An Ensemble Classifier for Drifting Concepts

Martin Scholz and Ralf Klinkenberg

University of Dortmund, 44221 Dortmund, Germany,
{scholz,klinkenberg}@ls8.cs.uni-dortmund.de,
WWW: http://www-ai.cs.uni-dortmund.de/

Abstract. This paper proposes a boosting-like method to train a classifier ensemble from data streams. It naturally adapts to concept drift and allows to quantify the drift in terms of its base learners. The algorithm is empirically shown to outperform learning algorithms that ignore concept drift. It performs no worse than advanced adaptive time window and example selection strategies that store all the data and are thus not suited for mining massive streams.

1 Introduction

Machine learning methods are often applied to problems, where data is collected over an extended period of time. In many real-world applications this introduces the problem that the distribution underlying the data is likely to change over time. Knowledge discovery and data mining from (time-changing) data streams and concept drift handling on data streams have become important topics in the machine learning community and have gained increased attention recently as illustrated by numerous conference tracks and workshops on these topics as well as specially dedicated journal issues (see e.g. [1] and [2]).

For example, companies collect an increasing amount of data like sales figures and customer data to find patterns in the customer behavior and to predict future sales. As the customer behavior tends to change over time, the model underlying successful predictions should be adapted accordingly. The same problem occurs in information filtering, i.e. the adaptive classification of documents with respect to a particular user interest. With the amount of online information and communication growing rapidly, there is an increasing need for automatic information filtering. Information filtering techniques are used e.g. to build personalized news filters, which learn about the news-reading preferences of a user [3]. The interest of the user, i.e. the concept underlying the classification of the texts, changes over time. A filtering system should be able to adapt to such concept changes. Machine learning approaches handling this type of concept drift have been shown to outperform more static approaches ignoring it [4].

After formalizing the concept drift problem in Sec. 2.1, previous approaches handling it are reviewed in Sec. 2.2. Sec. 3 discusses related work on ensemble methods for data streams and introduces a boosting-like algorithm for data streams that naturally adapts to concept drift. In Sec. 4 the approach is evaluated on two real-world datasets with different simulated concept drift scenarios, and on one economic dataset exhibiting real concept drift.

2 Concept Drift

2.1 Problem Definition

Throughout this paper, we study the problem of concept drift for the pattern recognition problem in the following framework [4, 5]. Each example $z = (x, y)$ consists of a feature vector $x \in \mathcal{X}$ and a label $y \in \{-1, +1\}$ indicating its classification. Data arrives over time in batches. Without loss of generality these batches are assumed to be of equal size, each containing m examples.

$$\underbrace{z_{(1,1)}, \ldots, z_{(1,m)}}_{\text{batch } 1}, \underbrace{z_{(2,1)}, \ldots, z_{(2,m)}}_{\text{batch } 2}, \cdots, \underbrace{z_{(t,1)}, \ldots, z_{(t,m)}}_{\text{batch } t}, \underbrace{z_{(t+1,1)}, \ldots, z_{(t+1,m)}}_{\text{batch } t+1}$$

$z_{(i,j)}$ denotes the j-th example of batch i. For each batch i the data is independently identically distributed with respect to a distribution $P_i(x, y)$. Depending on the amount and type of concept drift, the example distribution $P_i(x, y)$ and $P_{i+1}(x, y)$ between batches will differ. The learner aims to sequentially predict the labels of the next batch and to minimize the cumulated number of prediction errors. E.g., after batch t the learner can use any subset of the training examples from batches 1 to t to predict the labels of batch $t + 1$.

2.2 Related Work on Concept Drift

In machine learning, changing concepts are often handled by time windows of fixed or adaptive size on the training data [6–9] or by weighting data or parts of the hypothesis according to their age and/or utility for the classification task [10]. The latter approach of weighting examples has already been used for information filtering in the incremental relevance feedback approaches of [11] and [12].

For windows of fixed size, the choice of a *"good" window size* is a compromise between fast adaptivity (small window) and good generalization in phases without concept change (large window). A fixed window size makes strong assumptions about how quickly the concept changes. While heuristics can adapt to different speed and amount of drift, they involve many parameters that are difficult to tune. The basic idea of *adaptive window management* is to adjust the window size to the current extent of concept drift.

Drifting concepts can also be learned effectively and efficiently with little parameterization by an error minimization framework for adaptive time windows [4] and example weighting or selection [13, 14]. This framework makes use of support vector machines (SVMs) and their special properties, which allow an efficient and reliable error estimation after a single training run [15].

The methods of the framework either maintain an adaptive time window on the training data [4], select representative training examples, or weight the training examples [13, 14]. The key idea is to automatically adjust the window size, the example selection, and the example weighting, respectively, so that the estimated generalization error is minimized. The approaches do not require complicated parameterization and are both, theoretically well-founded as well as effective and efficient in practice.

3 Adapting Ensemble Methods to Drifting Streams

3.1 Ensemble Methods for Data Stream Mining

In the recent years many algorithms specifically tailored towards mining from data streams have been proposed. Apart from being able to cope with concept drift, scalability is one of the most important issues. For very large datasets, the induction of classifiers like decision trees is very efficiently possible: Using a sampling strategy based on Hoeffding bounds, the VFDT algorithm efficiently induces a decision tree in constant time [16]. By combining several trees to an ensemble of classifiers, techniques like bagging [17] and boosting [18] have been shown to significantly increase the predictive accuracy for many datasets. Corresponding variants for data streams have e.g. been suggested in [19, 20].

The SEA algorithm [21] induces an ensemble of decision trees from data streams and explicitly addresses concept drift. It splits the data into batches and fits one decision tree per batch. To predict a label, the base models are combined by an unweighted majority-vote, similar to bagging. Models are discarded by a heuristic. The authors do not report an increase in classification performance compared to a single decision tree learner, but state that the ensemble recovers from concept drifts. The recovery time of this approach seems unnecessarily long, as it only exchanges one model in each iteration and uses no confidence weights.

A more recent theoretical analysis suggests, that *weighted* base learners are a preferable alternative in domains with concept drift [22]. The analyzed AddExp algorithm steadily updates the weights of *experts* (base models in an ensemble), and adds a new expert each time the ensemble misclassifies an example. The new experts start to learn from scratch, using a weight that reflects the loss suffered by the ensemble for the current example.

Sec. 3.3 extends the efficient boosting procedure presented in Sec. 3.2 to streams. It trades off predictive performance versus scalability. To this end, the online algorithm reads examples aggregated to batches and decides for each batch, whether to add a new expert to the ensemble or not. Unlike in SEA, the base models of the ensemble are combined by a *weighted* majority vote. Subsequent models are trained after reweighting the examples of the new batch, and each new base classifier model is assigned a weight that depends on both, its own performance and the performance of the remaining ensemble. A continuous re-estimation adapts the weights of all ensemble members to concept drift.

3.2 Ensemble Generation by Knowledge-Based Sampling

The algorithm introduced in this paper is based on the sampling strategy suggested in [23]. Patterns are discovered iteratively. A pattern that is found in one iteration extends the user's prior knowledge in the next one. In each iteration the sampling procedure produces training sets that are "orthogonal" to the combined probability estimate corresponding to the prior knowledge. This aspect is very close to boosting classifiers. The idea of removing prior knowledge by biased sampling is formulated in terms of constraints. Formally, this step defines a new distribution, as close to the original function as possible, but orthogonal to the estimates produced by available prior knowledge. All that is accounted for in the next iteration is the unexpected component of each model. Technically this step can be realized by introducing example weights. In this paper the only kind of "prior" knowledge are the base models yielded by preceding iterations.

For a given instance space $\mathcal{X}$ and nominal class attribute $\mathcal{Y}$ examples are expected to be sampled i.i.d. from an initial distribution $D : \mathcal{X}, \mathcal{Y} \rightarrow \mathbb{R}^+$. Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ denote a base model from hypothesis space $\mathcal{H}$, predicting a class.

As a first constraint, the new distribution D' to be constructed should no longer support the knowledge encoded in the hypothesis h, so the predictions $h(x)$ and the true label should be independent considering D':

$$(\forall y', y^* \in \mathcal{Y}) : P_{x,y\sim D'}\left[y = y' \mid h(x) = y^*\right] = P_{x,y\sim D'}\left[y = y'\right] \tag{1}$$

If eq. (1) did not hold, then the same hypothesis would allow to derive further information about the true label's conditional distribution given the prediction.

As a second constraint, the probability of observing a specific class $y' \in \mathcal{Y}$ and the probability of a specific prediction $y^* \in \mathcal{Y}$ should not change from D to D'; it is sufficient and possible to remove only the *correlation* between the true label and the predicted label:

$$(\forall y' \in \mathcal{Y}) : P_{x,y\sim D'}\left[y = y'\right] = P_{x,y\sim D}\left[y = y'\right] \tag{2}$$

$$(\forall y* \in \mathcal{Y}) : P_{x,y\sim D'}\left[h(x) = y^*\right] = P_{x,y\sim D}\left[h(x) = y^*\right] \tag{3}$$

Eq. (2) ensures that the class skew does not change, which would result in an implicitly altered cost model for misclassifying examples [24]. Eq. (3) avoids to skew the marginal distribution unnecessarily.

Finally, within each partition sharing the same predicted label y^* and true class y', the new distribution is defined proportionally to the initial one; having a hypothesis h as prior knowledge, all instances within such a partition are indistinguishable. Changes to the conditional probabilities within one partition prefer some instance over the others, despite their equivalence with respect to the available prior knowledge. This translates into the following constraint:

$$\begin{aligned}&(\forall x, x' \in \mathcal{X})(\forall y', y^* \in \mathcal{Y}) :\\ &\quad P_{x,y\sim D'}(x = x' \mid h(x) = y^*, y = y') = P_{x,y\sim D}(x = x' \mid h(x) = y^*, y = y')\end{aligned} \tag{4}$$

Constraints (1)-(4) induce a unique target distribution. The following definition eases notation.

Definition 1. *The* LIFT *for a given hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$, predicted class $y* \in \mathcal{Y}$, and true class label $y' \in \mathcal{Y}$ is defined as*

$$\textsc{Lift}_D^{(h)}(y^* \rightarrow y') := \frac{P_{x,y\sim D}\left[h(x) = y^*, y = y'\right]}{P_{x,y\sim D}\left[h(x) = y^*\right] \cdot P_{x,y\sim D}\left[y = y'\right]}$$

Similar to precision, LIFT measures the correlation between a specific prediction and a specified true label. A value larger than 1 indicates a positive correlation.

1. Let D_1 denote the uniform distribution over example set $\mathcal{E} = \langle x_1, y_1 \rangle, \ldots, \langle x_m, y_m \rangle$.
2. For $i = 1$ to n do // n: user given number of iterations
   (a) Call $\textsc{BaseLearner}(D_i, \mathcal{E})$ to find an accurate model $h_i : \mathcal{X} \to \mathcal{Y}$.
   (b) Compute $\textsc{Lift}_{D_i}^{(h_i)}(y^* \to y')$ applying definition 1.
   (c) $D_{i+1}(x_j, y_j) \leftarrow D_i(x_j, y_j) \cdot \left(\textsc{Lift}_{D_i}^{(h_i)}(h(x_j) \to y_j)\right)^{-1}$ for all $\langle x_j, y_j \rangle \in \mathcal{E}$.
3. Output $\{h_1, \ldots, h_n\}$ and the $\textsc{Lift}$ values. Predict $P(y \mid x)$ with eq. (5).

Fig. 1. Algorithm KBS (Knowledge-Based Sampling)

Theorem 1. *For any initial distribution D and hypothesis $h \in \mathcal{H}$ constraints (1)-(4) are equivalent to*

$$\forall \langle x, y \rangle \in \mathcal{X} \times \mathcal{Y} : P_{D'}(x, y) = P_D(x, y) \cdot \left(\textsc{Lift}_D^{(h)}(h(x) \to y)\right)^{-1}.$$

A proof is given in [23]. Theorem 1 defines a new distribution to sample from, given a hypothesis h as prior knowledge. Assuming a single hypothesis is not very restrictive, since it is possible to directly incorporate each new base model into a single ensemble classifier. The theorem can be applied iteratively [25, 26]: Base model h_i is selected based on distribution D_i. Distribution D_{i+1} is defined by applying theorem 1 to D_i and h_i. A corresponding algorithm (KBS) is depicted in Fig. 1. It boosts weak base learners and has empirically been shown to be competitive to AdaBoost.M1 and LogitBoost [26].

The inverse of the reweighting strategy allows to approximately reconstruct the original distribution D_1 as a combination of the single hypotheses. The following formula estimates the odds $\beta(x)$ based on $\{h_1, \ldots, h_n\}$, a sequence of n hypotheses, each of which is the result of a separate iteration of learning:

$$\beta(x) := \frac{P(y = +1)}{P(y = -1)} \cdot \prod_{i=1}^{n} \frac{\textsc{Lift}_{D_i}^{(h_i)}(h_i(x) \ \to \ y = +1)}{\textsc{Lift}_{D_i}^{(h_i)}(h_i(x) \ \to \ y = -1)} \tag{5}$$

This allows to compute estimates of the conditional probabilities as

$$P(y = +1 \mid x) \approx \frac{\beta(x)}{1 + \beta(x)}.$$

The reweighting scheme used by KBS makes base classifiers rank models according to their contribution to the overall accuracy: After KBS "samples out" a model h, the accuracy of all overlapping (correlated) models with respect to the new distribution is reduced according to the degree of overlap. Because of constraint (1), the Lift of the subset with a common prediction $h(x)$ is 1. Predictive accuracy is a linear combination of the corresponding Lift values. Using examples that were reweighted with respect to a given h, the base classifier favors models according to their independent contributions [24, 25]. Similar to other boosting approaches, the example weights anticipate the expectation given the previously trained models. This is especially useful for handling smooth concept

Initialize empty ensemble $\mathbf{M} := \emptyset$.
While not end of stream, do

1. Read next batch $\mathcal{E}_k$ in iteration k.
2. Predict $P(y \mid x)$ for all $x \in \mathcal{E}_k$ with current ensemble $\mathbf{M}$ (eq. (5)).
3. Read true labels of $\mathcal{E}_k$.
4. If alternative ensemble $\mathbf{M}^*$ exists:
   (a) Compare accuracy of $\mathbf{M}$ and $\mathbf{M}^*$ wrt. $\mathcal{E}_k$.
   (b) $\mathbf{M} \leftarrow$ better ensemble, discard worse ensemble.
   (c) If $\mathbf{M}^*$ is discarded: $\mathcal{E}^* \leftarrow \mathcal{E}_{k-1}$ (shrink cache).
5. Initialize D_1: Uniform distribution over $\mathcal{E}_k$.
6. For $i \in \{1, \ldots, |\mathbf{M}|\}$, do:
   (a) Apply h_i to make predictions for $\mathcal{E}_k$.
   (b) Recompute $\text{LIFT}_{D_i}^{(h_i)}(y^* \to y')$ based on $\mathcal{E}_k$ (def. 1).
   (c) Update the LIFTs of h_i in $\mathbf{M}$.
   (d) $D_{i+1}(x_j, y_j) \leftarrow D_i(x_j, y_j) \cdot \left(\text{LIFT}_{D_i}^{(h_i)}(h(x_j) \to y_j)\right)^{-1}$ for all $\langle x_j, y_j \rangle \in \mathcal{E}_k$.
7. Call BASELEARNER($D_{|M|+1}$, $\mathcal{E}_k$), get new model $h_{|M|+1} : \mathcal{X} \to \mathcal{Y}$.
8. Compute $\text{LIFT}_{D_{|M|+1}}^{(h_{|M|+1})}(y^* \to y')$ (def. 1).
9. Add model $h_{|M|+1}$ (and its LIFTs) to ensemble $\mathbf{M}$.
10. If this is the first batch, then $\mathcal{E}^* = \mathcal{E}_k$ (no alternative ensemble).
11. Else
    (a) $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \mathcal{E}_k$, (extend last batch)
    (b) $\mathbf{M}^* \leftarrow$ clone($\mathbf{M}$)
    (c) discard last base model of $\mathbf{M}^*$
    (d) repeat steps 5-9 for $\mathcal{E}^*$ and $\mathbf{M}^*$ instead of for $\mathcal{E}_k$ and $\mathbf{M}$

Fig. 2. Algorithm KBS-Stream

drifts: While sudden drifts require a quick detection and a way to rapidly adjust the working hypothesis, for smooth drifts it is better to collect information on the new target concept over a period of time. Especially if the preceeding concept has been identified accurately at the point in time when a drift starts, removing the knowledge about the current concept from the data makes the learner already focus on the new concept.

3.3 A KBS-strategy to learn drifting concepts from data streams

The original KBS-algorithm (Fig. 1) assumes that the complete training set is available in main memory. The first step to adopt to data streams is to read and classify examples iteratively. For subsequent learning steps the reweighting strategy of KBS allows to compute example weights very efficiently. The data is assumed to arrive in batches large enough to train base classifiers.

The sizes of training sets that are effectively used for each model are determined dynamically by the algorithm. Processing a new batch yields two ensemble variants. The first variant appends the current batch to the batch that was used for training in the last iteration, and it refines the latest base model accordingly. The second variant adds a new model, which is trained using the latest batch, only. Only the ensemble variant performing better on the next batch is kept. The strategy serves two purposes. First, for stationary distributions a new model is trained only, if there is empirical evidence that this increases the accuracy of the resulting ensemble. This will generally happen if the learning curve of the latest model has leveled off, see e.g. [27], and the data set is well suited for boosting. Second, if sudden concept drift (concept shift) occurs, the same estimation procedure instantly suggests to add a new model, which will help to overcome the drift.

The second step of adopting KBS to data streams is to foresee a re-computation phase in which base model performances are updated with respect to the current distribution. In fact, we believe that this is a main advantage of using weighted ensembles in a concept drift scenario. For stationary distributions the weights vary marginally, while for smoothly drifting scenarios they are systematically shifted and even allow to quantify and interpret the drift in terms of previously found patterns or models. Even sudden drifts do not pose a problem, as they automatically result in radically reduced weights of previously trained models, and in high weights of subsequently trained models, if these parameters are re-estimated from new data. The response time to drifts is very short. Since the streaming variant of KBS is closely coupled to the accurate KBS boosting algorithm, the predictive performance is expected to outperform those of single base models for many datasets. Pruning of ensembles can efficiently be addressed during weight re-computation; whenever a model does not reach a fixed minimum advantage over random guessing on the latest batch, it is discarded.

The algorithm is depicted in Fig. 2. It loops until the stream ends. Lines 1-2 apply the current ensemble to the new batch without knowing the correct labels. Lines 3-4 check whether continuing the training of the latest model with the latest batch outperforms adding a new model trained on that batch[1]. Only the better of these ensembles is kept. Lines 5 and 6 recompute the Lift parameters of all base models. To this end, the models are iteratively applied to the new batch, and the weights are adjusted similarly to the learning phase. Finally, lines 8-11 train two variants of the ensemble again, $\mathbf{M}^*$ being the one extending the last batch and updating the newest model appropriately, and $\mathbf{M}$ being the one that adds a new model, which is trained using only the newest data.

One degree of freedom is left in line 2: The algorithm may use $\mathbf{M}$ or $\mathbf{M}^*$ to classify the new batch, as the performance of both is unknown at that time. For the experiments two variants have been implemented. The first one (KBS_{stream}) always uses ensemble $\mathbf{M}^*$, since models trained from larger batches are generally more reliable. The second variant (KBS_{hold_out}) uses a hold out set of 30% from the last batch to decide which ensemble to use.

If incremental base learners are used, then only the latest batch needs to be stored. The runtime is dominated by adjusting the most recent model to this data, and by applying all base models to it. This avoids the memory requirements and combinatorial explosion of more advanced techniques (see Sec. 4.1).

[1] The pseudocode does not assume an incremental base learner, but trains new models on cached data. For incremental base learners no cache is required.

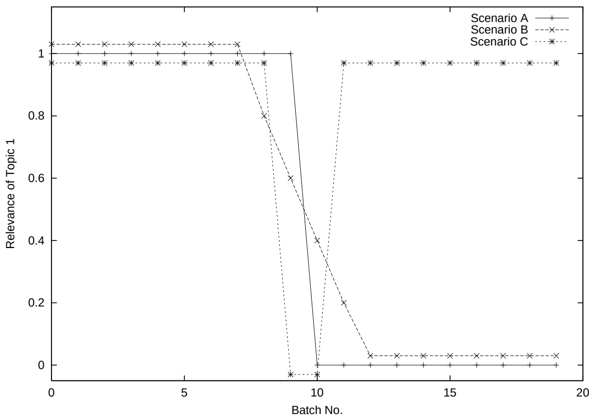

Fig. 3. Relevance of the first topic/concept in the concept change scenarios A, B, and C. The relevance of the second relevant topic/concept is *1.0 - relevance of topic 1.*

4 Experiments

4.1 Experimental Setup and Evaluation Scheme

In order to evaluate the KBS learning approach for drifting concepts, it is compared to the adaptive time window approach, to the batch selection strategy, and to three simple non-adaptive data management approaches.

Full Memory: The learner generates its classification model from all previously seen examples, i.e. it cannot "forget" old examples.

No Memory: The learner induces its hypothesis only from the most recent batch. This corresponds to using a window of the fixed size of one batch.

Window of "Fixed Size": A time window of a fixed size of $n = 3$ batches is used on the training data.

Adaptive Window: A window adjustment algorithm adapts the window size to the current concept drift situation (see Sec. 2.2 and [4]).

Batch Selection: Batches producing an error less than twice the estimated error of the newest batch, when applied to a model learned on the newest batch only, receive a weight of one, i.e. are selected for the final training set. All other examples are deselected by setting their weights to zero [13, 14].

The performance of the classifiers is measured by prediction error. All results reported in Sec. 4.2 and 4.3 are averaged over four runs. The experiments were conducted with the machine learning environment YALE [28], the SVM implementation *mySVM* [29], and two learners from the WEKA toolbox [30].

4.2 Evaluation on Simulated Concept Drift with TREC Data

The first experiments are performed in an information filtering domain, a typical application area for learning drifting concepts. Text documents are represented

	Full Memory	No Memory	Fixed Size	Adaptive Size	Batch Selection	KBS stream	KBS hold_out
Scen. A	21.11%	11.16%	9.03%	6.65%	6.15%	6.89%	**5.88%**
Scen. B	21.30%	12.64%	9.76%	9.06%	9.33%	**8.64%**	9.50%
Scen. C	8.60%	12.73%	11.19%	8.56%	**7.55%**	10.11%	8.27%

Table 1. Error of all time window and example selection methods vs. KBS.

as attribute-value vectors (*bag of words* model), where each distinct word corresponds to a feature whose value is the "ltc"-TF/IDF-weight [31] of that word in that document. The experiments use a subset of 2608 documents of the data set of the *Text REtrieval Conference (TREC)*. Each text is assigned to one or several categories, five of which are considered here.

Three concept change scenarios are simulated. The texts are randomly split into 20 batches of equal size containing 130 documents each. In all scenarios, a document is considered relevant at a certain point in time, if it matches the interest of the simulated user at that time. The user interest changes between two of the topics, while documents of the remaining three topics are never relevant. Fig. 3 shows the probability of being relevant for a document of the first category at each batch for each of the three scenarios; this also implies the probability of the second (sometimes) relevant topic. *Scenario A* is a concept shift from the first to the second topic in batch 10. In *Scenario B* the user interest changes slowly from batch 8 to batch 12. *Scenario C* simulates an abrupt concept shift in the user interest from the first to the second topic in batch 9 and back to the first in batch 11. Tab. 1 compares the results of all static and adaptive time window and batch selection approaches on all scenarios in terms of prediction error [4, 13, 5] to the two variants of KBS. In all cases the learning algorithm was a support vector machine with linear kernel.

The KBS algorithm manages well to adapt to all three kinds of concept drift. Tracking the ensembles revealed, that during statitionary distributions the current model was continuously refined. After a concept shift (*scenario A*) a new model was trained, and the old model received a significantly lower weight. It was not discarded, however, since it still helped to identify the three topics which are always irrelevant. The hold out set helped to identify the better of the two ensembles reliably at classification time. In *scenario B* five or more models were trained. The ensemble accurately adopted to the drift, but at classification time the systematic one-batch-delay of the hold out estimate was sometimes misleading. In *scenario C* the full memory approach is already competitive to all but the batch selection scenario. Without the hold out set KBS applies the depreciated model one iteration too long for each concept shift. This delay increases the error rate by about 2%. This problem is circumvented by using a hold out set. In essence the KBS algorithm performed very well on this domain, and it even outperformed computationally more expensive approaches. Only in *scenario C* the batch selection method is clearly superior, probably because it is the only method able to concatenate the data before the first and after the second concept shift to a single training set.

	J48 Fixed Memory	J48 Full Memory	AdaBoost+J48 Full Memory	KBS stream	KBS hold_out
Scen. A	11.06%	21.65%	20.51%	9.50%	**9.43%**
Scen. B	11.25%	21.22%	19.93%	11.60%	**10.92%**
Scen. C	12.70%	10.83%	**9.50%**	11.55%	10.45%

Table 2. Averaged prediction errors for satellite image dataset.

4.3 Experiments with Satellite Image Data

The second set of experiments used the satellite image dataset from the UCI library [32], a real-world dataset for classification. It contains no (known) drift over time, so we simulated a drift with the same technique as described in Sec. 4.2: The data set was split into 20 batches at random (321 examples per batch) and only two ("grey soil" and "very damp grey soil") of the six classes were selected to be relevant. The same drift scenarios *A*-*C* as in Sec. 4.2 were simulated, where the two classes corresponded to the two topics in the TREC experiments.

Since decision trees are a more typical base learner for ensemble methods, we chose the J48 algorithm from the Weka toolbox. We compared KBS to the non-adaptive "fixed size" (of 3 batches) and "full memory" strategies. Additionally to running J48 stand-alone, we tried to run AdaBoost on top of J48. For all learners the default settings were used. The results are listed in table 2. Unlike for the last experiment, the results of KBS could always be improved by using a hold out set. As before, *Scenario C* can well be tackled by a full memory approach, which is exploited by AdaBoost. For the other scenarios KBS is better than the fixed size learner, and much better than the full memory approach.

4.4 Predicting Phases in Business Cycles

The third evaluation domain is a task from economics based on real-world data. The quarterly data describes the West German business cycles from 1954 to 1994. Each of the 158 examples is described by 13 indicator variables. The task is to predict the current phase of the business cycle. In accordance to findings from Theis and Weihs [33] we use two phases instead of four for the description of the business data, as described in [34]. The following experiments compare the performance of the KBS data stream algorithm to previously reported results [5], so the same number of batches (5 and 15) were used. The timely order of the examples (quarters) was preserved and no artificial concept drift was simulated.

The results of these two evaluations are shown in Tab. 3. The column for the fixed time window approach lists the results for the fixed size that performed best. The fact that this approach performs well may be due to the cyclic nature of the domain. However, the size is generally not known in advance, and as shown in [5] using other fixed window sizes, leads to significant drops in performance. The results for 15 batches shows that KBS does not perform well, if each batch consists of less than a dozen examples. The reason is that it is not possible to get reliable probability estimates from such small data sets. The algorithm could cache older data in such cases, but it is more reasonable to choose larger batch

	Full Memory	No Memory	Fixed Size	Adaptive Size	Batch Selection	KBS stream	KBS hold_out
5 batches	32.80%	27.20%	24.00%	24.80%	24.80%	24.60%	**17.46%**
15 batches	28.08%	28.77%	**20.55%**	24.80%	23.29%	26.03%	28.77%

Table 3. Prediction error for business cycle data.

sizes. Using just 5 batches (31 examples) already improves the situation, so KBS performs similar to the fixed size, adaptive size, and to the batch selection approach. The hold out set turns out to be surprisingly effective for larger batches. This result provides first evidence that KBS is able to adapt classifier ensembles to different kinds of concept drift found in real-world datasets.

5 Conclusion

This paper presents a new ensemble method for learning from data streams. At each iteration, base models are induced and reweighted continuously, considering the latest batch of examples, only. Unlike other ensemble methods, the proposed strategy adapts very quickly to different kinds of concept drift. The algorithm has low computational costs. It has empirically been shown to be competitive to, and often to even outperform more sophisticated adaptive window and batch selection strategies.

References

1. Kubat, M., Gama, J., Utgoff, P.E.: Intelligent Data Analysis (IDA) Journal, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift, Vol.8, No.3. (2004)
2. Aguilar-Ruiz, J.S., Cohen, P.R.: Symposium on Applied Computing (SAC-2004), Special Track on Data Streams. (2004)
3. Lang, K.: NewsWeeder: Learning to filter netnews. In: Proc. of the 12th Int. Conf. on Machine Learning (1995), 331–339
4. Klinkenberg, R., Joachims, T.: Detecting concept drift with support vector machines. In: Proc. of the 17th Int. Conf. on Machine Learning (2000), 487–494
5. Klinkenberg, R.: Predicting phases in business cycles under concept drift. In: LLWA 2003 – Tagungsband der GI-Workshop-Woche, (2003) 3–10
6. Mitchell, T., Caruana, R., Freitag, D., McDermott, J., Zabowski, D.: Experience with a learning personal assistant. Communications of the ACM **37** (1994) 81–91
7. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23** (1996), 69–101
8. Klinkenberg, R., Renz, I.: Adaptive information filtering: Learning in the presence of concept drifts. In Workshop Notes of the ICML/AAAI-98 Workshop *Learning for Text Categorization*, AAAI Press (1998), 33–40
9. Hulten, G., Spencer, L., Domingos, P.: Mining Time-Changing Data Streams. In: 7th ACM Int. Conf. on Knowledge Discovery and Data Mining (2001), 97 – 106
10. Taylor, C., Nakhaeizadeh, G., Lanquillon, C.: Structural change and classification. In: Workshop Notes on Dynamically Changing Domains: Theory Revision and Context Dependence Issues, ECML '97. (1997) 67–78

11. Allan, J.: Incremental relevance feedback for information filtering. In: ACM SIGIR Conf. on Research and Development in Information Retrieval (1996), 270–278
12. Balabanovic, M.: An adaptive web page recommendation service. In: Proc. 1st Int. Conf. on Autonomous Agents, ACM Press (1997) 378–385
13. Klinkenberg, R., Rüping, S.: Concept drift and the importance of examples. In Franke, J., Nakhaeizadeh, G., Renz, I., eds.: Text Mining – Theoretical Aspects and Applications. Physica-Verlag (2003) 55–77
14. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. In: [1]
15. Joachims, T.: Estimating the generalization performance of a SVM efficiently. In: Proc. of the 17th Int. Conf. on Machine Learning (2000), 431–438
16. Domingos, P., Hulten, G.: Mining High Speed Data Streams. In: Proc. of the 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. (2000) 71–80
17. Breiman, L.: Bagging predictors. Machine Learning **13** (1996) 30–37
18. Freund, Y., Schapire, R.R.: A decision–theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences **55** (1997) 119 – 139
19. Oza, N.C., Russell, S.: Online Bagging and Boosting. In: 8th Int. Workshop on Artificial Intelligence and Statistics. (2001)
20. Lee, H.K.H., Clyde, M.A.: Lossless Online Bayesian Bagging. Journal of Machine Learning Research **5** (2004) 143–151
21. Street, W.N., Kim, Y.: A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification. In: Proc. of the 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. (2001) 377–382
22. Kolter, J., Maloof, M.: Using Additive Expert Ensembles to Cope with Concept Drift. In: 22nd Int. Conf. on Machine Learning (2005)
23. Scholz, M.: Knowledge-Based Sampling for Subgroup Discovery. In Morik, K., Boulicaut, J.F., Siebes, A., eds.: Local Pattern Detection. Springer (2005) 171–189
24. Fürnkranz, J., Flach, P.: ROC 'n' Rule Learning – Towards a Better Understanding of Covering Algorithms. Machine Learning **58** (2005) 39–77
25. Scholz, M.: Sampling-Based Sequential Subgroup Mining. In: Proc. of the 11th ACM SIGKDD Int. Conf. on Knowledge Discovery in Databases (2005), 265–274
26. Scholz, M.: Comparing Knowledge-Based Sampling to Boosting. Technical Report 26, SFB475, Universität Dortmund, Germany (2005)
27. John, G.H., Langley, P.: Static Versus Dynamic Sampling for Data Mining. In: Second Int. Conf. on Knowledge Discovery in Databases and Data Mining. (1996)
28. Fischer, S., Klinkenberg, R., Mierswa, I., Ritthoff, O.: Yale: Yet Another Learning Environment – Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund, Germany (2002). http://yale.sf.net/
29. Rüping, S.: mySVM Manual. Universität Dortmund, Lehrstuhl Informatik VIII (2000). http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/
30. Witten, I., Frank, E.: Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (2000)
31. Salton, G., Buckley, C.: Term weighting approaches in automatic text retrieval. Information Processing and Management **24** (1988) 513–523
32. Blake, C., Merz, C.: UCI repository of machine learning databases (1998). http://www.ics.uci.edu/∼mlearn/MLRepository.html
33. Theis, W., Weihs, C.: Clustering techniques for the detection of business cycles. SFB475 Technical Report 40, Universität Dortmund, Germany (1999)
34. Morik, K., Rueping, S.: A multistrategy approach to the classification of phases in business cycles. In: European Conf. on Machine Learning (2002), 307–318