Recovery Analysis for Adaptive Learning from Non-stationary Data Streams

Ammar Shaker and Eyke Hüllermeier

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.

1 Introduction

The development of methods for learning from so-called *data streams* has been a topic of active research in recent years [6, 9]. 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 [7].

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in which the entire data as a whole is to provided as an input to the learning algorithm (or "learner" for short), is no longer applicable.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,

Ammar Shaker · Eyke Hüllermeier
Department of Mathematics and Computer Science, University of Marburg, Germany
e-mail: {shaker,eyke}@mathematik.uni-marburg.de

R. Burduk et al. (Eds.): *CORES 2013*, AISC 226, pp. 289–298.
DOI: 10.1007/978-3-319-00969-8_28 © Springer International Publishing Switzerland 2013

the data must be processed in a single pass, which implies an incremental mode of learning and model adaptation.

Domingos and Hulten [3] 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 paper, we propose a new type of experimental analysis, called *recovery analysis*, which is aimed at assessing this ability of a learner. Roughly speaking, recovery analysis suggests a specific experimental protocol and a graphical presentation of a learner's generalization 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.

2 Learning under Concept Drift

We consider a setting in which an algorithm $\mathscr{A}$ is learning on a time-ordered stream of data $S = (z_1, z_2, z_3, \ldots)$. Since we are mainly interested in *supervised learning*, we suppose that each data item z_t is a tuple $(x_t, y_t) \in \mathbb{X} \times \mathbb{Y}$ consisting of an input 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 $\mathscr{A}$ is supposed to offer a predictive model $\mathscr{M}_t : \mathbb{X} \to \mathbb{Y}$ that has been learned on the data seen so far, i.e., on the sequence $S_t = (z_1, z_2, \ldots, z_t)$. Given a query input $x \in \mathbb{X}$, this model can be used to produce a prediction $\hat{y} = \mathscr{M}_t(x) \in \mathbb{Y}$ 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 $\mathscr{M}_t$ 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 $\mathscr{A}$ is truly incremental, it will produce $\mathscr{M}_t$ solely on the basis of $\mathscr{M}_{t-1}$ and z_t, that is, $\mathscr{M}_t = \mathscr{A}(\mathscr{M}_{t-1}, z_t)$. In other words, it does not store any of the previous observations $z_1, \ldots, z_{t-1}$. Most algorithms, however, store at least a few of the most recent data points, which can then also be used for model adaptation. In any case, the number of observations that can be stored is typically assumed to be finite, which excludes the possibility of memorizing the entire stream. A *batch learner* $\mathscr{A}_B$, on the other hand, would produce the model $\mathscr{M}_t$ on the basis of the complete set of data $\{z_1, \ldots, z_t\}$. Note that, although $\mathscr{A}$ and $\mathscr{A}_B$ have seen the same data, $\mathscr{A}_B$ can exploit this data in a more flexible way. Therefore, the models produced by $\mathscr{A}$ and $\mathscr{A}_B$ 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 z_t is generated at random according to $\mathbf{P}$, i.e., the probability to observe a specific $z \in \mathbb{Z}$ is given by[1]

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

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 z_t is generated by $\mathbf{P}_t$. One speaks of a *concept change* if these measures are not all equal [1].

In the literature, a distinction is made between different causes and types of concept change [8]. 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 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 $\mathbf{P}(x)$, i.e., the distribution of the inputs, while the concept itself, i.e., the conditional distribution $\mathbf{P}(y \,|\, x)$, remains unchanged [14]. 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.

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

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 $S = (z_1, z_2, \ldots, z_T)$. In experimental studies, such streams are commonly used to produce a performance curve showing the generalization performance of a model sequence $(\mathscr{M}_t)_{t=1}^T$ 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: First, for a data stream S, it is normally not known whether it contains any concept drift, let alone when such a drift occurs; this is a problem at least for real data, while obviously less of an issue if data is generated synthetically. 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.

3.1 Main Idea and Experimental Protocol

In order to overcome these problems, our idea is to work, not with a single data stream, but with three streams in parallel, two "pure streams" and one "mixture". The pure streams $S_A = (z_1^a, z_2^a, \ldots, z_T^a)$ and $S_B = (z_1^b, z_2^b, \ldots, z_T^b)$ 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. 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 $S_C = (z_1^c, z_2^c, \ldots, z_T^c)$ is produced by randomly sampling from the two pure streams:

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

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

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

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.

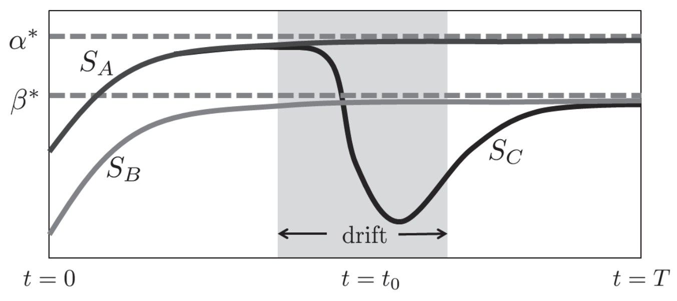

Fig. 1 Schematic illustration of a recovery analysis: The three performance curves are produced by training models on the pure streams S_A and S_B, as well as on the mixed stream S_C, each time using the same learner $\mathscr{A}$. The region shaded in grey indicates the time window in which the concept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

Now, suppose the same learning algorithm $\mathscr{A}$ 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 $\mathscr{A}$ 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 $\mathscr{A}$ 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 $\mathscr{A}$ starts to recover. Eventually, $\mathscr{A}$ may (or may not) reach the level β^*. This level is indeed an upper bound on the asymptotic performance, since $\mathscr{A}$ 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 $\mathscr{A}$ to deal with concept drift. In particular, the minimum of this curve indicates how strongly $\mathscr{A}$ is affected by the concept drift. Moreover, the curve informs about how quickly the performance deteriorates (giving an idea of how sensitive $\mathscr{A}$ is), how much time $\mathscr{A}$ 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 $\mathscr{A}$ 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

$$\gamma^*(t) = \sup_{\mathscr{M} \in \mathbf{M}} \gamma_{\mathscr{M}}(t) \qquad (2)$$

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 $\mathscr{A}$ can choose from), and $\gamma_{\mathscr{M}}(t)$ denotes the generalization performance of a model $\mathscr{M} \in \mathbf{M}$ 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 $\mathscr{M}_A, \mathscr{M}_B \in \mathbf{M}$ whose performance is $\alpha_{\mathscr{M}_A}(t) = \alpha(t)$ and $\beta_{\mathscr{M}_B}(t) = \beta(t)$. Now, suppose we were to apply the model $\mathscr{M}_A$ on the stream S_C. What is the expected generalization performance? If an example (x,y) on S_C is generated according to $\mathbf{P}_A$, the generalization performance (expected loss) of $\mathscr{M}_A$ on this example is the same as on S_A, namely $\alpha_{\mathscr{M}_A}(t)$. Otherwise, if the example is generated according to $\mathbf{P}_B$, nothing can be said about the performance of $\mathscr{M}_A$; thus, assuming that the performance measure takes values in the unit interval, 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 $\mathscr{M}_A$ is given by

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

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

$$\gamma^{\bullet}(t) = \max\left\{\lambda(t) \cdot \alpha_{\mathscr{M}_A}(t), (1-\lambda(t)) \cdot \beta_{\mathscr{M}_B}(t)\right\} \ . \qquad (3)$$

Obviously, since the supremum in (2) is not only taken over $\{\mathscr{M}_A, \mathscr{M}_B\}$ 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 (3) can be written more simply as

$$\gamma^{\bullet}(t) = \max\left\{\lambda(t) \cdot \alpha^*, (1-\lambda(t)) \cdot \beta^*\right\} \ . \qquad (4)$$

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.

Finally, we note that of course not all performance measures (loss functions) can naturally be normalized to $[0,1]$. Especially problematic in this regard are measures that are principally unbounded, such as the squared loss in regression. In such cases, an estimation similar to the one above can nevertheless be derived, provided the worst case performance can be bounded by a constant; this constant will then replace the constant in our estimation, which is simply 0.

3.3 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 $\mathscr{M}_t$ (induced by the learner $\mathscr{A}$) 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:[2] (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 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 some 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, $\mathscr{A}$ 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 $\mathscr{A}$ 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

[2] Both procedures are implemented in the MOA framework [2].

should not end while $\mathscr{A}$ 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 Illustration

This section is meant to illustrate our idea of recovery analysis by means of some practical examples. To this end, we conducted a series of experiments with different classification methods. All algorithms were implemented under MOA [2], except FLEXFIS, which is implemented in Matlab. MOA is a framework for learning on data streams. It includes data stream generators and several classifiers. Moreover, it offers different methods for performance evaluation.

Due to a lack of space, we present results only for a single data set, namely the weather data provided by the National Oceanic and Atmospheric Administration (NOAA).[3] Since this data originally contained missing values, we used it in the form as suggested by [5]. It contains eight daily weather measurements, such as temperature, visibility, etc. The goal is to predict whether it will be a rainy day or not. We used this 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 gradually changes from predicting whether it will be rainy to predicting whether it will not be rainy.

Performance curves were produced using the test-then-train procedure (cf. Section 3.3) in chunk mode (with chunk size 500) and averaging over ten random shuffles of the data. At each point of time, the evaluation curve shows the prediction performance on the most recent chunk.

The results for four different methods are shown in Figure 2: the eFPT method for learning evolving fuzzy pattern trees [12], the instance-based learner IBLStreams [11], the Hoeffding Trees classifier for learning decision trees [4] in its incremental form without any specific reaction mechanism to concept changes, and the FLEXFIS method for fuzzy rule induction [10].

From this figure, which also indicates the range of the drift and shows the lower bound on the optimal performance (3) as a reference, some interesting observations can be made. For example, by comparing the lower bound with the respective “performance valleys” of the methods, it can be seen that all methods are reacting with a certain delay, which is clearly expected; some methods, however, are obviously a bit faster than others. The same remark applies to the extent of performance loss, which is, for example, more pronounced for FLEXFIS than for eFPT. Overall, the instance-based learner IBLStreams seems to perform best, showing a curve that is close to the

[3] http://users.rowan.edu/~polikar/research/NIE_data/

reference (3). The Hoeffding Tree learner, on the other hand, is performing much worse. Its loss in performance is higher, and it recovers only very slowly. In fact, the decision trees produced by this learner are often very complex and, therefore, difficult to adapt if significant changes are needed. This may explain why Hoeffding Trees react more slowly than other learners (a tendency that we could also confirm on other data streams).

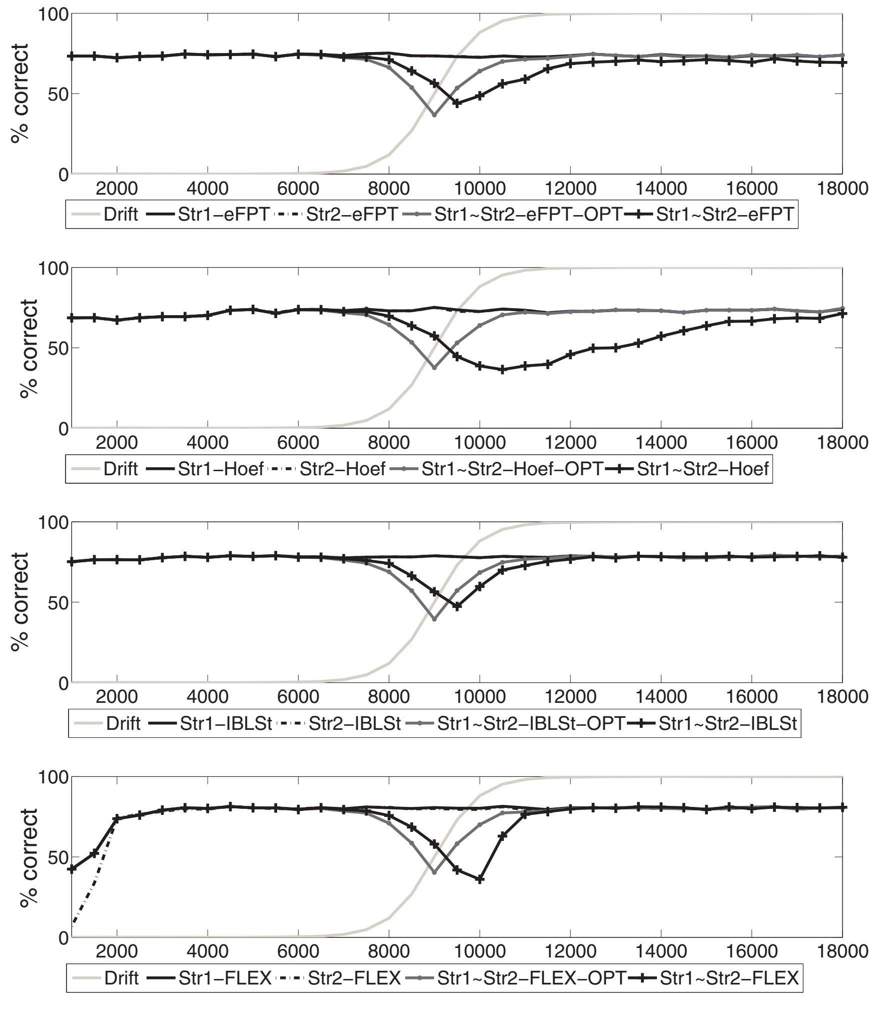

Fig. 2 Performance curves (classification rate) on the weather data. From top to bottom: fuzzy pattern trees, Hoeffding Trees, IBLStreams, FLEXFIS. The sigmoid in light grey indicates the range of the drift. The grey line shows the lower bound on the optimal performance (3).

5 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 shown results of an experimental study using a stream of weather data, on which we have compared four different learning algorithms.

In future work, we plan to further refine our approach to recovery analysis, for example by developing numerical measures to quantify specific aspects of a recovery curve (delay of reaction, duration, etc.). Moreover, we plan to use recovery analysis in large empirical studies as a tool for comparing different classes of learning methods with regard to their ability to handle concept drift [13].

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

References

1. Ben-David, S., Gehrke, J., Kifer, D.: Detecting change in data streams. In: Proc. VLDB 2004 (2004)
2. Bifet, A., Kirkby, R.: Massive Online Analysis Manual (August 2009)
3. Domingos, P., Hulten, G.: A general framework for mining massive data streams. Journal of Computational and Graphical Statistics 12 (2003)
4. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proc. KDD 2000, pp. 71–80 (2000)
5. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. IEEE Transactions on Neural Networks 22(10), 1517–1531 (2011)
6. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining data streams: A review. ACM SIGMOD Record 34(1) (2005)
7. Gama, J.: A survey on learning from data streams: current and future trends. Progress in Artificial Intelligence 1(1), 45–55 (2012)
8. Gama, J.: Knowledge Discovery from Data Streams. Chapman & Hall/CRC (2010)
9. Gama, J., Gaber, M.M.: Learning from Data Streams. Springer (2007)
10. Lughofer, E.: FLEXFIS: A robust incremental learning approach for evolving Takagi-Sugeno fuzzy models. IEEE Transactions on Fuzzy Systems 16(6), 1393–1410 (2008)
11. Shaker, A., Hüllermeier, E.: IBLStreams: A system for instance-based classification and regression on data streams. Evolving Systems 3(4), 235–249 (2012)
12. Shaker, A., Senge, R., Hüllermeier, E.: Evolving fuzzy pattern trees for binary classification on data streams. Information Sciences 220, 34–45 (2013)
13. Žliobaite, I., Pechenizkiy, M.: Reference framework for handling concept drift: An application perspective. Technical report (2010)
14. Widmer, G., Kubat, M.: Effective learning in dynamic environments by explicit context tracking. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, pp. 227–243. Springer, Heidelberg (1993)