathcal{Y} = \mathbb{R}$ in regression.

The current experience of the learning system is represented in terms of a set $\mathcal{D}$ of examples $\langle x_i, y_i\rangle$, $1 \leq i \leq n = |\mathcal{D}|$. 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 data seen so far. In case-based reasoning, it is also referred to as the *case base*.

Finally, suppose a novel instance $x_0 \in \mathcal{X}$ (a query) to be given. The NN principle prescribes to estimate the corresponding output y_0 by the output 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 y_0^{est} of y_0 is derived from the set $\mathcal{N}_k(x_0)$ of the k nearest neighbors of x_0. This prediction is expressed as an aggregation of the neighbors' output values, where the type of aggregation function used strongly depends on the type of prediction sought. In particular, the case of a discrete output (classification) must be distinguished from the case of a numeric output (regression).

2.1 Classification

In classification, a prediction is usually derived by means of a *majority vote*, i.e.,

$$y_0^{est} = \arg\max_{y \in \mathcal{Y}} \operatorname{card}\{x_i \in \mathcal{N}_k(x_0) \,|\, y_i = y\} \ . \tag{1}$$

Noting that y_0^{est} corresponds to the mode of the distribution on $\mathcal{Y}$ which is obtained by counting the frequencies of class labels in the neighborhood of x_0, this prediction can be justified as an empirical risk minimizer of the standard 0/1 loss function ($\ell(y_0, y_0^{est}) = 0$ if $y_0 = y_0^{est}$ and $= 1$ otherwise).

The estimation (1) can be generalized by weighting instances according to their distance from x_0:

$$y_0^{est} = \arg\max_{y \in \mathcal{Y}} \sum_{x_i \in \mathcal{N}_k(x_0) \,:\, y_i = y} w(x_i) \ , \tag{2}$$

with

$$w(x_i) = \frac{f(\Delta(x_i, x_0))}{\sum_{x_j \in \mathcal{N}_k(x_0)} f(\Delta(x_f, x_0))} \ . \tag{3}$$

Here, $f(\cdot)$ is a decreasing function $\mathbb{R}_+ \to \mathbb{R}_+$, which means that the smaller $\Delta(x_i, x_0)$, the higher the weight of y_i. Our implementation of IBL offers different types of (kernel) functions of this type (constant, linear, inverse, Gaussian, exponential), which can be selected by the user as a parameter of the system.

While the mode of a distribution is the minimizer of the expected 0/1 loss, it is not the optimal prediction for the absolute error loss $\ell(\lambda_i, \lambda_j) = |i - j|$, which is often used in the case of ordinal classification, where the classes are assumed to have a natural order $\lambda_1 < \lambda_2 < \ldots < \lambda_m$. Instead, the minimizer of this loss

function is given by the median of the distribution. Again, this prediction can be generalized by weighting each neighbor in accordance with its distance from x_0.

2.2 Regression

A common loss function in regression is the squared error loss $\ell(y_0, y_0^{est}) = (y_0 - y_0^{est})^2$, which suggests the mean of the neighbors' outputs as a prediction. More generally, one typically uses the weighted average

$$y_0^{est} = \sum_{x_i \in \mathcal{N}_k(x_0)} w(x_i) \cdot y_i \ . \tag{4}$$

Like in the case of classification, the basic assumption here is that the dependency to be learned is locally (i.e., in the neighborhood of x_0) constant, or can at least be approximated sufficiently well by a constant function. Relaxing this assumption from locally constant to locally linear gives rise to the idea of locally weighted linear regression. Here, a linear model

$$f(x) = \beta_0 + \sum_{j=1}^{m} \beta_j \cdot x[j] = \beta^T \begin{bmatrix} 1 \\ x \end{bmatrix} \ , \tag{5}$$

with $x[j]$ the j-th entry of the vector x, is fitted in the neighborhood of x_0. Thus, the vector of coefficients β is estimated by

$$\hat{\beta} = \left(X^T W X\right)^{-1} X^T W Y \ , \tag{6}$$

where the $k \times (m+1)$ matrix X is composed of the k neighbors $x_i \in \mathcal{N}_k(x_0)$ (plus the vector of ones modeling the intercept β_0) and Y is the $k \times 1$ vector of corresponding output values y_i. Moreover, W is a diagonal weight matrix $\mathrm{diag}(w_1, \ldots, w_k)$, which is determined by means of a kernel function $f(\cdot)$ centered at x_0; thus, the weight w_i of the neighbor $x_i \in \mathcal{N}_k(x_0)$ is of the form (3). In case $X^T W X$ is singular and its inverse does not exist, the weighted average is used for prediction instead. Situations of singularity or close-to-singularity, producing problems with numerical instability, may also occur if the weight vector $\mathrm{diag}(w_1, \ldots, w_k)$ is strongly dominated by a single entry. To prevent such problems, we take care that the kernel width in exponential or Gaussian weighting does never become too small.

Once (6) has been computed, the prediction y_0^{est} is obtained by evaluating (5) with $x = x_0$ and $\beta = \hat{\beta}$. The instance-based prediction strategy, both for classification and regression, is summarized in pseudo-code in Fig. 1.

It is worth mentioning that the learning (and combination) of locally linear models is also quite common in the field of evolving fuzzy systems, especially in learning so-called Takagi-Sugeno (TS) fuzzy models [33]; these are rule-based models in which the rule antecedents specify fuzzy predicates on the input attributes, while the rule consequents are linear functions of these attributes. From a modeling point of view, an important difference is that, while a linear function is fitted individually to each query $x_0 \in \mathcal{X}$ in instance-based learning, such a function refers to a larger part $X \subset \mathcal{X}$ of the input space (often identified through the use of clustering algorithms) in TS models. Besides, following the paradigm of lazy learning, the linear functions are only learned upon request in IBLStreams and immediately forgotten after a prediction has been made. As opposed to this, the combination of linear functions forms a global model in the case of TS fuzzy systems. This comparison gives rise to an interesting question, namely whether or not (lazy) instance-based learning might be preferable to (eager) model-based learning in the setting of data streams.

2.3 Model-based versus Instance-based Learning

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.

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 [34] and the learning of TS fuzzy models [25, 5], 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.

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 old examples. Whether or not this can be done more efficiently than adapting another type of models, such as a classification tree or a fuzzy system, does of course strongly depend on the particular model at hand. In any case, maintaining an implicit concept description by storing observations, as 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; at the best, it can be reduced gradually in the course of time.

3 Instance-Based Learning on Data Streams

This section introduces our approach to instance-based learning on data streams, referred to as IBLStreams. Our 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 IBLStreams, some general aspects and requirements of concept adaptation (case base maintenance) in a streaming context will be discussed.

3.1 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 win-

dow, 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 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.[1] 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,

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

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 output (strongly) differs from those in its neighborhood.

Many algorithms use only one indicator, either temporal relevance (e.g. window-based approaches), spatial relevance (e.g. Lightweight Frequency Counting, LWF [29]), or consistency (e.g. Instance Based learning algorithm 3, IB3 [3]). A few methods also use a second indicator, e.g., the approach of Klinkenberg (temporal relevance and consistency) [23], but only the window-based system FLORA4 (Floating Rough Approximation) [35] uses all three aspects.

3.2 IBLStreams

In this section, we describe the main ideas of IBLStreams, 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.

IBLStreams optimizes the composition and size of the case base autonomously. On arrival of a new example $\langle x_0, y_0 \rangle$, 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, determined according a distance measure Δ (see Appendix A), 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.

In the classification scenario, the most frequent class among the k_{cand} youngest examples in a larger test environment of size [2] $k_{test} = (k_{cand})^2 + k_{cand}$ is determined. If this class corresponds to the current class y_0, those candidates in C are removed that have a different class label and do not belong to k_{cand} youngest examples in

[2]The size of the test environment is quadratic as it is intended to cover the similarity environments of all examples in the similarity environment of x_0.

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.

In the regression scenario, the mode of the distribution is obviously unsuitable as a characterization of the "normal" output. Instead, the k_{cand} youngest examples in the neighborhood set C are used to determine a confidence interval $[\bar{y} - Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{k_{cand}}}, \bar{y} + Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{k_{cand}}}]$, where $\bar{y}$ is the average target value for the considered examples and σ the standard deviation; α is the significance level and chosen to be ≈ 0.001. A candidate case is then removed if it falls outside this confidence interval and is not one of the k_{cand} youngest instances in the larger test environment.

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 [19, 20]. If a corresponding change has been detected, a large number of examples will be removed instantaneously from the case base. In the classification scenario, 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 determined by testing the null hypothesis $H_0 : p \leq p_{min}$ against the alternative hypothesis $H_1 : p > p_{min}$. This is accomplished by 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.[3] Examples to be removed are chosen at random according to a distribution which is spatially uniform but temporally skewed; see [6] for details.

In the regression scenario, the above test is conducted with the mean absolute error instead of the classification rate, and the percentage of examples to be removed is determined by the relative increase of this error. Fig. 2 and Fig. 3 depict both algorithms dealing with concept drifts for the classification and regression case.

3.3 Parameter Adaptation in IBLStreams

Although instance-based learning does not induce a global model, its performance still depends on several parameters, such as the neighborhood size k. Thus, given its application in an evolving environment, some sort of adaptivity would clearly be desirable in this regard. In IBLStreams, two approaches for parameter adaptation are implemented (see Fig. 4).

In the first approach, we adapt the size k of the neighborhood. To this end, we continuously check whether it appears beneficial to increase or decrease the current value by 1. In order to make this decision, we monitor the mean error on a window formed by the last 100 instances, not only for the current IBLStreams version with k neighbors but also the variants with $k-1$ and $k+1$ neighbors. Whenever one of these two variants performs better in terms of the mean error, the current k is adapted correspondingly (see lines 1-10 in the pseudo-code in Fig. 4).

The second strategy controls the size of the neighborhood indirectly via the weighting function or, more specifically, the corresponding kernel width; this adaptation strategy can only be used in combination with the Gaussian or the exponential kernel. Like in the previous case, three variants of IBLStreams are compared in terms of their mean error on the last 100 instances, namely the current variant, the variant with a kernel width increased by 5%, and the variant with a kernel width decreased by 5% (see lines 11-21 in the pseudo-code in Fig. 4).

[3]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.

4 MOA

IBLStreams is implemented under the MOA (Massive Online Analysis) framework, an open source software for mining and analyzing large data sets in a stream-like manner. MOA is written in Java and is closely related to WEKA [36], the Waikato Environment for Knowledge Analysis, which is at present the most commonly used machine learning software.

MOA supports the development of classifiers that can learn either in a purely incremental mode, or in batch mode first (on an initial part of a data stream) and incrementally afterward. The implementation of an evolving classifier is supported by a Java interface called UpdateableClassifier. This operation simulates the case of online learning, which means that each instance is accessed only once. A few incremental classifiers are already included in MOA, notably the Hoeffding tree [21], a state-of-the-art classifier often used as a baseline in experimental studies. Some meta learning techniques are implemented, too, such as online bagging and boosting both for static [28] and evolving streams [8].

4.1 Stream Generators

MOA supports the simulation of data streams by means of synthetic stream generators. An example is the Hyperplane generator that was originally used in [21]. It generates data for a binary classification problem, taking a random hyperplane in d-dimensional Euclidean space as a decision boundary; a certain percentage of instances is corrupted with noise.

Another important stream generator is the RandomTree generator. Its underlying model is a decision tree for a desired number of attributes and classes. The tree is built by splitting on randomly chosen attributes and then giving random class labels to the leaf nodes. Instances are generated with uniformly distributed values in the attributes while the class label is determined by the tree.

MOA offers the ConceptDriftStream procedure for simulating concept drift. The idea underlying this procedure is to mix two pure distributions in a probabilistic way, smoothly varying the corresponding probability degrees. In the beginning, examples are taken from the first pure stream with probability 1, and this probability is decreased in favor of the second stream in the course of time. More specifically, the probability is controlled by means of the sigmoid function

$$f(t) = \left(1 + e^{-4(t-t_0)/w}\right)^{-1} \quad .$$

This function has two parameters: t_0 is the mid point of the change process, while

w is the length of this process.

4.2 Model Evaluation

The evaluation of an evolving classifier is clearly a non-trivial issue. In fact, compared to standard batch learning, simple one-dimensional performance measures such as classification accuracy are not immediately applicable, or at least not able to capture the time-varying behavior of a classifier in a proper way. MOA offers different solutions for this problem.

The holdout procedure is a generalization of the cross-validation procedure commonly used in batch learning. Here, the training and the testing phase of a classifier are interleaved as follows: the classifier is trained incrementally on a block of M instances and then evaluated (but no longer adapted) on the next N instances, then again trained on the next M and tested on the subsequent N instances, and so forth. Thus, it becomes possible to monitor the performance of the model as time progresses; this information can also be used as an indicator of possible changes of the underlying concept [7, 9].

While the holdout procedure uses an instance either for training or for testing, each instance is used for both in the prequential approach [12]: First, the model is evaluated on the instance, and then a single incremental learning step is carried out. The prequential error is advocated in [22], where it is also shown to converge to the holdout measure when using a sliding window or a fading factor (exponential weighting).

5 Experiments

In this section, we compare IBLStreams with state-of-the-art learners, namely Hoeffding trees for classification [21] and the FLEXFIS approach for regression [25], in terms of prediction performance and handling of concept drift. Hoeffding trees is a decision tree approach suitable for learning classifiers on data streams. For our experiments, we used the MOA implementation of Hoeffding trees in the default parameter setting.[4] For IBLStreams, we set $k_{cand} = 5$, initial $k = 16$, initial kernel width (for exponential and Gaussian kernels) $\sigma = 0.5$ and the maximum case base size=5000.

[4] gracePeriod $g = 200$, splitConfidence $c = 10^{-7}$, tieThreshold $t = 0.05$, numericEstimator n=GAUSS10

FLEXFIS constructs and maintains a specific kind of fuzzy rule-based model, namely a Takagi-Sugeno model [33]; apart from the original approach to regression, there is also a classification version of FLEXFIS [27], which can hence be used for both types of experiments. FLEXFIS is implemented in Matlab and offers a function for finding optimal parameter values. We used this function to fix all parameters except the so-called "forgetting parameter", for which we manually found the value 0.999 to perform best. Finally, pruning was enabled.

Experiments are not only conducted with real data sets, but also with synthetic data. As an important advantage of synthetic data, let us note that it allows for conducting experiments in a *controlled* way and, therefore, to investigate the performance of a method under specific conditions. In particular, synthetic data is useful for simulating a concept drift.

The experiments are performed in the MOA framework, using the holdout procedure for measuring predictive accuracy. The parameters M and N vary depending on the size of the data set (we take $M = 5000$ and $N = 1000$ in the first two experiments with synthetic data). For the experiments with real data, these parameters are adapted to the size of the respective data set. The real data sets are standard benchmarks taken from the Statlib archive[5] and the UCI repository [16]. Since they do not have an inherent temporal order, we average the performance curves over 100 randomly shuffled versions of these data sets.

5.1 Classification

For the classification experiments, we use IBLStreams in three variants:

C1: equal weighting of neighbors, adaptation of neighborhood size k
C2: weighting with exponential kernel, adaptation of kernel width
C3: equal weighting of neighbors, no adaptation

All other parameters in IBLStreams are set to their default values.[6]

5.1.1 Synthetic Data

The first two experiments are based on synthetic data with different characteristics (i.e., different types of decision boundaries). The first experiment uses

[5] `http://lib.stat.cmu.edu/`

[6] See the documentation of IBLStreams at `http://moa.cs.waikato.ac.nz/moa-extensions/`

data taken from the "hyperplane" generator. The ConceptDriftStream procedure mixing streams produced by two different hyperplanes simulates a rotating hyperplane. Using this procedure, we generated 12,000,000 examples connecting two hyperplanes in 4-dimensional space, with $t_0 = 500,000$ and $w = 100,000$.

As can be seen in Fig. 5, the different IBLStreams settings hardly differ in terms of performance. Moreover, they outperform both Hoeffding trees and FLEXFIS, with a very small decrease in performance when the drift occurs. IBLStreams recognises and adapts to the concept drift quite early, recovering its original performance as soon as the drift is over. The Hoeffding tree is more affected by the concept drift; it shows a more pronounced "valley" in the performance curve and also takes more time to recover. FLEXFIS is affected by the drift, too, with about 5% decrease in performance.

In a second experiment, we use the "random tree" generator to produce examples. This generator constructs a classification tree (by repeatedly choosing recursive splits) at random and uses this tree to label instances. Obviously, it is favorable for the Hoeffding tree. Again, the same ConceptDriftStream is used, but this time mixing two random tree generators. As can be seen in Fig. 6, the Hoeffding tree is now able to compete with IBLStreams in the first phase of the learning process; in fact, reaching an accuracy of close to 100%, which is not unexpected given that the Hoeffding tree is ideally tailored for this kind of data. Once again, however, the Hoeffding tree is much more affected by the concept drift than IBLStreams. The three variants of IBLStreams do not show any decrease in terms of classification rate. In contrast, they continue to improve the performance during the drift, whereas the Hoeffding tree loses about 40% of its accuracy, and FLEXFIS loses about 10%. The relative performance of FLEXFIS is similar to the case of the hyperplane data.

5.1.2 Real Data

In this experiment, we used the wine quality data, which is an ordinal classification problem, in which a wine (characterized by 11 chemical properties) is put into a discrete category ranging from 10 (best) to 0 (worst). We turned this problem into a binary classification task by grouping the top-5 and bottom-6 classes. Actually, the data set consists of two subsets, one for white wine (4889 examples) and one for red wine (1599 examples). For both data sets, the initial learning is done on 300 instances. In all our experiments on the wine quality data, we average the results over 100 randomly shuffled versions. For the evaluation on the red wine data, we used $M = 100$ and $N = 25$, because this data set is relatively

small; for white wine, we used $M = 200$ and $N = 50$. Fig. 7 and Fig. 8 show the results of both experiments. As can be seen, IBLStreams in its different settings is clearly superior to Hoeffding trees on these data sets.

The same data sets, only without grouping the output categories, were used to evaluate the multi-class case. The IBLStreams settings were also the same as for the binary case, with only one exception: Since the problem is now an ordinal classification task, the weighted median was used instead of the mode for prediction. As can be seen from Fig. 9 and Fig. 10, the performance of both IBLStreams and Hoeffding trees is now lower than for the binary case, an observation that is clearly expected. Still, IBLStreams remains superior on the whole stream.

Table 1 shows the execution time for the training and evaluation of instances. On both synthetic data sets for binary classification, IBLStreams is able to process 1000 instances in less than 1.5 seconds, and needs about 0.5 seconds for making the same amount of predictions. Similar results are obtained for the multi-class problems.

		binary class.				multi-class class.	
		hyperpl.	rand. tree	red	white	red	white
C1	tt	1.18±0.19	1.14±0.19	1.28±0.36	3.15±1.46	0.80±0.14	1.26±0.43
	st	0.41±0.09	0.40±0.09	0.61±0.16	1.52±0.67	0.36±0.06	0.58±0.19
C2	tt	1.19±0.19	1.19±0.18	1.43±0.40	3.59±1.71	1.14±0.22	1.84±0.65
	st	0.46±0.09	0.47±0.10	0.68±0.19	1.77±0.80	0.54±0.09	0.88±0.30
C3	tt	1.24±0.20	1.32±0.27	1.53±0.44	3.82±1.80	0.80±0.14	1.27±0.44
	st	0.48±0.09	0.52±0.15	6.12±1.76	1.88±0.86	0.37±0.08	0.60±0.21

Table 1: Average time (in seconds) for training (tt) and testing (ts) per 1000 instances.

5.2 Regression

In the regression case, we used IBLStreams in four different settings (while the rest of the parameters were again set to their default values):

- R1: weighted mean, equal weighting of neighbors, adaptation of neighborhood size k
- R2: weighted mean, weighting with exponential kernel, adaptation of kernel width

R3: local linear regression, equal weighting of neighbors, adaptation of neighborhood size k

R4: local linear regression, weighting with exponential kernel, adaptation of kernel width

5.2.1 Synthetic Data

For the case of regression, we modified the hyperplane generator in MOA as follows: The output for an instance x is not determined by the sign of $w^T x$, where w is the normal vector of the hyperplane, but by the absolute value $|w^T x|$. In other words, the problem is to predict the distance to the hyperplane. As an alternative, we also tried the squared distance $(w^T x)^2$ and the cubic distance $(w^T x)^3$. Again, ConceptDriftStream was used for simulating a concept drift by mixing two streams.

Fig. 11, 12 and 13 show the performance of IBLStreams and FLEXFIS, in terms of the root mean squared error (RMSE), for the (piecewise) linear, the quadratic case and the cubic case (and dimension $d = 4$), respectively. As can be seen, FLEXFIS is significantly outperformed by the different versions of IBLStreams. In fact, the RMSE is clearly lower for IBLStreams, not only under normal conditions but also in cases of a concept drift.

Comparing the IBLStreams variants amongst each other, it seems that local linear regression tends to perform better than just using the weighted mean as an estimator. This is clearly not unexpected, since the latter can be seen as a special case of the former, which is more flexible and, therefore, able to adapt to the data more easily. Yet, in the case of a concept drift, the simple weighted mean seems to be affected less strongly. This result is completely in agreement with the general observation that the more complex a model is, the more difficult it becomes to react and adapt to changing environmental conditions.

5.2.2 Real Data

In this experiment, we used the UCI data set about relative location of CT (computed tomography) slices on axial axis. This data set is extracted from 53500 images taken for 74 different patients (43 males and 31 females). Each CT image is described in terms of of 384 features. The target attribute is the relative location of the CT slice on the axial axis of the human body; this is a numeric value in the range $[0, 180]$, where 0 denotes the top of the head and 180 the soles of the feet. The data was ordered by the patient ID, which means that

images from the same patient are grouped. We kept this order of patient-wise blocks (since the transition from one patient to another one could be seen as a concept shift) but randomly shuffled the data within each block, so as to avoid a dependence of the order on the target value. Moreover, using a variant of the forward selection method as proposed in [25], the number of features was reduced to 9. Learning was started on an initial block of 1000 instances, and incremental learning was done using $M = 1000$ and $N = 500$. The results in Fig. 14 show that the different configurations of IBLStreams outperform FLEXFIS most of the time. Again, there is no big difference between the variants themselves.

The second experiment uses the concrete compressive strength data [37]. In this data set, a concrete is described in terms of its age and ingredients, giving rise to 9 attributes in total. The target attribute is the concrete compressive strength, which lies in the range $[2.33, 82.60]$. Since this data set is relatively small (1030 examples), we set $M = 50$ and $N = 10$ and used only 100 examples for initial learning. Fig. 15 shows that, on this data set, there is no clear winner. Again, the IBLStreams variants perform more or less equal most of the time. Comparing these variants with FLEXFIS, the latter is superior on some parts of the data stream and the former on others.

6 Discussion and Conclusion

We have presented a lazy algorithm for learning on data streams, using instance-based methods for tackling the tasks of classification and regression. This algorithm, called IBLStreams, has a number of desirable properties that are not, at least not as a whole, shared by existing alternative methods. In particular, two specifically designed editing strategies are used in combination in order to successfully deal with both gradual concept drift and abrupt concept shift.

The experiments presented in [6], complemented by those in this paper, suggest that IBLStreams is very flexible and thus able to adapt to an evolving environment quickly, a point of utmost importance in the data stream context. Indeed, the most important conclusions that can be drawn from our experiments are as follows:

— Compared to the other methods used in the experiments, IBLStreams seems to be less "inert" when a concept drift occurs and, moreover, recovers its original performance more quickly when the drift comes to an end. This is arguably due to the advantage of not having to adapt a possibly complex model.

— Besides, it seems that IBLStreams is relatively robust and produces good results when being used in a default setting for its parameters. In fact, in our experiments, IBLStreams was often better but never significantly worse than its competitors, even is cases where the data generating process is actually in favor of the latter.

IBLStreams is implemented in Java and can be downloaded, along with a documentation, from the Internet.[7] This implementation is supposed to be used under MOA[8], an open source framework for mining and learning from data streams, which is in the offing to be become a standard in this field [7].

References

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of 29th International Conference on Very Large Data Bases*, Berlin, Germany, 2003. Morgan Kaufmann.

[2] David W. Aha, editor. *Lazy Learning*. Kluwer Academic Publishers, Norwell, 1997.

[3] David W. Aha, Dennis F. Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[4] Plamen P. Angelov, Dimitar P. Filev, and Nik Kasabov. *Evolving Intelligent Systems*. John Wiley and Sons, New York, 2010.

[5] Plamen P. Angelov, Edwin Lughofer, and Xiaowei Zhou. Evolving fuzzy classifiers using different model architectures. *Fuzzy Sets and Systems*, 159(23):3160–3182, 2008.

[6] Jürgen Beringer and Eyke Hüllermeier. Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6):627–650, 2007.

[7] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[7] `http://www.uni-marburg.de/fb12/kebi/research/software/iblstreams`

[8] `http://moa.cs.waikato.ac.nz/moa-extensions/`

[8] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *KDD 2009, Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148, Paris, France, 2009. ACM.

[9] Albert Bifet and Richard Kirkby. *Massive Online Analysis Manual*, August 2009.

[10] Graham Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *ACM Symposium on Principles of Database Systems (PODS)*, San Diego, California, 2003. ACM.

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

[12] A. P. Dawid. Statistical theory: The prequential approach. In *Journal of the Royal Statistical Society-A*, pages 147:278–292. 1984.

[13] Pedro 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.

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

[15] Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12, 2003.

[16] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[17] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: A review. *ACM SIGMOD Record*, 34(1), 2005.

[18] João Gama and Mohamed Medhat Gaber. *Learning from Data Streams*. Springer-Verlag, Berlin, New York, 2007.

[19] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *SBIA 2004, Proceedings of 17th Brazilian Symposium on Artificial Intelligence*, Lecture Notes in Computer Science, pages 286–295, São Luis, Maranhão, Brazil, 2004. Springer.

[20] João 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 Mexico, USA, 2005. ACM.

[21] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97 – 106, San Francisco, CA, USA, 2001. ACM.

[22] Raquel Sebastião João Gama and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338, Paris, France, 2009. ACM.

[23] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *ICML 2000, Proceedings 17th International Conference on Machine Learning*, pages 487–494, Stanford, CA, USA, 2000. Morgan Kaufmann.

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

[25] Edwin Lughofer. FLEXFIS: A robust incremental learning approach for evolving Takagi-Sugeno fuzzy models. *IEEE Transactions on Fuzzy Systems*, 16(6):1393–1410, 2008.

[26] Edwin Lughofer. *Evolving Fuzzy Systems: Methodologies, Advanced Concepts and Applications*. Springer-Verlag, Berlin, Heidelberg, 2011.

[27] Edwin Lughofer, Plamen P. Angelov, and Xiaowei Zhou. Evolving single- and multi-model fuzzy classifiers with FLEXFIS-class. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 1–6, London, UK, 2007. IEEE.

[28] Nikunji C. Oza and Stuart Russell. Online bagging and boosting. *Artificial Intelligence and Statistics*, pages 105 – 112, 2001.

[29] Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, 1997.

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

[31] Ammar Shaker and Eyke Hüllermeier. Instance-based classification and regression on data streams. In Edwin Lughofer and Moamar Sayed Mouchaweh, editors, *Learning in Non-Stationary Environments: Methods and Applications*. Springer-Verlag. To appear.

[32] Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29:1213–1228, 1986.

[33] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.

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

[35] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

[36] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.

[37] I-Cheng Yeh. Modeling of strength of high performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.

A Distance Function

The distance function used in IBLStreams is an incremental variant of SVDM (Simple Value Difference Metric) which is a simplified version of the VDM (Value Difference Metric) distance measure [32] and was successfully used in the classification algorithm RISE [13, 14]. 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.[9] Then, $\delta_i(f_i, f_i')$ is defined by the Euclidean distance between the normalized values of f_i and f_i'.

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

For a discrete attribute F_j, the distance between two values f_j and f'_j is defined by the following measure:

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

where m is the number of classes and $P(\lambda \,|\, F = f)$ 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

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

Procedure Predict

Input: case base $\mathcal{D}$, example e=$\langle x_0, ? \rangle$
k number of considered instances
σ variance, used in the case of Gaussian or exponential weighting
WM weighting method (equal, inverseDistance, linear, gaussianKernel, exponentialKernel)
PM prediction method (Local_Lienar_Solution, wKNN)
Output: $\hat{\lambda}_{x_0}$

1: $S = \{k$ `nearest neighbor of` e `in` $\mathcal{D}\}$
2: **if** $\exists x_i \in S : x_i = x_0$ **then**
3: **return** λ_{x_i}
4: **end if**
5: $W = \text{getNormalizedWeightingVector}(e, WM, k, \sigma)$
6: **if** Classification **then**
7: $C = [0]_{\#classes}$
8: **for all** $x_i \in S$ **do**
9: $\text{C}[\lambda_{x_i}] = \text{C}[\lambda_{x_i}] + w_i$
10: **end for**
11: **if** Ordered Classification **then**
12: $\hat{\lambda}_{x_0} = median(C)$
13: **else**
14: $\hat{\lambda}_{x_0} = argmax_i \{c_i\}$
15: **end if**
16: **else**
17: {Regression}
18: **if** $PM =$ `wKNN` **then**
19: {Solve it as wKNN}
20: $\hat{\lambda}_{x_0} = W^T Y$
21: **else**
22: $X = [x_i 1]_{x_i \in S}$
23: $Y = [\lambda_{x_i}]_{x_i \in S}$
24: $\hat{\beta} = (X^T W X)^{-1} X^T W Y$
25: $\hat{\lambda}_{x_0} = [x_0 1]\hat{\beta}$
26: **end if**
27: **end if**
28: {dist is the Euclidean distance}
29: **return**

Figure 1: Algorithm for predicting the target value of a new instance.

Procedure ConceptDriftClassification

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

1: c = class estimate for x_0 derived from $\mathcal{D}$
2: compare c and λ_{x_0}, update statistics for the last 100 examples (error p and standard deviation s)
3: **if** $(1-p) <= 1.0/classCount$ OR $p + s > p_{min} + Z_\alpha s_{min}$ **then**
4: {and Warnings condition}
5: p_{diff} =(error of the last 20 training data) $-p_{first}$
6: **if** $p_{first} > 0.2$ **then**
7: delete $min(|\mathcal{D}|p_{diff}, |\mathcal{D}| - k_{test})$ cases from $\mathcal{D}$
8: reset p_{min}, s_{min}
9: **end if**
10: **end if**
11: $\{\alpha = 0.999 \Rightarrow Z_\alpha = 4\}$
12: **return**

Figure 2: Algorithm for checking and Handling of Concept Drifts in classification problems.

Procedure ConceptDriftRegression

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

1: c = the estimated target value for x_0 derived from $\mathcal{D}$
2: compare c and λ_{x_0}, update statistics for the last 100 examples (error p and standard deviation s)
3: **if** $p + s > p_{min} + Z_\alpha s_{min}$ **then**
4: {and Warnings condition}
5: $\tau = min(\frac{p - p_{min}}{p_{min}}, 0.5)$
6: delete $min(\tau|\mathcal{D}|, |\mathcal{D}| - k_{test}))$ cases from $\mathcal{D}$
7: reset p_{min}, s_{min}
8: **end if**
9: $\{\alpha = 0.999 \Rightarrow Z_\alpha = 4\}$
10: **return**

Figure 3: Algorithm for checking and Handling of Concept Drifts in classification problems.

Procedure UpdateClassifier

Input: case base $\mathcal{D}$, example e=$\langle x_0, \lambda_{x_0} \rangle$
k number of considered instances
σ variance, used in the case of Gaussian or exponential weighting
WM weighting method (equal, inverseDistance, linear, gaussianKernel, exponentialKernel)
PM prediction method (Local_Lienar_Solution, wKNN)
Output: k, σ
Global Variables:
$\{\sigma_0 = \sqrt{\#Att.}/10, k_0 = \#Att. * 4, \delta = 0.05\}$

```
1: if Different_ks then
2:    {p : mean error for the last 100 examples}
3:    update p_0 by Predict (e, WM, k − 1, σ)
4:    update p_1 by Predict (e, WM, k, σ)
5:    update p_2 by Predict (e, WM, k + 1, σ)
6:    if p_2 < p_1 then
7:       k = k + 1
8:    else if p_0 < p_1 then
9:       k = k − 1
10:   end if
11: else if Different_Segmas then
12:   {p : the mean error for the last 100 examples }
13:   update p_0 by Predict (e, WM, k, σ(1 − δ))
14:   update p_1 by Predict (e, WM, k, σ)
15:   update p_2 by Predict (e, WM, k, σ(1 + δ))
16:   if p_2 < p_1 then
17:      σ = σ(1 + δ)
18:   else if p_0 < p_1 then
19:      σ = σ(1 − δ)
20:   end if
21: end if
22: return
```

Figure 4: Algorithm for updating the parameters of the classifier.

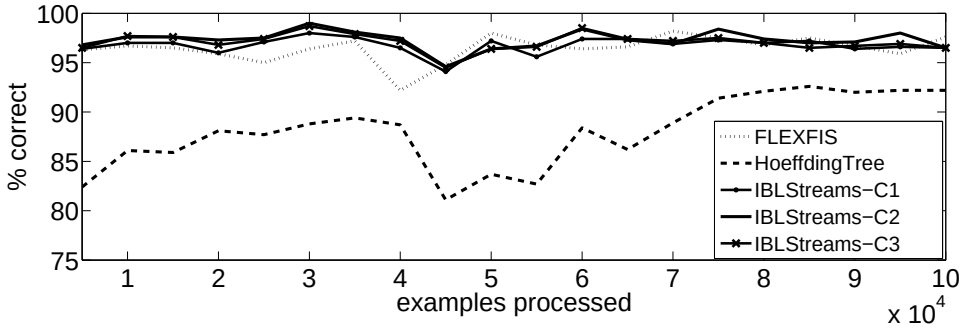

Figure 5: Classification rate on the hyperplane data (binary).

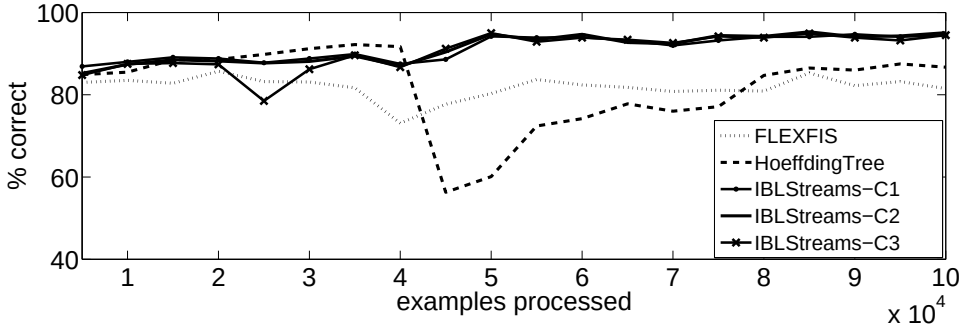

Figure 6: Classification rate on the random tree data (binary).

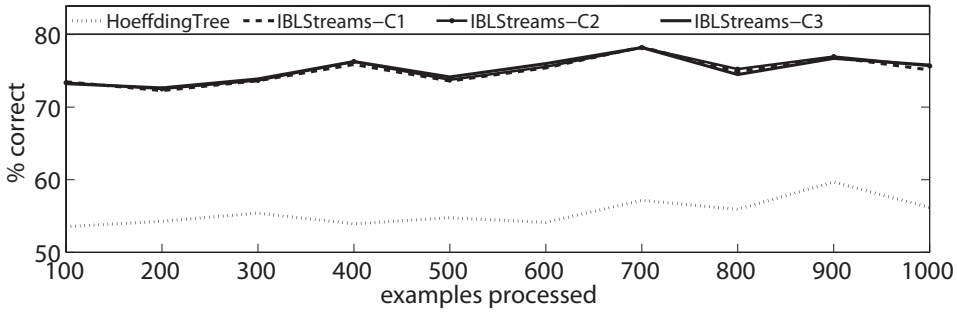

Figure 7: Classification rate on the red wine quality data set (binary).

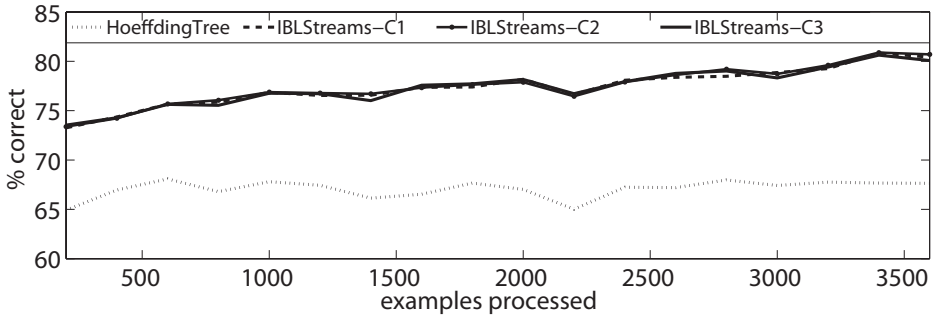

Figure 8: Classification rate on the white wine quality data set (binary).

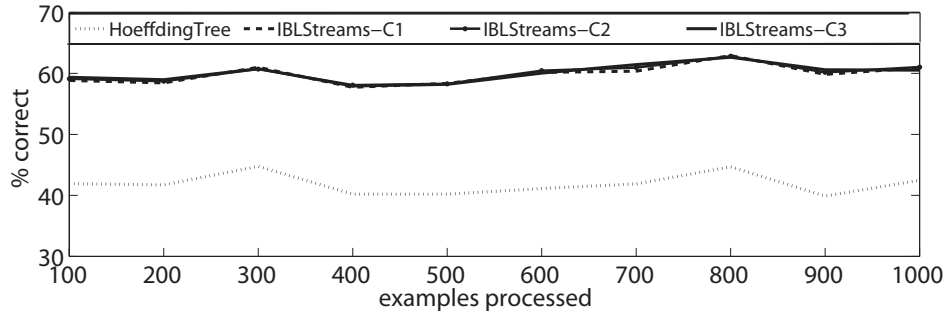

Figure 9: Classification rate on the red wine quality data set (multi-class).

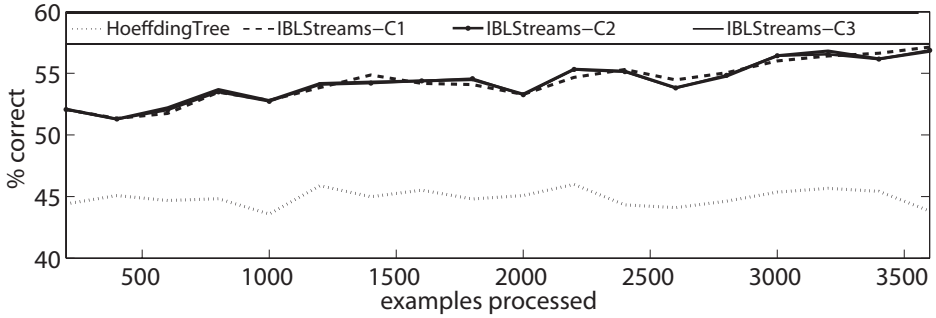

Figure 10: Classification rate on the white wine quality data set (multi-class).

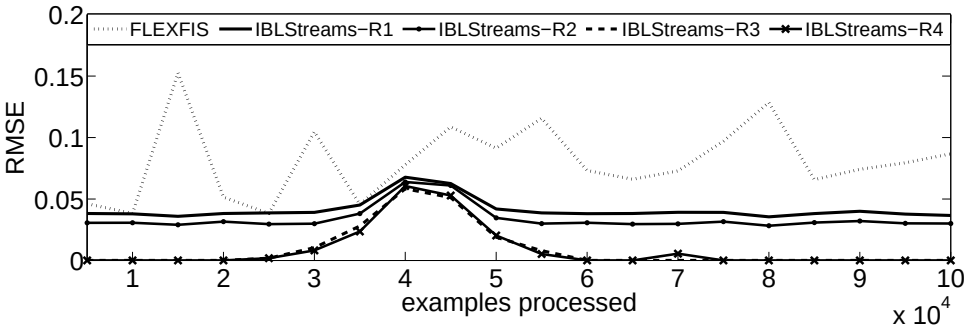

Figure 11: RMSE for the hyperplane data (regression, linear case).

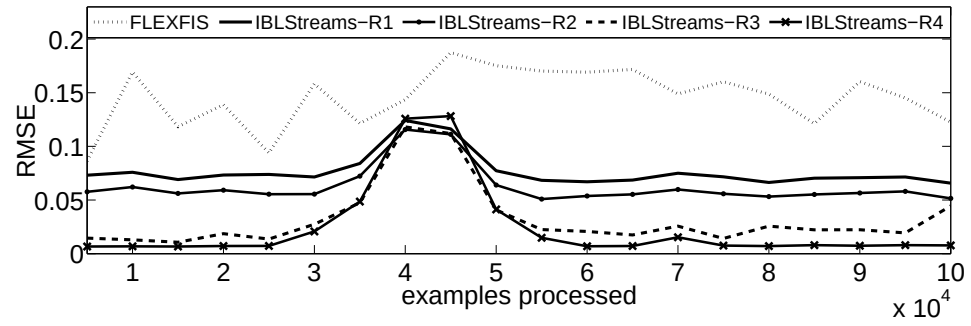

Figure 12: RMSE for the hyperplane data (regression, quadratic case).

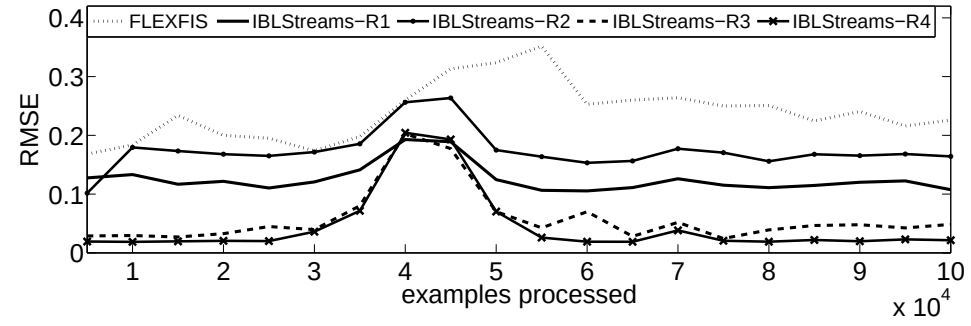

Figure 13: RMSE for the hyperplane data (regression, cubic case).

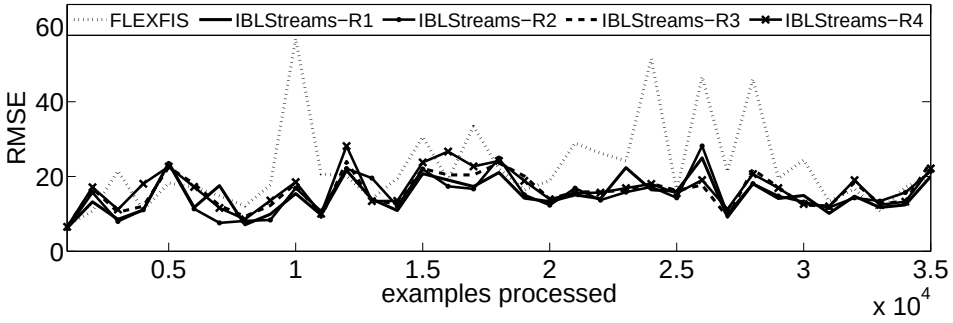

Figure 14: RMSE for the relative location of CT slices on axial axis (regression).

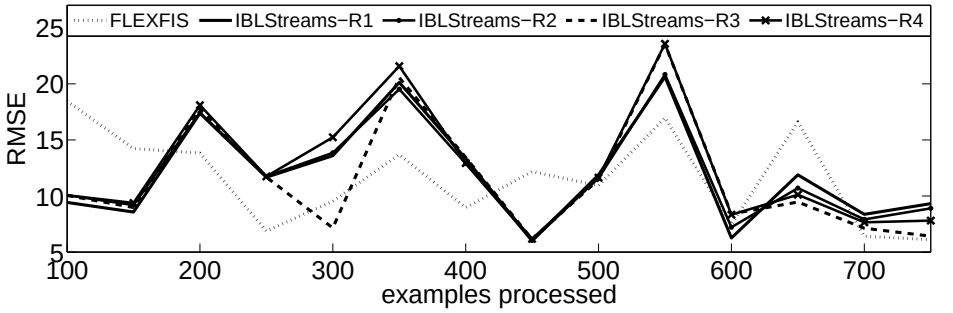

Figure 15: RMSE for the concrete compressive strength data (regression).