Learning with Local Drift Detection

João Gama[1,2] and Gladys Castillo[1,3]

[1] LIACC - University of Porto
Rua de Ceuta 118-6, 4050 Porto, Portugal
[2] Fac. Economics, University of Porto
[3] University of Aveiro
jgama@liacc.up.pt, gladys@mat.ua.pt

Abstract. Most of the work in Machine Learning assume that examples are generated at random according to some stationary probability distribution. In this work we study the problem of learning when the distribution that generates the examples changes over time. We present a method for detection of changes in the probability distribution of examples. The idea behind the drift detection method is to monitor the online error-rate of a learning algorithm looking for significant deviations. The method can be used as a wrapper over any learning algorithm. In most problems, a change affects only some regions of the instance space, not the instance space as a whole. In decision models that fit different functions to regions of the instance space, like Decision Trees and Rule Learners, the method can be used to monitor the error in regions of the instance space, with advantages of fast model adaptation. In this work we present experiments using the method as a wrapper over a decision tree and a linear model, and in each internal-node of a decision tree. The experimental results obtained in controlled experiments using artificial data and a real-world problem show a good performance detecting drift and in adapting the decision model to the new concept.

1 Introduction

In many applications, learning algorithms acts in dynamic environments where the data flows continuously. If the process is not strictly stationary (as most of real world applications), the target concept could change over time. Nevertheless, most of the work in Machine Learning assume that training examples are generated at random according to some stationary probability distribution. In [1], the authors present several examples of real problems where change detection is relevant. These include user modelling, monitoring in bio-medicine and industrial processes, fault detection and diagnosis, safety of complex systems, etc.

The PAC learning framework assumes that examples are independent and randomly generated according to some probability distribution D. In this context, some model-class learning algorithms (like Decision Trees, Neural Networks, some variants of k-Nearest Neighbours, etc) could generate hypothesis that converge to the Bayes-error in the limit, that is, when the number of training examples increases to infinite. All that is required is that D must be stationary, the distribution must not change over time.

In this work we review the Machine Learning literature for learning in the presence of drift, we propose a taxonomy for learning in dynamic environments, and present a method to detect changes in the distribution of the training examples. The method is presented as a wrapper over any learning algorithm. We show that it can be used to detect drift in local regions of the instance space, by using inside inner-nodes of a Decision Tree.

The paper is organized as follows. The next section presents related work in detecting concept drifting and proposes a taxonomy for drift detection. In Section 3 we present the theoretical basis of the proposed method. In Section 4 we discuss evaluation methods in time changing environments, and evaluate the proposed method in one artificial dataset and one real-world dataset. Section 5 concludes the paper and presents future work.

2 Tracking Drifting Concepts

Concept drift means that the concept about which data is being collected may shift from time to time, each time after some minimum permanence. Changes occur over time. The evidence for changes in a concept are reflected in some way in the training examples. Old observations, that reflect the behaviour of nature in the past, become irrelevant to the current state of the phenomena under observation and the learning agent must forget that information.

Suppose a supervised learning problem, where the learning algorithm observe sequences of pairs $(\boldsymbol{x_i}, y_i)$ where $y_i \in \{C_1, C_2, ..., C_k\}$. At each time stamp t the learning algorithm outputs a class prediction $\hat{y}_t$ for the given feature vector $\boldsymbol{x}_t$. Assuming that examples are independent and generated at random by a stationary distribution $\mathcal{D}$, some model class algorithms (e.g. Decision Trees, Neural Networks, etc) can approximate $\mathcal{D}$ with arbitrary accuracy (bounded by the *Bayes error*) whenever the number of examples increases to infinite.

Suppose now the case where $\mathcal{D}$ is not stationary. The data stream consists of sequences of examples $e_i = (\boldsymbol{x_i}, y_i)$. Suppose further that from time to time, the distribution that is generating the examples change. The data stream can be seen as sequences $< S_1, S_2, ..., S_k, ... >$ where each element S_i is a set of examples generated by some stationary distribution $\mathcal{D}_i$. We designate as *context* each one of these sequences. In that case, and in the whole dataset, no learning algorithm can guarantee arbitrary precision. Nevertheless, if the number of observations within each sequence S_i is large enough, we could approximate a learning model to $\mathcal{D}_i$. The main problem is to detect change points whenever they occur. In real problems between two consecutive sequences S_i and S_{i+1} there could be a transition phase where some examples of both distributions appear mixed. An example generated by a distribution $\mathcal{D}_{i+1}$ is noise for distribution $\mathcal{D}_i$. This is another difficulty faced by change detection algorithms. They must differentiate *noise* from *change*. The difference between noise and examples of another distribution is *persistence*: there should be a consistent set of examples of the new distribution. Algorithms for change detection must combine *robustness* to noise with *sensitivity* to concept change.

2.1 The Nature of Change

The nature of change is diverse and abundant. In this work, we identify two dimensions for analysis. The *causes* of change, and the *rate* of change. In a first dimension, the causes of change, we can distinguish between changes due to modifications in the context of learning, because of changes in hidden variables, from changes in the characteristic properties in the observed variables. Existing Machine Learning algorithms learn from observations described by a finite set of attributes. This is the *closed world* assumption. In real world problems, there can be important properties of the domain that are not observed. There could be *hidden* variables that influence the behaviour of nature [7]. Hidden variables may change over time. As a result, concepts learned at one time can become inaccurate. On the other hand, there could be changes in the characteristic properties of the nature.

The second dimension is related to the *rate of change*. The term *Concept Drift* is more associated to gradual changes in the target concept (for example the rate of changes in prices), while the term *Concept Shift* refers to abrupt changes. Usually, detection of abrupt changes are easier and require few examples for detection. Gradual changes are more difficult to detect. Detection algorithms must be resilient to noise. At least in the first phases of gradual change, the perturbations in data can be seen as noise by the detection algorithm. They often require more examples to distinguish change from noise.

Whenever a change in the underlying concept generating data occurs, the class-distribution changes, at least in some regions of the instance space. Nevertheless, it is possible to observe changes in the class-distribution without concept drift. This is usually referred as *virtual drift* [23].

2.2 Characterization of Drift Detection Methods

There are several methods in Machine Learning to deal with changing concepts: [13,12,11,23]. All of these methods assume that the most recent examples are the relevant ones. In general, approaches to cope with concept drift can be analysed into four dimensions: data management, detection methods, adaptation methods, and decision model management.

Data Management. The data management methods characterize the information about data stored in memory to maintain a decision model consistent with the actual state of the nature. We can distinguish:

1. *Full Memory.* Methods that store in memory sufficient statistics over all the examples. Examples include weighting the examples accordingly to their age. Weighted examples are based on the simple idea that the importance of an example should decrease with time. Thus, the oldest examples have less importance, see [16,17].
2. *Partial Memory.* Methods that store in memory only the most recent examples. Examples are stored in a *first-in first-out* data structure. Examples in the *fifo* define a time-window over the stream of examples. At each time

step, the learner induces a decision model using only the examples that are included in the window. The key difficulty is how to select the appropriate window size: a small window can assure a fast adaptability in phases with concept changes but in more stable phases it can affect the learner performance, while a large window would produce good and stable learning results in stable phases but can not react quickly to concept changes.

(a) *Fixed Size* windows. These methods store in memory a fixed number of the most recent examples. Whenever a new example is available, it is stored in memory and the oldest one is discarded. This is the simplest method to deal with concept drift and can be used as a baseline for comparisons.

(b) *Adaptive Size* windows. In this method, the set of examples in the window is variable. It is used in conjunction with a detection model. The most common strategy consists of decreasing the size of the window whenever the detection model signals drift and increasing otherwise.

Dynamic environments with non-stationary distributions require the forgetfulness of the observations not consistent with the actual behaviour of the nature. Drift detection algorithms must not only adapt the decision model to newer information but also forget old information. The memory model also indicates the *forgetting mechanism*. Weighting examples corresponds to a *gradual* forgetting. The relevance of old information is less and less important. Time windows corresponds to *abrupt* forgetting. The examples are deleted from memory. We can combine, of course, both forgetting mechanisms by weighting the examples in a time window, see [11].

Detection Methods. The Detection Model characterizes the techniques and mechanisms for drift detection. One advantage of the detection model is that they can provide meaningful description (indicating change-points or small time-windows where the change occurs) and quantification of the changes. They may follow two different approaches:

1. Monitoring the evolution of performance indicators. Some indicators (e.g. performance measures, properties of the data, etc.) are monitored over time (see [13] for a good overview of these indicators).
2. Monitoring distributions on two different time-windows.

Most of the work in drift detection follows the first approach. Relevant work in this approach is the FLORA family of algorithms developed by [23]. FLORA2 includes a window adjustment heuristic for a rule-based classifier. To detect concept changes, the accuracy and the coverage of the current learner are monitored over time and the window size is adapted accordingly. In the context of information filtering, the authors of [13] propose monitoring the values of three performance indicators: *accuracy*, *recall* and *precision* over time, and their posterior comparison to a confidence interval of standard sample errors for a moving average value (using the last M batches) of each particular indicator. In [12], Klinkenberg and Joachims present a theoretically well-founded method to recognize and handle concept changes using properties of Support Vector Machines. The key idea is to select the window size so that the estimated generalization error on new examples is minimized. This approach uses unlabelled data to reduce the need for labelled data, it does not require complicated parametrization and it works effectively and efficiently in practice.

An example of the latter approach, in the context of learning from Data Streams, has been present by [10]. The author proposes algorithms (statistical tests based on Chernoff bound) that examine samples drawn from two probability distributions and decide whether these distributions are different. In the same line [5] system VFDTc has the ability to deal with concept drift, by continuously monitoring differences between two class-distribution of the examples: the distribution when a node was built and the class-distribution when a node was a leaf and the weighted sum of the class-distributions in the leaves descendant of that node.

Adaptation Methods. The Adaptation model characterizes the adaptation of the decision model. Here, we consider two different approaches:

1. *Blind Methods*: Methods that adapt the learner at regular intervals without considering whether changes have really occurred. Examples include methods that weight the examples according to their age and methods that use time-windows of fixed size.
2. *Informed Methods*: Methods that only modify the decision model after a change was detected. They are used in conjunction with a detection model.

Blind methods adapt the learner at regular intervals without considering whether changes have really occurred. Examples of this approach are *weighted examples* and *time windows* of fixed size. Weighted examples are based on the simple idea that the importance of an example should decrease with time (references related to this approach can be found in: [13,12,18,19,23]).

Decision Model Management. Model management characterizes the number of decision models needed to maintain in memory. The key issue here is the assumption that data generated comes from multiple distributions, at least in the transition between contexts. Instead of maintaining a single decision model several authors propose the use of multiple decision models. A seminal work is the system presented by Kolter and Maloof [15]. The Dynamic Weighted Majority algorithm (DWM) is an ensemble method for tracking concept drift. DWM maintains an ensemble of base learners, predicts target values using a weighted-majority vote of these *experts*, and dynamically creates and deletes experts in response to changes in performance. DWM maintains an ensemble of predictive models, each with an associated weight. Experts can use the same algorithm for training and prediction, but are created at different time steps so they use different training set of examples. The final prediction is obtained as a weighted vote of all the experts. The weights of all the experts that misclassified the example are decreased by a multiplicative constant β. If the overall prediction is incorrect, a new expert is added to the ensemble with weight equal to the total weight of the ensemble. Finally, all the experts are trained on the example. Later, the same authors present the AddExp algorithm [14], a variant of DWM extended for classification and regression, able to prune some of the previous generated experts.

Another important aspect is the *granularity* of decision models. When drift occurs, it does not have impact in the whole instance space, but in particular regions. Adaptation in global models (like naive Bayes, discriminant functions, SVM) require reconstruction of the decision model. Granular decision models (like decision rules and decision trees[1] can adapt parts of the decision model. They only need to adapt those parts that cover the region of the instance space affected by drift. An instance of this approach is the CVFDT algorithm [9] that generate alternate decision trees at nodes where there is evidence that the splitting test is no more appropriate. The system replaces the old tree with the new one when the last becomes more accurate.

3 A Drift Detection Method Based on Statistical Control

In most of real-world applications of Machine Learning data is collected over time. For large time periods, it is hard to assume that examples are independent and identically distributed. At least in complex environments it is highly provable that class-distributions changes over time. In this work we assume that examples arrive one at a time. The framework could be easy extended to situations where data comes on batches of examples. We consider the online learning framework: when an example becomes available, the decision model must take a decision (e.g. a prediction). Only after the decision has been taken, the environment reacts providing feedback to the decision model (e.g. the class label of the example). In the PAC learning model [20], it is assumed that if the distribution of the examples is stationary, the error rate of the learning algorithm (p_i) will decrease when the number of examples (i) increases[2]. This sentence holds for any learning algorithm with infinite-capacity (e.g. decision trees, neural-networks, etc.). A significant increase in the error of the algorithm when trained using more examples, suggests a change in the intrinsic properties in the process generating examples and that the actual decision model is no more appropriate.

3.1 The Method

Suppose a sequence of examples, in the form of pairs $(\boldsymbol{x_i}, y_i)$. For each example, the actual decision model predicts $\hat{y_i}$, that can be either True ($\hat{y_i} = y_i$) or False ($\hat{y_i} \neq y_i$). For a set of examples, the error is a random variable from Bernoulli trials. The Binomial distribution gives the general form of the probability for the random variable that represents the number of errors in a sample of n examples.

[1] Nodes in a decision tree correspond to hyper-rectangles in particular regions of the instance space.

[2] For an infinite number of examples, the error rate will tend to the Bayes error.

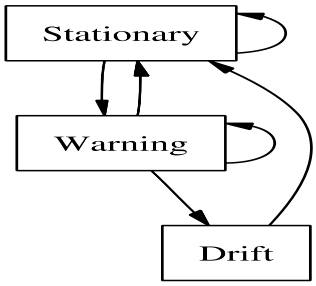

Fig. 1. The Space State Transition Graph

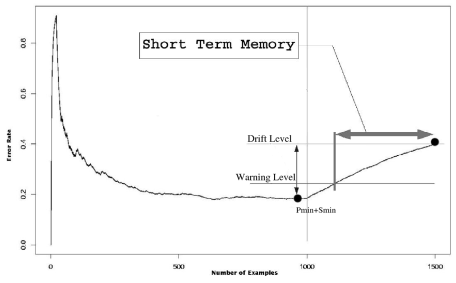

Fig. 2. Dynamically constructed Time Window. The vertical line marks the change of concept.

For each point i in the sequence, the error-rate is the probability of observe False, p_i, with standard deviation given by $s_i = sqrt(p_i(1-p_i)/i)$. The drift detection method manages two registers during the training of the learning algorithm, p_{min} and s_{min}. For each new processed example i, if $p_i + s_i$ is lower than $p_{min} + s_{min}$ these values are updated.

For a sufficient large number of example, the Binomial distribution is closely approximated by a Normal distribution with the same mean and variance. Considering that the probability distribution is unchanged when the context is static, then the $1-\alpha/2$ confidence interval for p with $n > 30$ examples is approximately $p_i \pm z * s_i$. The parameter z depends on the desired confidence level.

Suppose that in the sequence of examples, there is an example j with correspondent p_j and s_j. We define three possible states for the system:

– *In-Control*: while $p_j + s_j < p_{min} + \beta * s_{min}$. The error of the system is stable. The example j is generated from the same distribution of the previous examples.
– *Out-of-Control*: whenever $p_j + s_j > p_{min} + \alpha * s_{min}$. The error is increasing, and reach a level that is significantly higher from the past recent examples. With probability $1-\alpha/2$ the current examples are generated from a different distribution.
– *Warning*: whenever the system is in between the two margins. The error is increasing but without reaching an action level. This is a not decidable state. The causes of error increase can be due to noise, drift, small inability of the decision model, etc. More examples are needed to make a decision.

The graph describing the state transition is presented in figure 1. It is not possible to move from a stationary state to a drift state without passing the warning state. If is possible to move from a warning state to a stationary state. For example, we can observe an increase of the error reaching the warning level, followed by a decrease. We assume that such situations corresponds to a *false alarms*, most probably due to noisy data, without changing of context.

We use a warning level to define the optimal size of the context window. The context window will contain the old examples that are on the new context and a minimal number of examples on the old context. Suppose that, in the sequence of examples that traverse a node, there is an example i with correspondent p_i and s_i. In the experiments described next the confidence level for warning has been set to 95%, that is, the warning level is reached if $p_i + s_i \geq p_{min} + 2 * s_{min}$. The confidence level for drift has been set to 99%, that is, the drift level is reached if $p_i + s_i \geq p_{min} + 3 * s_{min}$. Suppose a sequence of examples where the error of the actual model increases reaching the warning level at example k_w, and the drift level at example k_d. This is an indication of a change in the distribution of the examples. A new context is declared starting in example k_w, and a new decision model is induced using only the examples starting in k_w till k_d.

Figure 2 details the dynamic window structure. With this method of learning and forgetting we ensure a way to continuously keep a model better adapted to the present context.

3.2 Discussion and Limitations

A main characteristic of the detection method we present here is the use of the variance of the error estimate to define the action boundaries. The boundaries are not fixed but decrease with the confidence increase in the error estimates (as far as we have more points for estimation). This method could be applied to any learning algorithm. It could be directly implemented inside online and incremental algorithms, and could be implemented as a wrapper to batch learners. The proposed method assumes that in the flow of training examples there are sequences of examples with a stationary distribution, and this sequences are large enough to ensure some kind of *convergence* of the classifier. We denote those sequences of examples as *context*.

From the practical point of view, when a drift is signalled, the method defines a dynamic time window of the more recent examples used to train a new classifier. Here the key point is how fast the change occurs. If the change occurs at slow rate, the prequential error will exhibit a small positive slope. More examples are needed to evolve from the warning level to the action level and the window size increases. For abrupt changes, the increase of the prequential error will be also abrupt and a small window (using few examples) will be chosen. In any case the ability of training a new accurate classifier depends on the rate of changes and the capacity of the learning algorithm to converge to a stable model. The last aspect depends on the number of examples in the context.

3.3 Local Drift Detection

In the previous section we described a general method for change detection. The method can be applied as a wrapper with any classification learning algorithm. The inconvenience of this approach is that a new decision model must be learned from scratch whenever a change is detected. This behaviour is useful if and only if the concept change in the instance space as a whole, which is rare. In most cases, changes occur in some regions of the instance space [4]. Some decision models fit different models to different regions of the instance space. This is the case of rule learners and decision trees. In decision trees, each leaf (and each internal node) corresponds to a hyper-rectangle in the instance space. The root node covers all the instance space, and descent nodes covers sub-spaces of the space covered by their parent node.

This is the rationale behind the algorithm that follows. Each inner node in a decision tree is bounded with a drift detection mechanism that traces the training examples traversing the node. If it triggers a change only the sub-tree rooted at that node is pruned. The node becomes a leaf, and a new subtree starts to be learned. In this way, changes that only affect particular regions in the instance space can be captured locally by the nodes that cover these regions. A major advantage of this schema is the reduced cost of updating the decision model.

The drift detection mechanism could be implemented in several ways. One possibility is to consider very simple classifiers (like the majority class) in each internal node of the tree. We have obtained a good traded between simplicity and faster detection rates using naive Bayes classifiers. We have implemented an incremental decision tree that maintains at each inner node a naive Bayes classifier. The decision tree algorithm is oriented towards processing data streams. It is based on VFDT algorithm [21] oriented towards continuous attributes. It uses the splitting criteria presented in UFFT [6] algorithm. The decision model starts with a single leaf. When there is statistical evidence in favor to a splitting test, the leaf becomes a decision node, and two descendant leaves are generated. The leaves store the sufficient statistics to computing the merit (information gain) of each splitting test. These sufficient statistics constitute also a naive Bayes classifier used for drift detection. After the leaf becomes a node, all examples that traverse the node will be classified by the naive-Bayes. The basic idea of the drift detection method is to control this online error-rate. If the distribution of the examples is stationary, the error rate of naive-Bayes decreases. If there is a change on the distribution of the examples the naive-Bayes error increases. When a naive Bayes detects a statistical significant increase of the error, it is signalled a change in the distribution of the examples that falls in the instance space covered by the corresponding node. This suggest that the splitting-test installed at this node is no longer appropriate. The subtree rooted at that node is pruned, and the node becomes a leaf. All the sufficient statistics of the leaf are initialized.

An advantage of this method is that it continuously monitors the online error of naive Bayes. It can detect changes at any time. All internal nodes contain naive Bayes to detect local changes. This correspond to detect shifts in different regions of the instance space. Nodes near the root should be able to detect abrupt changes in the distribution of the examples, while deeper nodes should detect localized, smoothed and gradual changes.

4 Experimental Evaluation

In this section we describe the evaluation of the proposed method in two scenarios: at a global level, as a wrapper over a decision tree learning algorithm[3], and at local level where drift detection is used in each node of a decision tree.

We use two datasets: the SEA concepts, previously used in concept drift detection [22] and the Electricity Market Dataset, a real-world problem previously used in [8]. Using artificial datasets allow us to control relevant parameters to evaluate drift detection algorithms. For example, we can measure how fast the detection algorithm reacts to drift.

Evaluation Methodology in Drift Environments. Changes occur over time. Drift detection algorithms assume that data is sequential. Standard evaluation methods, like cross-validation, assume examples are independent and identically distributed. How to evaluate learning algorithms in sequential data? The main constraint is that we must guarantee that all test examples have a time-stamp larger than those in the training data. Two viable alternatives are:

1. Train-Test: hold-out an independent test set. Given a large enough training and test sets, the algorithm learns a model from the training set and makes predictions for test set examples. It is assumed that all test examples have a time-stamp larger than those in training data.
2. Predictive Sequential (*Prequential*) [3], where the error of a model is computed from the sequence of examples. For each example in the sequence, the actual model makes a prediction for the next example. The prediction is then compared to the observed value. The prequential-error is a computed based on an accumulated sum of a loss function between the prediction and observed values.

The train-test method provides a single point estimate of the error. The prequential approach provides much more information. It uses all available data for training and test, providing a kind of learning curve that traces the evolution of the error. The main problem of this approach is its sensitivity to the order of the examples.

Error rate is the most relevant criteria for classifier evaluation. Other criteria relevant for change detection methods include:

1. The number of examples required to detect a change after the occurrence of a change.
2. Resilience to noise. That is, ability to not detect drift when there is no change in the target concept. We designate this type of errors as Type 1 error.
3. Type 2 error: does not detect drift when drift exist.

Evaluation on Stationary Environments. We have tested the method using the dataset *ADULT* [2]. This dataset was created using census data in a specific point of time. Most probable, the concept is stable. Using a decision tree as

[3] We have used the CART implementation in R.

Table 1. Error rate of Decision Tree algorithms on SEA Concepts. The table reports the error on an independent test set from the last concept. The second and third columns report the error setting on/off the drift detection method as a wrapper over the learning algorithm. Similarly, the fourth and fifth columns refer to local drift detection.

	Wrapper		**Local**		
	No detection	Detection	No Detection	Detection	CVFDT
Mean	15.71	13.72	14.65	12.51	14.72
Variance	0.29	0.28	1.26	1.89	1.06

inducer, the proposed method as a wrapper did not detect any drift; when used locally we observed 1.7% of drift signals (using 10-fold cross validation), all of them triggered by deeper nodes in the tree. This is an important aspect, because it presents evidence that the method is robust to false alarms.

Results on Artificial Domains - SEA concepts. The goal of these experiments is to study the effect of the proposed drift detection method on the generalization capacity of each learning algorithm. The use of artificial datasets allow us to design controlled experiments, defining where drift occurs and at which rate it occurs. We show the method is efficient in detecting drift, and is independent of the learning algorithm.

The *sea concepts* dataset has been used as a benchmark in drift detection problems [22]. For our analysis, it is interesting to describe the process used to generate data. We first generate 60000 random points in a three-dimensional space. All features take values between 0 and 10. Those points are divided into 4 blocks with different concepts. In each block, a data point belongs to class + if $f_1 + f_2 < \theta$, where f_i represents feature i and θ represents a threshold value between the two classes. We use threshold values of 8,9,7, and 9.5 for the four data blocks. We introduce class noise by changing the class label in 10% of the data points in each block. We should note that attribute f_3 is irrelevant. Transitions between concepts are abrupt. The first transition (at example 15k) affects a smaller region of the instance space than the other two changes (at examples 30k and 45k).

The Electricity Market Dataset. The data used in this experiments was first described by M. Harries [8]. The ELEC2 dataset contains 45312 instances dated from 7 May 1996 to 5 December 1998. Each example of the dataset refers to a period of 30 minutes, i.e. there are 48 instances for each time period of one day. Each example on the dataset has 5 fields, the day of week, the time stamp, the NSW electricity demand, the Vic electricity demand, the scheduled electricity transfer between states and the class label. The class label identifies the change of the price related to a moving average of the last 24 hours. The class level only reflects deviations of the price on a one day average and removes the impact of longer term price trends. The interest of this dataset is that it is a real-world dataset. It is not known if there exists drift or when it occurs.

We consider two problems. The first problem (last-day) consists in short term predictions. The test set contains the examples corresponding to the last day. The

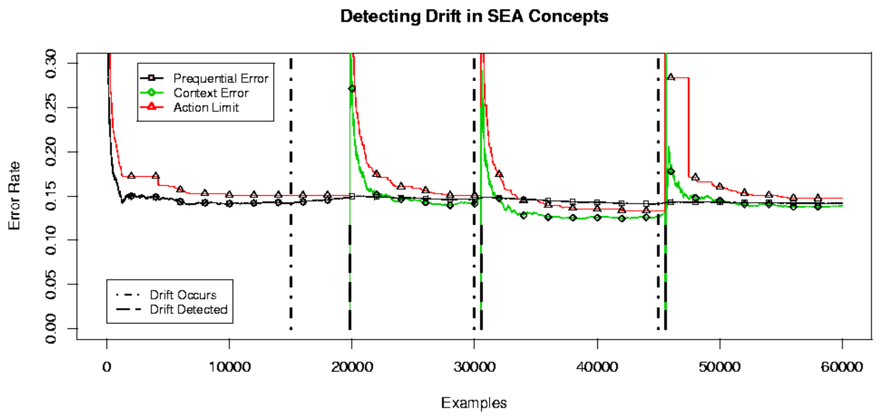

Fig. 3. Detecting Drift in SEA concepts. The figure plots the prequential error, the context error and the drift action level using a Decision tree learning algorithm.

Table 2. Evolution of the context error at the end of each Context of a decision tree and a GLM model. For each context we show the number of example needed to reach the warning and action levels. For both algorithms no false alarm was signalled.

	Context 1			Context 2			Context 3			Context 4
Algorithm	Error	Warn	Detect	Error	Warn	Detect	Error	Warn	Detect	Error
Decision Tree	14.18	1788	4822	14.12	143	527	12.63	150	526	13.84
GLM	11.30	290	1220	11.03	245	604	11.25	70	507	10.52

Table 3. Error rate on an independent test set using examples from the last day and last week on the Elect problem. The second and third columns report the error setting on/off the drift detection method as a wrapper over the learning algorithm. Similarly, the fourth and fifth columns refer to local drift detection.

	Wrapper		**Local**		
	No detection	Detection	No Detection	Detection	CVFDT
Last Week	23.52	20.45	25.89	19.64 (8)	18.21
Last Day	18.75	12.50	16.67	12.50 (8)	12.50

second problem (last-week)consists in predicting the changes in the prices relative to the last week of examples recorded. The learning algorithm used is the implementation of CART available in R. The algorithm learns a model from the training data. We use the detection algorithm as a wrapper over the learning algorithm. After seeing all the training data, the final model classifies the test data. In the 1-day dataset, the trees are built using only the last 3836 examples on the training dataset. In the 1-week dataset, the trees are built with the 3548 most recent examples. This is the data collected since 1998/09/16. Table 3 shows the error rate obtained with the 1-day and 1-week prediction.In both problems the controlling drift allow substantial improvements in performance. This is an evidence indicating the presence of drift in the dataset and the advantages of using control mechanisms to detect and reacting to drift in real-world applications.

5 Conclusions

In this work we analyse and discuss change detection methods for machine learning. We present a method for concept drift detection in the distribution of the examples. The drift detection method continuously monitors the prequential error of a learning algorithm searching for significant deviations from previous stable state. It can be applied to problems where the information is available sequentially over time. The method can be used either as a wrapper over any learning algorithm or locally at each internal node of a decision tree. When a drift is detect, the wrapper approach requires learning a new model, the local approach only require to adapt the part of the decision model covering the region of the instance space affected by the change. In the experimental section we present controlled experiments using artificial data that illustrate the ability of the method for detecting abrupt and smoothed changes. The method is resilient to false alarms, and improves the learning capability of learning algorithms when modelling non-stationary problems. We are now exploring the application of the method with other loss-functions. Preliminary results in the regression domain using *mean-squared error* loss function confirm the results presented here.

Acknowledgements. The authors reveal its gratitude to the projects Adaptive Learning Systems II(POSI/EIA/55340/2004), RETINAE, and FEDER through the plurianual support to LIACC.

References

1. Michele Basseville and Igor Nikiforov. *Detection of Abrupt Changes: Theory and Applications*. Prentice-Hall Inc, 1987.
2. C. Blake, E. Keogh, and C.J. Merz. UCI repository of Machine Learning databases, 1999.
3. A. P. Dawid. Statistical theory: The prequential approach. *Journal of the Royal Statistical Society-A*, 147:278–292, 1984.
4. Wei Fan. Systematic data selection to mine concept-drifting data streams. In J. Gehrke and W. DuMouchel, editors, *Proceedings of the Tenth International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2004.
5. J. Gama, R. Fernandes, and R. Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1):23–46, 2006.
6. Joï¿½o Gama, Pedro Medas, and Pedro Rodrigues. Learning decision trees from dynamic data streams. In Hisham Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 573–577. ACM Press, March 2005.
7. M. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Machine Learning*, 32:101, 1998.

8. Michael Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales, 1999.
9. Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM Press, 2001.
10. Daniel Kifer, Shai Ben-David, and Johanes Gehrke. Detecting change in data streams. In *VLDB 04: Proceedings of the 30th International Conference on Very Large Data Bases*, pages –. Morgan Kaufmann Publishers Inc., 2004.
11. R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 2004.
12. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In Pat Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 487–494, Stanford, US, 2000. Morgan Kaufmann Publishers.
13. R. Klinkenberg and I. Renz. Adaptive information filtering: Learning in the presence of concept drifts. In *Learning for Text Categorization*, pages 33–40. AAAI Press, 1998.
14. J. Kolter and M. Maloof. Using additive expert ensembles to cope with concept drift. In L Raedt and S. Wrobel, editors, *Machine Learning, Proceedings of the 22th International Conference*. OmniPress, 2005.
15. Jeremy Z. Kolter and Marcus A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the Third International IEEE Conference on Data Mining*, pages 123–130. IEEE Computer Society, 2003.
16. I. Koychev. Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning. Berlin, Germany*, pages 101–106, 2000.
17. I. Koychev. Learning about user in the presence of hidden context. In *Proceedings of Machine Learning for User Modeling: UM-2001.*, 2001.
18. C. Lanquillon. *Enhancing Text Classification to Improve Information Filtering.* PhD thesis, University of Madgdeburg, Germany, 2001.
19. M. Maloof and R. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41:27–52, 2000.
20. Tom Mitchell. *Machine Learning.* McGraw Hill, 1997.
21. Domingos P. and Hulten G. Mining High-Speed Data Streams. In *Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, 2000.
22. W. Nick Street and YongSeog Kim. A streaming ensemble algorithm SEA for large-scale classification. In *Proc. seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM Press, 2001.
23. Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.