Recovery Analysis for Adaptive Learning from Non-Stationary Data Streams: Experimental Design and Case Study

Ammar Shaker and Eyke Hüllermeier

Department of Computer Science
University of Paderborn, Germany
{amma.shaker, eyke}@upb.de

—-draft version of a paper to be published in Neurocomputing—-

Abstract

The extension of machine learning methods from static to dynamic environments has received increasing attention in recent years; in particular, a large number of algorithms for learning from so-called *data streams* has been developed. An important property of dynamic environments is *non-stationarity*, i.e., the assumption of an underlying data generating process that may change over time. Correspondingly, the ability to properly react to so-called *concept change* is considered as an important feature of learning algorithms. In this paper, we propose a new type of experimental analysis, called *recovery analysis*, which is aimed at assessing the ability of a learner to discover a concept change quickly, and to take appropriate measures to maintain the quality and generalization performance of the model. We develop recovery analysis for two types of supervised learning problems, namely classification and regression. Moreover, as a practical application, we make use of recovery analysis in order to compare model-based and instance-based approaches to learning on data streams.

1 Introduction

The development of methods for learning from so-called *data streams* has been a topic of active research in recent years [Gaber et al., 2005, Gama and Gaber, 2007]. Roughly speaking, the key idea is to have a system that learns incrementally, and maybe even in real-time, on a continuous and potentially unbounded 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 already been developed for different machine learning and data mining tasks, such as clustering and classification [Gama, 2012].

An extension of data mining and machine learning methods to the setting of data streams comes with a number of challenges. In particular, the standard "batch mode" of learning, in which the entire data as a whole is provided as an input to the learning algorithm (or "learner" for short), is no longer applicable. Correspondingly, the learner is not allowed to make several passes through the data set, which is commonly done by standard methods in statistics and machine learning. Instead, the data must be processed in a single pass, which implies an incremental mode of learning and model adaptation.

Domingos and Hulten [Domingos and Hulten, 2003] 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 change (i.e., any change of the underlying data generating process) in a proper way and maintain a model that always reflects the current concept.

This last property is often emphasized as a key feature of learning algorithms, since non-stationarity is arguably the most important difference between static and dynamic environments. Indeed, while the idea of an incremental learning is crucial in the setting of data streams, too, it is not entirely new and has been studied for learning from static data before. The ability of a learner to maintain the quality and generalization performance of the model in the presence of concept drift, on the other hand, is a property that becomes truly important when learning under changing environmental conditions.

In this article, which is an extended version of the conference paper [Shaker and Hüllermeier, 2013], we propose a new type of experimental analysis, called *recovery analysis*. With the help of recovery analysis, we aim at assessing a learner's ability to maintain its generalization performance in the presence of concept drift. Roughly speaking, recovery analysis suggests a specific experimental protocol and a graphical presentation of the learner's performance that provides an idea of how quickly a drift is recognized, to what extent it affects the prediction performance, and how quickly the learner manages to adapt its model to the new condition. Our method makes use of real data, albeit in a modified and specifically prepared form, which is a main prerequisite for conducting controlled experiments under suitable conditions; therefore, it could be seen as a "semi-synthetic" approach.

Another contribution of the paper is an experimental study, which illustrate the usefulness of recovery analysis by comparing different types of learning methods with regard to their ability to handle concept drift. In particular, we turn our attention to the comparison of instance-based and model-based approaches to learning on data streams.

The remainder of the paper is organized as follows. By way of background, the next section recalls some important aspects of learning on data streams, with a specific emphasis on handling concept drift. Our method of recovery analysis is then introduced in Section 3. In Section 4, we outline the learning methods that we included in our case study, prior to describing the experiments and results in Section 5. We end the paper with some concluding remarks and an outlook on future work in Section 6.

2 Learning under concept drift

We consider a setting in which an algorithm $\mathcal{A}$ is learning on a time-ordered stream of data $S = (\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \ldots)$. Since we are mainly interested in *supervised learning*, we suppose that each data item $\boldsymbol{z}_t$ is a tuple $(\boldsymbol{x}_t, y_t) \in \mathbb{X} \times \mathbb{Y}$ consisting of an input $\boldsymbol{x}_t$ (typically represented as a vector) and an associated output y_t, which is the target for prediction. In classification, for example, the output space $\mathbb{Y}$ consists of a finite (and typically small) number of class labels, whereas in regression the output is a real number.

At every time point t, the algorithm $\mathcal{A}$ is supposed to offer a predictive model $\mathcal{M}_t : \mathbb{X} \to \mathbb{Y}$ that has been learned on the data seen so far, i.e., on the sequence $S_t = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_t)$. Given a query input $\boldsymbol{x} \in \mathbb{X}$, this model can be used to produce a prediction

$$\hat{y} = \mathcal{M}_t(\boldsymbol{x}) \in \mathbb{Y}$$

of the associated output. The accuracy of this prediction can be measured in terms of a loss function $\ell : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}$, such as the $0/1$ loss in the case of classification or the squared error loss in regression. Then, the prediction performance of $\mathcal{M}_t$ is defined in terms of the expected loss, where the expectation is taken with respect to an underlying probability measure $\mathbf{P}$ on $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. This probability measure formally specifies the data generating process.

If the algorithm $\mathcal{A}$ is truly incremental, it will produce $\mathcal{M}_t$ solely on the basis of $\mathcal{M}_{t-1}$ and $\boldsymbol{z}_t$, that is, $\mathcal{M}_t = \mathcal{A}(\mathcal{M}_{t-1}, \boldsymbol{z}_t)$. In other words, it does not store the entire sequence of previous observations $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}$. Many algorithms, however, store at least a few of the previous data points, typically the most recent ones, which can then also be used for model adaptation. In any case, the number of observations that can be stored is normally assumed to be finite, which excludes the possibility of memorizing the entire stream. A *batch learner* $\mathcal{A}_B$, on the other hand, would produce the model $\mathcal{M}_t$ on the basis of the complete set of data $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t\}$. Note that, although $\mathcal{A}$ and $\mathcal{A}_B$ have seen the same data, $\mathcal{A}_B$ can exploit this data in a more flexible way. Therefore, the models produced by $\mathcal{A}$ and $\mathcal{A}_B$ will not necessarily be the same.

As mentioned before, the data generating process is characterized by the probability measure $\mathbf{P}$ on $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. Under the assumptions of stationarity and independence, each new observation $\boldsymbol{z}_t$ is generated at random according to $\mathbf{P}$, i.e., the probability to observe a specific $\boldsymbol{z} \in \mathbb{Z}$ is given by[1]

$$\mathbf{P}(\boldsymbol{z}) = \mathbf{P}(\boldsymbol{x}, y) = \mathbf{P}(\boldsymbol{x}) \cdot \mathbf{P}(y \,|\, \boldsymbol{x}) \ .$$

Giving up the assumption of stationarity (while keeping the one of independence), the probability measure $\mathbf{P}$ generating the next observation may possibly change over time. Formally, we are thus dealing, not with a single measure $\mathbf{P}$, but with a sequence of measures $(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \ldots)$, assuming that $\boldsymbol{z}_t$ is generated by $\mathbf{P}_t$. One speaks of a *concept change* if these measures are not all equal [Ben-David et al., 2004].

In the literature, a distinction is made between different causes and types of concept change [Gama, 2010]. The first type refers to a sudden, abrupt change of the underlying concept to be learned and is often called *concept shift* ($\mathbf{P}_t$ is very different from $\mathbf{P}_{t-1}$). Roughly speaking, in

[1]We slightly abuse notation by using the same symbol for the joint probability and its marginals.

the case of a concept shift, any knowledge about the old concept may become obsolete and the new concept has to be learned from scratch. The second type refers to a gradual evolution of the concept over time. In this scenario, old data might still be relevant, at least to some extent. Finally, one often speaks about *virtual* concept drift if the change only concerns $\mathbf{P}(\boldsymbol{x})$, i.e., the distribution of the inputs, while the concept itself, i.e., the conditional distribution $\mathbf{P}(y \mid \boldsymbol{x})$, remains unchanged [Widmer and Kubat, 1993]. To guarantee optimal predictive performance, an adaptation of the model might also be necessary in such cases. In practice, virtual and real concept drift will often occur simultaneously.

Learning algorithms can handle concept change in a direct or indirect way. In the indirect approach, the learner does not explicitly attempt to detect a concept drift. Instead, the use of outdated or irrelevant data is avoided from the outset. This is typically accomplished by considering only the most recent data while ignoring older observations, e.g., by sliding a window of fixed size over a data stream. To handle concept change in a more direct way, appropriate techniques for discovering the drift or shift are first of all required, for example based on statistical tests.

3 Recovery analysis

In practical studies, data streams are of course never truly infinite. Instead, a "stream" is simply a large data set in the form of a long yet finite sequence $S = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_T)$. In experimental studies, such streams are commonly used to produce a performance curve showing the generalization performance of a model sequence $(\mathcal{M}_t)_{t=1}^{T}$ over time. Although many of these studies are interested in analyzing the ability of a learner to deal with concept drift, such an analysis is hampered by at least two problems:

- Ignorance about drift: First, for a real data stream S, it is normally not known whether it contains any concept drift, let alone when such a drift occurs.
- Missing baseline: Second, even if a concept drift is known to occur, it is often difficult to assess the performance of a learner or to judge how well it recovers after the drift, simply because a proper *baseline* is missing: The performance that could in principle be reached, or at least be expected, is not known.

Obviously, these problems are less of an issue if data is generated synthetically. In fact, for synthetic data, the "ground truth" is always known. Moreover, synthetic data has the big advantage of enabling *controlled* experiments. For example, one might be interested in how an algorithm reacts to a drift depending on certain characteristics of the drift, such as its strength and duration. While real data will (at most) contain a single drift, these characteristics can easily be varied in experiments with synthetic data.

On the other hand, purely synthetic data begs the danger of being unrealistic or overly idealized. Therefore, in our approach to recovery analysis, we try to find a reasonable compromise by using a setting that we would qualify as "semi-synthetic". As will be explained in more detail in the remainder of this section, we are making use of real data, albeit in a "manipulated" form that circumvents the above problems and allows us to conduct controlled experiments.

3.1 Main idea and experimental protocol

Instead of using a single data stream, our idea is to work with three streams in parallel, two "pure streams" and one "mixture". The pure streams

$$S_A = (\boldsymbol{z}_1^a, \boldsymbol{z}_2^a, \ldots, \boldsymbol{z}_T^a)$$
$$S_B = (\boldsymbol{z}_1^b, \boldsymbol{z}_2^b, \ldots, \boldsymbol{z}_T^b)$$

are supposed to be stationary and generated, respectively, according to distributions $\mathbf{P}_A$ and $\mathbf{P}_B$; in the case of real data, stationarity of a stream can be guaranteed, for example, by permuting the original stream at random.[2] These two streams must also be compatible in the sense of sharing a common data space $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. The mixture stream $S_C = (\boldsymbol{z}_1^c, \boldsymbol{z}_2^c, \ldots, \boldsymbol{z}_T^c)$ is produced by randomly sampling from the two pure streams:

$$\boldsymbol{z}_t^c = \begin{cases} \boldsymbol{z}_t^a & \text{with probability } \lambda(t) \\ \boldsymbol{z}_t^b & \text{with probability } 1 - \lambda(t) \end{cases} \tag{1}$$

A concept drift can then be modeled, for example, by specifying the (time-dependent) sample probability $\lambda(t)$ as a sigmoidal function:

$$\lambda\,(t) = \left(1 + \exp\left(\frac{4(t - t_0)}{w}\right)\right)^{-1} \quad . \tag{2}$$

This function has two parameters: t_0 is the mid point of the change process, while w controls the length of this process. Using this transition function, the stream S_C is obviously drifting "from S_A to S_B": In the beginning, it is essentially identical to S_A, in a certain time window around t_0, it moves away from S_A toward S_B, and in the end, it is essentially identical to S_B. Thus, we have created a gradual concept drift with a rate of change controlled by w.

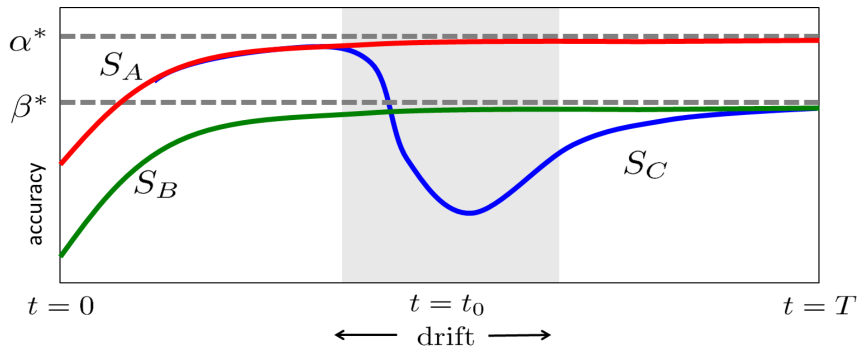

Figure 1: Schematic illustration of a recovery analysis: The three performance curves are produced by training models on the pure streams S_A and S_B, as well as on the mixed stream S_C, each time using the same learner $\mathcal{A}$. The region shaded in grey indicates the time window in which the concept drift (mainly) takes place. While the concept is drifting, the performance (here, for example, the classification rate) on S_C will typically drop to some extent.

[2]Permutation of data streams is also used in [Žliobaitė, 2014], albeit in a different way and for a different purpose.

Now, suppose the same learning algorithm $\mathcal{A}$ is applied to all three streams S_A, S_B and S_C. Since the first two streams are stationary, we expect to see a standard learning curve when plotting the generalization performance (for example, the classification accuracy) as a function of time. In the following, we denote the performance curves for S_A and S_B by $\alpha(t)$ and $\beta(t)$, respectively. These curves are normally concave, showing a significant increase in the beginning before reaching a certain saturation level later on; see Figure 1 for an illustration. The corresponding saturation levels α^* and β^* provide important information, namely information about the best performance that can be expected by the learner $\mathcal{A}$ on the pure streams S_A and S_B, respectively.

Even more interesting, however, is the performance curve $\gamma(t)$ for the stream S_C, which exhibits concept drift. In the beginning, this curve will be effectively identical to the curve for S_A, so that the learner $\mathcal{A}$ should reach the level α^*. Then, upon the beginning of the concept drift, the performance is expected to drop, and this decrease is supposed to continue until the drift ends and the learner $\mathcal{A}$ starts to recover. Eventually, $\mathcal{A}$ may (or may not) reach the level β^*. This level is indeed an upper bound on the asymptotic performance, since $\mathcal{A}$ cannot do better even when being trained on S_B from the very beginning. Thus, reaching this level indicates an optimal recovery.

Obviously, the performance curve for S_C provides important information about the ability of $\mathcal{A}$ to deal with concept drift. In particular, the minimum of this curve indicates how strongly $\mathcal{A}$ is affected by the concept drift. Moreover, the curve informs about how quickly the performance deteriorates (giving an idea of how sensitive $\mathcal{A}$ is), how much time $\mathcal{A}$ needs to recover, and whether or not it manages to recover optimally.

3.2 Bounding the optimal generalization performance

As explained above, the performance curve produced by a learner $\mathcal{A}$ on the stream S_C is expected to decrease while this stream is drifting from S_A to S_B. In order to judge the drop in performance, not only relatively in comparison to other learners but also absolutely, it would be desirable to have a kind of reference performance as a baseline. This leads to an interesting question: Is it possible to quantify our expectations regarding the drop in performance? More specifically, what is the optimal generalization performance

$$\gamma^*(t) \; = \; \sup_{\mathcal{M} \in \mathbf{M}} \gamma_{\mathcal{M}}(t) \tag{3}$$

we can expect on the stream S_C at time t? Here $\mathbf{M}$ is the underlying model class (i.e., the class of models that $\mathcal{A}$ can choose from), and $\gamma_{\mathcal{M}}(t)$ denotes the generalization performance of a model $\mathcal{M} \in \mathbf{M}$ on the mixture distribution (1), i.e.,

$$\mathbf{P}_C(t) = \lambda(t)\mathbf{P}_A + (1 - \lambda(t))\mathbf{P}_B \;\; .$$

Our experimental setup indeed allows for answering this question in a non-trivial way. To this end, we exploit knowledge about the performance levels $\alpha(t)$ and $\beta(t)$ that can be reached on S_A and S_B, respectively. Thus, there are models $\mathcal{M}_A, \mathcal{M}_B \in \mathbf{M}$ whose performance is $\alpha_{\mathcal{M}_A}(t) = \alpha(t)$ and $\beta_{\mathcal{M}_B}(t) = \beta(t)$. Now, suppose we were to apply the model $\mathcal{M}_A$ on the stream S_C. What is the expected generalization performance? To be concrete, consider the case of classification, with the classification rate as a performance measure; this measure assumes values in the unit interval, with 0 and 1 indicating the worst and best performance, respectively.

If an example $(\boldsymbol{x}, y)$ on S_C is generated according to $\mathbf{P}_A$, the generalization performance of $\mathcal{M}_A$ on this example is the same as on S_A, namely $\alpha_{\mathcal{M}_A}(t)$. Otherwise, if the example is generated according to $\mathbf{P}_B$, nothing can be said about the performance of $\mathcal{M}_A$; thus, we can only assume the worst case performance of 0. Since the first case occurs with a probability of $\lambda(t)$ and the second one with a probability of $1 - \lambda(t)$, the overall expected performance of $\mathcal{M}_A$ is given by

$$\lambda(t) \cdot \alpha_{\mathcal{M}_A}(t) + (1 - \lambda(t)) \cdot 0 = \lambda(t) \cdot \alpha_{\mathcal{M}_A}(t) \ .$$

Using the same line of reasoning, the performance of the model $\mathcal{M}_B$ on the stream S_C is given by $(1 - \lambda(t))\beta_{\mathcal{M}_B}(t)$. Thus, choosing optimally from the two candidate models , we can at least guarantee the performance

$$\gamma^\bullet(t) = \max\left\{\lambda(t) \cdot \alpha_{\mathcal{M}_A}(t), (1 - \lambda(t)) \cdot \beta_{\mathcal{M}_B}(t)\right\} \ . \tag{4}$$

Obviously, since the supremum in (3) is not only taken over $\{\mathcal{M}_A, \mathcal{M}_B\}$ but over the entire model class $\mathbf{M}$, $\gamma^\bullet(t)$ is only a lower bound on the optimal performance $\gamma^*(t)$, that is, $\gamma^\bullet(t) \leq \gamma^*(t)$. We also remark that, if the performance levels $\alpha(t)$ and $\beta(t)$ are already close enough to the optimal levels α^* and β^*, respectively, then (4) can be written more simply as

$$\gamma^\bullet(t) = \max\left\{\lambda(t) \cdot \alpha^*, (1 - \lambda(t)) \cdot \beta^*\right\} \ . \tag{5}$$

Strictly speaking, this estimation is not correct, since α^* and β^* are only limit values that will not necessarily be attained. Practically, however, this is of no importance, especially since we have to work with estimations of these values anyway.

The above estimation can be generalized to other performance measures or loss functions in a straightforward way, provided these measures assume values in a bounded range (not necessarily $[0, 1]$). In fact, as soon as the range is bounded, the worst performance can easily be derived. For example, consider the case of regression with the root mean square error as a loss function, and assume that $\mathbb{Y} = [y_{min}, y_{max}]$. Then, a bound on the worst performance of an optimal model, analogous to (5), is

$$\begin{aligned}\gamma^\bullet(t) = \min\Bigg\{&\sqrt{(\alpha^*)^2 \cdot (1 - \lambda(t)) + l^2 \cdot \lambda(t)}, \\ &\sqrt{(\beta^*)^2 \cdot (\lambda(t)) + l^2 \cdot (1 - \lambda(t))}\Bigg\} \ ,\end{aligned} \tag{6}$$

where $l = |y_{min} - y_{max}|$, $(\alpha^*)^2$ is the mean square error of $\mathcal{M}_A$ on S_A and $(\beta^*)^2$ the mean square error of $\mathcal{M}_B$ on S_B. Note that, in contrast to the (the-higher-the-better) classification rate, we are now estimating a (the-less-the-better) loss. Therefore, (6) is an upper bound on the optimal performance: $\gamma^*(t) \leq \gamma^\bullet(t)$.

Practically, the above estimate is not unproblematic. First, the output variable in regression does normally not have natural bounds y_{min} and y_{max}. Moreover, even if such limits can be established, the bound (6) will be very loose. A more realistic bound can be obtained by estimating the true performance of $\mathcal{M}_A$ on S_B and the true performance of $\mathcal{M}_B$ on S_A. In our experimental setting, this can be done by computing the average performance $\tilde{\alpha}$ of the model $\mathcal{M}_A$ (once it has stabilized and reached the performance α^* on S_A) on the data S_B. Likewise, the average performance $\tilde{\beta}$ of $\mathcal{M}_B$ can be obtained on S_A. Replacing l^2 in (6) by these estimates then yields

$$\begin{aligned}\gamma^*(t) \leq \min \Bigg\{ &\sqrt{(\alpha^*)^2 \cdot (1 - \lambda(t)) + (\tilde{\alpha})^2 \cdot \lambda(t)}, \\ &\sqrt{(\beta^*)^2 \cdot (\lambda(t)) + (\tilde{\beta})^2 \cdot (1 - \lambda(t))} \Bigg\} \ .\end{aligned} \tag{7}$$

3.3 Recovery measures

The major goal of our recovery analysis is to provide insight into how an algorithm behaves in the presence of a concept drift. This behavior is most clearly represented by the recovery curves that are produced as a graphical output (cf. Figure 1). Yet, in some cases, it may also be desirable to have a more quantitative summary of the algorithms's recovery, comparable to the use of metrics for performance evaluation [Gama et al., 2013], even if a quantification in terms of a scalar measure will necessarily come along with a certain loss of information. In this section, we propose concrete examples of measures of such kind.

The **duration** measures the (relative) length of the recovery phase or, more specifically, the sub-optimal performance of the algorithm. It is defined as

$$\frac{t_2 - t_1}{T} \in [0, 1] \ ,$$

where t_1 is the time at which the curve S_C drops below 95% of the performance curve S_A, t_2 the time at which S_C recovers up to 95% of the performance of S_B, and T the length of the entire stream.

The **maximum performance loss** measures the maximal drop in performance. In the case of the classification rate, it compares S_C with the pointwise minimum

$$S(t) = \min\{S_A(t), S_B(t)\}$$

as a baseline and derives the maximum relative performance loss

$$\max_{t \in T} \frac{S(t) - S_C(t)}{S(t)} \tag{8}$$

as compared to this baseline. In the case of regression, the measure is defined analogously as

$$\max_{t \in T} \frac{S_C(t) - S(t)}{S(t)} = \max_{t \in T} \frac{S_C(t) - \max\{S_A(t), S_B(t)\}}{\max\{S_A(t), S_B(t)\}} \ .$$

3.4 Defining pure streams

As mentioned above, the two streams S_A and S_B have to be compatible in the sense of sharing a common data space $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$. An important practical question is of course where these streams are coming from. Ideally, two separate data sets of this kind are directly available. An example is the wine data from the UCI repository repository [Frank and Asuncion, 2010], with input attributes describing a wine in terms of physicochemical properties and the output a quality level between 1 and 10. This data comes in two variants, one for red wine and one for white wine. Thus, using the first data set for S_A and the second one for S_B, one can simulate a transition from rating red wine to rating white wine.

If ideal data of that kind is not available, it needs to be produced in one way or the other, preferably on the basis of a single source data stream S. In the following, we suggest a few possibilities for such a construction. However, we would not like to prescribe one specific way of data generation. In fact, even in reality, there is not only one type of concept drift, or a single source for a drift. Instead, concept drift can occur for many reasons, which one may try to mimic.

In the above example of wine data, one could also imagine a single data set in which the type of wine (red or white) is added as a binary attribute. The other way around, instead of merging two data sets into a single one, one could split a data set S on the basis of a binary attribute X, using those examples with the first value for S_A and those with the second value for S_B (and removing X itself from both data sets). Again, the idea is to have a hidden variable that defines the context. Obviously, this approach can be generalized from binary to any type of attributes, simply by using appropriate splitting rules.

In some data sets S, the role of the attributes as either input (predictor) variable or output (response) variable is not predetermined. If the data contains two attributes A and B that are both of the same kind and can both be used as outputs, then S_A and S_B can be obtained , respectively, by using these attributes as a target for prediction.

Obviously, there are many other ways of "manipulating" a data set S in order to simulate a concept change, for example by changing the order of some of the input attributes. In the experiments later on, we also apply another simple idea: S_A is given by the original data S, while S_B is constructed by copying S and reversing or shifting the output attribute.

3.5 Further practical issues

Our discussion of recovery analysis so far has left open some important practical issues that need to be addressed when implementing the above experimental protocol. An obvious question, for example, is how to determine the generalization performance of a model $\mathcal{M}_t$ (induced by the learner $\mathcal{A}$) at time t, which is needed to plot the performance curve. First of all, it is clear that this generalization performance can only be estimated on the basis of the data given, just like in the case of batch learning from static data. In the literature, two procedures are commonly used for performance evaluation on data streams:[3] (i) In the *holdout* approach, the training and the test phase of a learner are interleaved as follows: The model is trained incrementally on a block of M data points 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. (ii) In the *test-then-train* approach, every instance is used for both training and testing. First, the current model is evaluated on the observed instance, and then this instance is used for model adaptation. The evaluation

[3] Both procedures are implemented in the MOA framework [Bifet and Kirkby, 2009].

measure in this scenario is updated incrementally after each prediction(prequential evaluation). This approach can also be applied in a chunk mode, where a block of size M (instead of a single instance) is used for evaluation first and training afterwards.

The test-then-train procedure has advantages over the holdout approach. For example, it obviously makes better use of the data, since each example is used for both training and testing. More importantly, it avoids “gaps” in the learning process: In the holdout approach, $\mathcal{A}$ only learns on the training blocks but stops adaptation on the evaluation blocks in-between. Such gaps are especially undesirable in the presence of a concept drift, since they may bias the assessment of the learner’s reaction to the drift. This is the main reason for why we prefer the test-then-train procedure for our implementation of recovery analysis.

Another practical issue concerns the length of the data streams. In fact, to implement recovery analysis in a proper way, the streams should be long enough, mainly to make sure that the learner $\mathcal{A}$ will saturate on all streams: First, it should reach the saturations levels α^* and β^* on S_A and S_B, respectively. Moreover, the streams should not end while $\mathcal{A}$ is still recovering on S_C; otherwise, one cannot decide whether or not an optimal recovery (reaching β^*) is accomplished.

Finally, to obtain smooth performance curves, we recommend to repeat the same experiment with many random permutations S_A and S_B of the original streams, and to average the curves thus produced. Obviously, averaging is legitimate in this case, since the results are produced for the same data generating processes (specified by the distributions $\mathbf{P}_A$, $\mathbf{P}_B$ and their mixture $\mathbf{P}_C$).

4 A comparison of algorithms

The remainder of the paper is devoted to a case study, in which we compare a number of different learning algorithms with respect to their ability to handle concept drift. A main goal of this study is to see whether there are important differences between the methods, and whether recovery analysis helps uncover them. In particular, we are interested in a comparison between *model-based* and *instance-based* approaches to learning on data streams.

Model-based machine learning methods induce a general model (theory) from the data and use that model for further reasoning. As opposed to this, instance-based learning (IBL) algorithms simply store parts of the data itself and defer its processing until a prediction (or some other type of query) is actually requested, a property which qualifies them as a *lazy* learning paradigm [Aha, 1997]. Predictions are then derived by combining the information provided by the stored examples.

While a lot of effort has been invested in the development of model-based approaches, instance-based learning on data streams has not received much attention so far, with only a few notable exceptions [Law and Zaniolo, 2005, Beringer and Hüllermeier, 2007, Shaker and Hüllermeier, 2012]. This is arguably surprising, especially in light of some appealing properties of IBL. Obviously, IBL algorithms are inherently incremental, since adaptation basically comes down to editing the current case base (which essentially represents the model) by either 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, since the incremental update of a model, such as a classification tree or a fuzzy system, is often quite complex and in many cases assumes the storage of a considerable amount of additional information.

Regarding the handling of concept drift, model adaptation in IBL 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 model cannot be answered in general but depends on the particular model at hand. In any case, maintaining an implicit concept description by storing observations, as done by IBL, is very simple, facilitates "forgetting" examples that seem to be outdated and allows for flexible updating and adaptation strategies. 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.

	Learning problem			Model type			Approach		Change detection	
	Binary Classification	Multiclass Classification	Regression	Tree Structure	Rule-based	Instance-based	Fuzzy	Hoeffding Bound	Page-Hinkley	Stat. Hypot. Testing
eFPT	✓			✓			✓			✓
IBLStreams	✓	✓	✓			✓				✓
FLEXFIS			✓		✓		✓			
AdpHoef	✓	✓		✓				✓	✓	
AMRules			✓		✓			✓	✓	
FIMTDD			✓	✓				✓	✓	

Table 1: Summary of the learning algorithms and their main characteristics.

Table1 provides a summary of the methods that we included in our study as well as their main properties. Our selection was based on the following considerations:

- Since one of our main goals is to compare model-based and instance-based learning, we included both types of algorithms. Yet, since the former clearly prevail in the literature, there are five model-based and only one instance-based approach.
- The algorithms should be representative and reflect the state of the art. Many of the methods are therefore based on tree induction or rule learning.
- The idea of adaptive learning in dynamical environments has not only received attention in machine learning but also in the computational intelligence community, where it is inten-

sively studied under the notion of “evolving fuzzy systems” [Angelov et al., 2010]. Therefore, we also included methods for the adaptive, incremental learning of fuzzy systems on data streams.

Last but not least, the availability of implementations was of course an important criterion, too. All approaches except FLEXFIS are implemented and run under MOA [Bifet and Kirkby, 2009], a framework for learning on data streams that includes data stream generators, different methods for performance evaluation and implementations of several classifiers. MOA is also able to interact with the popular WEKA machine learning environment [Witten and Frank, 2005]. IBLStreams and eFPT can be downloaded as extensions of MOA, while the other methods can be acquired from the 2013.11 release of MOA.

A brief description of each of the algorithms, without going into technical detail, is given in Appendix A. With regard to the handling of concept drift, it is noticeable that most of the methods do not only react in an indirect way, by adapting to the most recent observations. Instead, by using statistical tools for drift detection, the algorithms are made “drift-aware”. Once a drift has been detected, they start to run in a specific mode, using strategies for learning and model adaptation that differ from those in the “normal” mode.

5 Experiments and results

All data sets are collected from the UCI[4] repository [Frank and Asuncion, 2010] unless otherwise stated. Table 2 provides a summary of the data sets and their main properties (attributes, size, nature and source).

We conducted experiments with three different drift settings. To this end, we varied the speed of change by modifying the width parameter w in the sigmoid function (2). More specifically, we control the angle θ of the tangent of this function at $t = t_0$, which is inversely proportional to w, i.e., $\tan(\theta) = \frac{1}{w}$. Figure 2 depicts the three drift velocities:

- $\theta = \frac{\pi}{75}$ for a slow concept drift,
- $\theta = \frac{\pi}{30}$ for concept drift with a modest speed,
- $\theta = \frac{\pi}{2}$ for a sudden concept change (concept shift).

Table 2 also contains the size of the evaluation window, which is always adapted to the size of the data set. In fact, since the data sets do not share the same size, we define the length of the window as $w = \frac{\ell}{100 \tan(\theta)}$, where ℓ is the size of the data set. Thus, given the drift angle θ, the proportion of the entire stream that is subject to drift is the same for all data sets.

5.1 Binary classification

RandomTrees This is a synthetic data set obtained from the RandomTreeGenerator class offered by MOA. The data is produced by randomly generated decision trees (splits are randomly made on

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

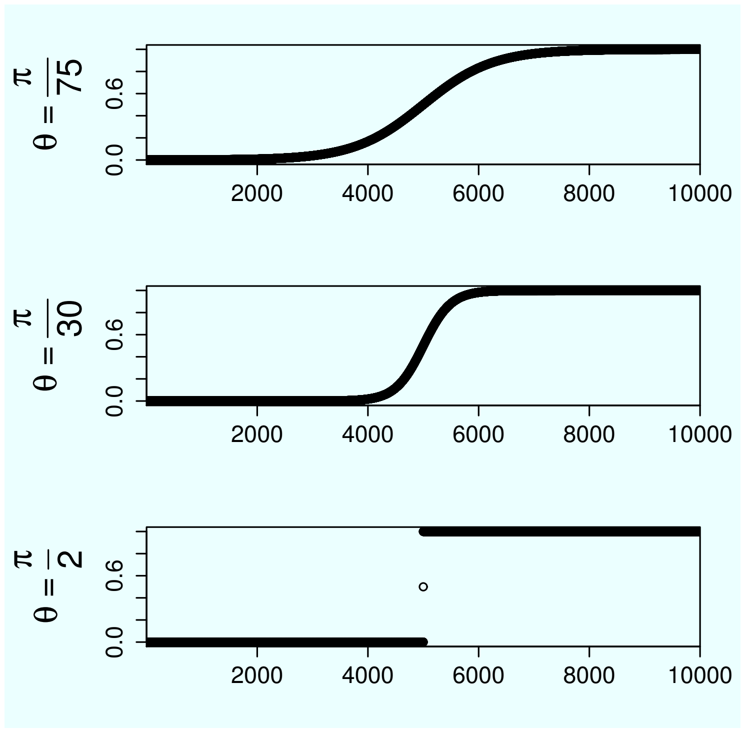

Figure 2: Sigmoid transition function modeling different types of concept drift: slow drift (top), moderate drift (middle), sudden drift (bottom).

attribute values and class labels are randomly assigned to the leaf nodes). Instances are generated according to a uniform distribution on the input space. The number of attributes and classes can be provided as parameters. We use 4 numerical attributes to describe each instance. Moreover, we produced two copies of this data using different random seeds. These copies define our pure streams S_A and S_B.

Figure 3 shows the recovery curves for different drift velocities. As can be seen, the different approaches reach different saturation levels α^* and β^* on the pure streams. Despite showing different "recovery patterns", they all manage to recover from level α^* to level β^*. For example, eFPT exhibits a rather smooth recovery with almost no drop in performance, whereas AdpHoef deteriorates quite significantly. Interestingly, all approaches seem to perform better than the estimated lower bound.

Mushroom This data set represents a binary classification problem with the objective of predicting whether a given mushroom is edible or poinsonous. The data takes samples of 23 species of gilled mushrooms. Each mushroom is described by 22 nominal attributes (we removed one of those attributes, which has missing values in 30% of the cases). This data set is relatively large, compared to other real data sets, with about 8124 instances. We used the original data as a first pure stream S_A and an "inverted copy" as a pure stream S_B. In this copy, we simply inverted the target attribute. Thus, the problem on the mixture stream S_C changes from predicting whether a mushroom is edible to predicting whether it is poinsonous.

Figure 4 shows that both IBLStreams and AdpHoef recover quickly and very well to the optimal performance curve. Nonetheless, AdpHoef shows a drastic drop in performance during the drift. This could be explained by the cost for repairing the model: The original tree becomes invalid and needs to be transformed into a valid one through successive replacements of internal nodes or complete subtrees. For eFPT, on the other hand, it seems that the more drastic the change, the better the recovery. This might be due to the use of statistical tests for discovering changes: the more obvious the change, the easier it can be detected.

Breast Cancer Wisconsin This data comprises a set of reported clinical cases with the task of classifying them as benign or malignant. Each case is described with 9 integer numbers between 1 and 10. Removing the cases with missing values (about 2% of the whole data), the number of remaining cases is 683. Like for mushroomMCR, we produce a stream S_B by "inversion" of the original stream.

This problem appears to be quite difficult. As can be seen in Figure 5, only IBLStreams recovers well, but even this learner shows a significant drop in performance (below the estimated bound) during the drift. The tree-based methods eFPT and AdpHoef never manage to recover. Apparently, the data stream is too short to accomplish the complex process of tree reconstruction.

5.2 Multiclass classification

5-class RandomTrees This data is used in the same way as for binary classification, but now with five classes. Again, we created two copies using different random seeds.

Figure 6 shows that both IBLStreams and AdpHoef recover well, although AdpHoef is a bit slower. It seems that the quicker the drift, the deeper the performance drop for IBLStreams. One explanation for this observation is that, in the case of a (detected) drift, IBLStreams removes an amount of data from the case base proportional to the error rate. This removal could harm its predictive performance during the drift, especially when the case base becomes almost empty.

Page Blocks The task in this data set is to classify blocks in an image into one of five classes: text, horizontal line, picture, vertical line, graphic. The blocks are detected through segmentation and described by various attributes and pixel information. In total, the data contains 5473 blocks, each described by 10 numerical attributes. We simulate a drift by means of a cyclic shift of the class labels, replacing label i by $1 + i \mod 5$.

Figure 7 shows that only IBLStreams recovers, whereas AdpHoef remains on a very low level of performance after the drift occurred.

Letter Recognition The problem here is to recognize 26 English letters, which are written in 20 different fonts. After post-processing of the original graphical representation, each letter is described by 16 primitive numerical features. In total, the data set comprises 20,000 examples. Like previously, we simulate a drift by shifting the class labels in a circular manner.

As can be seen in Figure 8, the results are qualitatively similar to the 5-class RandomTree data. In particular, only IBLStreams is able to recovers.

5.3 Regression

HyperDistance This is another synthetic data set that we produced by modifying the HyperplaneGenerator (for classification data) in MOA as follows: The output for an instance $\boldsymbol{x}$ is not determined by the sign of $\boldsymbol{w}^\top \boldsymbol{x}$, where $\boldsymbol{w}$ is the normal vector of the hyperplane, but by the absolute value $y = f_1(\boldsymbol{x}) = |\boldsymbol{w}^\top \boldsymbol{x}|$. In other words, the problem is to predict the distance of $\boldsymbol{x}$ to the hyperplane. As an alternative, we also used the cubic distance $y = f_3(\boldsymbol{x}) = (\boldsymbol{w}^\top \boldsymbol{x})^3$, which is arguably more difficult to learn than the absolute distance.

In a first experiment, we generated S_A using f_1 and S_B using f_3; thus, the drift is from the simpler to the more difficult problem. Figure 9 shows the recovery curves of the learning methods. As can be seen, both IBLStreams and FLEXFIS have a relatively small error on the first problem, compared to AMRules and FIMTDD. During the drift, both suffer from a high drop in performance, which is visible as a bell-shaped peak. Nonetheless, IBLStreams recovers quite well, whereas FLEXFIS fails to do so. AMRuless and FIMT, on the other hand, show very similar performance curves, and both manage to recover in a comparable way.

In a second experiment, we changed the order of the problems: S_A was generated using f_3 and S_B using f_1; thus, the drift is now from the more to the less difficult problem. As shown in Figure 10, IBLStreams and FLEXFIS suffer from the same drop in performance, but this time they both succeed to recover. Again, AMRules and FIMTDD share the same performance, with the exception that FIMTDD smoothly and perfectly recovers without any loss in the end.

Bank32h This data is acquired from the DELVE[5] repository. It is generated by simulating the way in which the customers from different rural areas select and reject banks, based on the waiting queues in a series of banks they successively visit in order to accomplish tasks of different complexity. The data set contains 8192 cases, each of which has 32 numeric attributes. The second pure stream S_B is created by "inverting" the target values: For an example $(\boldsymbol{x}, y)$, the original output y is replaced by $y_{max} + y_{min} - y$, where y_{min} and y_{max} are the smallest and largest values in the data set.

Figure 11 shows that all methods manage to recover quite well on this data, despite a visible drop in performance in the middle of the drift region.

Census-House This data was again obtained from the DELVE repository. It is a collection of data sets designed on the basis of the US census data collected in 1999. We use the House8L data, the purpose of which is to predict the median price of houses in different regions based on the demographic properties. Each house has 8 attributes, with a total number of 22784 houses. The second stream S_B was created by inverting the original outputs in the same way as in the previous data set.

[5] `http://www.cs.utoronto.ca/~delve/data/datasets.html`

As can be seen in Figure 12, all approaches again recover quite well, except for FIMTDD, which has a comparatively long recovery phase.

The results are generated by plotting the average evaluation, accuracy for classification and the root mean square error for regression, on each data chunk. To this end, we use the test-then-train approach which works in an sample by sample way.

5.4 Recovery measures

In the above experiments, we also derived the quantitive recovery measures proposed in Section 3.3. Figures 13, 14 and 15 plot the duration against the maximal performance loss for the different learning algorithms on the different data sets. As can be seen, IBLStreams is often better than the other methods in terms of both measures, at least for classification problems.

5.5 Summary of the experiments

Although our experimental study is quite comprehensive, it is of course neither complete nor fully conclusive. In fact, the main goal of this study is not to "prove" the superiority of any method over any other method, but rather to illustrate the potential of our recovery analysis. Nevertheless, from the results we obtained, we can at least extract some trends and draw some preliminary conclusions, which are summarized in the following observations:

- For classification problems, both IBLStreams and Hoeffding trees are recovering quite well on synthetic data sets. On real data sets, however, Hoeffding trees recover less quickly and tend to require a larger amount of data. In fact, while they successfully recover on the large streams (see Figure 4), they do not completely recover on relatively short ones (see Figures 5, 7 and 8). Besides, the larger the number of classes in the problem, the more data the Hoeffding trees need to recover. This is exemplified, for example, by comparing the good recovery on the binary data (8K instances) in Figure 5 with the incomplete recovery on the 26 classes problem (20K instances) in Figure 8.
- For regression problems, FLEXFIS and IBLStreams are quite strong in terms of absolute accuracy, compared to the other methods (FIMTDD and AMRules). Nevertheless, they tend to have slightly higher peaks (maximum performance loss) in the area of drifts.
- In terms of recovery, IBLStreams appears to be the strongest method. In fact, it recovers well in all experiments.
- FLEXFIS tends to have problems with adapting to increasingly difficult situations: When the second stream is more complex than the first one, it often fails to recover (see Figure 9).
- Overall, FIMTDD and AMRules perform quite similarly, regardless of the problem and the type of drift (slow or sudden). This is perhaps not surprising, given that both methods are quite comparable in terms of their model structure (trees and rules are closely related). Moreover, both are using the Hoeffding bound for model adaptation and PH for drift detection (see Figures 9–12).

- Interestingly, FIMTDD recovers especially smoothly and with almost no drop in performance when drifting from a difficult concept to a simpler one (see Figure 11).

6 Conclusion

We have introduced *recovery analysis* as a new type of experimental analysis in the context of learning from data streams. The goal of recovery analysis is to provide an idea of a learner's ability to discover a concept drift quickly, and to take appropriate measures to maintain the quality and generalization performance of the model.

To demonstrate the usefulness of this type of analysis, we have presented an experimental study, in which we analyzed different types of learning methods on classification as well as regression problems. Our results clearly reveal some qualitative differences in how these methods react to concept drift, how much they are affected and how well they recover their original performance, as well as some important factors that seem to be responsible for these differences, such as the number of classes (in classification problems) and whether the drift is from a simpler to a more difficult problem or the other way around.

Overall, our results also provide evidence in favor of our conjecture that instance-based approaches to learning on data streams are not only competitive to model-based approaches in terms of performance, but also advantageous with regard to the handling of concept drift. This is arguably due to their "lightweight" structure: Removing some outdated examples is all the more simpler than completely reconstructing a possibly complex model.

Acknowledgments

This work was supported by the German Research Foundation (DFG).

	Learning Problem			Properties				Source	
	Binary Cl.	Multiclass Cl.	Regression	#Attributes	#Instances	#Eval. Window	Target Attribute	Origion	**R**eal/**A**rtif./**S**im.
Breast	✓			9	683	25	2-Class	UCI	R
mushroom	✓			22	8124	100	2-Class	UCI	R
Page Blocks		✓		10	5473	100	5-Class	UCI	R
Letter		✓		26	20,000	200	26-Class	UCI	R
Bank32h			✓	32	8192	100	[0,0.819665]	DELVE	S
House8L			✓	8	22784	200	[0,500001]	DELVE	R
FIMTDD									
RandomTrees									
RandomTrees									
Binary	✓			4	125000	500	2-Class	MOA	A
5-Classes		✓		4	125000	500	2-Class	MOA	A
HyperDistance									
Distance			✓	4	125000	500	[0.0388,1.7016]	MOA	A
Cubic Distance			✓	4	125000	500	[0.0001,4.9271]	MOA	A

Table 2: Summary of the data sets used in the experiments.

References

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

[Angelov et al., 2010] Angelov, P. P., Filev, D. P., and Kasabov, N. (2010). *Evolving Intelligent Systems*. John Wiley and Sons, New York.

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

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

[Bifet and Gavaldà, 2008] Bifet, A. and Gavaldà, R. (2008). Mining adaptively frequent closed unlabeled rooted trees in data streams. In *Proceedings of KDD 2008, the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 34–42, Las Vegas, Nevada, USA. ACM.

[Bifet and Gavaldà, 2009] Bifet, A. and Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Proceedings of IDA 2009, the 8th International Symposium on Intelligent Data Analysis*, pages 249–260, Lyon, France.

[Bifet and Kirkby, 2009] Bifet, A. and Kirkby, R. (2009). *Massive Online Analysis Manual*.

[Domingos and Hulten, 2000] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press.

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

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

[Gaber et al., 2005] Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2005). Mining data streams: A review. *ACM SIGMOD Record, ACM Special Interest Group on Management of Data*, 34(1).

[Gama, 2010] Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC.

[Gama, 2012] Gama, J. (2012). A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55.

[Gama and Gaber, 2007] Gama, J. and Gaber, M. M. (2007). *Learning from Data Streams*. Springer-Verlag, Berlin, New York.

[Gama et al., 2013] Gama, J., Sebastião, R., and Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346.

[Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

[Huang et al., 2008] Huang, Z., Gedeon, T. D., and Nikravesh, M. (2008). Pattern trees induction: A new machine learning method. *IEEE Transactions on Fuzzy Systems*, 16(4):958–970.

[Ikonomovska et al., 2011] Ikonomovska, E., Gama, J., and Dzeroski, S. (2011). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168.

[Klement et al., 2002] Klement, E., Mesiar, R., and Pap, E. (2002). *Triangular Norms*. Kluwer Academic Publishers.

[Law and Zaniolo, 2005] Law, Y. and Zaniolo, C. (2005). An adaptive nearest neighbor classification algorithm for data streams. In *Proc. PKDD–05, European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 108–120, Porto, Portugal.

[Ljung, 1999] Ljung, L. (1999). *System identification (2nd ed.): theory for the user*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

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

[Mouss et al., 2004] Mouss, H., Mouss, D., Mouss, N., and Sefouhi, L. (2004). Test of Page-Hinkley, an approach for fault detection in an agro-alimentary production system. In *Proceedings of the Asian Control Conference, Volume 2*, pages 815–818.

[Senge and Hüllermeier, 2010] Senge, R. and Hüllermeier, E. (2010). Pattern trees for regression and fuzzy systems modeling. In *Proceedings WCCI–2010, World Congress on Computational Intelligence*, Barcelona, Spain.

[Senge and Hüllermeier, 2010] Senge, R. and Hüllermeier, E. (2011). Top-down induction of fuzzy pattern trees. *IEEE Transactions on Fuzzy Systems*, 19(2):241–252.

[Shaker and Hüllermeier, 2012] Shaker, A. and Hüllermeier, E. (2012). IBLStreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4):235–249.

[Shaker and Hüllermeier, 2013] Shaker, A. and Hüllermeier, E. (2013). Recovery analysis for adaptive learning from non-stationary data streams. In Burduk, R., Jackowski, K., Kurzynski, M., Wozniak, M., and Zolnierek, A., editors, *Proceedings CORES 2013, 8th International Conference on Computer Recognition Systems*, pages 289–298, Wroclaw, Poland. Springer-Verlag.

[Shaker et al., 2013] Shaker, A., Senge, R., and Hüllermeier, E. (2013). Evolving fuzzy pattern trees for binary classification on data streams. *Information Sciences*, 220:34–45.

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

[Widmer and Kubat, 1993] Widmer, G. and Kubat, M. (1993). Effective learning in dynamic environments by explicit context tracking. In *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, volume 667, pages 227–243. Springer-Verlag.

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

[Žliobaitė, 2014] Žliobaitė, I. (2014). Controlled permutations for testing adaptive learning models. *Knowledge and Information Systems*, 39(3):565–578.

A Methods

A.1 Instance-based Learning

We choose the instance-based learner IBLStreams[6] as a representative of the lazy learning paradigm [Shaker and Hüllermeier, 2012]. IBLStreams can be applied to classification and regression problems. The core of IBLStreams is a strategy for maintaining and updating its case base. Whenever

[6] `www.uni-marburg.de/fb12/kebi/research/software/iblstreams`

a new example is produced by the data stream, the algorithm decides whether or not to store it, perhaps at the cost of other examples that are removed from the case base. This decision is based on several criteria and indicators of the usefulness of the example for future predictions.

This case base editing strategy allows IBLStreams to adapt to smooth concept changes. Apart from that, it makes use of statistical hypothesis testing to detect more abrupt changes. As soon as a change of that kind has been detected, a certain percentage (pdiff) of the case base is immediately discarded.

A prediction for a new query instance is obtained by retrieving its k nearest neighbors from the case base (typically using Euclidean distance as a dissimilarity measure) and combining their outputs in an appropriate way. In the case of classification, the simple majority rule is used for combination, i.e., the class that occurs most frequently among the neighbors is predicted. This prediction rule can be generalized, using a kernel function for weighting the votes of the neighbors according their distance from the query. In the case of regression, the mean or the weighted mean is predicted. Alternatively, IBLStreams also allows for deriving a prediction by locally weighted linear regression.

For technical details of the method, we refer to [Shaker and Hüllermeier, 2012]. In our experiments, we use IBLStreams in its default settings (in which a Gaussian kernel is used for weighing, and the width of this kernel is dynamically adapted in a data-driven manner).

A.2 Adaptive Hoeffding Trees

The Hoeffding tree [Domingos and Hulten, 2000] is an incremental decision tree approach tailored for classification on data streams. Upon the arrival of new training data, the algorithm examines each inner node of the tree and decides whether the current split (attribute) is still optimal, or whether an alternative split appears to be advantageous. In the latter case, the corresponding subtree is modified correspondingly. The decision regarding the optimal split attribute is made based on statistical hypothesis testing. More specifically, the Hoeffding bound [Hoeffding, 1963] is used to check whether the information gain of an alternative attribute is significantly higher than the gain of the currently chosen attribute.

An adaptive version of the Hoeffding Tree (AdpHoef) has been presented in [Bifet and Gavaldà, 2009]. This algorithm maintains a drift detection statistics in each node to judge the compatibility of the current tree/subtree with the latest data. For each of these nodes, an alternative tree is maintained and learned on the most recent data only. Whenever a drift detector signals a change at a node, the subtree rooted at that node is replaced by the alternative tree. This variant of Hoeffding trees uses the ADWIN [Bifet and Gavaldà, 2008] technique, a parameter–free method for detecting the rate of change in data streams. In our experiments, we use the AdpHoef algorithm for binary and multiclass classification.

A.3 Adaptive Model Rules

Adaptive Model Rules, AMRules, is a rule induction method for regression on data streams [Mouss et al., 2004]. Each rule is specified by a conjunction of literals on the input attributes in the premise part, and a (linear) function minimizing the root mean squared error in the consequent. Adaptive statistical measures are attached to each rule describing the instance subspace covered by that rule. Moreover, the performance of each rule is monitored by the Page-Hinkley (PH) test [Mouss et al., 2004], and a rule is pruned as soon as it seems to suffer from an increased error due to a concept change.

Each rule is initialized with a single literal and successively expanded with new literals. The best literal to be added, if any, is chosen on the basis of the Hoeffding bound, in a way quite similar to the expansion of Hoeffding trees. The linear function in the consequent part is learned online by stochastic gradient descent.

A.4 Fast Incremental Model Trees with Drift Detection

The Fast Incremental Model Trees with Drift Detection (FIMTDD) algorithm is a tree-based approach for inducing model trees for regression on data streams [Ikonomovska et al., 2011]. It combines properties of Hoeffding trees and AMRules. Just like Hoeffding trees, it uses the Hoeffding bound to choose the best splitting attribute; in this case, where the problem is regression instead of classification, attributes are evaluated in terms of the reduction of standard deviation (of the output attribute in the subtree) they achieve.

Each leaf node of the induced tree contains a linear function fitting the subspace covered by the instances falling into that leaf, just like the rule consequents in AMRules. Moreover, again like in AMRules, the function is learned using stochastic gradient descent. The Page-Hinckley (PH) test is used for change detection at internal and leaf nodes. A high increase in the error indicates a concept change and triggers the exchange of a subtree by an alternative subtree.

A.5 Flexible Fuzzy Inference Systems

The FLEXible Fuzzy Inference Systems [Lughofer, 2008], FLEXFIS, learns so-called Takagi-Sugeno (TS) fuzzy systems [Takagi and Sugeno, 1985] for modeling $\mathbb{R}^p \to \mathbb{R}$ mappings in an online manner. A single rule in a (single output) TS fuzzy system is of the form

$$\begin{array}{rl} \text{IF} & (x_1 \text{ IS } \mu_{i1}) \text{ AND ... AND } (x_p \text{ IS } \mu_{ip}) \\ \text{THEN} & l_i = w_{i0} + w_{i1}x_1 + w_{i2}x_2 + ... + w_{ip}x_p \end{array}$$

where $\boldsymbol{x} = (x_1, \ldots, x_p)$ is the p-dimensional input vector and μ_{ij} the fuzzy set describing the j-th antecedent of the rule. Typically, these fuzzy sets are associated with a linguistic label such as “small” or “large”. The AND connective is modeled in terms of a triangular norm, i.e., a generalized logical conjunction [Klement et al., 2002]. The output $l_i = l_i(\boldsymbol{x})$ is the so-called consequent function of the rule.

The output of a TS system consisting of C rules is a linear combination of the outputs produced by the individual rules, where the contribution of each rule is given by its normalized degree of activation:

$$\hat{f}(\boldsymbol{x}) = \hat{y} = \sum_{i=1}^{C} \Psi_i(\boldsymbol{x}) \cdot l_i(\boldsymbol{x}) \tag{9}$$

with

$$\Psi_i(\boldsymbol{x}) = \frac{\mu_i(\boldsymbol{x})}{\sum_{j=1}^{C} \mu_j(\boldsymbol{x})} \ , \tag{10}$$

where $\mu_i(\boldsymbol{x})$ denotes the activation degree of the i-th rule. The latter is defined by the conjunctive combination of the rule antecedents, i.e., the degrees of membership of the feature values x_j in the fuzzy sets μ_{ij}:

$$\mu_i(\boldsymbol{x}) = \bigotimes_{j=1}^{p} \mu_{ij}(x_j) \ , \tag{11}$$

where $\otimes$ is a triangular norm. As can be seen, a TS fuzzy model is a parameterized mapping which is defined by the choice of the input space (including its dimensionality p), the number of rules, C, and the parameters of each single rule, i.e., the fuzzy sets μ_{ij} and the weight vector $w_i = (w_{i0}, w_{i1}, \ldots w_{ip})$.

FLEXFIS allows for learning and adapting these components incrementally. It makes use of online clustering techniques (including the creation of new clusters and the merging or removal of existing ones) in order to specify rule antecedents and recursive weighted least squares estimation [Ljung, 1999] for fitting the linear functions is the consequent parts. For technical details of the method, we refer to [Lughofer, 2008].

FLEXFIS is implemented in Matlab and offers a function for finding optimal values for different parameters and thresholds. We used this function to fix all parameters except the so-called "forgetting parameter" (responsible for down-weighing the influence of previous examples), for which we manually found the value 0.999 to perform best. Finally, the pruning option was enabled.

A.6 Evolving fuzzy pattern trees

Fuzzy pattern trees have recently been introduced as a novel machine learning method for classification [Huang et al., 2008, Senge and Hüllermeier, 2010] and regression [Senge and Hüllermeier, 2010]. Roughly speaking, a fuzzy pattern tree is a hierarchical, tree-like structure, whose inner nodes are marked with generalized (fuzzy) logical and arithmetic operators. It implements a recursive function that maps a combination of attribute values, entered in the leaf nodes, to a number in the unit interval, which is produced as an output by the root of the tree.

An evolving variant of FPT was introduced in [Shaker et al., 2013] for binary classification problems. The basic idea of the evolving version of fuzzy pattern tree learning (eFPT) is to maintain an ensemble of pattern trees, consisting of a current (active) model and a set of neighbor models. The current model is used to make predictions, while the neighbor models can be seen as *anticipated adaptations*: they are kept ready to replace the current model in case of a drop in performance, caused, for example, by a drift of the concept to be learned. More generally, the current model is replaced, or say, the anticipated adaptation is realized, whenever its performance appears to be significantly worse than the performance of one of the neighbor models. Roughly speaking, the adaptive behavior of the eFPT can be seen as pruning some leaf nodes of the currently used tree model or extending other leaf nodes whenever this adaptation step has a significant improvement on the performance. Significance of a change in performance is decided by means of statistical hypothesis testing. We analyze the recovery behavior of eFPT on binary classification problems.

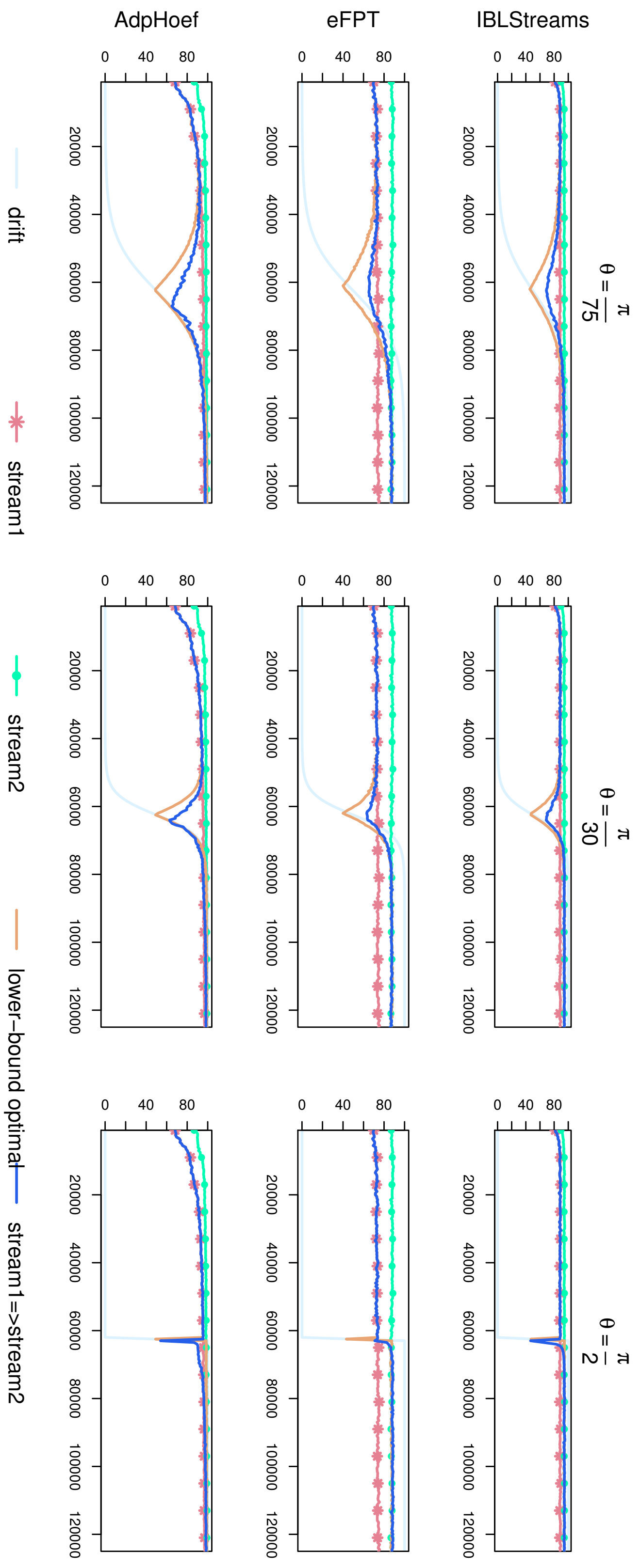

Figure 3: Performance curves (accuracy) on the RandomTree data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

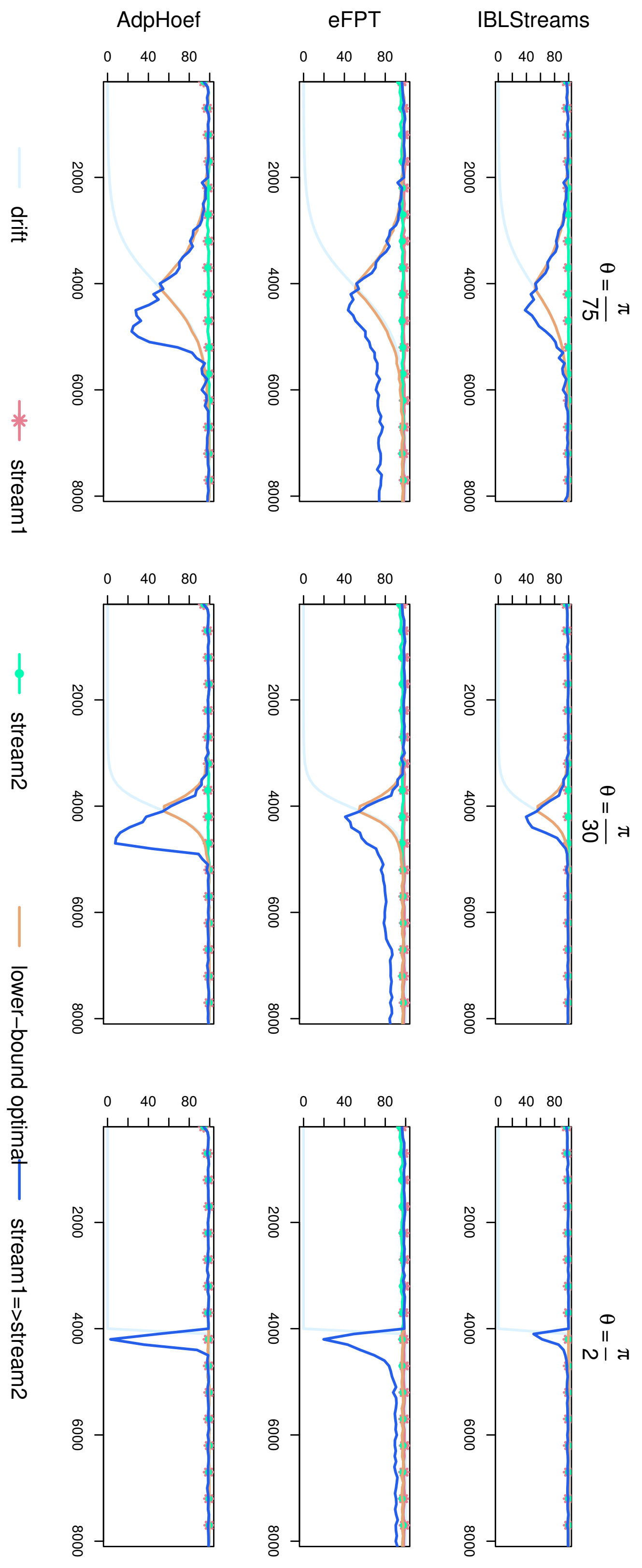

Figure 4: Performance curves (accuracy) on the Mushroom data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

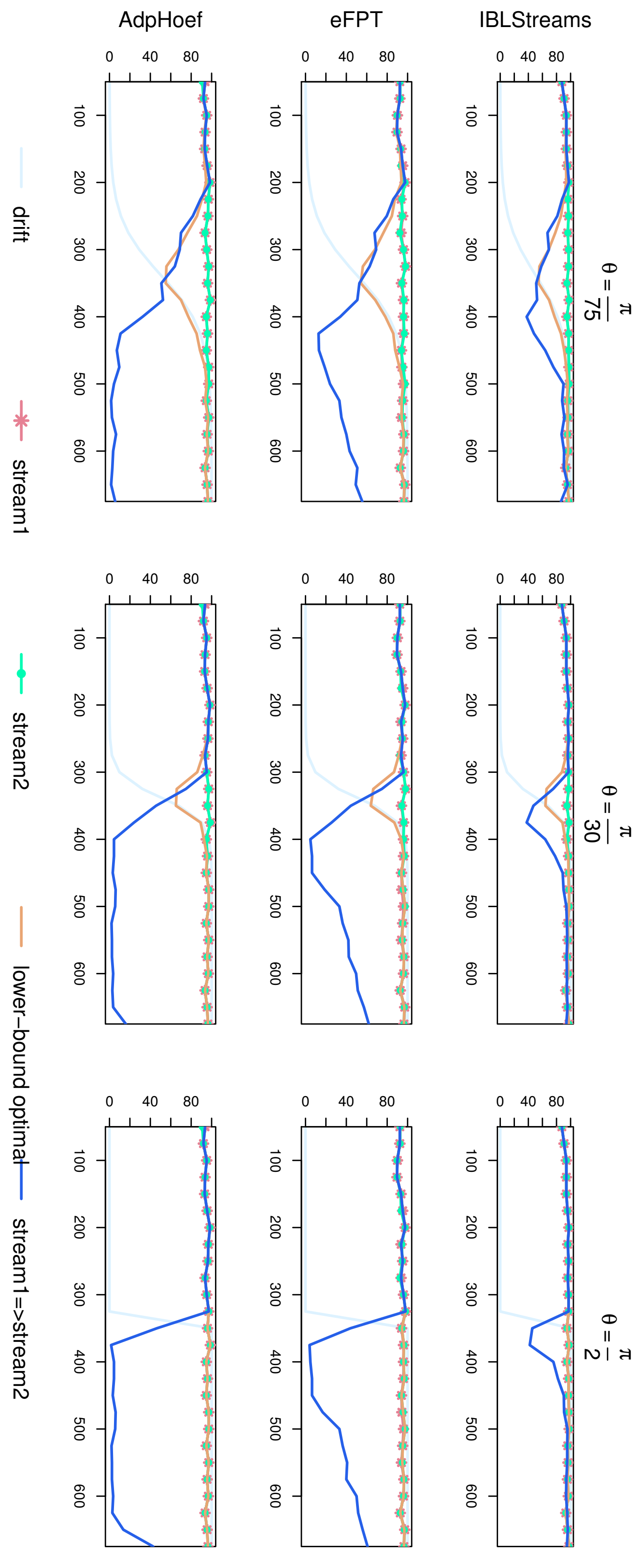

Figure 5: Performance curves (accuracy) on the Breast Cancer Wisconsin data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

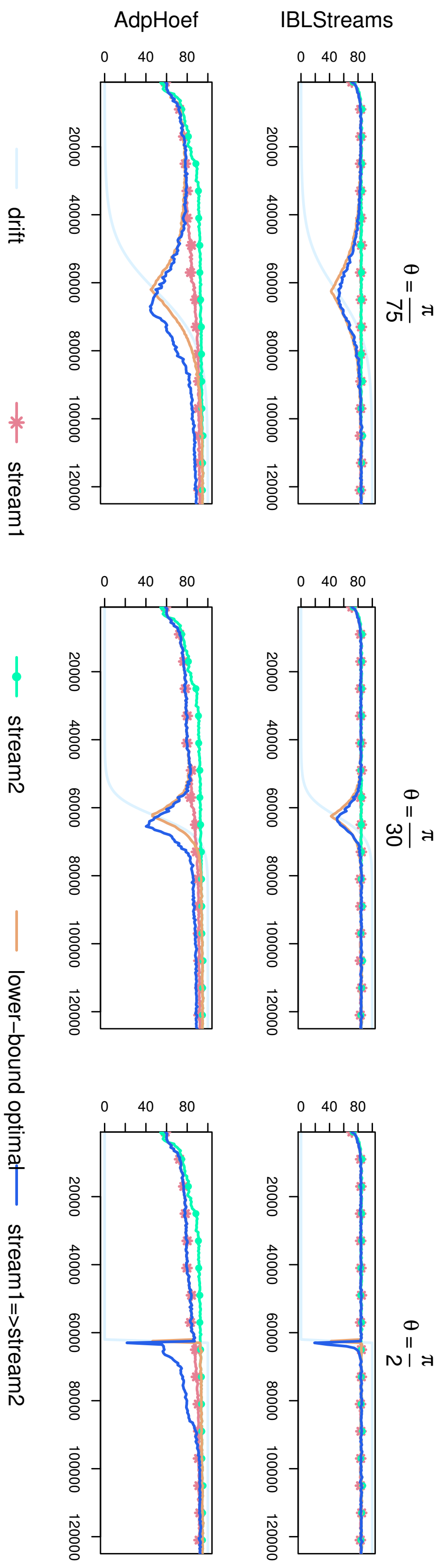

Figure 6: Performance curves (accuracy) on the RandomTree 5-classes data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

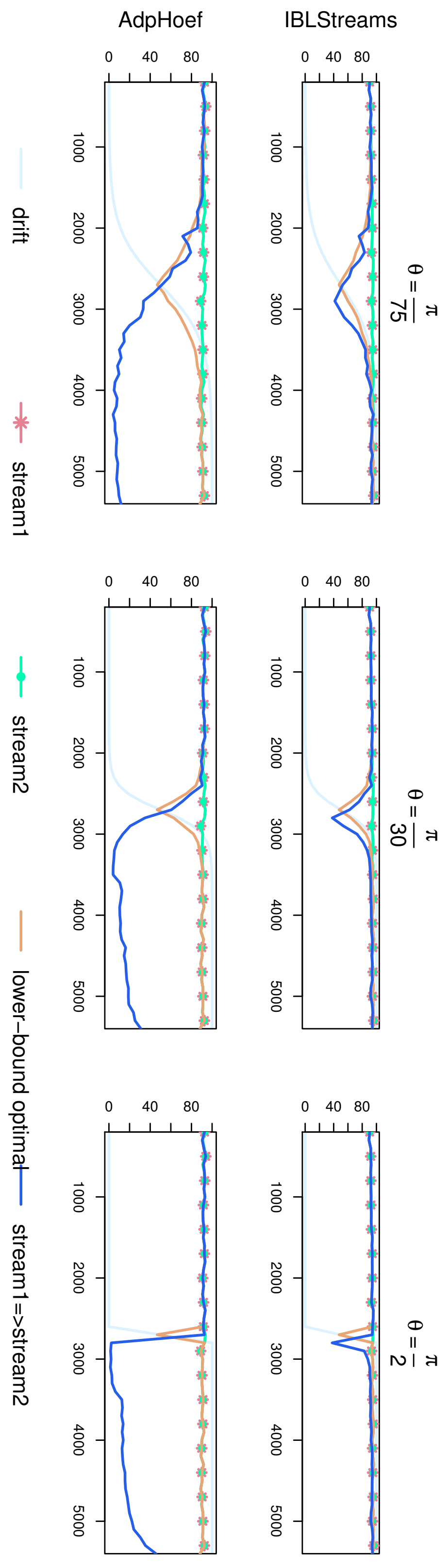

Figure 7: Performance curves (accuracy) on the Page Blocks data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

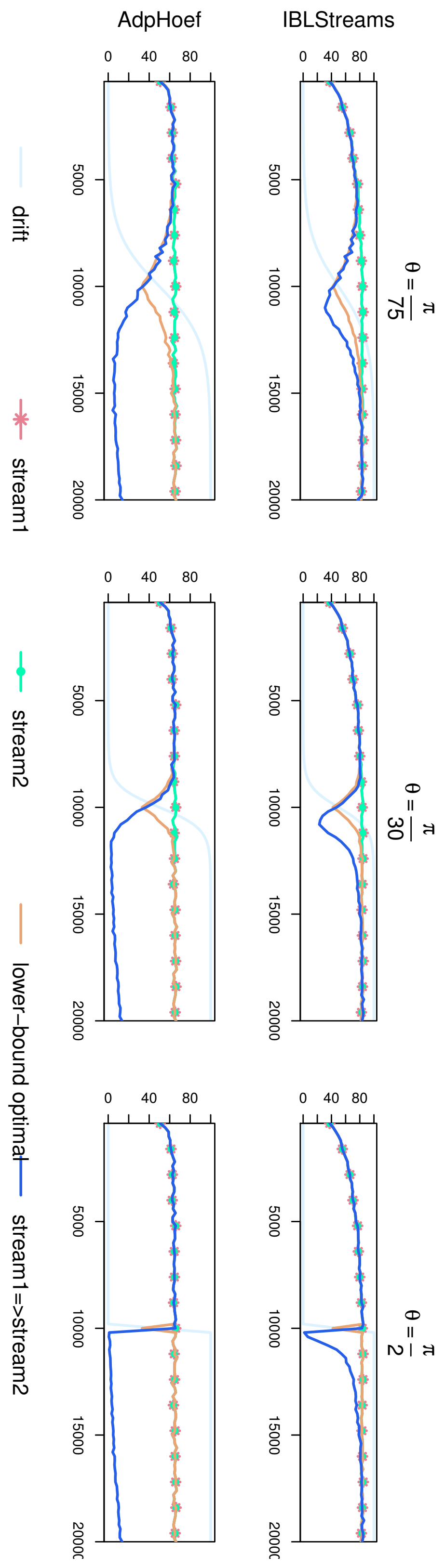

Figure 8: Performance curves (accuracy) on the Letter Recognition data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

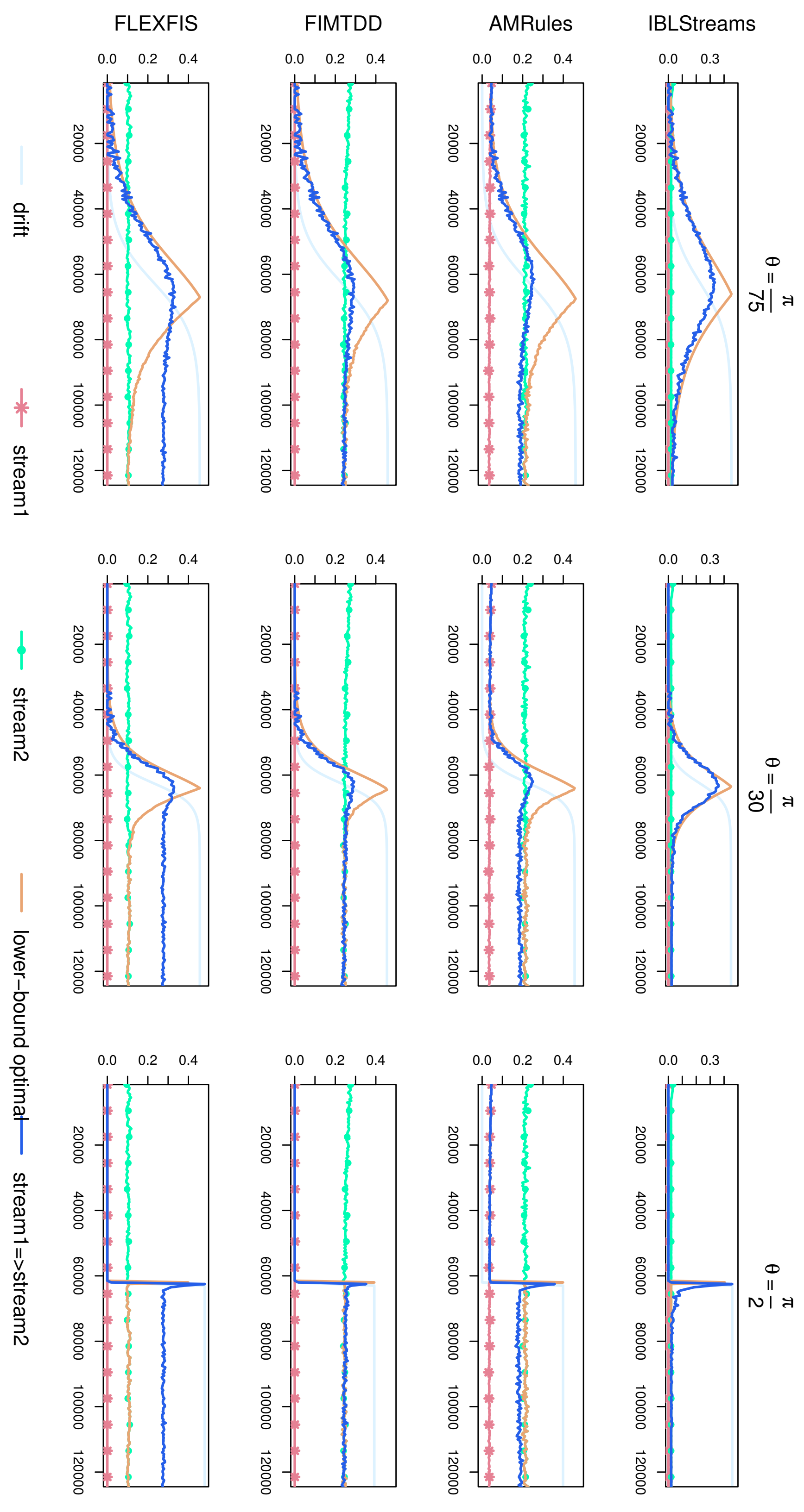

Figure 9: Performance curves (RMSE) on the HyperDistance data, with a drift from f_1 to f_3. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

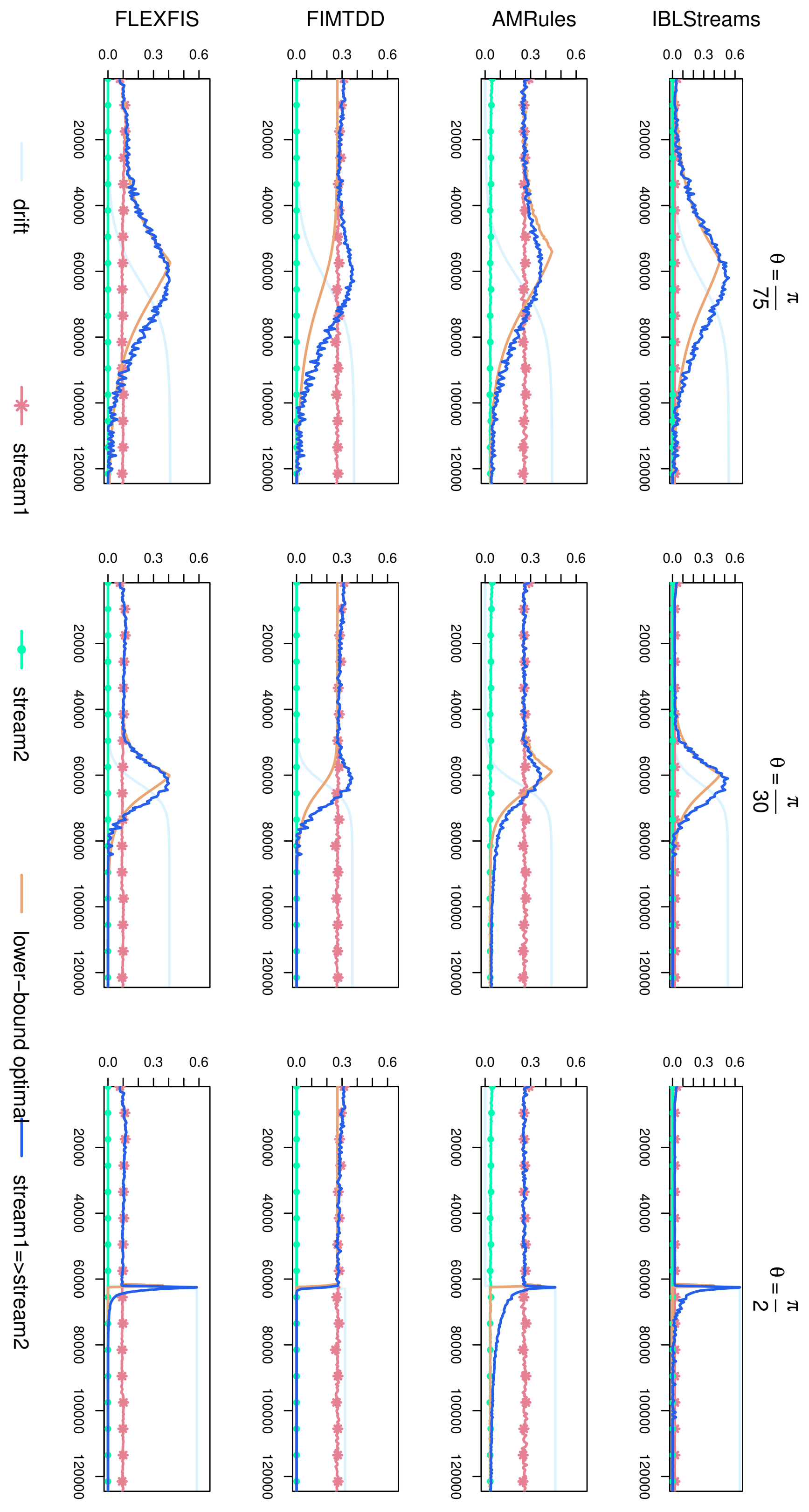

Figure 10: Performance curves (RMSE) on the HyperDistance data, with a drift from f_3 to f_1. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

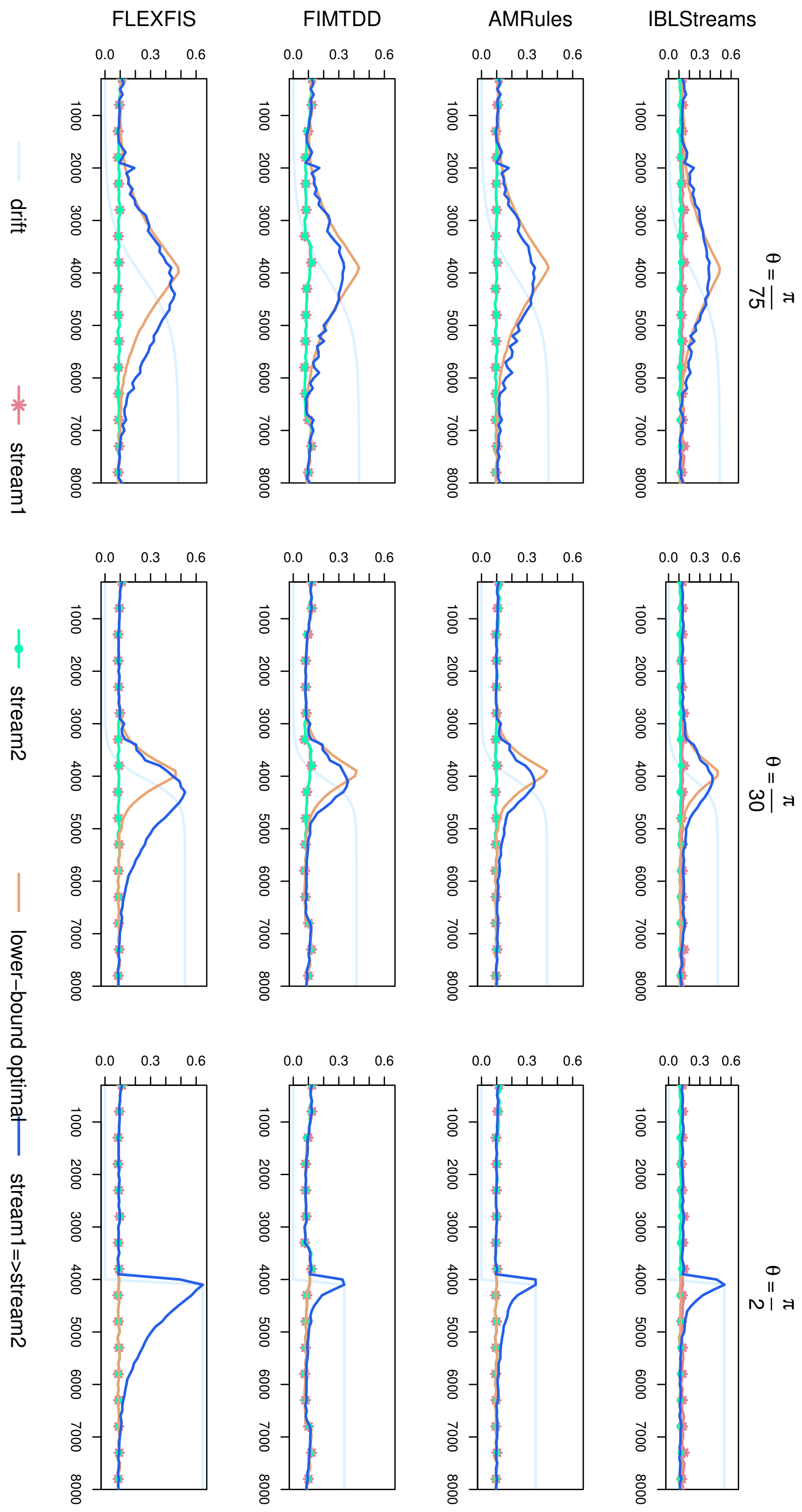

Figure 11: Performance curves (RMSE) on the Bank32h data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

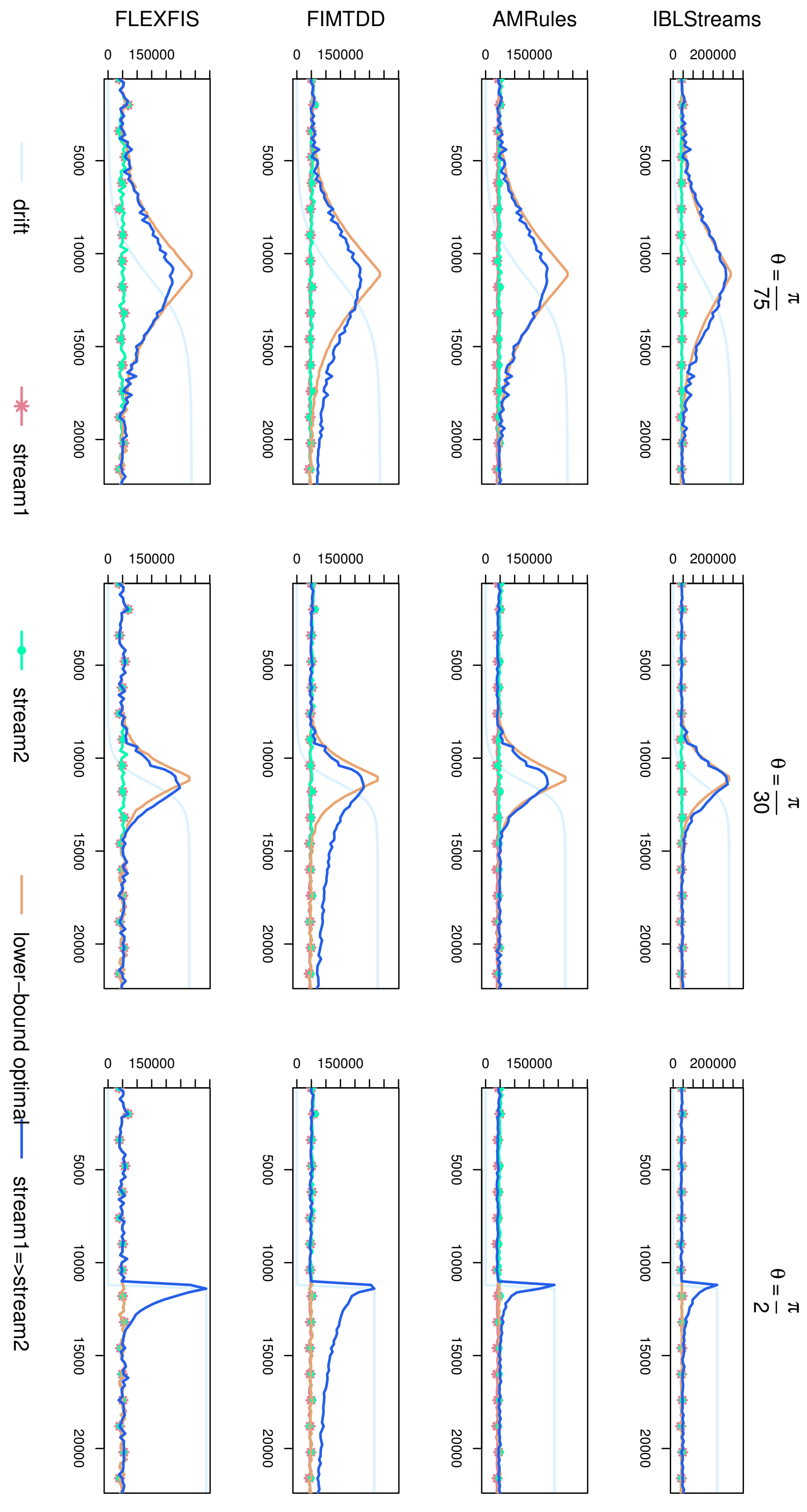

Figure 12: Performance curves (RMSE) on the House8L data. The sigmoid in light grey indicates the range of the drift. The brown line shows the lower bound on the optimal performance.

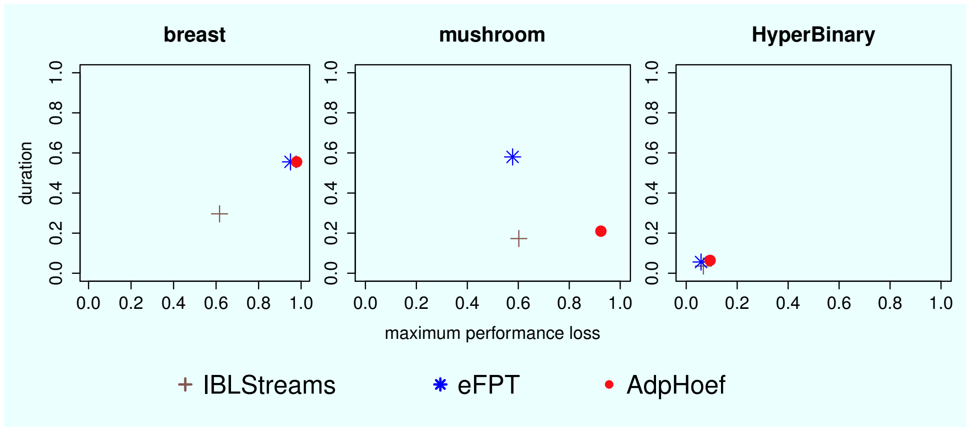

Figure 13: Duration versus maximal performance loss of different methods on the binary classification problems.

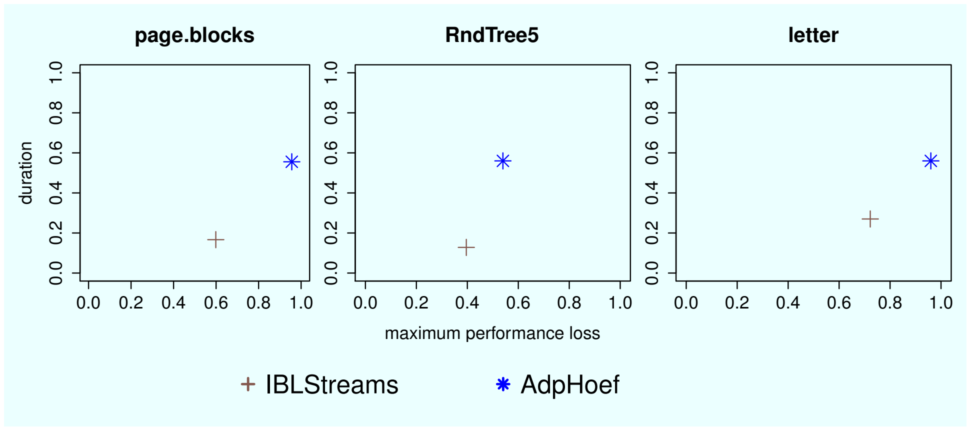

Figure 14: Duration versus maximal performance loss of different methods on the multi-class classification problems.

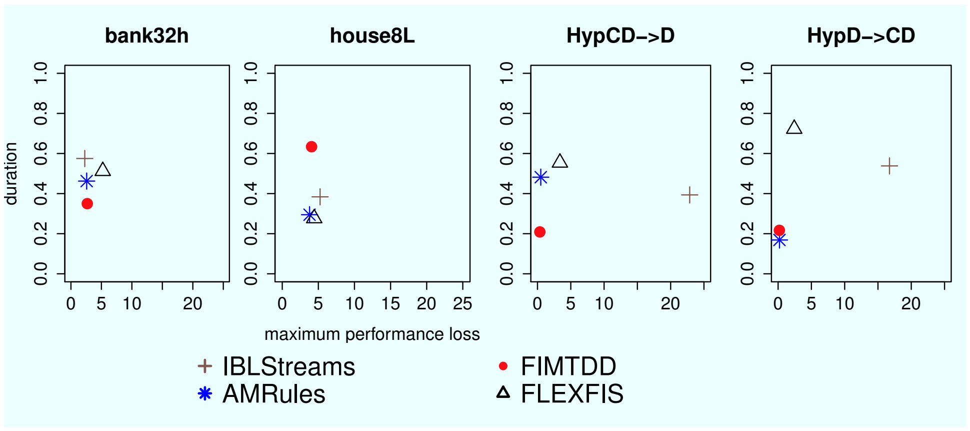

Figure 15: Duration versus maximal performance loss of different methods on the regression problems.