Chapter 8
Instance-Based Classification and Regression on Data Streams

Ammar Shaker and Eyke Hüllermeier

Abstract In order to be useful and effectively applicable in dynamically evolving environments, machine learning methods have to meet several requirements, 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. This paper advocates an instance-based learning algorithm for that purpose, both for classification and regression problems. This algorithm has a number of desirable properties that are not, at least not as a whole, shared by currently existing alternatives. Notably, our method is very flexible and thus able to adapt to an evolving environment quickly, a point of utmost importance in the data stream context. At the same time, the algorithm is relatively robust and thus applicable to streams with different characteristics.

8.1 Introduction

The idea of adaptive learning in dynamical environments has recently received increasing attention in different research communities, for example, in the database and data mining community under the slogan of "learning from data streams" [17, 18], and in the computational intelligence community under the notion of "evolving fuzzy systems" [4, 5, 24, 25]. 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.

A. Shaker • E. Hüllermeier (✉)
Department of Mathematics and Computer Science, Philipps-Universität
Marburg, D-35032 Marburg, Germany
e-mail: shaker@Mathematik.Uni-Marburg.de; eyke@mathematik.uni-marburg.de

Systems with these properties have been developed for different machine learning and data mining problems, such as clustering [1], classification [22], and frequent pattern mining [10].

Domingos and Hulten [15] list a number of properties that an ideal stream mining system should possess, 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 [32] (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, the core of this paradigm, is a standard method in machine learning, pattern recognition, and related fields. In this chapter, 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 classification and regression. To this end, we build on [6], in which our approach to classification was introduced.

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, IBL-DS, is introduced in Sect. 8.3. In Sect. 8.4, we provide some information about the MOA (Massive Online Analysis) framework for mining data streams, in which IBL-DS is implemented. Experimental results are presented in Sect. 8.5.

8.2 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 [23, 27, 28]. 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 $\mathscr{X}$ denote the instance space, where an instance corresponds to the description x of an object (usually although not necessarily in attribute-value form). $\mathscr{X}$ is endowed with a distance measure $\Delta(\cdot)$, i.e., $\Delta(x,x')$ is the distance between instances $x,x' \in \mathscr{X}$. $\mathscr{Y}$ is the output space and $\langle x,y\rangle \in \mathscr{X} \times \mathscr{Y}$ is called a labeled instance, a case, or an example. In classification, $\mathscr{Y}$ is a finite (usually small) set comprised of m classes $\{\lambda_1,\ldots,\lambda_m\}$, whereas $\mathscr{Y} = \mathbb{R}$ in regression.

The current experience of the learning system is represented in terms of a set $\mathscr{D}$ of examples $\langle x_i,y_i\rangle$, $1 \le i \le n = |\mathscr{D}|$. From a machine learning point of view, $\mathscr{D}$ 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, $\mathscr{D}$ is only a subset of the training set. In case-based reasoning, it is also referred to as the *case base*.

Finally, suppose a novel instance $x_0 \in \mathscr{X}$ (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 $k \ge 1$ nearest neighbors of x_0 into account. That is, an estimation y_0^{est} of y_0 is derived from the set $\mathscr{N}_k(x_0)$ of the k nearest neighbors of x_0. In classification, this is usually done by means of a *majority vote*, i.e.,

$$y_0^{\mathrm{est}} = \arg\max_{\lambda \in \mathscr{L}} \#\{x_i \in \mathscr{N}_k(x_0) \,|\, y_i = \lambda\}, \tag{8.1}$$

with $\mathscr{L}$ the set of class labels, whereas in regression, a weighted average of the outputs of the neighbors is predicted:

$$y_0^{\mathrm{est}} = \sum_{x_i \in \mathscr{N}_k(x_0)} w(x_i) \cdot y_i, \tag{8.2}$$

with

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

Here, $f(\cdot)$ is a decreasing function $\mathbb{R}_+ \to \mathbb{R}_+$, which means that the smaller $\Delta(x_i,x_0)$, the stronger the weight of y_i.

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 [30], 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 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 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.

8.3 Instance-Based Learning on Data Streams

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

8.3.1 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 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, nonwindow-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 do not 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.

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

- 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), or consistency (e.g., Instance-Based learning algorithm 3, IB3). A few methods also use a second indicator, e.g., the approach of Klinkenberg (temporal relevance and consistency), but only the window-based system FLORA4 (Floating Rough Approximation) uses all three aspects.

8.3.2 IBL-DS

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

IBL-DS optimizes the composition and size of the case base autonomously. On arrival of a new example $\langle x_0, y_0 \rangle$, 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 Sect. 8.7), 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] $k_{\text{test}} = (k_{\text{cand}})^2 + k_{\text{cand}}$ 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 the 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. The similarity environment constitutes the set of instances in the vicinity of the query instance, while the test environment can be seen as the union of the similarity environments of the neighbored instances.

[2]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.

In the regression scenario, the k_{cand} youngest examples in the neighborhood set C determines a confidence interval $\left[\bar{y} - Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{k_{\text{cand}}}}, \bar{y} + Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{k_{\text{cand}}}}\right]$, where $\bar{y}$ is the average target value for the considered examples and σ is the standard deviation. A class values y_0 outside this interval indicates an unexpected change in the neighborhood when this instance was generated. In this case, instances not belonging to the confidence interval are removed from the larger test environment.

Using this strategy, the algorithm is able to adapt to concept drift but will also have a high accuracy for nondrifting 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.

[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.

8.4 MOA

IBL-DS 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 [31], the Waikato Environment for Knowledge Analysis, which is presently 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 [22], 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 [26] and evolving streams [8].

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 [22]. 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

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

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

8.4.2 Model Evaluation

The evaluation of an evolving classifier is clearly a nontrivial 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 [21], where it is also shown to converge to the holdout measure when using a sliding window or a fading factor (exponential weighting).

8.5 Experiments

In this section, we compare IBL-DS with state-of-the-art learners in terms of performance and handling of concept drift, namely Hoeffding trees for classification [22] and the FLEXFIS approach for regression [24]. Hoeffding trees is a decision tree approach suitable for learning on data streams, whereas FLEXFIS constructs and maintains a specific kind of fuzzy rule-based model, namely a model of the Takagi–Sugeno type [29]. Our study is not meant as an extensive empirical evaluation that supports statistically valid conclusions. Instead, it is only supposed to serve an illustration purpose. We refer to [6] for more experiments with classification problems.

We use IBL-DS in its default setting unless otherwise stated (in some binary classification problems, we try different values for the maximum size of the instance base). 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

Table 8.1 Summary of the data sets used in the experiments

Data Set	Instances	Attributes	Holdout evaluation
Statlog (shuttle)	58,000	9	$M = 5{,}000$ and $N = 1{,}000$
Red wine	1,599	11	$M = 100$ and $N = 25$
White wine	4,889	11	$M = 200$ and $N = 50$
YearPredictionMSD	515,345	90	$M = 200$ and $N = 50$

depending on the size of the data set (we take $M = 5{,}000$ and $N = 1{,}000$ 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; see Table 8.1 for an overview of the main characteristics of these data sets. The real data sets are standard benchmarks taken from the Statlib archive[4] 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.

8.5.1 *Classification*

8.5.1.1 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 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 four-dimensional space, with $t_0 = 500{,}000$ and $w = 100{,}000$.

We compare the performance of two different settings of IBL-DS, one with a value of 400 for the maximum size of the instance base and the other one with 5,000. Figure 8.1 shows that both versions of IBL-DS initially outperform the Hoeffding tree. The Hoeffding tree is also more affected by the concept drift, showing a more pronounced "valley" in the performance curve, and also taking more time to recover. IBL-DS recognizes and adapts to the concept drift quite early, recovering its original performance as soon as the drift is over.

In a second experiment, we use the random tree generator to produce examples. Obviously, this generator 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. 8.2, the Hoeffding tree is now able to outperform IBL-DS 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

[4]http://lib.stat.cmu.edu/.

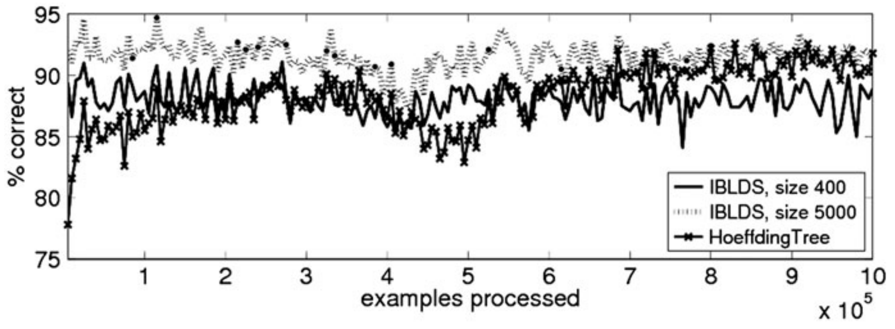

Fig. 8.1 Classification rate on the hyperplane data (binary)

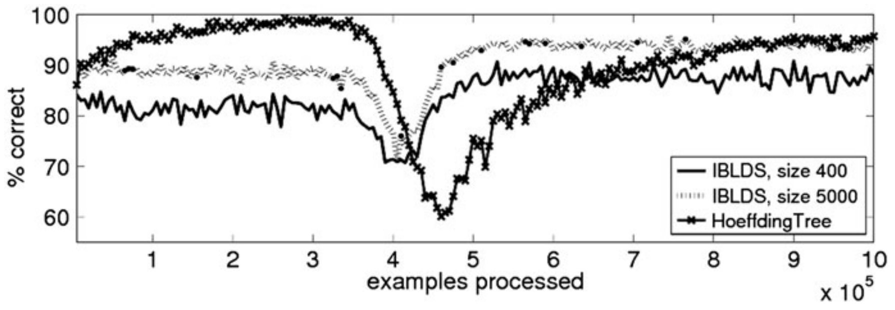

Fig. 8.2 Classification rate on the RandomTree data (binary)

concept drift than the IBL-DS. Both variants of IBL-DS suffer from a drop of about 15% in terms of classification rate, and recover quickly during the phase of the drift, whereas the Hoeffding tree loses about 40% of its accuracy.

8.5.1.2 Real Data

In this experiment, we used the Shuttle data from the Statlog repository, for which the task is to predict the class of a shuttle. The data set is highly imbalanced, with 80% of the instances belonging to one class and the remaining 20% distributed among six other classes; in order to obtain a binary problem, we grouped these six classes into a single one. The new problem thus consists of predicting whether a shuttle belongs to the majority class or not. Both algorithms were initially trained on 300 instances in batch mode; for the holdout evaluation, we used $M = 200$ and $N = 50$. Figure 8.3 shows the results averaged over 100 randomly shuffled versions of the data set. As can be seen, IBL-DS starts with a very strong performance, close to 99% accuracy; the Hoeffding tree reaches this accuracy, too, but not before observing three quarters of the whole stream.

The wine quality data is an ordinal classification problem, in which a wine (characterized by several chemical properties) is put into a discrete category ranging

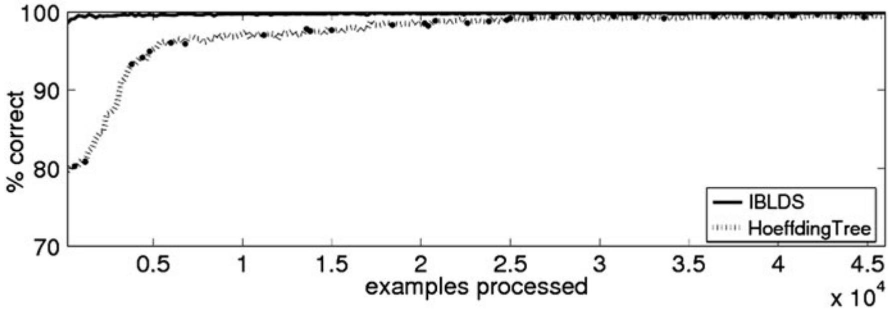

Fig. 8.3 Classification rate on the Shuttle data (binary)

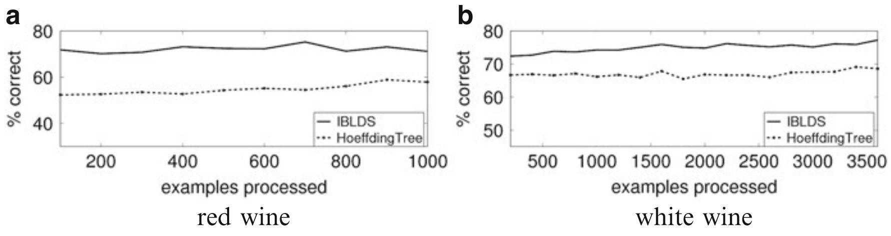

Fig. 8.4 Classification rate on the wine quality data set (binary)

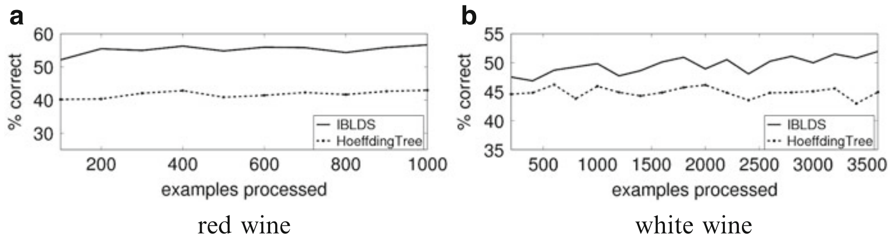

Fig. 8.5 Classification rate on the wine quality data set (multiclass)

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 and one for red wine. 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 (about 1,600 examples); for white wine, we used $M = 200$ and $N = 50$. Figure 8.4 shows the results of both experiments. As can be seen, IBL-DS is clearly superior to Hoeffding trees on these data sets.

For evaluating the muticlass case, we used the same real data sets as above, but without grouping the output categories. As can be seen from Fig. 8.5, the performance of both IBL-DS and Hoeffding trees on the wine data is lower than

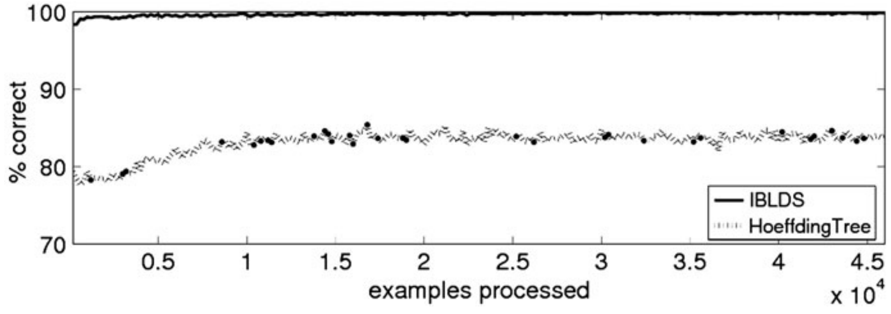

Fig. 8.6 Classification rate on the Shuttle data (multiclass)

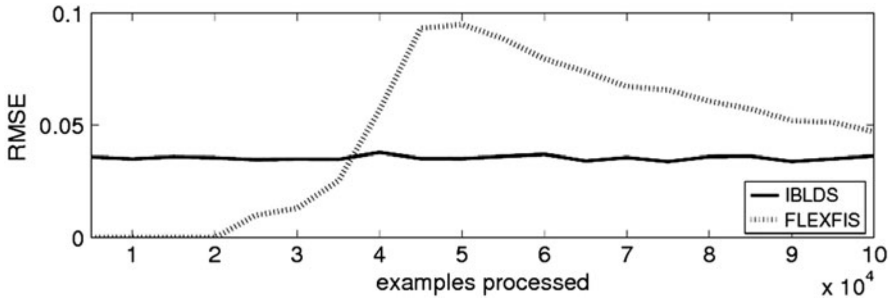

Fig. 8.7 RMSE for the hyperplane data (regression, linear case)

that for the binary case, an observation that is clearly expected. Still, IBD-DS remains superior on the whole stream. For the Shuttle data, Fig. 8.6 shows that the performance of IBL-DS remains almost the same, compared to the binary case, whereas the Hoeffding tree again starts with low classification rate and never exceeds the 85% limit.

8.5.2 Regression

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 $w^{\mathrm{T}}x$, where w is the normal vector of the hyperplane, but by the absolute value $\left|w^{\mathrm{T}}x\right|$. In other words, the problem is to predict the distance to the hyperplane. As an alternative, we also tried $\left(w^{\mathrm{T}}x\right)^2$, i.e., the squared distance. Again, ConceptDriftStream was used for simulating a concept drift by mixing two streams.

Figures 8.7 and 8.8 show the performance of IBL-DS and FLEXFIS, in terms of the root mean squared error (RMSE), for the (piecewise) linear and the quadratic case (and dimension $d = 4$), respectively. As can be seen, FLEXFIS performs quite

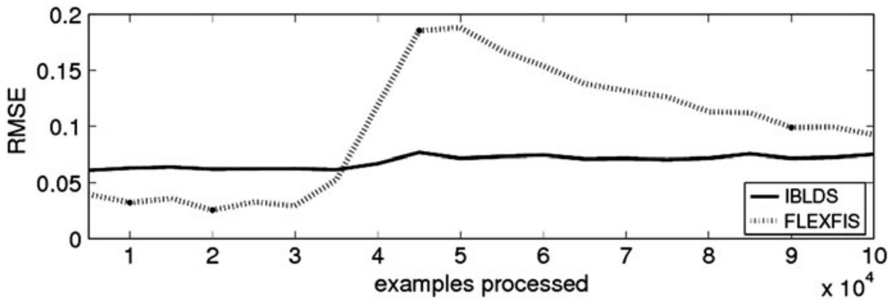

Fig. 8.8 RMSE for the hyperplane data (regression, quadratic case)

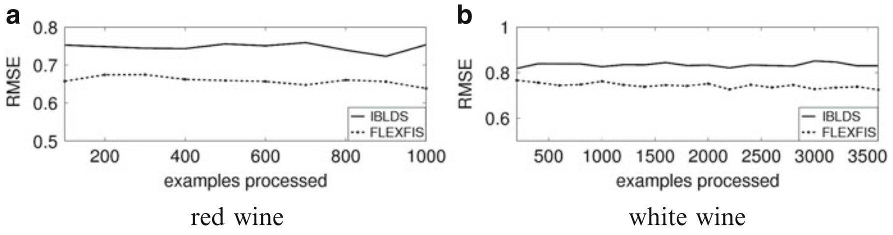

Fig. 8.9 RMSE for wine quality data set (regression)

well in the linear case. This behavior is expected and can easily be explained by its model structure (FLEXFIS uses fuzzy rules with linear functions as consequent parts). What is more interesting, however, is the observation that IBL-DS is much less affected by the concept drift, both in the linear and the quadratic case. In fact, while FLEXFIS deteriorates significantly and needs quite some time to recover, the performance of IBL-DS remains almost unchanged.

As a real data set, we again used the wine data, this time treating the quality level as a numerical value. Figure 8.9 shows that IBL-DS is slightly worse than FLEXFIS [24] on these two data sets.

8.6 Summary

We have presented an instance-based algorithm for classification and regression 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. The experiments presented in [6], complemented by those in this paper, 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. An implementation of IBL-DS under the MOA framework, along with a documentation, can be downloaded under the following address: http://www.uni-marburg.de/fb12/kebi/research/software/iblstreams/.

8.7 Distance Function

The distance function used in IBL-DS is an incremental variant of SVDM (Simple Value Difference Metric) which is a simplified version of the VDM (Value Difference Metric) distance measure [28] 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 $x=(f_1,\dots,f_\ell)\in D_1\times\cdots\times D_\ell$.

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.[5] 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:

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

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

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

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of VLDB 2003, the 29th International Conference on Very Large Data Bases. Berlin, Germany (2003)
2. Aha, D.W. (ed.): Lazy Learning. Kluwer Academic Publ., Dordrecht, Netherlands (1997)
3. Aha, D.W., Kibler, D.F., Albert, M.K.: Instance-based learning algorithms. Machine Learning **6**(1), 37–66 (1991)

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

4. Angelov, P.P., Filev, D.P., Kasabov, N.: Evolving Intelligent Systems. John Wiley and Sons, New York (2010)
5. Angelov, P.P., Lughofer, E., Zhou, X.: Evolving fuzzy classifiers using different model architectures. Fuzzy Sets and Systems **159**(23), 3160–3182 (2008)
6. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. Intelligent Data Analysis **11**(6), 627–650 (2007)
7. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. Journal of Machine Learning Research **11**, 1601–1604 (2010)
8. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 139–148. Paris, France (2009)
9. Bifet, A., Kirkby, R.: Massive Online Analysis Manual (2009)
10. Cormode, G., Muthukrishnan, S.: 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)
11. Dasarathy, B.V. (ed.): Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. IEEE Computer Society Press, Los Alamitos, California (1991)
12. Dawid, A.P.: Statistical theory: The prequential approach. In: Journal of the Royal Statistical Society-A, pp. 147:278–292 (1984)
13. Domingos, P.: Rule induction and instance-based learning: A unified approach. In: C. Mellish (ed.) Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 95, vol. 2, pp. 1226–1232. Morgan Kaufmann, Montral, Qubec, Canada (1995)
14. Domingos, P.: Unifying instance-based and rule-based induction. Machine Learning **24**, 141–168 (1996)
15. Domingos, P., Hulten, G.: A general framework for mining massive data streams. Journal of Computational and Graphical Statistics **12** (2003)
16. Frank, A., Asuncion, A.: UCI machine learning repository (2010). URL http://archive.ics.uci.edu/ml
17. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining data streams: A review. ACM SIGMOD Record, ACM Special Interest Group on Management of Data **34**(1) (2005)
18. Gama, J., Gaber, M.M.: Learning from Data Streams. Springer-Verlag, Berlin, New York (2007)
19. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Proceedings SBIA 2004, the 17th Brazilian Symposium on Artificial Intelligence, pp. 286–295. São Luis, Maranhão, Brazil (2004)
20. Gama, J., Medas, P., Rodrigues, P.: Learning decision trees from dynamic data streams. In: SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, pp. 573–577. ACM Press, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1066677.1066809
21. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: Proceedings of 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Paris, France (2009)
22. Hulten, G., Spencer, L., Domingos, P.: Mining timechanging data streams. In: Proceedings 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 97–106. San Francisco, CA, USA (2001)
23. Kolodner, J.L.: Case-based Reasoning. Morgan Kaufmann, San Mateo (1993)
24. Lughofer, E.: FLEXFIS: A robust incremental learning approach for evolving takagi-sugeno fuzzy models. IEEE Transactions on Fuzzy Systems **16**(6), 1393–1410 (2008)
25. Lughofer, E.: Evolving Fuzzy Systems: Methodologies, Advanced Concepts and Applications. Springer-Verlag, Berlin, Heidelberg (2011)
26. Oza, N.C., Russell, S.: Online bagging and boosting. Artificial Intelligence and Statistics pp. 105–112 (2001)
27. Salzberg, S.: A nearest hyperrectangle learning method. Machine Learning **6**, 251–276 (1991)

28. Stanfill, C., Waltz, D.: Toward memory-based reasoning. Communications of the ACM **29**, 1213–1228 (1986)
29. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Transactions on Systems, Man, and Cybernetics **15**(1), 116–132 (1985)
30. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning **4**, 161–186 (1989)
31. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2 edn. Morgan Kaufmann, San Francisco (2005)
32. Widmer, G. and Kubat, M.: Learning in the Presence of Concept Drift and Hidden Contexts. Machine Learning **23**, 69–101 (1996)