Applied Soft Computing 8 (2008) 1283–1294

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Info-fuzzy algorithms for mining dynamic data streams☆

Lior Cohen [a,1], Gil Avrahami [a,1], Mark Last [a,*], Abraham Kandel [b]

[a] Ben-Gurion University of the Negev, Department of Information Systems Engineering, Beer-Sheva 84105, Israel
[b] Department of Computer Science and Engineering, University of South-Florida, Tampa, FL 33620, USA

ARTICLE INFO

Article history:
Received 8 October 2007
Accepted 8 November 2007
Available online 26 March 2008

Keywords:
Real-time data mining
Data streams
Incremental learning
Online learning
Concept drift
Info-Fuzzy Networks

ABSTRACT

Most data-mining algorithms assume static behavior of the incoming data. In the real world, the situation is different and most continuously collected data streams are generated by dynamic processes, which may change over time, in some cases even drastically. The change in the underlying concept, also known as concept drift, causes the data-mining model generated from past examples to become less accurate and relevant for classifying the current data. Most online learning algorithms deal with concept drift by generating a new model every time a concept drift is detected. On one hand, this solution ensures accurate and relevant models at all times, thus implying an increase in the classification accuracy. On the other hand, this approach suffers from a major drawback, which is the high computational cost of generating new models. The problem is getting worse when a concept drift is detected more frequently and, hence, a compromise in terms of computational effort and accuracy is needed. This work describes a series of incremental algorithms that are shown empirically to produce more accurate classification models than the batch algorithms in the presence of a concept drift while being computationally cheaper than existing incremental methods. The proposed incremental algorithms are based on an advanced decision-tree learning methodology called "Info-Fuzzy Network" (IFN), which is capable to induce compact and accurate classification models. The algorithms are evaluated on real-world streams of traffic and intrusion-detection data.

1. Introduction

Data mining is known as the core stage of Knowledge Discovery in Databases (KDD), which is defined by Fayyad et al. [12] as: "the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". In recent years, there is an ongoing demand for systems, which are capable to mine massive and continuous streams of real-world data. The use of such systems can be in the fields of temperature monitoring, precision agriculture, urban traffic control, stock market analysis, network security, etc. The complex nature of real-world data has increased the difficulties and the challenges of data mining in terms of data processing, data storage, and model storage requirements [20]. One of the main difficulties in mining dynamic continuous data streams is to cope with the changing data concept. The fundamental processes generating most real-world data streams may change over years, months and even seconds, at times drastically. In case of the classification task, this change, also known as concept drift [15], causes the data-mining model generated from past data to become less accurate in the classification of new records. Therefore, the most important characteristic of such a system is to deal with noise, uncertainty, and asynchrony of the real-world data [8].

Batch classification algorithms like CART [2], ID3 [28], C4.5 [29], and IFN [25] are not suitable for mining continuous data streams. The main problem of these algorithms is their tendency to store and process the entire set of training data. The continuous arrival of training data increases their storage and processing effort, which eventually results in insufficient memory or prohibitively long computation times. In addition, when a certain data-mining algorithm considers all past training examples, the induced patterns may not be valid and relevant to the new data because of changes in the dynamic process, which generates the data. In practical terms, this means an increasing error rate in classifying new records with the existing model.

Algorithms and methods, which extract patterns from continuous and potentially dynamic data streams, are known as *incremental (online) learning*. According to [14], a learning task is

☆ This paper is partially based on the following non-archival publication: L. Cohen, G. Avrahami, M. Last, A. Kandel, and O. Kipersztok, "Incremental Classification of Nonstationary Data Streams", Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams, pp. 117–124, October 7, 2005, Porto, Portugal.

* Corresponding author. Tel.: +972 8 6461397; fax: +972 8 6477527.
E-mail addresses: clior@bgu.ac.il (L. Cohen), gilav@bgu.ac.il (G. Avrahami), mlast@bgu.ac.il (M. Last), kandel@csee.usf.edu (A. Kandel).
[1] Tel.: +972 8 6461397; fax: +972 8 6477527.

1568-4946/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2007.11.003

defined as incremental if the training examples used to solve it become available over time, usually one at a time. The basic approach of pure incremental algorithms is to induce patterns in an incremental manner based on every new incoming instance. This means that instead of building a new model, an incremental learning algorithm updates the current model. This approach saves a significant amount of computer resources such as processing time and memory. In the area of incremental learning with decision-tree classification algorithms, there are several methods such as VFDT [9], CVFDT [17], and OLIN [22], which in general are able to process continuous data streams.

In this paper, we present a series of novel incremental algorithms that produce more accurate classification models than the batch algorithms in the presence of a concept drift and are computationally cheaper than existing incremental methods (OLIN and CVFDT). In our work, we use classification models, which are "oblivious" decision trees generated by the IFN (Info-Fuzzy Network) algorithm introduced by Maimon and Last in [25]. The proposed incremental algorithms are evaluated on real-world streams of traffic and intrusion-detection data. The algorithms are also compared to a leading incremental approach to mining dynamic data streams called CVFDT (Concept adapting Very Fast Decision Tree) of [17] and the results show that our incremental methods outperform the CVFDT performance in terms of run time while maintaining nearly the same level of predictive accuracy.

The rest of this paper is organized as follows. Section 2 presents the related work in the areas of incremental learning and real-time data mining. Section 3 presents a brief overview of IFN and OLIN algorithms, which are the basis for this work. Section 4 presents the proposed incremental algorithms, and Section 5 is the evaluation part. In Section 6, we conclude our paper with a discussion of experimental results and propositions for future work.

2. Related work

2.1. Incremental learning

Pratt and Tschapek [27] claim that the change in outcome distribution (concept drift) may occur in two ways. First, an existing predictive rule may keep its accuracy level, but the rule may be invoked more or less often due to a change in the frequency of occurrence of its feature values. Second, the accuracy of a certain rule may decrease because its underlying features have become irrelevant. In this case, those rules should be discarded and replaced with new rules that depend on newly relevant features. They also propose a visualization technique that uses brushed, parallel histograms to help in identifying a concept drift in multidimensional problem spaces.

In order to overcome the concept drift issue, the optimal situation is to have data-mining systems, which operate continuously, constantly processing data received so that potentially valuable information is never lost. To achieve this goal, many methods were developed, all under the general title of incremental (online) learning, which is aimed at extracting patterns from continuous data streams. The main concept behind the incremental learning methods is that upon receiving a new example, it is cheaper to update an existing model than to build a new one. The pure incremental learning methods take into account every new instance that arrives. Widmer and Kubat [32] have described a series of pure incremental learning algorithms called FLORA that flexibly react to concept drift and can take advantage of situations where a context repeats itself. The main idea behind all their algorithms is to:

1 Keep a window of current examples and a hypothesis.
2 Keep concept descriptions and use them again when they reappear.
3 Monitor the window of examples and the concept descriptions (the system behavior).

The FLORA family includes a number of incremental algorithms; the first one is FLORA2, which maintains a dynamically adjustable window during the learning process (the latest training examples). The heuristic for adjusting the size of the window is known as WAH (window adjustment heuristic). WAH shrinks the window and forgets old instances when a concept drift seems to occur (a drop in predictive accuracy) and keeps the window size fixed as long as the concept seems to be stable. Otherwise, the window keeps growing until the concept seems to be stable. FLORA3 stores concepts in stable situations and reuses them whenever a similar context re-appears. In environments with small number of contexts, the process of relearning speeds up due to storage of past concepts. FLORA4 is designed to be exceptionally robust with respect to noise in the training data as it is very difficult in incremental learning to distinguish between slight irregularities due to noise and a real concept drift.

A recent paper [21] presents a methodology for adaptive modeling and discovery of dynamic relationship rules from continuous data streams using Evolving Fuzzy Neural Networks (EFuNN). After each consecutive chunk of data is entered into the system, extracted rules are compared in order to discover new patterns of interaction between input and output variables. The approach of [21] is demonstrated on two simple case studies.

2.2. Real-time data mining

The main characteristic of a real-time data-mining algorithm is its ability to cope with dynamic and high-speed data streams. As mentioned above, dynamic data streams contain concept drifts resulting from changes in the underlying process. In addition, the arrival rate of the data may be very high.

Domingos and Hulten [9] proposed the VFDT (Very Fast Decision Trees learner) algorithm in order to overcome the long training times issue. The VFDT algorithm is based on a decision-tree learning method combined with sub-sampling of the entire data stream. The size of the sub-sample is calculated using distribution-free bounds called *Hoeffding bounds* under the assumption that the data is generated by a stationary distribution. For this reason, the method can process each example in constant time and memory and is able to incorporate tens of thousands of examples per second using off-the-shelf hardware. In addition, the VFDT algorithm is an anytime algorithm in the sense that at any time (after the first few examples are seen) there is a ready-to-use model of gradually increasing quality. The main drawback of the VFDT algorithm is its inability to cope with concept drifts. The ideas initially proposed in the VFDT algorithm are generalized in [18], where the authors describe the VFBN (Very Fast Bayesian Networks), which also use the concept of Hoeffding bounds in order to calculate the size of the sub-sample.

Hulten et al. [17] have proposed an improved version of the VFDT algorithm called the CVFDT (Concept-adapting Very Fast Decision Trees learner) algorithm. The CVFDT algorithm utilizes VFDT advantages of speed and accuracy and in addition has the ability to discover and react to dynamic changes in the data. It works with a fixed sliding window of examples and builds a model in an incremental manner. Every time a new example arrives, CVFDT updates the sufficient statistics at its nodes by incrementing the counts corresponding to the new example and decrementing the counts corresponding to the oldest example in the window (which now needs to be forgotten). If there is no concept drift, the

updates will have no effect on the current model. However, if a concept drift has been discovered, some splits that previously passed the Hoeffding test will no longer do so, because an alternative attribute now has a higher gain value. In that case, the algorithm begins to grow an alternative sub-tree with the new best attribute at its root. If the alternative sub-tree appears to be more accurate on new data than the old one, the new sub-tree replaces the old one. The advantages of CVFDT are its high accuracy and low time complexity. CVFDT develops a model which is almost as accurate as the one that would be developed by applying VFDT to a sliding window of examples every time a new example arrives, but with O(1) complexity per example, as opposed to (O(w), where w is the size of the window.

Gama et al. [13] have proposed the VFDTc algorithm, which is an extension to the VFDT algorithm in two aspects: the ability to cope with continuous data and the use of more powerful classification techniques at the tree leaves. For the first issue, the information gain of a continuous attribute is calculated by an exhaustive method, which evaluates the quality of all possible discretization points. For the second issue, Naïve Bayes Classifiers are implemented at the tree leaves.

The CVFDT algorithm [17] mines high-speed data streams under the approach of one-pass mining. Aggarwal et al. [1] claim in their paper that the problem with the one-pass mining approach is that it disables the recognition of changes in the mining model during the data arrival process. Although the CVFDT algorithm seems to be an effective method for incremental updating of a classification model induced from a dynamic data stream, the claim is that the accuracy of such an incremental model cannot be greater than a model, which was created, based on the best sliding window model for the current time. The empirical results in [1] show that the true behavior of a data stream is best represented in a temporal model, which is sensitive to the level of evolution of a data stream.

The challenge of mining concept drifting data streams was also studied by Wang et al. [30]. The authors proposed to use weighted classifier ensembles to mine data streams that contain concept drifts. While most incremental algorithms continuously revise a single model, the authors propose to train an ensemble of classifiers from sequential data chunks in the stream. This approach helps in avoiding overfitting and the problem of conflicting concepts by giving each classifier a weight based on its expected predictive accuracy on the current test examples.

Fan [11] presents the StreamMiner, which is a random decision-tree ensemble based engine for mining a data stream. The StreamMiner's choice of the optimal model depends on the amount of data needed and the detection of a concept drift. The results of the experiments show the potential of the StreamMiner in classifying streams with changing concept. An interesting question is how the algorithm deals with high-speed streams. This issue is not discussed by [11].

Domingos and Hulten [10] proposed a general framework for mining massive data streams. Their framework consists of three steps:

1 Derive an upper bound for the time complexity of the chosen data-mining algorithm, depending on the number of training examples required at each step.
2 Derive an upper bound for the accuracy loss between the finite-data and the infinite-data models, as a function of the number of examples required at each step of the finite-data algorithm.
3 Minimize the time complexity by decreasing the number of examples required at each step, subject to pre-specified limitation on the accuracy loss.

This general framework assists in two main issues related to mining of high-speed data streams. The first one is how much data is enough for producing a model, which is almost as accurate as the one produced from infinite-data. The second one is how to keep the existing model updated when the data stream is dynamic and concept drifts are discovered.

To deal with dynamic data streams, Last [22] developed the OLIN (On Line Information Network) classification algorithm that induces an oblivious decision-tree model called Info-Fuzzy Network (IFN). As shown in [24] and [25], IFN is characterized by almost the same accuracy level as CART and C4.5 but has a much more compact structure. Its anytime properties, shown in [23], make IFN particularly appropriate for the task of real-time data mining in high-speed data streams. OLIN gets a continuous stream of data and builds an Info-Fuzzy Network based on a sliding window of latest examples. Unlike the CVFDT algorithm having a fixed window size, OLIN dynamically adjusts the size of the training window and the rate of the model reconstruction to the current rate of the concept drift measured by fluctuations in the classification error rate.

In [3–6], we have extended the ideas suggested by Last [22] in order to enhance the performance of the OLIN system. We have suggested a series of incremental algorithms also dealing with continuous dynamic data streams, but this time instead of creating a new classification model for each new sliding window of examples, the algorithms update or replace a current model according to the new data unless a major concept drift is detected. This paper summarizes the above work by presenting, for the first time, a comprehensive evaluation of these incremental methods in comparison to the regenerative (OLIN) approach and the state-of-the-art algorithm for mining dynamic data streams (CVFDT). The evaluation results presented in this paper also include, for the first time, the experiments with the intrusion detection dataset [16], which is much larger than the traffic data streams used in our previous publications. A full description of the methods and the experiments is provided in the subsequent sections.

3. Information networks: an overview

3.1. The batch learning algorithm (IN)

Many batch and online learning methods use the information theory to induce classification models. One of the batch information-theoretic methods, developed by Last and Maimon [24,25], is the Info-Fuzzy Network algorithm (also known as Information Network-IN). IN is an oblivious decision-tree classification model designed to minimize the total number of predicting attributes. The underlying principle of the IN-based methodology is to construct a multi-layered network in order to maximize the Mutual Information (MI) between the input and the target attributes. Each hidden layer is uniquely associated with a specific predictive attribute (feature) and represents an interaction between that feature and features represented by preceding layers. Unlike such popular decision-tree algorithms as CART [2] and C4.5 [29], the IN algorithm is using a pre-pruning strategy: a node is split if the split leads to a statistically significant decrease in the conditional entropy of the target attribute (equal to an increase in the conditional mutual information[2]). If none of the remaining candidate input attributes provides a statistically significant increase in the mutual information, the network construction

[2] The conditional mutual information [7], or information gain [28], is an information-theoretic measure of association between two random variables X and Y, which is defined as a decrease in the entropy (uncertainty) of Y as a result of knowing X (and vice versa).

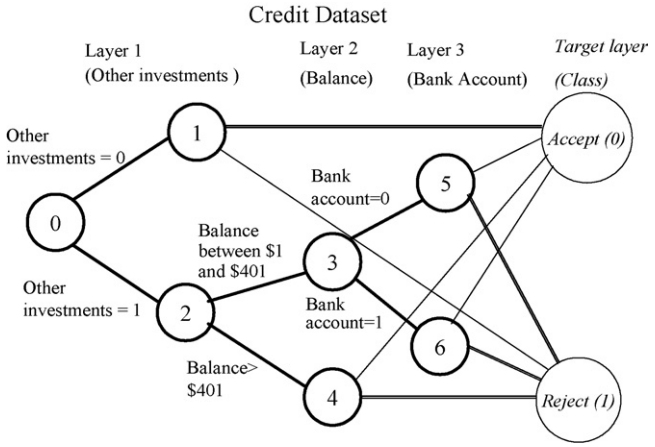

Fig. 1. Sample Info-Fuzzy Network-Credit Approval Dataset.

stops. The output of the IN algorithm is a classification network, which can be used as a decision tree to predict the value (class) of the target attribute. For continuous target attributes, each prediction refers to a discretization interval rather than a specific class.

Fig. 1 illustrates a sample structure of a three-layer information network for the well-known Credit Approval Dataset [26]. The IN algorithm has selected only three predictive features to be included in the network: Other Investments, Balance, and Bank Account.

The network construction algorithm can be summarized as follows (based on [24]):

Input: the set of n training instances; the set CI of m candidate input attributes (discrete and continuous); the target (classification) attribute T; the minimum significance level *sign* for splitting a network node (default: *sign* = 0.1%).

Output: a set I of selected input attributes and an information-theoretic network *IN*. Each input attribute has a corresponding hidden layer in the network.

Step 1 – Initialize the information-theoretic network: a single root node representing all training instances, no hidden layers ($I = \varnothing$, $l = 0$), and a target layer for the values of the target attribute.

Step 2 – While the number of layers $l < m$ (number of candidate input attributes) **do**

Step 2.1 – **for each** candidate input attribute $A_i \notin I$ **do**

if A_i is discrete **then**

Return the statistically significant conditional mutual information $cond\_MI_i$ between A_i and T.

Else return the best threshold splits of A_i and the statistically significant conditional mutual information $cond\_MI_i$ between A_i and the target attribute T.

Step 2.2 – find the candidate input attribute A_{i^*} maximizing $cond\_MI_i$

Step 2.3 – **If** $cond\_MI_{i^*} = 0$, **then**

End do.

Else

Step 2.3.1 – expand the network by a new hidden layer associated with the attribute A_{i^*} and increment the number of layers l.

Step 2.3.2 – Update the set I of selected input attributes $I = I \cup A_{i^*}$

Step 3 – Return the set I of selected input attributes and the network structure

---

The network predictions can be used to compute the reliability degrees of the *actual* values of a target attribute in each training instance. In [25], data reliability is defined as a fuzzy measure representing the degree of certainty that the value of a target attribute A in a specific instance k is correct from the user's point of view. The calculation of reliability degrees in [25] is based on the difference between estimated probabilities of the actual target value (class) and the value (class) predicted by the Info-Fuzzy Network. It is also based on the user-specified coefficient β, which represents the subjective attitude of a particular user to reliability of "unexpected" data. Low values of β (around 1) provide a continuous range of reliability degrees between 0 and 1 for different values of a target attribute. Higher values of β (like 10 or 20) assign a reliability degree of zero to nearly any value, which is different from the predicted one.

Low data reliability cannot be used as an indication of low model accuracy. To clarify this issue, we are providing two extreme examples of binary (two-class) classification problems. If a model is close to a random guess, its accuracy is very low (e.g., 51.1% only). However, according to the reliability function defined in [25], the reliability degree of minority class instances is very close to 1, since they have almost the same probability as the instances of the majority class. On the other hand, in a very accurate model of 99.9% accuracy, the minority class instances are considered extremely unreliable. In general, data reliability, unlike model accuracy, is determined by the *difference* between class probabilities rather than by the probability of any specific class.

3.2. The online learning algorithm (OLIN)

The OLIN algorithm [22] extends the IN algorithm for mining continuous and dynamic data streams. The system repeatedly applies the IN algorithm to a sliding window of training examples and changes the size of the training window (and thus the reconstruction frequency) according to the current rate of concept drift. The purpose of the system is to predict at any given time the correct class for the next arriving example.

The architecture of the OLIN-based system is presented in Fig. 2.

The online learning system contains three main parts: the Learning Module is responsible for applying the IN algorithm to the current sliding window of examples; the Classification Module is

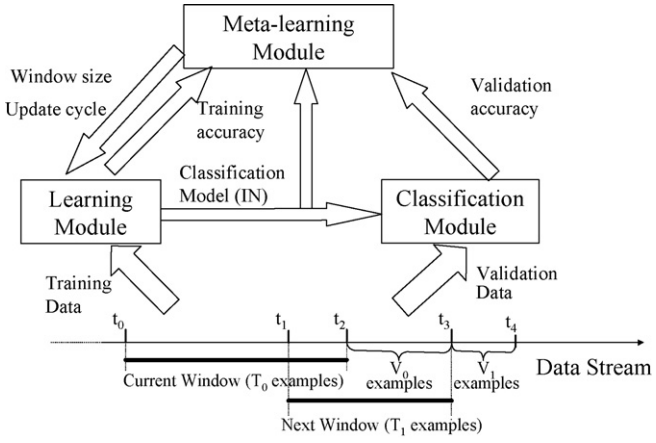

Fig. 2. OLIN-based System Architecture (from [22]).

responsible for classifying the incoming examples using the current network; and the Meta Learning Module controls the operation of the Learning Module. As shown in Fig. 2, the system builds a network from T_0 training examples; afterwards, the next V_0 examples are classified according to the induced network. According to the regenerative approach of OLIN, when more examples arrive, a completely new network is built from the most recent T_1 examples, where the last example belonging to T_1 is also the last example of V_0. In order to ensure that each example in the data stream will be classified only once, the validation intervals have to be disjoint and consecutive. After constructing a new network, the set V_1 is classified using that network. This regenerative process continues indefinitely if the data stream never stops.

The Meta Learning Module gets as input the error rates of the training and the validation examples classified by the current IN model. Those error rates are denoted as E_{tr} and E_{val}, respectively. In addition, it gets the description of the model itself (selected attributes, entropy information, etc.). Using all these inputs, the module re-calculates the size of the next training window (interval) and the number of validation examples to be classified by the next model.

The OLIN system is based on the assumption that if the concept is stable, the training and the validation examples, should conform to the same distribution. Thus, the error rates in classifying those examples using the current model should not be significantly different. On the other hand, a statistically significant difference may indicate a possible concept drift. The variance of the difference between the error rates is calculated by the following formula based on a Normal Approximation to the Binominal distribution (Eq. (9) in [22]):

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

where W is the size of the training window and Add_Count is the number of validation examples.

The algorithm tests the null hypothesis that the concept is stable, in which case the maximum difference between the training and validation error rates, at the 99% confidence level is:

$$\text{Max\_Diff} = Z_{0.99}\sqrt{\text{Var\_Diff}} = 2.326\sqrt{\text{Var\_Diff}} \quad (2)$$

A concept drift is detected by the algorithm when the difference between the error rates is greater than Max_Diff implying that the null hypothesis can be rejected. In that case, the algorithm re-calculates the size of the next training window using the following formula (based on Eq. (8) in [22]):

$$W = \frac{\chi^2_\alpha((NI_i - 1)\cdot(NT-1))}{2\ln 2(H(T) - H(E_{\text{tr}}) - E_{\text{tr}}\log_2(NT-1))} \quad (3)$$

where α is the significance level *sign* used by the network construction algorithm (default: $\alpha = 0.1\%$), NI_i is the number of values (or discretized intervals) for the first input attribute A_i in the Info-Fuzzy Network, NT is the number of target values, $H(T)$ is the entropy of the target, and E_{tr} is the training error of the current model.

In addition, the number of examples in the next validation interval is reduced by Red_Add_Count. Otherwise, the concept is considered stable and both the training window and the validation interval are increased by Add_Count examples up to their maximum sizes of Max_Add_Count and Max_Win, respectively.

4. Incremental information network algorithms

The OLIN online classification algorithm [22] deals with the potential concept drift in a non-stationary data stream by simply generating a new model for every new sliding window. On one hand, this regenerative approach ensures accurate and relevant models over time and therefore an increase in the classification accuracy. On the other hand, OLIN's major shortcoming is the high computational cost of generating new models. In this section, we present four incremental learning algorithms, which are aimed at reducing the high computational cost of the regenerative approach. As shown in the evaluation section of this paper, the incremental methods achieve almost the same and sometimes even higher accuracy rates than OLIN and are significantly cheaper, since there is no need of producing a new model for every new window of examples.

4.1. Basic incremental approach

We start this section with a detailed description of the Basic IOLIN algorithm initially introduced by us in [3]. In general, the basic incremental method updates a current model as long as the concept is stable. If a concept drift has been detected, the algorithm creates a completely new network. The operations of the *Update_Current_Network* procedure include checking split validity, examining the replacement of the last layer, and adding new layers as necessary. Each of the update operations is aimed at reducing the error rate of the existing model, when using it for classifying new instances. The first update procedure *Check_Split_Validity* should eliminate nodes, which are not relevant for the model according to the current training window. The split validity checking starts from the root and goes downwards the network. The general idea is to ensure that the current split of a specific node actually contributes to the mutual information calculated from the current training set. Elimination of non-relevant splits should decrease the error rate. From experiments conducted with the Regenerative OLIN [22], we discovered that when there is no major concept drift between two adjacent windows, the main differences between the newly constructed model and the current one are in the last hidden layer of the information network. For this reason, the second procedure *Check_Replacement_of_Last_Layer* is applied in order to determine which attribute is most appropriate to correspond to the last (final) hidden layer. The attributes under consideration include the attribute that is already associated with the last layer or the second best attribute for the last layer of the previous model. The attribute selected for the last layer of the new model will be the one with the highest conditional mutual information based on the current training window. The last procedure *New_Split_Process* attempts to split the nodes of the last layer on attributes, which are not yet participating in the updated network.

The basic intuition behind the incremental approach is to update the current classification model with the current training window concept as long as no *major* concept drift has been detected by a statistically significant drop in predictive accuracy and to build a new model only in case of a major concept drift. Actually, the regenerative approach can be viewed as a special case of the incremental approach. If a major concept drift is detected at each arrival of new data, the incremental approach is identical to the regenerative one. So the incremental approach becomes useful only when the concept stays stable between at least one pair of adjacent windows. This condition can be expected in most data streams, especially those where the size of the training window is relatively small. The pseudo-code outline of the Basic Incremental OLIN (Basic IOLIN) algorithm and its main procedures is presented below.

The **IN_Control** procedure (see Fig. 3) is responsible for managing the application. It gets as input a continuous stream of examples (S). The initial size of the training window W_{init} is

IN_Control

Input: continuous stream of examples (S)

Output: Updated / New network model

Calculate the initial size of the training window W_{init}

Set $W = W_{init}$

Obtain IN model by applying the IN algorithm to the sliding window (W)

While S is not finished

  IN model = Incremental_IN (W, IN model)

Return *IN model*

Fig. 3. IN_Control procedure.

calculated using the following expression [22]:

$$W_{\text{init}} = \frac{\chi^2_\alpha(NT-1)}{2\ln 2(\log_2(NT) - H(P_e) - P_e\log_2(NT-1))} \quad (4)$$

where α is the significance level *sign* used by the network construction algorithm (default: $\alpha = 0.1\%$), NT is the number of target values (classes), P_e is the maximum allowable prediction error of the model (default = 0.50), H is Shannon's entropy [7], and $\chi^2_\alpha(n)$ is the inverse of the Chi-squared distribution with probability α and n degrees of freedom. The general idea behind the formula (4) is to find the minimum number of training examples required to construct an information network for predicting an NT-valued target attribute, while preserving the prediction error below P_e. As explained in [22], the nominator of (4) refers to the critical value of the Chi-square distribution required to confirm the statistical significance α of splitting the root node on an input attribute while the denominator represents the mutual information between the input and the target attributes. Fig. 4 shows the initial size of the training window W_{init} as a function of the number of classes NT while the maximum allowable prediction error P_e is kept at the fixed level of 20%. As one can see, the number of required examples is decreasing as the target attribute takes more values, since preserving the same error rate for more classes implies a higher decrease in error vs. the default (majority rule) model for uniform priors. Furthermore, a higher decrease in the error rate of the classification model indicates a smaller amount of uncertainty (noisiness) presenting in the data, which means that the mutual information (the denominator of Eq. (4)) is higher. In other words, we need less training examples to induce a model of the same accuracy from less noisy data.

Incremental_IN

Input: training window (W), current information network model (*IN model*)

Output: Updated / New network model

Calculate the training error rate E_{tr} of *IN model*

Calculate the validation error rate E_{val} of *IN model*

Find the maximum expected difference between the last training and the validation errors *Max_Diff*

If (E_{val} - E_{tr}) < *Max_Diff* //concept is stable

  IN_Model = Update_Current_Network (IN_Model, W)

Else

  IN_Model = IN (W)

Calculate *New_Training_Window_Size W*

Fig. 5. Incremental_IN procedure.

Afterwards, the initial network model is produced by applying the IN algorithm to the initial window of examples. The algorithm will run until the end of the data stream. As long as a data stream continues, the algorithm will keep running and producing up-to-date IN classification models.

The **Incremental_IN** procedure (see Fig. 5) is responsible for calculating both error rates of the training and validation examples after running the current model on those data sets. The maximum expected difference between those errors *Max_Diff* is calculated at the 99% confidence level using a Normal Approximation to the Binominal distribution (see Eq. (2) above). It is important to mention our basic assumption that the correct classifications for the window of validation examples are immediately available. This is a reasonable assumption in process monitoring, traffic control, web usage mining, video stream monitoring, and other real-time data-mining domains. The correct classifications are used for calculating the validation error rate and determine whether a major concept drift has occurred.

A concept is considered stable if the actual difference between the validation and the training errors (E_{val}–E_{tr}) is smaller than the

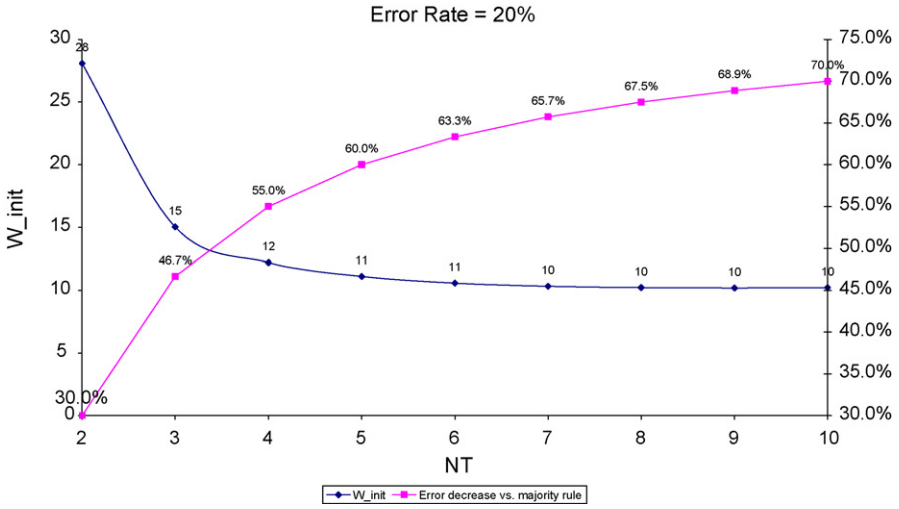

Fig. 4. W_{init} and error decrease as a function of NT.

maximum expected difference *Max_Diff*. This means that the examples in the training window and in the subsequent validation interval conform to the same distribution and there is no statistically significant difference between the model's training and validation error rates. In this case, operations for updating the network are applied. However, if there is a statistically significant increase in the error rate, it indicates that a major concept drift occurs and the basic incremental approach creates a new network. The size of the calculated training window is based on the concept stability (see [22]). In general, the size of the training window increases if the concept appears to be stable and shrinks if a concept drift has been detected.

The **Update_Current_Network** procedure gets as inputs the current network structure and a sliding window. This procedure activates another procedure (*Check_Split_Validity)* for checking the split validity of the current network. Afterwards, it replaces the last layer of the network if needed. Finally, it activates the *New_Split_Process* procedure attempting to split the last layer (whether it was replaced or not) and add a new hidden layer to the network, if necessary (Fig. 6).

Check_Split_Validity is responsible to verify that the current split of each node actually contributes to the conditional mutual information calculated from the current training set (Fig. 7).

New_Split_Process is responsible for splitting the nodes of the last layer on attributes, which are not yet included in the updated network. From the candidate attributes, which are not yet participating in the network structure, this procedure finds the best one for splitting the current last hidden layer according to the MI value. The split by the new input attribute is made on the relevant nodes of the last hidden layer and a new hidden layer is added to the network (Fig. 8).

The Basic IOLIN creates an information network based on a certain number of past examples. When the network construction completes, the algorithm applies it to online data in order to predict the target value of each new record. This means that the algorithm gets two windows as inputs: the first one is a sliding window of past examples and their actual classification, which is used for the construction of the network. The second one is a window of new records, which need to be classified.

As mentioned above, the assumption is that the true classification of the incoming records becomes available after a short while. The true classification can then be used to start the construction of a new network for future classification. As long as the true classification is not available from the monitored system, we can keep the current network. Another alternative is to use unsupervised learning methods in order to detect changes in the distribution of unlabelled data and update the current network accordingly. Zeira et al. [33] present a methodology for change detection and segmentation based on a set of statistical estimators. This methodology can be used for detecting statistically significant changes in incrementally built classification models of data mining.

Update_Current_Network
Input: training window (*W*), current information network model (*IN model*)
Output: Updated network model

Check_Split_Validity (IN_Model, W) on the last layer of the current model
Calculate the conditional MI of *Sec_Best_Attr* based on the current training set (*W*)
IF (conditional MI of the current last layer < conditional MI of *Sec_Best_Attr*)
  Replace last layer with *Sec_Best_Attr*
  New_Split_Process (IN) on the last layer of the current model
Return *IN_Model*

Fig. 6. Update_Current_Network procedure.

Check_Split_Validity
Input: training window *W*, current information network model (*IN model*)
Output: Updated network model

For *i* = 1 to *i* = *total_number_of_layers*-1
  For *j* = 1 to *j* = *number of nodes in hidden layer i*
    If node *j* is split
      Calculate the estimated conditional MI of *j* and the target attribute
      Calculate the Likelihood-ratio* statistic of *j*
      If the Likelihood-ratio statistic of *j* is significant
        Leave the node split
      Else
        Remove the splitting and make *j* a terminal node

Fig. 7. Check_Split_Validity Procedure.
(*) The likelihood-ratio statistic is discussed in [22].

The time complexity of the suggested incremental algorithm depends on the number of detected concept drifts. As long as no concept drift has been detected, the major computational cost is to add a new layer to the existing network. Checking the split validity of the nodes in the network is relatively cheap (O(n) where n is the number of nodes in the network). In case of concept drift, a new network should be constructed from scratch, and the cost of the network construction procedure is linear in the number of records, linear in the number of distinct attribute values, and quadratic in the number of candidate input attributes [24]. When a candidate input attribute is continuous, the number of distinct attribute values is equal to the number of potential splits, which is bounded by the size of the training set. Thus, as shown in [23], the time complexity of discretizing each continuous attribute in a given layer of the network is quadratic in the number of training records. Discretization also requires identification and sorting of distinct values for each continuous attribute before construction of a new network. The computational complexity of these operations is at most quadratic in the number of records.

4.2. Multiple-Model IOLIN

The second incremental approach, introduced in [6], also updates a current model as long as the concept is stable. However, if a concept drift has been detected *for the first time*, the algorithm searches for the best model representing the current data from all previous networks. The idea is to utilize potential periodicity in the data by re-using previously created models rather than generating

New_Split_Process (*IN model*)
Input:current network model
Output: Updated network model

Repeat for every candidate input attribute *i'* which is still not an input attribute
  Repeat for every node *z* of the final hidden layer
Calculate the estimated conditional MI of *i'* and the target attribute given *z*
Calculate the Likelihood-ratio statistic of *i'* and the target attribute given *z*
  If the Likelihood-ratio statistic of *i'* is significant
  Make *i'* the best input attribute and define a new layer for *i'*

Fig. 8. New_Split_Process procedure.

a new model with every concept drift as suggested by the Basic IOLIN approach. The search is made as follows: when a concept drift is being detected, the algorithm calculates the target attribute's entropy and compares it to the target's entropy values of each previous training window, where a completely new network was constructed. The previous network to be chosen is the one, which was built from the training window having the closest target entropy to the current target's entropy. The chosen network is used for classification of the examples arriving in the next validation session. In the case of recurrence of a concept drift, a new network is created like under the Basic IOLIN approach. Following is the pseudo-code outline of the Multiple-Model IOLIN algorithm:

Multiple Model IOLIN

Input: *Training window T, current network model, previous network models*

Output: *Updated/Former/New network model*

For each new training window

If concept drift is detected

Search for the best network from the past by:

1. Comparing the value of the target entropy (based on the current window data) to entropy values in all windows where a completely new network was built.
2. Choose Network with *Min* $|E_{\text{current}}(T)-E_{\text{former}}(T)|$.
3. Add new layers if possible.

If concept drift is detected again with the chosen network

Create completely new network using the Info-Fuzzy algorithm

Else update the existing network (see below)

Else

Update the existing network as follows:

1. Eliminate non-relevant nodes.
2. Replace the last layer with a better one.
3. Add new layers if possible.

4.3. Pure Multiple Model IOLIN

In addition to the use of former models in the presence of concept drift, we also proposed in [6] another variation of the multiple model approach, which includes the use of former models even in the case when the concept is stable. This means that when the concept is stable, we will use a former model instead of updating the current model. This approach should save additional run time because an appropriate model is likely to be found in the adjacent former windows. This approach is called the *Pure Multiple Model IOLIN* and its pseudo-code is given below:

Pure Multiple Model IOLIN

Input: *Training window T, current network model, past network models*

Output: *Former/New network model*

For each new training window

Search for the best network from the past by:

1. Comparing the value of the target entropy (based on the current window data) to entropy values in all windows where a totally new network was built.
2. Choose Network with *Min* $|E_{\text{current}}(T)-E_{\text{former}}(T)|$.
3. Add new layers if possible.

If a concept drift is detected again with the chosen network

Create totally new network using the IN algorithm

4.4. Advanced IOLIN

In the fourth proposed incremental algorithm, initially introduced by us in [5] [6], we enhance the update operations of the incremental algorithms by maintaining the information-theoretic quality of the current model disregarding its predictive accuracy on the new data. We re-calculate the conditional mutual information of each layer using the top-down approach: we start with the first layer, and replace the input attributes in that layer and all subsequent layers in case of a significant decrease in the layer's conditional mutual information with the target attribute. A decrease in the mutual information of the first layer will trigger a complete re-construction of the model. The current model will be retained only if there is no significant decrease in mutual information in any layer. In that case, we will try to add new layers representing attributes, which are not yet participating in the network.

This approach does not deal directly with the issue of concept drift detection. The information network is constantly updated and the occurrence of a concept drift can be concluded if all network layers have been replaced. For each window, the conditional mutual information (MI) value of every layer is saved. In the model update process, a comparison is made between the former MI value of the *i*th layer and the current MI value. If the current value is nearly as high as the former one (up to 5% difference), the *i*th layer is kept as is. If the current MI value is significantly lower, the *i*th layer is replaced with a new one. Following is the pseudo-code outline of the Advanced IOLIN algorithm:

Advanced IOLIN

Input: *Training window, current network model, conditional mutual information of each network layer I (Former_Conditional_MI(i))*

Output: *Updated network model*

For each new training window

For each Layer *i* in existing network

Calculate *Cond_MI(i)*

If *Cond_MI(i)* $\geq$ *Former_Conditional_MI(i)**95%

Keep *i*th layer as is and move to the next layer

Else

Continue the network construction by adding new layers

Save value: *Former_Conditional_MI(i)* = *Conditional_MI(i)*

If reached the last layer

Try adding new layers to the network

The time complexity of the advanced approach depends on the number of layers that have been replaced. In the case of a layer replacement, the algorithm searches for the best input attribute from the candidate inputs. This search costs $O(n)$ where n is the number of candidate input attributes. In the replacement of all layers, which is actually the construction of a totally new network, the cost will be $O(n^*l)$ where l is the number of the network layers.

Table 1
Traffic attributes

Attribute	Source	Values	Attribute type
Date	Sensor		None
Hour	Sensor	0–23, as the beginning hour of the interval	Input
Year	Derived	1999–2002	Input
Month of year	Derived	1–12	Input
Week of year	Derived	1–52	Input
Day of week	Derived	1–7	Input
Day of year	Derived	1–365	Input
Day of month	Derived	1–31	Input
Day type	Derived	0 = regular day, 1 = a holiday eve, 2 = holiday	Input
Previous hour volume	Sensor		Input
Previous Day hourly volume	Sensor		Input
Previous week hourly volume	Sensor		Input
Current hourly volume	Sensor		Target

5. Evaluation

In this section, the proposed incremental algorithms are evaluated vs. the original Regenerative OLIN algorithm [22] on several real-world streams of dynamic data. In addition, the IN-based methods are compared to the CVFDT incremental decision-tree learner [17] available as part of the VFML toolkit [19]. We have examined two aspects of all incremental algorithms: first, we have evaluated their predictive accuracy on incoming examples and secondly we have compared their processing time per the same number of arriving records. All IN-based algorithms were implemented with the single-pass approach used by CVFDT [17], which implies that the algorithm will process each incoming record only once—either for training, or for testing. The real-world data streams were obtained from two different domains: urban traffic control and network intrusion detection.

5.1. Traffic data streams

5.1.1. Data acquisition and preparation

The data acquisition and preprocessing of this dataset are extensively discussed in [3,4], so here we describe this data only in brief. The data streams we used include traffic flow information from under-road sensors at a signaled three-way junction of Tahon and Uruguay streets in Jerusalem, Israel. The vehicles can cross the intersection in five different directions. The resulting five data streams related to these directions have included the incoming traffic volumes for 24 h a day, seven days a week during the period of more than 3 years (1999–2002). Traffic count records have been saved to the Traffic Sensors Database every 15 min, for every lane in the intersection.

In the data cleaning stage, duplicate records have been removed and missing volume quantities have been filled in with the average of their preceding and successive traffic volumes. Cases with missing successive traffic volumes were completely ignored. In order to represent the traffic volumes per hour and per direction, the traffic volume records of all lanes in the same direction were combined and then the 15-min records were summed up to 1-h records for each direction. Eventually, each original data stream has been converted into a data set, where a record contains twelve candidate attributes representing the exact time (date, hour, day in week, etc.) when the traffic volume was measured and traffic volumes at earlier points of time (the previous hour, the same hour of the previous day, etc.). The resulting relational dataset is described in Table 1.

The target attribute *Current Hourly Volume* represents the volume of traffic during a given hour. Since the target attribute is continuous, we have manually discretized it to four equal-frequency intervals of traffic volume. The traffic data was partitioned into five separate data tables for the five traffic directions. Each table corresponding to a given direction contained about 30,000 hourly records.

5.1.2. Results

The runs of the algorithms were carried out on a Pentium IV processor with 256 MB of RAM. In the experiments, the online learning of the traffic data starts after inducing the initial model from the first 500 records, which leaves the system to work with about 30,000 records for each direction in every year. Table 2 presents the results of applying the IN-based single-pass algorithms and the CVFDT algorithm to the traffic data sets. The testing error rates and the processing times are shown graphically in Figs. 9 and 10, respectively

Table 2 and Fig. 9 show that, in most cases, the IOLIN-based algorithms outperform the accuracy rate of the CVFDT algorithm,

Table 2
Traffic data results

Algorithm	Data source	Run time (s)	Testing error rate (%)
CVFDT	Direction1	50.76	28.63
	Direction2	49.79	23
	Direction3	55.76	23.7
	Direction4	50.48	23.23
	Direction5	49.55	22.43
Regenerative IFN	Direction1	22.11	18.7
	Direction2	11.63	14.1
	Direction3	16.07	6.5
	Direction4	11.89	4.8
	Direction5	14.2	23.5
Basic Incremental IFN	Direction1	8.78	26.4
	Direction2	4.54	13.2
	Direction3	3.62	8.7
	Direction4	3.83	6.2
	Direction5	4.67	25.3
Multi-Model IFN	Direction1	9.59	26.1
	Direction2	6.77	13.2
	Direction3	4.32	8.7
	Direction4	4.46	6.2
	Direction5	6.56	25.3
Pure Multi-Model IFN	Direction1	19.99	27.4
	Direction2	6.62	12.9
	Direction3	4.73	8.6
	Direction4	4.09	5.4
	Direction5	4.63	25.6
Advanced Incremental IFN	Direction1	8.86	19.5
	Direction2	10.47	12.9
	Direction3	16.09	6.4
	Direction4	9.05	4.5
	Direction5	12.94	22.2

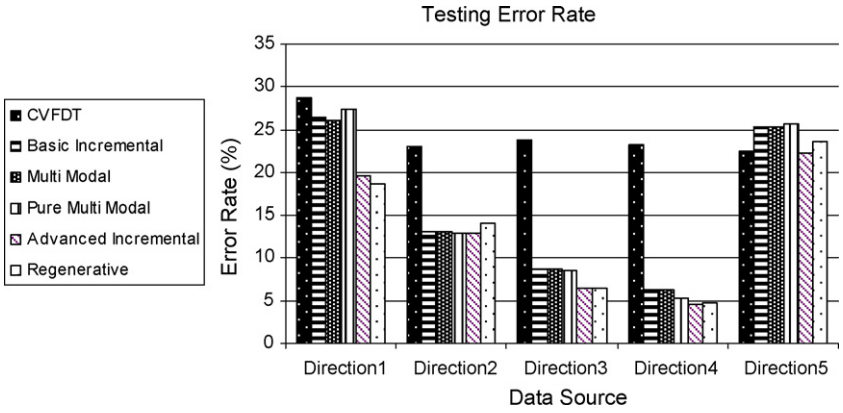

Fig. 9. Testing error rates (traffic data).

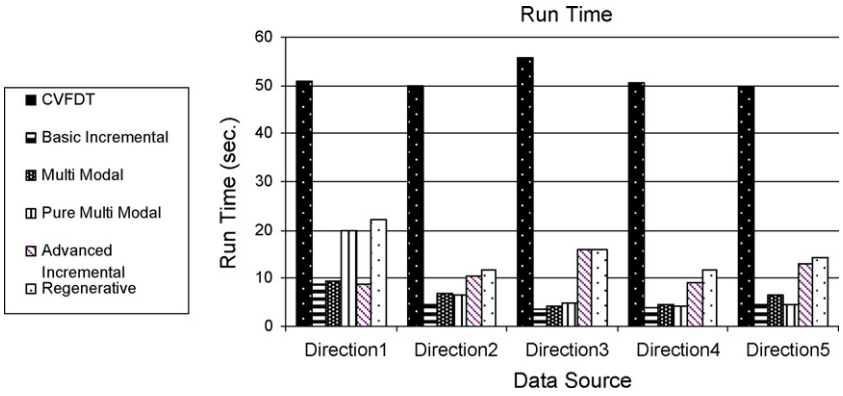

Fig. 10. Run time (traffic data).

with Advanced IOLIN being the most accurate algorithm in all five data streams. The differences in the error rates across traffic directions imply that Directions 2–4 are less noisy than Directions 1 and 5. In practical terms, this means that the temporal behavior of car drivers crossing the intersection in Directions 1 and 5 is less predictable than of those crossing in all other directions.

Fig. 10 presents the running time results. CVFDT is clearly inferior to all IN-based approaches, while the Basic IOLIN and the Multi-Model algorithms consume the least run time for processing the incoming examples. One can also notice that the Advanced IOLIN requires considerably longer run times than the other incremental approaches. The reason for that is the update procedure, which checks if changes should be made in each layer. On one hand, this process makes the model more accurate with respect to the testing window but on the other hand, it consumes longer processing times.

5.2. The Intrusion Detection (ID) dataset

5.2.1. Data acquisition

The data set used for the experiments, was originally developed by DARPA and then used for the Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 (The Fifth International Conference on Knowledge Discovery and Data Mining). The competition task was to build a network intrusion detector, which is actually a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. The database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment. The data is available from the UCI KDD Archive [16].

The raw TCP data was accumulated for 9 weeks from a local-area network (LAN) simulating a typical U.S. Air Force LAN. The simulated LAN was operated as if it was a true Air Force environment suffering from multiple attacks. The raw training data was about 4 GB of compressed binary TCP dump data from network traffic. It was processed into about five million connection records. A connection is defined as a sequence of TCP packets running from source IP to target IP, starting and ending at defined times. Each connection is labeled as either normal or an attack with exactly one specific attack type. The main attack categories are:

- DOS: denial-of-service
- R2L: unauthorized access from a remote machine
- U2R: unauthorized access to local super user (root) privileges
- Probing: surveillance and other probing (for example, port scanning)

The classification task in our experiments was also to determine the type of the connection according to given past and present values of the data attributes.

5.2.2. Preprocessing the intrusion-detection data

The ID data set contains 41 candidate attributes and one class attribute. These 41 attributes can be partitioned into three distinct categories: basic features of individual TCP connections, content features within a connection representing domain knowledge, and traffic features computed using a 2-s time window.

For the purpose of the experiments, we used only a portion of the data (about 500,000 records). For each discrete attribute, rare

Table 3
Intrusion-detection data results

Algorithm	Run time (s)	Testing error rate (%)
CVFDT	1001.15	2.35
Regenerative IFN	1079.67	3.9
Basic Incremental IFN	322.48	3
Multi-Model IFN	222.47	3.5
Pure Multi-Model IFN	169.07	4
Advanced Incremental IFN	228.88	3.3

and uncommon values were eliminated. The basic rule for value elimination was as follows: if a value of an attribute appears in less than 1% of the entire data set (less than 5,000 records), the value should be removed. The domain of the target attribute was also reduced from 22 values to seven using the same rule. Initial experiments have supported the elimination of non-relevant values by producing almost the same mining models in less run time and with almost the same accuracy rates. For continuous attributes, missing values were completed using a simple moving average with a lag equal to 5.

Using the Weka software [31], a feature selection procedure was applied, based on the information gain measure of each candidate input attribute X with respect to the target attribute:

$$IG(X) = H(D) - \sum_{j=1}^{p} \frac{|D_j|}{|D|} H(D_j) \quad (5)$$

where D is the data set before splitting on the values of X, p is the number of values of X, D_j is the data subset satisfying $X = j$, $H(D)$ is the entropy of the target attribute given the entire dataset D, and $H(D_j)$ is the target attribute entropy given the subset D_j.

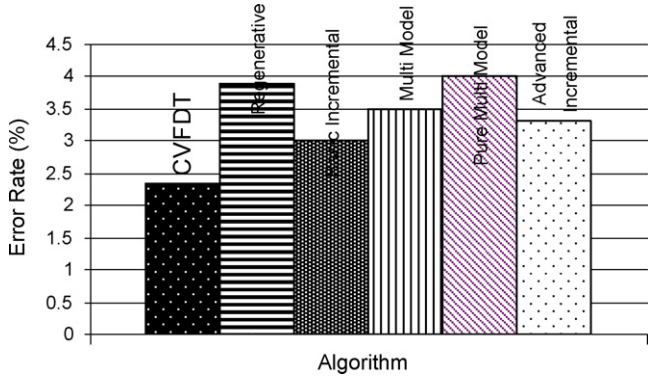

Fig. 11. Testing error rates (ID data).

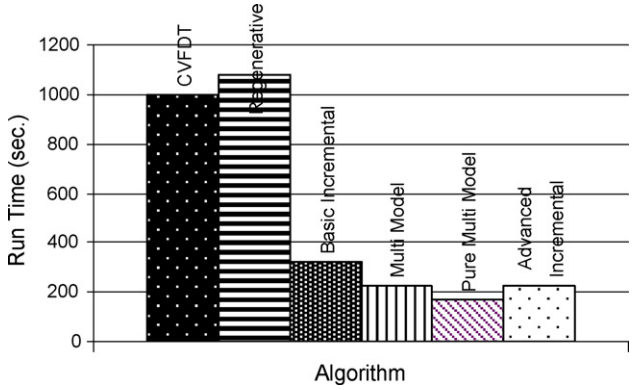

Fig. 12. Run time (ID data).

The obtained information gain values were between 1.52 and 0. Attributes with an information gain of less than 0.5 were eliminated. The remaining attributes were: Duration, Protocol_Type, Service, Src_Bytes, Dst_Bytes, Count, Srv_Count, Dst_Host_Count, Dst_Host_Srv_Count, Dst_Host_Same_Srv_Rate, Dst_Host_Diff_Srv_Rate, Dst_Host_Same_Src_Port_Rate, Dst_Host_Serror_Rate, Dst_Host_Srv_Serror_Rate, Dst_Host_Rerror_Rate, Dst_Host_Srv_Rerror_Rate.

5.2.3. Results

The runs of the algorithms were again carried out on a Pentium IV processor with 256 MB of RAM. In the experiments, the online learning on the ID data starts after inducing the initial model from the first 500 records, which leaves the system to work with about 499,500 records. Table 3 presents the results of applying the IN-based single-pass algorithms and the CVFDT algorithm to the ID data set.

The testing error rates are shown graphically in Fig. 11. Though in this data stream, CVFDT has provided the lowest error rates, the error rates of the most accurate IN-based online algorithms (Basic and Advanced IOLIN) were higher by less than 1%.

Fig. 12 presents the run time results. The incremental algorithms clearly outperform the performance of the CVFDT algorithm and the regenerative approach (OLIN) in terms of the run time.

6. Conclusions and future work

This paper has performed a comprehensive evaluation of a series of novel real-time data-mining algorithms, aimed at optimizing the classification performance under arrival of dynamic data. Unlike existing techniques for mining continuous data streams, the real-time algorithms adapt themselves automatically to the rate of data change ("concept drift"). The Learning Module of the proposed real-time data-mining methods is based on an advanced decision-tree induction algorithm called Info-Fuzzy Network (IFN).

The first algorithm is the basic incremental IN. The intuition behind the basic incremental approach is to update the current classification model with the current training window concept as long as no major concept drift has been detected by a statistically significant drop in predictive accuracy and to build a new model in case of a major concept drift. The second, multi-model approach also updates a current model as long as the concept is stable. In addition, if a concept drift has been detected, the algorithm searches for the best model for the current data from all the past networks. The third algorithm uses the pure multi-model approach—it searches for the best model from the past even if the concept is stable. The fourth approach is the advanced incremental IN. In this approach, we have enhanced the update operations of the incremental IN algorithm by maintaining the information-theoretic quality of the current model disregarding its predictive accuracy on the new data.

The proposed real-time algorithms were implemented and then compared to the regenerative approach, which applies the batch IN algorithm to every new training window and the CVFDT algorithm that constructs Hoeffding decision trees from dynamic data streams. From the results it can be clearly stated that the IN-based incremental algorithms achieve reasonably high predictive accuracy in the classification tasks while significantly decreasing the processing time of the training data when scanning the data only once (using the single-pass approach). Among the incremental approaches, the Advanced IOLIN was found to be the most accurate one but also the most expensive in terms of the processing time. The reason for that is the update process in each layer of the network.

The two features of high accuracy and short processing time are very important when the task is to classify high-speed data streams in real time. The information-theoretic incremental algorithms seem to be quite accurate and at the same time cheaper in terms of the processing time than the CVFDT algorithm based on Hoeffding bounds.

The future research can examine multi-thread implementations of incremental algorithms aimed at further reduction of the processing times per record. Another important issue for future research can be the calculation of the training window size. Currently, the size of the training window depends on the accuracy rates achieved on the training and the testing samples. The problem is how to calculate the size of the window if the accuracy rate of the testing sample is not available. In addition, the incremental algorithms can be evaluated using more real-world data streams from the fields of meteorology, agriculture, image processing, etc.

Acknowledgements

We would like to thank the Traffic Control Center of Jerusalem for granting us the permission to use their traffic database. This work was partially supported under a research contract from the Israel Ministry of Defense and by the National Institute for Systems Test and Productivity at University of South Florida under the USA Space and Naval Warfare Systems Command Grant No. N00039-01-1-2248.

References

[1] C. Aggarwal, J. Han, J. Wang, P.S. Yu, On demand classification of data streams, in: Proceedings KDD'04, ACM Press, New York, NY, (2004), pp. 503–508.
[2] L. Breiman, J.H. Friedman, R.A. Olshen, P.J. Stone, Classification and Regression Trees, CRC Press, Boca Raton, FL, 1984.
[3] L. Cohen, M. Last, G. Avrahami, Incremental info-fuzzy algorithm for real time data mining of non-stationary data streams, in: Proceedings of the TDM 2004-ICDM 2004 Workshop on Temporal Data Mining: Algorithms, Theory and Applications, Brighton UK, 2004.
[4] L. Cohen, G. Avrahami, M. Last, A. Kandel, O. Kipersztok, Real-Time Data Mining of Non-Stationary Data Streams from Sensor Networks, Inf. Fusion J. 9 (2008) 344–353.
[5] L. Cohen, G. Avrahami, M. Last, A. Kandel, O. Kipersztok, incremental knowledge discovery in traffic sensors data, in: Proceedings of the SENSORFUSION 2005, Workshop on Information Fusion and Dissemination in Wireless Sensor Networks, Budapest, Hungary, 2005.
[6] L. Cohen, G. Avrahami, M. Last, A. Kandel, O. Kipersztok, Incremental classification of nonstationary data streams, in: Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams, Porto, Portugal, (2005), pp. 117–124.
[7] T.M. Cover, J.A. Thomas, Elements of Information Theory, second ed., Wiley-Interscience, New York, NY, 2006.
[8] D.E. Culler, W. Hong, Wireless sensor networks: introduction, Commun. ACM 47 (6) (2004) 30–33.
[9] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings KDD 2000, ACM Press, New York, NY, USA, (2000), pp. 71–80.
[10] P. Domingos, G. Hulten, A general framework for mining massive data streams, J. Comput. Graphical Stat. 12 (4) (2003) 945–949.
[11] W. Fan, StreamMiner: a classifier ensemble-based engine to mine concept drifting data streams, in: Proceedings VLDB'2004, Toronto, (2004), pp. 1257–1260.
[12] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery: an overview, in: U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, Menlo Park, CA, 1996, pp. 1–36.
[13] J. Gama, R. Rocha, P. Medas, Accurate Decision Trees for mining high-speed Data Streams, in: Proceedings of the 9th ACM SIGKDD International Conference in Knowledge Discovery and Data Mining, ACM Press, New York, NY, (2003), pp. 523–528.
[14] C. Giraud-Carrier, A note on the utility of incremental learning, AI Commun. 13 (4) (2000) 215–223.
[15] D.P. Helmbold, P.M. Long, Tracking drifting concepts by minimizing disagreements, Machine Learn. 14 (1994) 27–45.
[16] S. Hettich, S.D. Bay, The UCI KDD Archive [http://kdd.ics.uci.edu] (Irvine, CA, University of California, Department of Information and Computer Science, 1999).
[17] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings KDD 2001, ACM Press, New York, NY, (2001), pp. 97–106.
[18] G. Hulten, P. Domingos, Mining complex models from arbitrarily large databases in constant time, in: Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining, ACM Press, New York, NY, (2002), pp. 525–531.
[19] G. Hulten, P. Domingos, VFML—a toolkit for mining high-speed time-changing data streams, [http://www.cs.washington.edu/dm/vfml/] (University of Washington, 2003).
[20] H. Kargupta, Distributed Data Mining for Sensor Networks Tutorial, ECML/PKDD, Pisa, 2004.
[21] N. Kasabov, Adaptation, Interaction in dynamical systems: modeling and rule discovery through evolving connectionist systems, Appl. Soft Comput. 6 (3) (2006) 307–322.
[22] M. Last, Online classification of nonstationary data streams, Intell. Data Anal. 6 (2) (2002) 129–147.
[23] M. Last, A. Kandel, O. Maimon, E. Eberbach, Anytime algorithm for feature selection, in: Proceedings of the RSCTC 2000, Lecture Notes in Computer Science, vol. 2005, Springer-Verlag, Berlin, (2001), pp. 532–539.
[24] M. Last, O. Maimon, A compact and accurate model for classification, IEEE Trans. Knowledge Data Eng. 16 (2) (2004) 203–215.
[25] O. Maimon, M. Last, Knowledge Discovery and Data Mining—the Info-Fuzzy Network (IFN) Methodology, Kluwer Academic Publishers, Boston, 2000.
[26] D.J. Newman, S. Hettich, C.L. Blake, C.J. Merz, UCI Repository of machine learning databases [http://www.ics.uci.edu/mlearn/MLRepository.html] (Irvine, CA: University of California, Department of Information and Computer Science, 1998).
[27] K.B. Pratt, G. Tschapek, Visualizing concept Drift, in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, New York, NY, (2003), pp. 24–27.
[28] J.R. Quinlan, Induction of Decision Trees, Machine Learn. 1 (1) (1986) 81–106.
[29] J.R. Quinlan, C4. 5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.
[30] H. Wang, W. Fan, P. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), ACM Press, New York, NY, (2003), pp. 226–235.
[31] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, second ed., Morgan Kaufmann, San Francisco, CA, 2005.
[32] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, Machine Learn. 23 (1) (1996) 69–101.
[33] G. Zeira, O. Maimon, M. Last, L. Rokach, Change detection in classification models induced from time series data, in: M. Last, A. Kandel, H. Bunke (Eds.), Data Mining in Time Series Databases, Series in Machine Perception and Artificial Intelligence, vol. 57, World Scientific, Singapore, 2004, pp. 101–125.