Online Classification of Nonstationary Data Streams

Mark Last[1]
Ben-Gurion University of the Negev
Department of Information Systems Engineering
Beer-Sheva 84105, Israel
Email: mlast@bgumail.bgu.ac.il

Abstract

Most classification methods are based on the assumption that the data conforms to a stationary distribution. However, the real-world data is usually collected over certain periods of time, ranging from seconds to years, and ignoring possible changes in the underlying concept, also known as *concept drift*, may degrade the predictive performance of a classification model. Moreover, the computation time, the amount of required memory, and the model complexity may grow indefinitely with the continuous arrival of new training instances. This paper describes and evaluates OLIN, an online classification system, which dynamically adjusts the size of the training window and the number of new examples between model re-constructions to the current rate of concept drift. By using a fixed amount of computer resources, OLIN produces models, which have nearly the same accuracy as the ones that would be produced by periodically re-constructing the model from all accumulated instances. We evaluate the system performance on sample segments from two real-world streams of non-stationary data.

Keywords. Classification, incremental learning, online learning, concept drift, info-fuzzy networks.

---

[1] Also with Department of Computer Science and Engineering, University of South Florida, USA

1. Introduction

The so-called Information Age has provided us with huge amounts of digital data, which keep growing at unprecedented pace. Business information systems, Internet servers, and telecommunication providers are producing and storing new data every day, hour, and even minute. The reported rates include the maximum amount of 17,400 Web page requests per minute at a single university campus [6], 500,000 transactions recorded during less than one day by the Lycos search engine [1], and tens of millions of queries sent every day to Yahoo! Search [25]. The accumulated data may contain some valuable information for an organization that is storing it. Data mining is the core stage of the KDD (Knowledge Discovery in Databases) process, which is aimed at "identifying valid, novel, potentially useful, and ultimately understandable patterns in data" [9]. The problems of knowledge discovery in online data include real-time monitoring of manufacturing processes, prediction of stock prices, and intrusion detection in computer networks.

Continuous streams of data pose new problems to classification methods of data mining, like CART [4], ID3 [28], C4.5 [29], IFN [23], and many others. The common approach of these methods is to store and process the entire set of training examples. The growing amounts of training data increase the processing requirements of data mining systems up to a point, where they either run out of memory, or their computation time becomes prohibitively long. Furthermore, even if all the available examples can be handled by the system, the patterns discovered by an algorithm in the data from the past, may be hardly valid and useful for the new data obtained hours or even minutes later due to unexpected changes in the data-generating process (e.g., a political event which affects the stock prices).

Methods for extracting patterns from continuous streams of data are known as *incremental (online) learning algorithms*. The basic idea of incremental induction is that upon receiving a new instance, it is much less expensive to update an existing model than to build a new one. On the other hand, as indicated in [6], the incremental algorithms suffer from several shortcomings, like high sensitivity to the order of training examples and longer training times than the non-incremental (batch) methods. Pure incremental methods consider *every* new instance, which may be impractical in environments, where transactions arrive at the rate of thousands per second. The algorithms for purely incremental learning include WINNOW [21], COBWEB [10], ITI [31], and the stochastic gradient descent algorithm for training a neural network (see [27]). Most of these methods are focused on efficient ways of inducing a classification model from *stationary* data streams.

The general problem of learning a *drifting concept* that is a concept which changes over time is studied by Helmbold and Long in [12]. The rate of drift is defined there as the probability that the target function disagrees over two successive examples. Disagreements are minimized by an algorithm, which is polynomial in the sample size, given that the rate of drift is bounded. Thus, for high-volume non-stationary data streams, where the actual rate of drift is unknown in advance, the run time of the algorithm may grow indefinitely.

Widmer and Kubat [32] describe a family of purely incremental algorithms for learning in the presence of drift. These algorithms use a simple representational framework, called FLORA, which stores descriptions of positive, negative, and noisy examples in separate description sets. One of the incremental algorithms, FLORA2, maintains a dynamically adjustable window of the latest training examples. Whenever a concept drift is suspected, due to a drop in predictive accuracy or an explosion in the number of descriptions, the window size is decreased, by discarding the oldest examples. If the concept appears to be stable, the window size is left unchanged. As long as the presence of drift is uncertain, no examples are forgotten, thus incrementally increasing the window size. According to [32], this window adjustment strategy may efficiently detect radical changes in the underlying concept, subject to a relatively low *rate of change*. The FLORA algorithms also assume a limited rate of data arrival, since they process one example at a time.

A recent paper by Domingos and Hulten [6] deals directly with the problem of mining high-speed streams of data. Their data mining system, called VFDT, builds decision trees from symbolic attributes by using sub-sampling of the entire data stream generated by a stationary process. A similar assumption of stationary concepts is used by the incremental method of Fan et al. [7]. The sample size is determined in VFDT from distribution-free Hoeffding bounds. A new version of the VFDT system, called CVFDT [16], learns decision trees from continuously changing data streams by repeatedly applying the VFDT algorithm to a sliding window of fixed size. CVFDT is aimed at detecting only one type of concept drift at the node level of the tree: namely, the importance of the current input attribute vs. other attributes. The algorithm grows an alternative subtree for each attribute having a relatively high information gain and replaces the old subtree when a new one becomes more accurate.

Harries and Sammut [11] have developed an off-line meta-learning system (based on C4.5) for partitioning the data stream into a set of time-dependent "conceptual clusters." The off-line approach is aimed at analyzing concept drifts in historic data rather than at the online detection of evolving changes. A strategy for dynamically updating a linear regression classifier by using a linear model of dynamic behavior is presented in [17]. The linear approach of[17] is computationally efficient, but it obviously restricts the search space by eliminating nonlinear concepts and nonlinear (especially, abrupt) patterns of concept drift.

Time-dependent changes in the class distributions of rules induced from data can be detected by the CD3 algorithm of Black and Hickey [3][13]. CD3 treats the time-stamp as an additional input attribute in a decision tree. Consequently, paths where the value of the time-stamp attribute refers to the old period(s) represent rules, which are out of date. When the process is stable for a long period of time, the time-stamp attribute should not appear in any path of the tree.

Closely associated with the problem of change detection is the task of discovering the *robust knowledge*, which is unlikely to be affected by database changes. A Bayesian Network model for evaluating robustness of database rules is described in [14]. The network probabilities are estimated by an off-line procedure, which assumes stationarity of the database transactions.

Another related area is *change detection*, including an increasingly important problem of *intrusion detection* in computers and computer networks. Lane and Brodley [18] suggest a compromise between purely batch and purely incremental learning for the task of detecting abnormal (possibly hostile) behavior of a computer user: their algorithm is trained on a batch of the most recent user transactions. However, the minimum batch size (80 examples) makes the system vulnerable to short-term cyber attacks. According to Fawcett and Provost [8], the most efficient method for detecting an intruder is the *profiling* approach, where a model of normal (stationary) activity is built and then used to alarm on significant deviations from the normal.

This paper proposes an online classification system, which uses an info-fuzzy network [23], or IFN, as a base classifier. As shown in [19][23], the IFN method is able to produce much more compact models than other decision-tree methods, like CART and C4.5, while preserving nearly the same level of predictive accuracy. Moreover, it can also be used as an efficient feature selection method [20]. The proposed system, called OLIN for On-Line Information Network, adapts itself automatically to the rate of concept drift in a non-stationary data stream by dynamically adjusting the size of the training window and the rate of model update. The system does not impose any limitations on the rate, the extent, or the type of change in the underlying concept. Like the batch version of the IFN method, it can handle both discrete and continuous attributes. OLIN saves computer resources by increasing the update cycle when the concept appears to be stable and it shrinks the size of the training window, whenever a concept drift is detected. Thus, OLIN can be applied to a time-changing data stream of arbitrary duration. The cumulative accuracy of the models produced by OLIN tends to be higher than the accuracy obtained with a fixed-size sliding window though it may be slightly lower than the accuracy of an incremental system that does not "forget" any past examples.

This paper is organized as follows. In the next section, we provide a brief overview of the IFN method and its features. The OLIN system is described in the following section. We then present the empirical results of applying OLIN to real-world streams of non-stationary data coming from two different domains: semiconductor manufacturing and stock market. The paper concludes with a discussion of results and future work.

2. Information Networks

2.1. Network Structure

IFN, or info-fuzzy network [23] is a tree-like classification model, which is designed to minimize the total number of predicting attributes. Beyond classification, this model can be used for the tasks of discretization [23], feature selection [20], rule induction [19], and evaluation of data reliability [24]. An info-fuzzy network has the following components:

1) I - a subset of *input* (predicting) attributes used by the model. Input attributes are selected from the set C of *candidate input* attributes (available features).

2) $|I|$ - total number of *hidden* layers (levels) in a network. Unlike the standard decision tree structure used in CART [4], ID3 [28], and C4.5 [29], where the nodes of the same tree level are independent of each other, all nodes of a given network layer are labeled by the same input attribute associated with that layer. This is why the number of network layers is equal to the number of input attributes. In layers associated with continuous attributes, an information network uses multiple splits, which are identical at all nodes of the corresponding layer. Most other decision-tree methods apply only binary splits in each layer. The first layer in the network (Layer 0) includes only the root node and is not associated with any input attribute.

3) L_l - a subset of nodes *z* in a hidden layer *l*. Each node represents an attribute-based test, similarly to a standard decision tree. If a hidden layer *l* is associated with a nominal input attribute, each outgoing edge of a non-terminal node corresponds to a distinct value of an attribute. For continuous features, the outgoing edges represent consecutive intervals obtained from the discretization process. If a node has no outgoing edges, it is called a terminal node.

4) *K* - a subset of target nodes representing distinct values of the target (classification) attribute. For continuous target attributes, the target nodes represent disjoint intervals in the attribute range. A target layer is missing in the standard decision-tree structure. The connections between terminal nodes and the nodes of the target layer may be used for extracting information-theoretic rules from a network [19].

In Figure 1, a structure of a two-layered info-fuzzy network (based on two selected input attributes) is shown. The first input attribute has three values, represented by nodes no. 1,2, and 3 in the first layer, but only nodes no. 1 and 3 are split by the network construction procedure described in sub-section 2.2 below. The second layer has four nodes standing for the combinations of two values of the second input attribute with two split nodes of the first layer. The target attribute has three values, represented by three nodes in the target layer. New examples can be classified by an info-fuzzy network in a similar way to standard decision trees: we start at the root node, test the attribute associated with the first layer, then move along the network path corresponding to the value of the first input attribute. The process continues until a terminal node is encountered (nodes 1,1; 1,2; 2; 3,1; and 3,2 in the network of Figure 1), at which time an example is labeled with a single predicted class having the maximum probability at the node or with the probability distribution of classes.

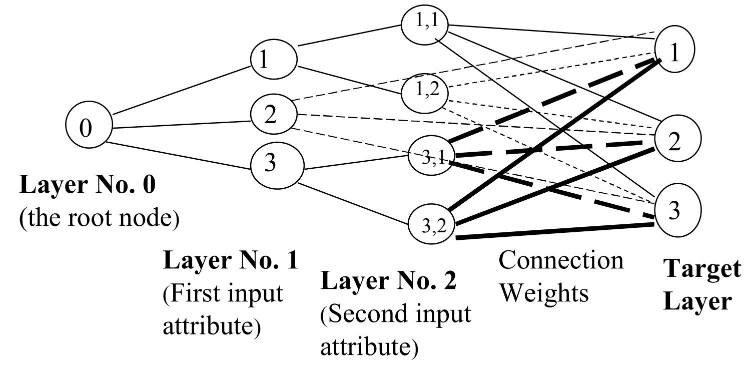

Figure 1 Info-Fuzzy Network - Two-Layered Structure

The connectionist nature of the info-fuzzy network (each terminal node is connected to every target node) resembles the topological structure of multi-layer neural networks (see [27]), which also have input and output nodes and a variable number of hidden layers. Consequently, we define our model as a *network* and not as a *tree*.

2.2. Network Construction Procedure

The network construction algorithm (called here IN for Information Network) starts with defining the target layer and the "root" node representing an empty set of input attributes. Unlike CART [4] and C4.5 [29], IFN is built only in one direction (top-down). After the construction is stopped, there is no bottom-up post-pruning of the network branches. As explained below, the network is *pre-pruned* by applying statistical significance testing.

A node z can be split on an input attribute $A_{i'}$ only if the split provides a statistically significant increase in the *mutual information* of z and the target attribute A_i. Mutual information (see [5]) is an information-theoretic measure of association between two random variables *X* and *Y,* which is defined as a decrease in the entropy of *Y* as a result of knowing *X* (and vice versa). An increase in mutual information, also called *conditional mutual information* [5] or *information gain* [28], of a candidate input attribute $A_{i'}$ and the target attribute A_i, given a node *z,* is calculated by the following expression (based on [5]):

$$MI(A_{i'}; A_i / z) = \sum_{j=0}^{M_i - 1} \sum_{j'=0}^{M_{i'} - 1} P(V_{ij}; V_{i'j'}; z) \bullet \log \frac{P(V_{i'j'}^{ij} / z)}{P(V_{i'j'} / z) \bullet P(V_{ij} / z)} \quad (1)$$

where

M_i / $M_{i'}$ - number of distinct values of the target attribute *i* /candidate input attribute *i'*.

$P(V_{i'j'}/z)$ - an estimated conditional (*a posteriori*) probability of a value *j'* of a candidate input attribute *i'* given a node *z*

$P(V_{ij}/z)$ - an estimated conditional (*a posteriori*) probability of a value *j* of the target attribute *i* given a node *z*.

$P(V_{i'j'}^{ij}/z)$ - an estimated conditional (*a posteriori*) probability of a value *j'* of a candidate input attribute *i'* and a value *j* of the target attribute *i* given a node *z*.

$P(V_{ij}; V_{i'j'}; z)$ - an estimated joint probability of a value *j* of the target attribute *i*, a value *j'* of a candidate input attribute *i'* and a node *z*.

Conditional mutual information measures the benefit of adding an input attribute to the information network. If the input and the target attributes are conditionally independent given a node *z*, their conditional joint probability should be equal to the product of their individual conditional probabilities. This makes the logarithmic terms in Equation (1) equal to zero. On the other hand, if the knowledge of an input value either increases, or decreases the conditional probability of a target value, the corresponding summation term in (1) becomes either positive, or negative respectively.

If a tested attribute is continuous, its values in Equation (1) correspond to thresholds, which maximize an increase in mutual information. Prior to adding a new layer, the algorithm re-computes the best threshold splits of each continuous attribute that is not in the network. More details on discretizing and selecting continuous attributes are provided in [20] and [23].

The statistical significance of the estimated conditional mutual information, is evaluated by using the likelihood-ratio statistic (based on [1]):

$$G^2 (A_{i'} ; A_i / z) = 2 \bullet (\ln 2) \bullet E^*(z) \bullet MI (A_{i'} ; A_i / z) \qquad (2)$$

Where $E^*(z)$ is the number of records associated with the node z

The Likelihood-Ratio Test [30] is a general-purpose method for testing the null hypothesis H_0 that two discrete random variables are statistically independent. As can be seen from Equation (1), independence of two attributes implies that their expected mutual information is zero. If H_0 holds, then the likelihood-ratio test statistic $G^2 (A_{i'}; A_i / z)$ is distributed as chi-square with $(NI_{i'} (z) - 1) \bullet (NT_i (z) - 1)$ degrees of freedom, where $NI_{i'}(z)$ is the number of values of a candidate input attribute i' at node z and $NT_i (z)$ is the number of values of the target attribute i at node z (based on [30]). Thus, $MI (A_{i'} ; A_i / z)$ is considered statistically significant if H_0 can be rejected at the significance level α:

$$G^2 (A_{i'} ; A_i / z) \geq \chi^2_{\alpha} ((NI_{i'} (z) - 1) \bullet (NT_i (z) - 1)) \qquad (3)$$

The default value of α used by the IN algorithm is 0.1%. We have found empirically that in most datasets, higher values of α tend to decrease the generalization performance of the model.

At each iteration, the algorithm builds a new hidden layer by choosing an input attribute (either discrete, or continuous), which provides the maximum significant increase in mutual information relative to the previous layer. The nodes of a new layer are defined for a Cartesian product of split nodes of the previous layer and the values of a new input attribute. The chain rule of the information theory (see [5]) implies that the mutual information between an info-fuzzy network and the target attribute is equal to the sum of drops in conditional entropy (information gains) across all hidden layers. If there is no candidate input attribute significantly increasing the mutual information, the network construction is stopped and the algorithm outputs the final network structure.

3. The OLIN System

3.1. Algorithm Description

The OLIN (On-Line Information Network) system is a wrapper for the Information Network (IN) algorithm presented in the previous section. OLIN is receiving a continuous stream of data examples. The system repeatedly applies the IN algorithm to a sliding window of training examples and it dynamically adapts the size of the training window and the frequency of model re-construction to the current rate of concept drift. The OLIN system is limited neither by the overall duration of a data stream, nor by the cumulative number of examples arrived. At each point in time, the purpose of the system is to predict a correct class for the next arriving example by using a current classification model. We assume that immediately afterwards the correct classification becomes available to the wrapper. This assumption, used by many incremental learning methods

[32], is reasonable as long as the delay between receiving an example and receiving its class is negligible with respect to the time between the arrivals of successive examples.

The general architecture of the OLIN system is presented in Figure 2. The system has three modules: the Learning Module, which implements the IN algorithm to produce an info-fuzzy network; the Classification Module, which uses the current network to classify the incoming examples; and the Meta-Learning Module, which controls the operation of the Learning Module. In the sample data stream of Figure 2, each one of V_0 examples in the validation interval $[t_2, t_3]$ is classified by a model induced from T_0 examples of the training interval $[t_0, t_2]$. The number of examples in the training and the validation intervals do not have to be equal. At the time t_3 the network is re-constructed by the Learning Module using T_1 examples from the training interval $[t_1, t_3]$ and subsequently applied to V_1 examples in the validation interval $[t_3, t_4]$. We assume here that the first example in the interval $[t_3, t_4]$ arrives after the network construction has been completed. In massive data streams, examples that arrive in the process of network construction may be classified by a partially constructed model using the anytime nature of the IN algorithm (see [23]).

The Meta-Learning Module obtains as input the training and the validation accuracy rates of the model, measured on the training and the validation intervals respectively. It also gets the description of the model itself (selected attributes, entropy information, etc.). Using the OLIN algorithm (see below), the module re-calculates the size of the next training window (interval) and the number of validation examples to be classified with the new model. The last validation example of the current window is always the last training example of the next window. To classify every example in the data stream exactly one time, the validation intervals (*[t_2, t_3], [t_3, t_4]*, etc.) have to be disjoint and consecutive. However, the training windows may overlap with each other (but not with their respective validation intervals).

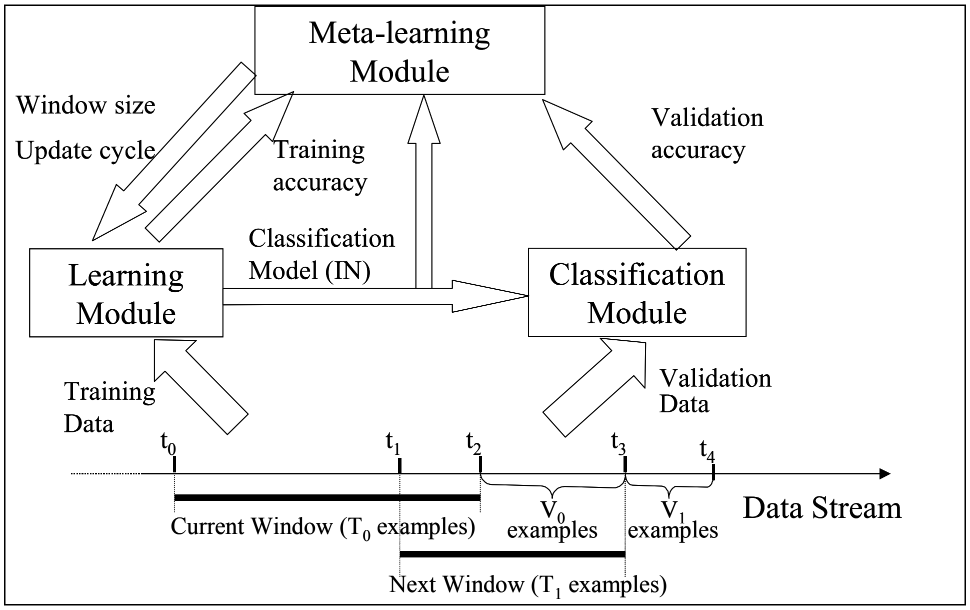

Figure 2 OLIN System Architecture

The basic intuition behind the OLIN algorithm resembles the FLORA2 framework [32]: a narrow training window may be too small to identify any stable concept with a high degree of confidence; on the other hand, a wide training window may completely miss short-term changes in the underlying concept. Since the timing, the rate, and the extent of concept drift in a data stream are not known in advance, the window size should be adjusted dynamically in the process of data arrival. A good heuristic is to shrink the training window to a minimum size, when a drift seems to occur, and increase the window (up to a certain limit dictated by the data size and the available computer resources) if stability is observed. OLIN uses the statistical significance of the difference between the training and the validation accuracy of the current model as an indicator of concept stability.

The purely incremental approach of FLORA2 does not seem appropriate for massive data streams, where the concepts are not expected to change drastically between the arrivals of consecutive instances. For this reason, the CVFDT algorithm of Hulten et al. [16] checks for drift only once in a fixed number of examples (20,000). The OLIN approach is to adjust dynamically the number of examples between model re-constructions by using the following heuristic: keep the current model for more examples if the concept appears to be stable and reduce drastically the size of the validation window, if a concept drift is detected.

Table 1 shows the pseudo-code outline of the OLIN algorithm. The algorithm does some initializations and then processes a user-specified number of incoming examples from a continuous data stream. If the user does not provide the number of the last example to be classified by the system, the algorithm runs indefinitely. The following parameters are calculated by OLIN: initial size of the training window, updated window size, and the maximum difference between the training and the validation errors. These calculations are described in the next sub-sections.

The IN algorithm, like other batch decision-tree methods, stores all the training examples in the computer memory. Thus, applying IN to an indefinitely growing training window is not feasible. Consequently, we have limited the maximum size of the training window used by OLIN to *Max_Win* examples. This number can be adjusted to the amount of available memory on a given computer. Another important parameter is the training time required per each new example. As shown by us in [23], the computation time of the IN algorithm is directly proportional to the number of training examples for each discrete attribute and to the square of this number, if an attribute is continuous. This imposes an additional limitation on the size of the training window, which can be handled by a given computer system. Since the size of the training window is bounded by *Max_Win* and the minimum number of examples in a validation interval is *Min_Add_Count*, we can say that during the slowest periods of its operation, the training time of OLIN per a discretely-valued new example can be proportional to *Max_Win* / *Min_Add_Count.* With continuous attributes, this value increases to $(Max\_Win)^{2}$ / *Min_Add_Count.* However, when the concept appears to be stable, the algorithm training time will go down to the order of *Max_Win* / *Max_Add_Count* or $(Max\_Win)^{2}$ / *Max_Add_Count*, depending on the nature of the data stream attributes.

Table 1 The OLIN Algorithm

Inputs:	*S* -	A continuous stream of examples
	n_{min}	The number of the first example to be classified by the system ($n_{min} – 1$ examples have already arrived)
	n_{max}	The number of the last example to be classified by the system (if unspecified, the system will run indefinitely)
	C	A set of candidate input attributes (discrete and continuous)
	Sign	A user-specified significance level
	P_e	Maximum allowable prediction error of the model
	Init_Add_Count	The number of new examples to be classified by the first model
	Inc_Add_Count	Amount (percentage) to increase the number of examples between model re-constructions
	Max_Add_Count	Maximum number of examples between model re-constructions
	Red_Add_Count	Amount (percentage) to reduce the number of examples between model re-constructions
	Min_Add_Count	Minimum number of examples between model re-constructions
	Max_Win	Maximum number of examples in a training window
Output:	*IFN*	Info-fuzzy network

```
Procedure OLIN
Calculate the initial size of the training window $W_{init}$ (using Equation 7)
Let the training window size $W = W_{init}$
Initialize the index $i$ of the first training example to $n_{min} - W$
Initialize the index $j$ of the last training example to $W$
Initialize the number of validation examples Add_Count to Init_Add_Count
While $j < n_{max}$ Do
        Obtain a model (IFN) by applying the IN algorithm to $W$ latest training examples
        Calculate the training error rate $E_{tr}$ of the obtained model on $W$ training examples
        Calculate the index of the last validation example $k = j + Add\_Count$;
        Calculate the validation error rate $E_{Val}$ of the obtained model on Add_Count validation examples
        Update the index of the last training example $j = k$
        Find the maximum difference between the training and the validation errors Max_Diff (using Equation 10)
        If $(E_{Val} - E_{tr}) < Max\_Diff$      // concept is stable
                Add_Count =Min( Add_Count * (1+ (Inc_Add_Count/100)), Max_Add_Count)
                W = Min (W + Add_Count, Max_Win)
        Else                                     //concept drift detected
                Re-calculate the size of the training window W (using Equation 8)
                Update the index of the first training record $i = j - W$
                Add_Count = Max (Add_Count * (1- (Red_Add_Count/100)), Min_Add_Count)
        Return the current model (IFN)
End Do
```

3.2. Calculating the Number of Training Examples

We assume that the OLIN system can be applied to a continuous data stream at its beginning or after any number of incoming examples. If the data stream has just started, the question is, how many examples need to arrive before the algorithm can induce an initial model at a given significance level? If, on the other hand, we have an indefinite number of past examples, what is a minimum number of latest examples that we need to store in the computer memory? Since the IN algorithm uses an information-theoretic heuristic in constructing the information network, the minimum number of examples $E^*(z)$ required to confirm the statistical significance α of an attribute $A_{i'}$ at a node z can be found by combining Equations 2 and 3:

$$E^*(z) \geq \frac{\chi^2_\alpha((NT_{i'}(z)-1)\cdot(NT_i(z)-1))}{2\ln 2MI(A_{i'};A_i/z)} \qquad (4)$$

Equation 4 is less conservative than the distribution-independent Hoeffding bound [6], since it is based on the assumption that the split heuristic (conditional mutual information) is distributed as chi-square with a specific number of degrees of freedom. However, before we apply OLIN to a new data stream, we do not know the conditional mutual information $MI\ (A_{i'};\ A_i\,/\,z)$ for any of the network nodes. In fact, we do not even know the network structure in terms of the total number of nodes and the selected attributes to be associated with each layer. To estimate the minimum number of examples, we first assume, without loss of generality, that the network has one layer only. This is a good approximation, since empirical results [23] show that most nodes of the first layer are usually terminal nodes and only a small portion of examples is associated with subsequent layers. In general, the minimum number of examples in the training window can always be derived from the number of examples required to split every node in the network. We also assume that the first input attribute selected by the algorithm has only two values: $NT_{i'}(z) = 2$. This number can be adjusted to the actual domain size of candidate input attributes in a given dataset.

Since we consider only the attribute that is used to split the root node, the conditional mutual information $MI(A_{i'}; A_i / z = 0)$ is equal to the mutual information $MI(A_{i'}; A_i)$, which can be expressed as [5]):

$$MI(A_{i'}; A_i) = H(A_i) - H(A_i / A_{i'}) \qquad (5)$$

Where $H(A_i)$ is the unconditional entropy of the target attribute A_i and $H(A_i / A_{i'})$ is the conditional entropy of A_i given an input attribute $A_{i'}$. The unconditional entropy $H(A_i)$ can only be estimated by processing some training examples. However, it can be approximated by its maximum value of $\log_2(NT_i)$, where NT_i is the number of classes [5]. If a dataset is very unbalanced, NT_i may include only the most common classes. The upper bound for the conditional entropy can be found from Fano's inequality[5]:

$$H(A_i / A_{i'}) \le H(P_e) + P_e \log_2(NT_i - 1) \qquad (6)$$

Where P_e is the error rate of the information network having conditional entropy of $H(A_i / A_{i'})$. Since the right-hand expression is a non-decreasing function of P_e, we can assume that P_e represents the maximum allowable error rate of the model. The assumption of $NT_{i'}(z) = 2$ and Equations 4 – 6 lead to the following expression for the initial size of the training window:

$$W_{init} = \frac{\chi^2_\alpha(NT_i - 1)}{2\ln 2(\log_2(NT_i) - H(P_e) - P_e \log_2(NT_i - 1))} \qquad (7)$$

Intuitively, the expression for W_{init} can be interpreted as follows. A higher significance level α requires more examples, since it increases the corresponding value of chi-square. More examples are needed to distinguish between a larger number of target classes. Given that other factors remain unchanged, the error rate P_e is also proportional to the number of required examples, since the algorithm needs more examples to confirm the significance of a less accurate model. It is important to note that a high statistical significance of the likelihood-ratio test does not necessarily imply a high predictive accuracy of the resulting model. A significance level just represents the probability that the model is not random, i.e., it is *more* accurate than the default (majority) prediction rule.

The expression for the size of an updated window is also based on Equations 4 – 6, but it uses information, which is available from the latest window of the training examples:

$$W = \frac{\chi^2_\alpha((NT_{i'} - 1)\cdot(NT_i - 1))}{2\ln 2(H(A_i) - H(E_{tr}) - E_{tr}\log_2(NT_i - 1))} \qquad (8)$$

Where $NT_{i'}$ is the number of values (or discretized intervals) for the first attribute $A_{i'}$ in the info-fuzzy network, $H\ (A_i)$ is the entropy of the target, and E_{tr} is the training error of the current model. The resulting number of examples W should be sufficient to confirm the significance of the first layer in the latest trained model. We assume that a new concept, which is still unknown, can be learned from at least the same number of examples. If a new concept pertains for fewer examples than W, the algorithm cannot detect it at all.

3.3. Comparing the Training and the Validation Error Rates

If a concept is stable, the examples in the training window and in the subsequent validation interval should conform to the same distribution. Consequently, there should not be a statistically significant difference between the training and the validation error rates of the IFN model, as we have seen in our previous studies of static datasets [23]. On the other hand, a sharp increase in the error rate may indicate a possible concept drift [32]. Using a Normal approximation to the Binomial distribution, we calculate the variance of the difference between error rates by the following formula (based on [27]):

$$Var\_Diff = \frac{E_{tr}(1-E_{tr})}{W} + \frac{E_{val}(1-E_{val})}{Add\_Count} \qquad (9)$$

If the concept is stable, the maximum difference between the error rates, at the 99% confidence level, is:

$$Max\_Diff = z_{0.99}\sqrt{Var\_Diff} = 2.326\sqrt{Var\_Diff} \qquad (10)$$

If the difference between the error rates exceeds *Max_Diff*, a concept drift is detected and the size of the next training window is re-calculated by using the Equation 8. Also, the number of examples in the next validation interval is reduced by *Red_Add_Count* percent. Otherwise, the concept is considered stable and both the training window and the validation interval are increased up to their maximum sizes.

4. Empirical Results

4.1. Manufacturing Data

We have applied OLIN to a sample of yield data recorded at a semiconductor plant. In semiconductor industry, the yield is defined as the ratio between the number of good parts (microelectronic chips) in a completed batch and the maximum number of parts, which can be obtained from the same batch, if no chips are scraped at one of the fabrication steps. Due to high complexity and variability of modern microelectronics industry, the yield is anything but a stationary process. It is affected daily by hundreds of material-related, equipment-related, and human-related factors. Maintaining current models for yield prediction, as well as timely detection of changes in yield behavior, usually called "yield excursions", constitute the primary tasks of process engineers.

A semiconductor company has provided us with the records of 1,378 manufacturing batches that completed their production during a four months period. Due to the confidentiality of the original data, we omit here the name of the company, the description of the products, and many other details, which are irrelevant to the evaluation of our method. The records include seven candidate input attributes, which represent the main properties of a given batch (chip size, production priority, etc.). The target (dependent) attribute is the percentage of outgoing yield, discretized to three intervals of approximately equal frequency. These intervals can be characterized as a low yield, a normal yield, and a high yield.

In our experiments, we have assumed, for the sake of convenience, that the online learning starts after the completion of the first 378 batches, which leaves us with exactly 1,000 records for validating the performance of the constructed models. Table 2 shows the settings used by OLIN in this dataset. The parameter values have been chosen experimentally to provide the best classification performance of the system. All runs were carried out on a Pentium III processor with 128 MB of RAM.

Table 2 OLIN Parameter Values (Manufacturing Data)

Parameter	Meaning	Value
n_{min}	The number of the first example to be classified by the system	378
n_{max}	The number of the last example to be classified by the system	1377
Sign	A user-specified significance level	0.1%
P_e	Maximum allowable prediction error of the model	0.50
Init_Add_Count	Initial number of examples to be classified by the first model	10
Inc_Add_Count	Amount to increase the number of examples between model re-constructions (if the model is stable)	50%
Max_Add_Count	Maximum number of examples between model re-constructions	100
Red_Add_Count	Amount to reduce the number of examples between model re-constructions (if a concept drift is detected)	75%
Min_Add_Count	Minimum number of examples between model re-constructions	1
Max_Win	Maximum number of examples in a training window	1,000

In Table 3, we compare the overall performance of OLIN to other methods of online learning. The “laziest” approach, implemented in Run 0, is to build a model from all the examples available at the beginning of the run (378) and to subsequently apply it to all the validation examples (1,000) without ever re-training the IN algorithm. The “no re-training” approach is based on the assumption that the data stream is stationary.

Another extreme approach (“no-forgetting”) is to repeatedly re-construct the model from all the past instances after the arrival of every new example. The “no-forgetting” learning requires longer training times as more examples arrive, and, eventually, its training time may exceed the time between arrivals of successive examples or surpass the limits of the computer memory. Runs no. 1 and 2 in Table 3 show the results of this method for initial training windows of 50 and 100 examples respectively.

The next three runs (3–5) show the experiments, which apply *static windowing* by adding and removing the same number examples from the training window at each iteration. Finally, Run no. 6 presents the results of OLIN, which uses dynamic windowing. The columns of the table present the total number of training windows in the process of classifying 1,000 examples, the average number of examples in a training window, the total run time of the system on the stream of 378 training and 1,000 validation examples, the aggregated error rate on the validation records (including its variance), and the probability that the error rate in a given run is significantly different from the error rate of OLIN. One and two asterisks designate 5% and 1% significance levels respectively.

Table 3 Manufacturing Data: Summary of Experiments

Run No.	Initial Window	Add Count	Remove Count	Number of Windows	Av. Window Size	Run Time (sec.)	Error Rate	Variance	p-value
0	378	1000	0	1	1000	0.14	0.558	0.2466	0.027 *
1	50	1	0	1000	550	32.57	0.507	0.2500	0.360
2	100	1	0	1000	600	35.81	0.512	0.2499	0.447
3	50	1	1	1000	50	5.16	0.563	0.2460	0.016 *
4	50	10	10	100	50	0.60	0.598	0.2404	0.000 **
5	100	50	50	20	100	0.28	0.556	0.2469	0.033 *
6	**117**	**Dynamic**	**Dynamic**	**23**	**348.2**	**0.66**	**0.515**	**0.2498**	

The high error rate of the "no re-training" approach (Run 0) indicates that even during a relatively short period of several months, the yield does not preserve a stationary pattern. It is also clear that OLIN (Run 6) is significantly more accurate on this data than the static windowing (Runs 3-5). The no-forgetting approach (Runs 1-2) appears slightly more accurate than OLIN, but the differences are not statistically significant. In terms of system resources, OLIN with its average window size of 348.2 examples requires more memory than the static windowing. On the other hand, the no-forgetting method requires an indefinite amount of memory to store its ever-growing window. In terms of run time, OLIN is comparable with the static windowing and it is considerably faster than the no-forgetting approach even for a limited stream of 1,000 records.

Figure 3 shows how OLIN adjusts the window size to the rate of concept drift. To remove short-term effects, the error rate is averaged over past 100 validation examples. During the first segment of the run, approximately up to Example no. 500, the error rate keeps going up until the system recognizes a concept drift and the window size is drastically decreased. As a result of this window adjustment, the error rate goes down rapidly from 70% to less than 40%. However, between Examples no. 660 and 760 there is again a sharp increase in the average error rate, followed by a very slow decline in the error. The second peak in the error rate is not high enough to be identified as a concept drift, and, hence, the training window continues to grow until the end of the run, where it approaches the maximum size of 1,000 examples.

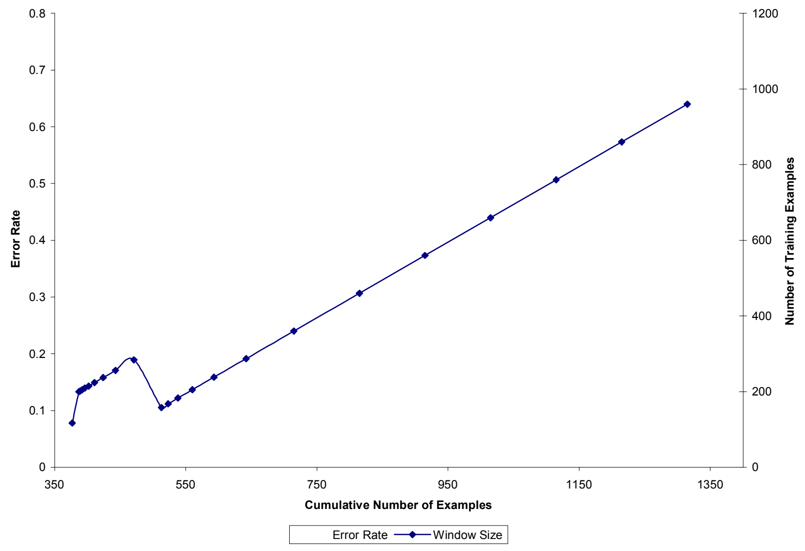

Figure 3 Manufacturing Data: Adjusting to Drift

Figure 4 compares the performance of four methods for online learning: no re-training, no-forgetting (initial size = 100), static windowing (size = 100, increment = 50), and dynamic windowing (OLIN). The error rates of all methods are calculated as moving averages of the past 100 validation examples. The no re-training approach is leading in the beginning of the run, but eventually its error goes up and it becomes inferior to other methods most of the time, probably due to changes in the yield behavior. The no-forgetting method is consistently providing the most accurate predictions for nearly the entire length of the data stream. The static windowing is doing better than OLIN in the first part of the stream, up to approximately Example no. 900. Afterwards, there is a sharp increase in the error rate of the static windowing, while OLIN and the no-forgetting provide the lowest error. In other words, by the end of the run, the large windows of OLIN and the no-forgetting method are more accurate than the small, fixed-size windows of the static method.

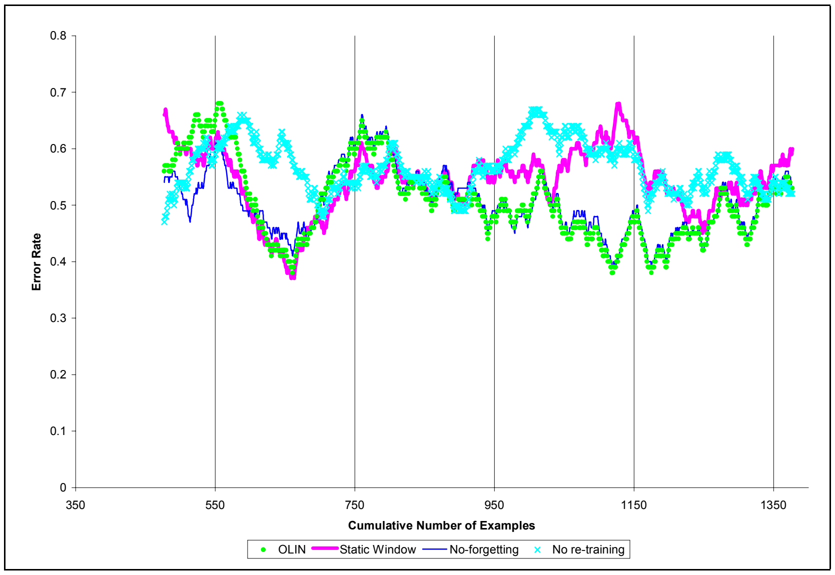

Figure 4 Manufacturing Data: Comparison of On-line Learners

4.2. Stock Market Data

The second set of experiments has been performed on a stock market dataset, initially used by us in [19] for evaluation of a batch learning algorithm (IFN). The raw data represents the daily stock prices of 373 companies from the Standard & Poor's 500 index [33] over a 5-year period (from 8/29/94 to 8/27/99) and it has been obtained from the Microsoft™ MoneyCentral web site [34]. In [19], we have applied signal-processing techniques to partition each series of daily stock values into a sequence of intervals having distinct slopes (trends). An average of 15.64 intervals per company have been identified. The classification problem has been defined as predicting the correct length of the current interval based on the known characteristics of the current and the preceding intervals. Consequently, we have converted every sequence of *m* intervals related to a specific stock into *m-1* interval-pairs each containing information about two consecutive intervals. This resulted in a total of 5,462 records of interval-pairs. The candidate input attributes include the duration, the slope, and the fluctuation measured in each interval, as well as the major sector of the corresponding stock (a static attribute). The target attribute, which is the duration of the second interval in a pair, has been discretized to five sub-intervals of nearly equal frequency. These sub-intervals have been labeled as very short, short, medium, etc.

To restore the original order of data arrival, we have sorted the records by the dates of change points between the adjacent intervals. The online classification task is to predict the timing of the next change point. To make the calculations more convenient, we have started the online learning after the arrival of the first 462 records, which has left us with exactly 5,000 records for validation. The parameters of OLIN were kept as similar as possible to the settings of the experiment with semiconductor data in sub-section 4.1. Due to a larger size of this dataset, the initial number of validation examples was set to 100 and it was allowed to vary between 10 and 400 examples. The number of the last example to be classified is 5,461. The values of other parameters remained unchanged and the runs were performed on the same computer that was used for the first experiment.

Table 4 compares the performance of OLIN to other methods of online learning. Run 0 represents the “no re-training” learner. Runs 1 –3 were performed with the no-forgetting learner for a varying number of new examples accumulated before each model re-construction. We could not reduce this number below 100 due to extremely long training times (more than 24 hours for *Add_Count* = 10). Runs 4 – 6 show the results for a static window size, while Run 7 represents dynamic windowing with OLIN. Like in Table 3, one and two asterisks denote 5% and 1% significance levels respectively vs. dynamic windowing.

Table 4 Stock Data: Summary of Experiments

Run No.	Initial Window	Add Count	Remove Count	Number of Windows	Av. Window Size	Run Time (sec.)	Error Rate	Variance	p-value
0	462	5000	0	1	5000	1.26	0.450	0.2475	0.000 **
1	100	100	0	50	2600.0		0.384	0.2365	0.001 **
2	400	100	0	50	2900.0	18691.09	0.392	0.2383	0.010 **
3	400	200	0	25	2900.0		0.398	0.2396	0.042 *
4	100	100	100	50	100	3.02	0.423	0.2441	0.209
5	400	100	100	50	400	28.73	0.412	0.2423	0.380
6	400	200	200	25	400	15.27	0.424	0.2442	0.181
7	**41**	**Dynamic**	**Dynamic**	**41**	**274.0**	**76.90**	**0.415**	**0.2428**	

The non-stationary nature of the stock data is confirmed by the "no re-training" approach having the highest error rate in Table 4 (see Run 0). The no-forgetting method (Runs 1-3) is significantly more accurate on this data than OLIN, though the difference between the error rates of the two methods is about 2-3% only. Table 4 also demonstrates how fast is an increase in the computation time, if no examples are removed from the training window: after the arrival of 5,000 records, it takes more than five hours to get an overall accuracy, which is just by 2% higher than the accuracy, which could be obtained within a little more than a minute! The static windowing (Runs 4-6), which appears to be the fastest method, is not significantly worse on this dataset than OLIN, though its error rates tend to be slightly higher. One should also note that the static windowing provides the best result with a fixed window of 400 examples, while OLIN's dynamic windows contain an average of 274 examples only. As indicated above, the training window size is directly related to the memory requirements of the learning system.

Figure 5 shows how OLIN adjusts the window size to the rate of the concept drift. It also shows the error rate averaged over the past 400 validation examples. Between the beginning of the run and approximately Example no. 2,700, the concept appears to be stable and there is even some decrease in the error rate vs. the initial part of the run. Due to this apparent stability, OLIN keeps growing the window size. After Example 2700, we observe a sharp increase in the error rate, which causes OLIN to reduce the size of the training window. Subsequently, OLIN performs frequent updates of the window size, trying to keep track of a rapidly changing concept. However, OLIN fails to decrease the error rate for about the next 1,400 examples. The steep decline of the error rate in the last

500 examples, accompanied by an increase in OLIN's window size, reveals the nature of data pre-processing rather than a stock market phenomenon: the last interval of each stock was truncated at the end of the period covered by the dataset (five years). Consequently, the last intervals tend to be much shorter than the preceding intervals, and their classification becomes almost deterministic.

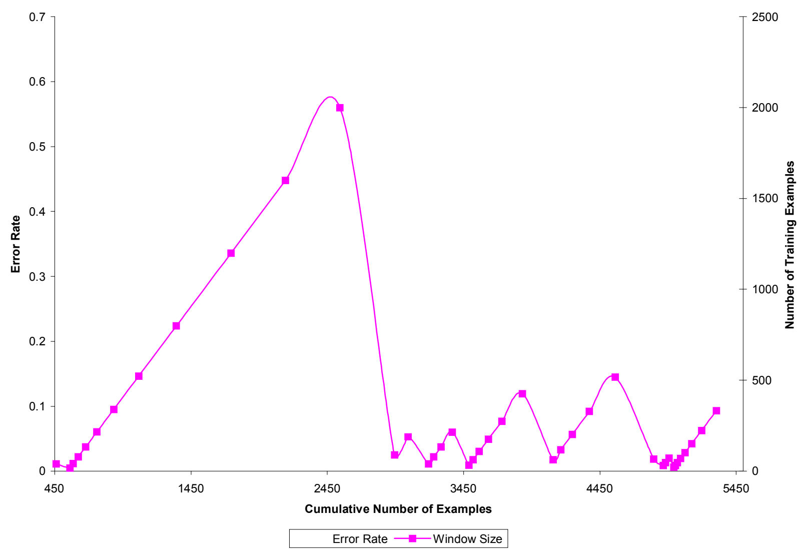

Figure 5 Stock Data: Adjusting to Drift

Figure 6 compares the performance of the "no re-training" approach, no-forgetting (initial size = 100, increment = 100), static windowing (size = 400, increment = 200), and dynamic windowing (OLIN). The error rates of all methods are calculated as moving averages of the past 400 validation examples. During the first 40% of the run, while the concept appears to be stable, no method is consistently better or consistently worse than the other two, which is quite reasonable. In the next segment, which is characterized by a high rate of concept drift, the static windowing approach is less successful than OLIN. However, OLIN itself performs worse in this segment than the no-forgetting learner. This implies that in the second segment, retaining all the past examples improves the classification accuracy, though it requires an increasing amount of computer resources. A similar result has been obtained by Hulten et al. [16] in a comparison between a static and a dynamic system. At the same time, one can see that the gap between OLIN and the no-forgetting method remains nearly constant as more examples arrive. It is also noteworthy that OLIN is better at adapting to a new, though artificial concept in the last segment of the run: its error rate goes down much faster than the error rates of the other methods, especially the no-forgetting learner. The "no re-training" approach is close to static windowing most of the time, but it becomes extremely inaccurate for the last 800 examples of the run, since it does not adjust itself to an abrupt change in the underlying concept.

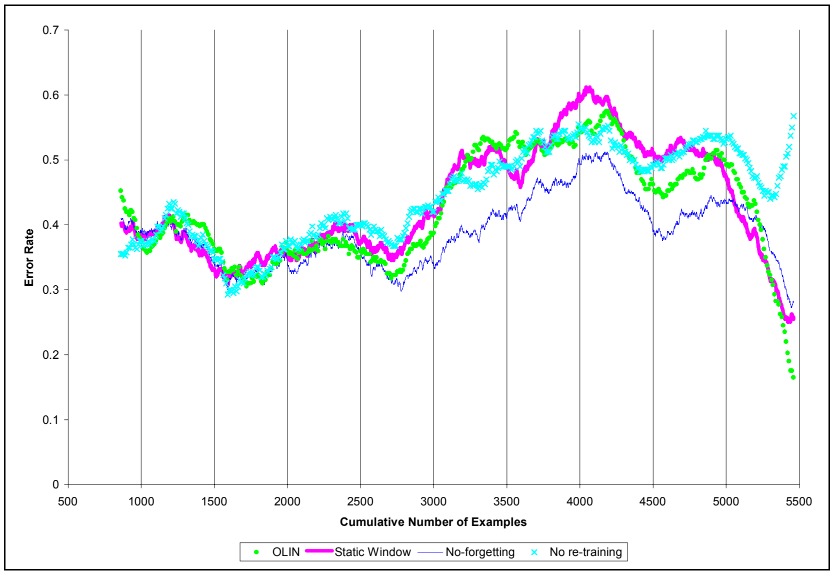

Figure 6 Stock Data: Comparison of On-line Learners

5. Conclusions

This paper has presented OLIN, a new system for on-line classification of continuous data streams. To sum-up, the system is based on the following principles. The induced concept is represented in the form of an info-fuzzy network (IFN), a tree-like classification model. The system is repeatedly constructing a new network from a sliding window of latest examples. The latest model classifies the examples that arrive before the subsequent network re-construction. A concept drift is detected by an unexpected rise in the classification error rate. When a concept is stable, the system increases the size of the training window, up to a pre-specified limit, and reduces the frequency of model updates. With the detection of a concept drift, the system re-calculates the size of the training window by using the principles of information theory and statistics.

The efficiency of OLIN has been confirmed by experiments on two real-world sets of online data. In the case of 1,000 records from a manufacturing dataset, it took OLIN 50 times less computation time to obtain practically the same classification accuracy like the "brute force" no-forgetting method. In the second experiment, performed on 5,000 records of stock data, the ratio between the computation times of "no-forgetting" and OLIN was even more dramatic: about 240. In terms of classification accuracy, the no-forgetting approach has outperformed OLIN by 2% only.

Hulten and Domingos [15] present a number of criteria for designing an online data mining system. These criteria include constant time and memory per incoming record, single scan of data, availability of a usable model at any time, equivalence to an ordinary data mining algorithm, and adaptability to time-changing concepts along with the preservation of concepts from the past that are still relevant. With respect to these criteria, we have seen that OLIN requires a limited amount of time and memory per record, does not revisit old records after they are removed from the training window, uses an anytime algorithm for constructing an info-fuzzy network (see [23]), is almost as accurate as models induced from all the available instances and has a method for adjusting the system parameters in response to a concept drift. The empirical results presented in this paper show that when applied to nonstationary data, OLIN tends to be more accurate than the static windowing methods. However, OLIN is usually less accurate than the extremely inefficient "no-forgetting" approach due to apparent presence of long-term concepts, which are eventually forgotten by OLIN, while being retained by the no-forgetting learner.

The future work includes evaluating OLIN on more real-world datasets, as well as on artificially built data streams. Considering the time dimension as an explicit input attribute available to the system (similar to the approach of [3][13]) is another promising direction. Finding a better trade-off between the efficiency of OLIN and the accuracy of the no-forgetting method should also be examined. Other topics to study include incremental updating of information network structure and application of a similar approach to other classification methods. A visual environment for running beta versions of the batch algorithm (IN) and the online algorithm (OLIN) is available for download at http://www.ise.bgu.ac.il/faculty/mlast/ifn.htm.

Acknowledgments. This work was partially supported by the National Institute for Systems Test and Productivity at the University of South Florida, under SPAWAR

Research Grant N00039-01-1-2248. My thanks to Prof. Eric Backer for his helpful comments.

References

[1] F. Attneave, Applications of Information Theory to Psychology, Holt, Rinehart and Winston, 1959.

[2] S.D. Bay, The UCI KDD Archive [http://kdd.ics.uci.edu], 1999.

[3] M. Black and R. J. Hickey, Maintaining the Performance of a Learned Classifier under Concept Drift, Intelligent Data Analysis, No. 3, pp. 453-474, 1999.

[4] L. Breiman, J.H. Friedman, R.A. Olshen, & P.J. Stone, Classification and Regression Trees, Wadsworth, 1984.

[5] T. M. Cover, Elements of Information Theory, Wiley, 1991.

[6] P. Domingos and G. Hulten, Mining High-Speed Data Streams, Proc. of KDD 2000, pages 71-80, 2000.

[7] W. Fan, S.J. Stolfo, J. Zhang, The Application of AdaBoost for Distributed, Scalable and On-line Learning, Proc. of KDD-99, pages 362-366, 1999.

[8] T. Fawcett and F. Provost, Activity Monitoring: Noticing interesting changes in behavior, Proc. of KDD-99, 1999.

[9] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, From Data Mining to Knowledge Discovery: An Overview, In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 1-30, AAAI/MIT Press, 1996.

[10] D. H. Fisher, Knowledge Acquisition Via Incremental Conceptual Clustering, Machine Learning, No. 2, pp. 139-172, 1987.

[11] M.B. Harries, C. Sammut, Extracting Hidden Context, Machine Learning, No. 32, pp. 101-126, 1998.

[12] D.P. Helmbold and P.M. Long, Tracking Drifting Concepts by Minimizing Disagreements, Machine Learning, No. 14, pp. 27-45, 1994.

[13] R. J. Hickey and M. M. Black, Refined Time Stamps for Concept Drift Detection during Mining for Classification Rules, Proc. of TSDM2000, LNAI 2007, pages 20-30, 2000.

[14] C. -N. Hsu and C.A. Knoblock, Discovering Robust Knowledge from Databases that Change, Data Mining and Knowledge Discovery, No. 2, pp. 1-28, 1998.

[15] G. Hulten and P. Domingos, Catching Up with the Data: Research Issues in Mining Data Streams, Proc. of Workshop on Research Issues in Data Mining and Knowledge Discovery, 2001.

[16] G. Hulten, L, Spencer, and P. Domingos, Mining Time-Changing Data Streams, Proc. of KDD-2001, ACM Press, 2001.

[17] M.G. Kelly, D.J. Hand, and N.M. Adams, The Impact of Changing Populations on Classifier Performance, Proc. of KDD-99, pages 367-371, 1999.

[18] T. Lane and C.E. Brodley, Approaches to Online Learning and Concept Drift for User Identification in Computer Security, Proc. of KDD-98, pages 259-263, 1998.

[19] M. Last, Y. Klein, A. Kandel, Knowledge Discovery in Time Series Databases, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 31, Part B, No. 1, pp. 160-169, 2001.

[20] M. Last, A. Kandel, O. Maimon, Information-Theoretic Algorithm for Feature Selection, Pattern Recognition Letters, Vol. 22, No. 6, pp. 799-811, 2001.

[21] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm, Machine Learning, No. 2, pp. 285-318, 1988.

[22] H. Liu and H. Motoda, Feature Selection for Knowledge Discovery and Data Mining, Kluwer, 1998.

[23] O. Maimon and M. Last, Knowledge Discovery and Data Mining, The Info-Fuzzy Network (IFN) Methodology, Kluwer Academic Publishers, 2000.

[24] O. Maimon, A. Kandel, and M. Last, Information-Theoretic Fuzzy Approach to Data Reliability and Data Mining, Fuzzy Sets and Systems, Vol. 117, No. 2, pp. 183-194, 2001.

[25] U. Manber, A. Patel, and J. Robison, Experience with Personalization on Yahoo!, Communications of the ACM, Vol. 43, No. 8, pp. 35-39, 2000.

[26] W. Mendenhall, J.E. Reinmuth, R.J. Beaver, Statistics for Management and Economics, Duxbury Press, 1993.

[27] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[28] J.R. Quinlan, Induction of Decision Trees, Machine Learning, Vol. 1, No. 1, pp. 81-106, 1986.

[29] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.

[30] C.R. Rao and H. Toutenburg, Linear Models: Least Squares and Alternatives, Springer-Verlag, 1995.

[31] P.E. Utgoff, An Improved Algorithm for Incremental Induction of Decision Trees, Proc. of the Eleventh International Conference on Machine Learning, pages 318-325, 1994.

[32] G. Widmer and M. Kubat, Learning in the Presence of Concept Drift and Hidden Contexts, Machine Learning, Vol. 23, No. 1, pp. 69-101, 1996.

[33] Standard & Poor's Index at http://www.spglobal.com/

[34] The Microsoft™ MoneyCentral home page at http://moneycentral.msn.com/