Knowl Inf Syst (2011) 29:335–347
DOI 10.1007/s10115-010-0345-5

REGULAR PAPER

A hybrid decision tree training method using data streams

Michal Wozniak

Received: 13 April 2010 / Revised: 12 August 2010 / Accepted: 11 September 2010 /
Published online: 5 October 2010

Abstract Classical classification methods usually assume that pattern recognition models do not depend on the timing of the data. However, this assumption is not valid in cases where new data frequently become available. Such situations are common in practice, for example, spam filtering or fraud detection, where dependencies between feature values and class numbers are continually changing. Unfortunately, most classical machine learning methods (such as decision trees) do not take into consideration the possibility of the model changing, as a result of so-called *concept drift,* and they cannot adapt to a new classification model. This paper focuses on the problem of concept drift, which is a very important issue, especially in data mining methods that use complex structures (such as decision trees) for making decisions. We propose an algorithm that is able to co-train decision trees using a modified NGE (*Nested Generalized Exemplar*) algorithm. The potential for adaptation of the proposed algorithm and the quality thereof are evaluated through computer experiments, carried out on benchmark datasets from the UCI Machine Learning Repository.

Keywords Nested generalized exemplar · Nearest hyperrectangle · Concept drift · Decision tree · Incremental learning · Pattern recognition

1 Introduction

Advances in computer science technologies have resulted in many institutions collecting huge amounts of data, the manual analysis of which is virtually impossible. Nowadays, for the efficient management of an average enterprise, simple data analysis methods do not suffice because to make smart decisions, hidden knowledge must be extracted from the data [27]. The biggest disadvantage of most of these methods is that they "assume" that the statistical properties of the discovered concept (that is, the target predicted by the model) remain

M. Wozniak (✉)
Department of Systems and Computer Networks, Faculty of Electronics,
Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
e-mail: Michal.Wozniak@pwr.wroc.pl

unchanged. In real situations, so-called *concept drift* occurs frequently [15,37]. The potential for considering new training data [46] is an important feature of machine learning methods used in security applications (like spam filters or IDS/IPS) [30] or decision support systems for marketing departments, which need to follow the changing client behavior [9].

It is obvious that the smaller the data structures used by such systems in making decisions, the more likely the systems can adapt. As an example, minimal distance methods, known as *lazy classifiers* [2], can take into consideration each new training element because no structure is used in making the decision. On one hand, such methods are very adaptable, but on the other hand, the cost of their decision making is high. Other kinds of machine learning methods invest in building data structures that allow them to make inexpensive and rapid decisions. Unfortunately, these methods are not at all adaptable; for example, although a decision tree is able to make decisions quickly, the training thereof is costly in terms of time. Therefore, the design of data mining methods, especially classification ones for data streams, is currently the focus of intense research [1,3,26,38].

This paper deals with the problem of incremental learning of a given decision tree. As is well known, the structure of the tree is not predisposed to change as new data become available. The following alternative approaches can be considered:

1. Rebuilding the tree if new data become available, which is very expensive and impossible from a practical point of view, especially if the *concept drift* occurs rapidly;
2. Detecting concept changes in the new data [7] and if these changes are sufficiently "significant", then rebuild the tree;
3. Adopting an incremental learning algorithm for the decision tree [6];
4. Generating the tree using a well-known method and then enabling incremental learning of the given concept by another method—the so-called hybrid approach.

This article focuses on the final alternative. We show how to use the NGE (*Nested Generalized Exemplar*) algorithm to improve the results of a given decision tree by considering new training data.

2 Machine learning algorithms

2.1 Decision tree

One of the most useful and popular trends in data mining is classification, also known as pattern recognition [8]. The aim of a pattern recognition task is to classify the object into one of the predefined categories according to its feature values. These methods are usually applied to many practical areas, such as credit approval, prediction of customer behavior, fraud detection, designing IPS/IDS, and medical diagnoses, to name but a few. Numerous approaches have been proposed for constructing efficient classifiers, such as neural networks, statistical learning, and symbolic learning [4].

Of the different concepts and methods for machine learning, induction of a decision tree is both attractive and efficient [43]. Usually, decision trees are divided into two categories: (1) classification trees that classify objects into one of the predefined categories (e.g., letters during text recognition) and (2) regression trees that predict an actual value (e.g., the profitable price of new consumer goods). One of the most popular algorithms for regression tree training is MARS (*Multivariable Adaptive Regression Splines*), introduced by Friedman [14] for training a decision tree by fitting piecewise linear regressions. A very interesting proposition that combines the two types of decision trees is CART (*Classification and Regression*

Decision Tree) proposed by Breiman et al. [10]. In our research, we focus on classification trees to propose a method for discrete function approximation that can be adapted to the classification task. The most well-known decision tree algorithms are ID3 [31], its extension C4.5 [32], and their descendant C5, which is a commercial version of the new algorithm developed by Quinlan. As reported on his company's (RuleQuest Research Pty Ltd) web page [19], the performance of C5 surpasses that of C4.5, while at the same time having a very low time complexity. Unfortunately, this algorithm is not open source.

As mentioned above, ID3 is a typical decision tree induction algorithm. It introduces information entropy as the splitting attribute's selection measure and trains a tree from the root to the leaves, in a top-down manner. The central choice in the ID3 algorithm is selecting "the best" attribute to be tested at each node in the tree. The proposed algorithm uses *information gain,* which is a measure of how well the given attribute separates the training examples according to the target classification. C4.5 improves the appropriate attribute selection measure, avoids data overfitting, reduces error pruning, handles attributes with different weights, improves computational efficiency, handles missing value data and continuous attributes, and performs other functions. It uses the information gain ratio instead of information gain as in ID3 [32]. Other decision tree induction methods differ from that presented above, for example, in their use of different measures for attribute selection, such as the Gini or Twoing criteria used by CART [10] among others, or another statistical measure.

One of the greatest advantages of a decision tree is that it can easily be converted into a set of rules. Assume that each decision path is a rule. Let the set of rules RS consist of subsets of rules that point at the same class (assuming we have M class labels).

$$RS = \{RS_1, RS_2, \ldots, RS_M\}. \tag{1}$$

$$RS_i = \{r_i^{(1)}, r_i^{(2)}, \ldots, r_i^{(N_i)}\} \tag{2}$$

The analysis of different practical examples leads to the following general form of rule $r_i^{(k)}$.

$$\text{If } x \in D_i^{(k)}, \text{ then the object belongs to class } i,$$

where $D_i^{(k)}$ denotes the set of conditions associated with a single path of the decision tree (i.e., with rule $r_i^{(k)}$).

In [31], Quinlan notes that the computational complexity of ID3 (for discrete attributes) at each node of the tree is $O(N_{LS}\ N_A)$, where N_{LS} is the number of examples in the learning set and N_A is the number of attributes in the node. For continuous attributes, the computational complexity is quadratic in the size of the learning set [29]. In this case, to speed up the examination of the candidates, ID3 sorts the examples using the continuous attributes as the sort key. The computational complexity of this operation is $O(N_{LS}log_2N_{LS})$, which means that a great deal of time is needed for a large dataset. Another time-consuming problem is pruning, the complexity of which is hard to estimate because it depends on the decision tree size and structure.

There are several works that deal with the problem of speeding up decision tree generation. Some of these make additional assumptions that allow the tree to be built faster, such as in [34] where the authors assume conditional independence of the attributes. This leads to a reduction in the computational complexity, compared with that of the naïve Bayes or a one-level decision tree [17]. This concept called NT (*Naïve Tree*) performs slightly better than C4.5 and remarkably better than the Naïve Bayes. Of course, the authors warn of uncritical use of this method, especially where the independence assumption is violated (e.g., in text classification). Another interesting proposition is presented by Domingos et al. [12]. The

authors describe and evaluate a novel algorithm for a VFDT (*Very Fast Decision Tree*) that trains decision trees using constant memory and constant time per example. The algorithm uses Hoeffding bounds to guarantee that its output is asymptotically almost identical to that of a conventional learner.

In many of the comparative studies on classifiers reported in the literature, the decision trees achieve good results and are thus usually used in business applications [22]. To speed up decision tree generation and to deal effectively with huge databases, we need time-efficient distributed decision tree induction methods with the ability to use a network computational system [44]. Various parallel decision tree algorithms have been proposed. SLIQ [28] and its extension SPRINT [35] use a pre-sorting technique in the tree-growing phase and propose new pruning procedures. In [25], a data distributed parallel formulation of C4.5 is given, in which only frequency statistics from the data are used to select the best attribute. BOAT [16] trains an approximate tree using a fixed size subsample of the training set and then improves on this using the whole dataset. SPIES, a parallel algorithm for decision tree induction with almost linear speedup is presented in [21]. In [36], synchronous and partitioned decision tree induction algorithms are presented. Additionally, the authors compare these propositions and formulate a hybrid algorithm. The study presented in [45] is interesting in that three parallel versions of the decision tree algorithm are proposed, namely (i) feature-based parallelization, (ii) node-based parallelization, and (iii) data-based parallelization. The results of experiments evaluating the relationship between speedup and the number of processors are also given.

The above mentioned algorithms focus on constructing a decision tree for a given set of data. If new data become available, the algorithms have to start from the beginning, because the structure of a decision tree is difficult to modify. This characteristic of the methods is not too disadvantageous if the database grows at a slow rate, but for fast-growing databases it could cause serious problems. The need for tree rebuilding is usually caused by *concept drift*, which means that the recognition task model is not stable and depends on time. Generally speaking, concept drift could be caused by changes in the probabilities of classes or conditional probability distributions of classes [23].

Various decision tree algorithms for streaming data can be found in [6]. The modification of the VFDT algorithm to handle a changing environment was proposed in [18]. The method proposed in [20] builds a decision tree using horizontal parallelism based on an online method for building histograms from new data. In the paper, the authors show that the classification error of their proposition for a distributed version of the decision tree induction is slightly worse than the original one, but the bound on the error is usually acceptable for practical implementation.

As mentioned in the introduction, we are interested in developing a hybrid adaptive decision tree training method. To achieve this aim, we make use of the NGE algorithm, which produces a data structure consisting of so-called hyperrectangles that can easily be used in conjunction with the decision making concept described below.

2.2 Nearest hyperrectangle algorithm

The *nearest hyperrectangle* algorithm was described in [33], with subsequent modifications and an evaluation through computational experiments presented in [39,40]. As previously mentioned, the greatest advantage of this method is its low computational complexity. This is achieved by reducing the size of the data structure (compared with minimal distance methods) stored in memory, which is required to make a decision.

In [33], the following knowledge representation was proposed. $H_{k,i}$ denotes the k-th hyperrectangle pointing to the i-th class:

$$H_{k,i} = \begin{bmatrix} \underline{x}_{k,i}^{(1)} & \bar{x}_{k,i}^{(1)} \\ \underline{x}_{k,i}^{(2)} & \bar{x}_{k,i}^{(2)} \\ \vdots & \vdots \\ \underline{x}_{k,i}^{(d)} & \bar{x}_{k,i}^{(d)} \end{bmatrix} \tag{3}$$

where $\underline{x}_{k,i}^{(l)}$ and $\bar{x}_{k,i}^{(l)}$ denote constraints on the l-th feature $x^{(l)}$ of vector x, i.e., $\underline{x}_{k,i}^{(l)} \leq x^{(l)} \leq \bar{x}_{k,i}^{(l)}$.

The proposed measure of distance between object x_m and hyperrectangle $H_{k,i}$ is given by

$$d\left(x_m,\ H_{k,i}\right) = \gamma_{k,i} \sqrt{\sum_{l=1}^{d} \left[\alpha^{(l)} \left(\frac{d_{k,i}\left(x_m^{(l)}\right)}{\bar{x}_{k,i}^{(l)} - \underline{x}_{k,i}^{(l)}} \right) \right]^2}, \tag{4}$$

where

$$d_{k,i}\left(x_m^{(l)}\right) = \begin{cases} x_m^{(l)} - \bar{x}_{k,i}^{(l)} & \text{if } x_m^{(l)} > \bar{x}_{k,i}^{(l)} \\ \underline{x}_{k,i}^{(l)} - x_m^{(l)} & \text{if } \underline{x}_{k,i}^{(l)} > x_m^{(l)} \\ 0 & \text{if } \underline{x}_{k,i}^{(l)} \leq x_m^{(l)} \leq \bar{x}_{k,i}^{(l)} \end{cases}. \tag{5}$$

In the above equation, the weighted distance between the object and hyperrectangle is used as the distance measure. A weight $\gamma_{k,i}$ denotes the importance given to the hyperrectangle. In [39], a distance measure for discrete features is proposed, which is proportional to the number of measures between the given hyperrectangle and the training objects during learning. In addition, the distance measure takes into consideration the importance of each feature, by multiplying the distance between the l-th feature and the edge of the hyperrectangle by weight $\alpha^{(l)}$. The authors suggest setting these weights to 1 in practice. Further discussion of this can be found in [40].

2.3 Nested generalized exemplar algorithm

First, note that the NGE is a learning method that generalizes a given training set *TS* into a set of hyperrectangles in an n-dimensional Euclidean space, with

$$TS = \{(x_1,\ j_1), (x_2,\ j_2), \ldots, (x_n,\ j_n)\}, \tag{6}$$

where for the i-th element, x_i and j_i denote, respectively, observations (feature vector values) and the correct class label.

The NGE could be considered an amplification of minimal distance classifiers. The main advantage of the NGE over classical minimal distance classifiers is its low computational complexity. The computational load of a minimal distance classification of a single object is O(nd), where d is the dimension of feature vectors and n is the number of training samples. Therefore, the classification task requires a large amount of time, particularly if n is large. Minimal distance classifiers differ from most other techniques of classification in that training objects are needed during classification, whereas in typical inductive learning methods, they are needed only during the training. Various experiments evaluating the classification speedup of the NGE are given in [11].

2.4 iDTt-NGE algorithm

Here, we propose a novel iDTt-NGE (*incremental Decision Tree training NGE*) algorithm by modifying the NGE. This method is an improved version of the rb-NGE (*rule-based* NGE) presented in [42]. The iDTt-NGE, distinct from the rb-NGE, has been adapted for training using data streams. The algorithm works in two main phases.

1. Phase 1: Decision tree induction and its transform into a set of hyperrectangles.
2. Phase 2: Rebuilding the set of hyperrectangles using the NGE algorithm.

Given below is a detailed discussion of these phases in the iDTt-NGE algorithm.

2.4.1 Phase 1: Decision tree induction and its transform into a set of hyperrectangles

In this step, we use a decision tree induction algorithm, e.g., C4.5 to train a tree based on *TS* and then we convert it into a set of rules *RS* (1) taking into consideration that each decision path is a *de facto* individual rule. We assume that conditions formulated in the rules apply to continuous features belonging to a closed interval. It the rule does not include any conditions for a given feature, we use the smallest possible value for the feature and the largest one as the constraint connected to the feature. If a feature does not belong to a closed interval, the constraints have to be established in terms of the lowest and highest values of the given feature in the subset of *TS* in which the class label is consistent with the label given by the rule. Finally, we create hyperrectangles equivalent to the rules from *RS*. For example, $H_{k,i}$ is created based on $r_i^{(k)}$, as

$$H_{k,i} = \begin{bmatrix} \underline{x}^{(1)} & \bar{x}^{(1)} \\ \underline{x}^{(2)} & \bar{x}^{(2)} \\ \vdots & \vdots \\ \underline{x}^{(d)} & \bar{x}^{(d)} \end{bmatrix}, \tag{7}$$

where $\underline{x}^{(s)}$ and $\bar{x}^{(s)}$ denote the smallest and highest values, respectively, of the s-th feature of x allowed by $r_i^{(k)}$.

2.4.2 Phase 2: rebuilding the set of hyperrectangles using NGE algorithm

The original NGE algorithm uses randomly chosen elements of *TS* (known as seeds) to initialize the hyperrectangles and the rest of the training objects to generalize them. We initialize the hyperrectangles using a decision tree induction algorithm. If new data become available and we choose to consider new training elements, then we use these to change the concept hidden in the set of hyperrectangles.

Mainly, we take advantage of the functions proposed in the original NGE algorithm. Here, we examine some of the important steps of this phase.

(a) **Generalize** H_i based on (x_l, j_l) This procedure expands the constraints given by H_i so that (x_l, j_l) fulfills the condition of the hyperrectangle.
(b) **Generalize the new hyperrectangle** based on (x_l, j_l) This procedure creates a hyperrectangle so that the upper and lower values of the constraints are equal to x_l given in Fig. 1.
(c) **Find the nearest two hyperrectangles to** (x_l, j_l) This procedure uses the function for finding the n-th nearest hyperrectangle given in Fig. 2.

```
procedure generalize_hyperrectangle( H_{k,i} , (x_l, i_l) )
for s:=1 to d:
        if  D_i^(k)(x^(s)) > x_l^(s)  then D_i^(k)(x^(s)) = x_l^(s)
        if  D_i^(k)(x̄^(s)) < x_l^(s)  then  D_i^(k)(x̄^(s)) = x_l^(s)
endfor
```

Fig. 1 Pseudocode for generalized hyperrectangle used by iDTt-NGE

```
procedure find_next_nearest_hyperrectangle (n,(x_l, i_l))
1.  compute distance between (x_l, i_l) and each hyperrectangles
2.  sort hyperrectangles according mentioned distance (x_l, i_l)
3.  return n-th hyperrectangle
```

Fig. 2 Pseudocode to find the n-th nearest hyperrectangle used by RB-NGE

(d) **Prune the set of hyperrectangles** To protect the classifier from overfitting, we use the rule post-pruning method proposed by Quinlan [32]. First, every hyperrectangle is pruned to remove redundant conditions. This is probably the most popular method for pruning (e.g., used in C4.5), and in practice, it yields high-accuracy performance. A flowchart of the proposed iDTt-NGE algorithm is given in Fig. 3.

3 Experiments

To evaluate the quality of the proposed modification, two experiments were carried out.

3.1 Experiment 1

The objectives of this experiment were as follows:

- To investigate whether the number of initial hyperrectangles has an influence on the quality of classifiers trained by RB-NGE
- To compare the classification accuracy of the following algorithms:
  - NN—nearest neighbor,
  - NGE-3—NGE initialized by 3 seeds,
  - NNGE—non-nested NGE initialized by 3 seeds [11],
  - iDTt-NGE3—iDTt-NGE initialized by 3 rules and co-trained using 20% of the original elements,
  - iDTt -NGE6 - iDTt -NGE initialized by 6 rules (for Iris dataset, only 5 rules were used) and co-trained as in the previous algorithm.

Initial rules for iDTt -NGE were randomly chosen from the set of rules given by the C4.5 algorithm.

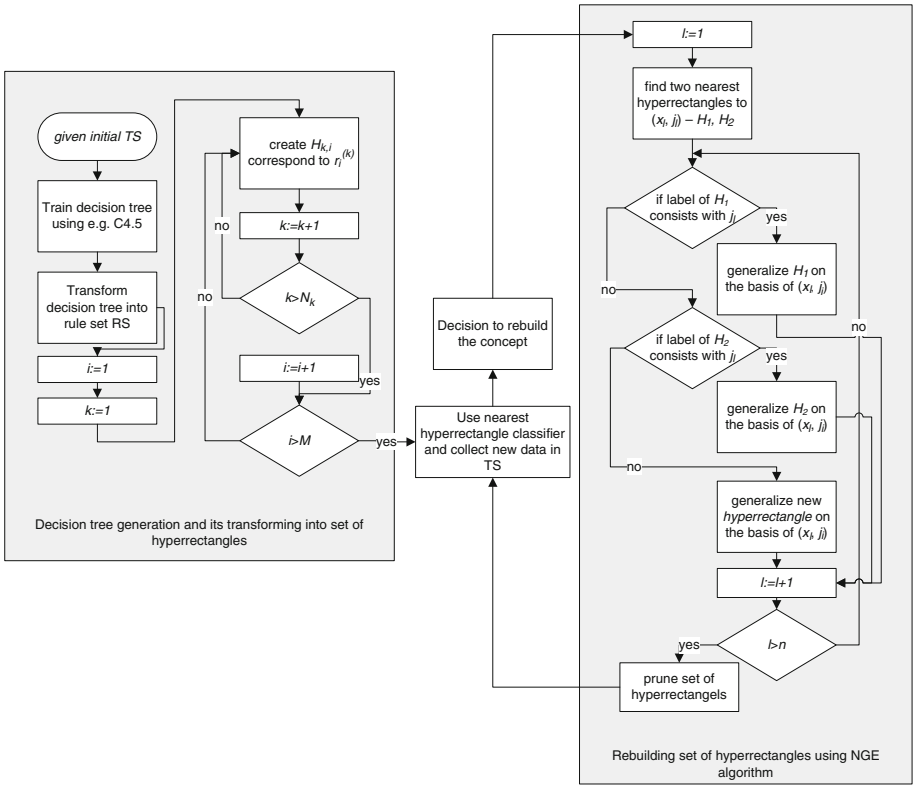

Fig. 3 Flowchart of iDTt-NGE algorithm

3.1.1 Setup of experiment 1

The setup of this experiment was as follows.

1. All experiments were carried out in WEKA [41] and Matlab (with the PRTools toolbox) environments [13].
2. Classifier errors were estimated using separate validation sets consisting of 30–50% of the original elements. Each experiment was repeated 10 times with the averaged values presented as the results.

All experiments were carried out on 5 benchmark datasets from the UCI Machine Learning Repository [5] as described in Table 1.

3.1.2 Results

Results for NNGE, NN, and NGE-3 have been published in [11,40], but to ensure we repeated the experiments exactly and we also obtained our own results, which are very similar. These are presented in Fig. 4.

3.1.3 Evaluation of experimental results

For this experiment, the following observations can be made.

Table 1 Description of datasets used in Experiment 1

Dataset	Number of			Short description
	Instances	Attributes	Classes	
Iris	150	4	3	One of the most popular benchmark datasets devoted to the recognition of a type of iris plant.
LED*	1000	7	10	Dataset consists of digit representations on a 7 segment LED display. Problem is complicated by adding noise which means that each segment could be inverted with a 10% probability [10].
Waveform*	5000	40	3	Problem of recognizing 3 classes of waves introduced by Breiman et al. [10].
Heart-dis	303	14	5	This is a very popular benchmark dataset for heart disease diagnosis.
Voting	435	16	2	Dataset includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes in 1984.

* We generated a number of instances using software from UCI; for the LED dataset we applied 10% noise

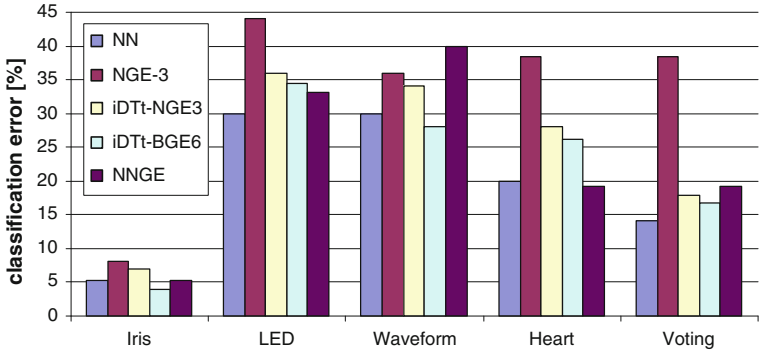

Fig. 4 Classification errors using different datasets from UCI

- As anticipated, the quality of NGE and iDTt -NGE is worse than that of NN for 4 of the databases, although for the *Waveform* dataset our proposed approach achieves the best quality.
- The quality of NNGE is very similar to that of NN.
- An increasing number of initial rules enable the iDTt -NGE to achieve better performance.

Additionally, as discussed in [42], the quality of iDTt -NGE is almost independent of the training process. It is worth noting that the original NGE and NN algorithms are highly dependent on the number of training objects.

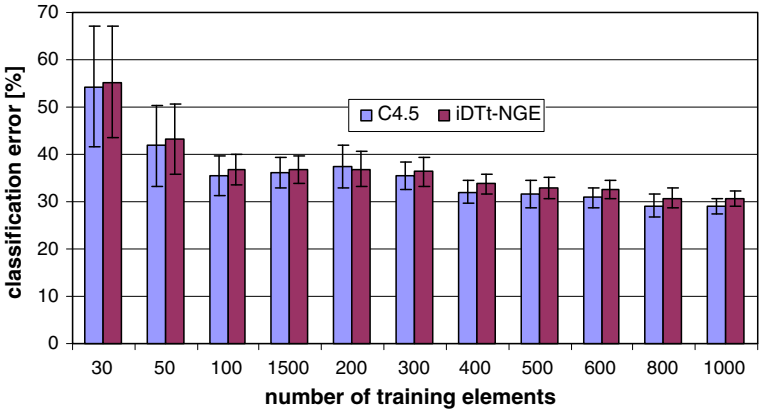

Fig. 5 Classification error for different sizes of LED dataset

3.2 Experiment 2

The main goal of the second experiment was to assess the potential to adapt the iDTt-NGE algorithm. We devised the following scenario to evaluate this characteristic.

- For a given number of training elements, train a decision tree using the C4.5 algorithm.
- Then, randomly choose 20 training elements and train another decision tree based on the remaining elements. The obtained tree is then co-trained using iDTt-NGE with the randomly chosen elements.
- This scenario was repeated for varying numbers of training elements.

3.2.1 Setup of experiment 2

The setup of this experiments was as follows.

1. All experiments were carried out in WEKA [41] and Matlab (with the PRTools toolbox) environments [13].
2. We used artificial datasets of the LED dataset, generated by the WEKA software. For each fixed number of training elements, we generated *TS* independently 10 times using different seeds and with the application of 10% noise.
3. Classifier errors were estimated using the ten-fold cross-validation method [24].

3.2.2 Results

The results of the experiments, i.e., the relationship between the classification error and size of the learning set for the LED dataset, are presented in Fig. 5.

3.2.3 Evaluation of experimental results

The following observations can be made regarding this experiment.

- C4.5 co-trained by iDTt-NGE achieves only a slightly worse quality than the original one, but what is more important is that the computational cost of the co-training procedure is

significantly lower than the cost of training a new tree. It is worth noting that the optimal Bayes classifier error is 26% [10].
- The co-trained classifier is slightly more stable (smaller standard deviation) than the original tree.

4 Conclusions

A modification of the NGE algorithm has been presented in this paper. This modification could be interpreted as an information fusion model, because in producing a knowledge representation in the form of hyperrectangles, it uses rules and a training set simultaneously. The proposed algorithm was evaluated through computer experiments.

The results obtained are encouraging, enabling us to continue working on algorithms from the NGE family. It is worth noting that classifiers obtained from algorithms based on the NGE concept are easily adaptable because if new training objects become available, we can improve the hyperrectangles by starting an NGE procedure for the new objects. This is not commonly the case in classifiers trained by machine learning methods, e.g., we cannot improve given decision trees by learning new objects. A second advantage of using the NGE method is the low computational cost for classification. Additionally, the quality of the non-nested NGE makes it worthwhile to start working on an implementation thereof as a method for decision tree co-training.

The results obtained are promising and as such, we intend to carry out new experiments on a wider range of datasets.

Acknowledgments This work is supported in part by the Polish State Committee for Scientific Research under a grant for the period 2010–2013.

References

1. Aggarwal ChC (2009) On classification and segmentation of massive audio data streams. Knowl Inf Syst 20(2):137–156
2. Aha DW (ed) (1997) Lazy learning. Kluwer, Dordrecht
3. Aksela M, Laaksonen J (2007) Adaptive combination of adaptive classifiers for handwritten character recognition. Pattern Recognit Lett 28(1):136–143
4. Alpaydin W (2010) Introduction to Machine Learning, 2nd edn. The MIT Press, London
5. Asuncion A, Newman DJ (2007) UCI Mach.Learn. Rep. [http://www.ics.uci.edu/~mlearn/MLRepository.html]. University of California, School of Information and Computer Science, Irvine, CA
6. Ben-Haim Y, Yom-Tov E (2011) A streaming parallel decision tree algorithm. J Mach Learn Res 11:849–872
7. Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In Proceedings of SIAM International Conference on Data Mining (SDM'07)
8. Bishop ChM (2006) Pattern recognition and machine learning. Springer, Berlin
9. Black M, Hickey R (2002) Classification of customer call data in the presence of concept drift and noise. In: Proceedings of the 1st international conference on computing in an imperfect world soft-ware 2002. Springer, Berlin, pp 74–87
10. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth & Brooks, Monterey
11. Brendt M (1995) Instance-based learning: nearest neighbour with generalization, Techical Report of Department of Computer Science. University of Waitako, New Zealand

12. Domingos P, Hulten G (2000) Mining highSpeed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. Boston, Massachusetts, United States pp 71–80
13. Duin RPW et al (2004) PRTools4, A Matlab Toolbox for Pattern Recognition. Delft University of Technology
14. Friedman JH (1991) Multivariate adaptive regression splines. Ann Stat 19(1):1–67
15. Gaber MM, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. ACM SIGMOD Record 34(1):18–26
16. Gehrke J, Ganti V, Ramakrishnan R, Loh W-L (1999) BOAT: optimistic decision tree construction. In: Proceedings of the 1999 ACM SIGMOD international conference on management of data. Philadelphia, PA, pp 169–180
17. Holte R (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 11:63–91
18. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceeding of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 97–101
19. Is See5/C5.0 Better Than C4.5?, RuleQuest Research Pty Ltd. http://rulequest.com/see5-comparison.html. Accessed 10 August 2010
20. Jin R, Agrawal G (2003) Communication and memory efficient parallel decision tree construction. In: Proceedings of the 3rd SIAM conference on data mining. pp 119–129
21. Jin R, Agrawal G (2003) Efficient decision tree construction on streaming data. In: Proceedings of the 9th ACM international conference on knowledge discovery and data mining (SIGKDD). Washington, D.C. pp 571–576
22. Jiang L, Li Ch, Cai Z (2009) Learning decision tree for ranking. Knowl Inf Syst 20(1):123–135
23. Kelly M, Hand D, Adams N (1999) The impact of changing populations on classifier performance. In: Proceedings of the 5th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 367–371
24. Kohavi R (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th international joint Conference on artificial intell. San Mateo, pp 1137–1143
25. Kufrin R (1997) Decision trees on parallel processors. In: Geller J, Kitano H, Suttner CB (eds) Parallel processing for artificial intelligence, vol.3. Elsevier Science, Amsterdam, pp 279–306
26. Liu H, Lin Y, Han J Methods for mining frequent items in data streams: an overview. Knowl Inf Syst doi:10.1007/s10115-009-0267-2
27. Liu S, Duffy AHB, Whitfield RI, Boyle IM (2010) Integration of decision support systems to improve decision support performance. Knowl Inf Syst 22(3):261–286
28. Mehta M, et al (1996) SLIQ: A fast scalable classifier for data mining. In: Proceedings of the 5th international conference on extending database technology, pp 18–32
29. Paliouras G, Bree DS (1995) The effect of numeric features on the scalability of inductive learning programs. Lecture notes in computer science 912: 218–231
30. Patcha A, Park J (2007) An overview of anomaly detection techniques: existing solutions and latest technological trends. Comput. Netw. 51(12):3448–3470
31. Quinlan JR (1986) Induction on decision tree. Mach Learn 1:81–106
32. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, Los Altos
33. Salzberg S (1991) A nearest hyperrectangle learning method. Mach Learn 6:251–276
34. Su J, Zhang H (2006) A fast decision tree learning algorithm. In: Proceedings of the twenty-first AAAI conference on artificial intelligence. Boston, Massachusetts July 16–20, pp 500–505
35. Shafer J et al (1996) SPRINT: a scalable parallel classifier for data mining. In the Proceedings of the 22nd VLBD conference, pp. 544–555
36. Srivastava A et al (1999) Parallel formulations of decision tree classification algorithms. Data Min Knowl Discov 3(3):237–261
37. Tsymbal A (2004) The problem of concept drift: Definitions and related work. Technical report Department of Computer Science, Trinity College: Dublin, Ireland
38. Ulaş A, Semerci M, Yıldız OT, Alpaydın E (2009) Incremental construction of classifier and discriminant ensembles. Inf Sci 179(9):1298–1318
39. Wettschereck D (1994) A hybrid nearest-neighbor and nearest-hyperrectangle algorithm. In: Proceedings of the European Conference on machine learning, pp 323–335
40. Wettschereck D, Dietterich TG (1995) An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. Mach Learn 19:5–27
41. Witten IH, Frank E (2000) Data mining: practical machine learning tools and techniques with Java implementations. Morgan Kaufmann Publisher, Los Altos

42. Wozniak M (2009) Modification of nested hyperrectangle exemplar as a proposition of information fusion method. LNCS 5788:687–694
43. Wu X, Kumar V, Quinlan JR, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou Z-H, Steinbach M, Hand DJ, Steinberg D (2008) Top 10 algorithms in data mining. Knowl Inf Syst 14(1):1–37
44. Yang Ch-T, Tsai ST, Li K-Ch (2005) Decision tree construction for data mining on grid computing environments. In: Proceedings of the 19th international conference on advanced information networking and applications AINA'05. Taipei, Taiwan, pp 421–424
45. Yidiz OT, Dikmen O (2007) Parallel univariate decision trees. Pattern Recognit Lett 28(7):825–832
46. Zhu X, Wu X, Yang Y (2006) Effective classification of noisy data streams with attribute-oriented dynamic classifier selection. Knowl Inf Syst 9(3):339–363

Author Biography

Michal Wozniak is Professor of Computer Science in the Department of Systems and Computer Networks, Faculty of Electronics, Wroclaw University of Technology, Poland. He received an M.S. degree in Biomedical Engineering in 1992 from the Wroclaw University of Technology, Ph.D. and D.Sc. (habilitation) degrees in Computer Science in 1996 and 2007 respectively, from the same university. His research focuses on multiple classifier systems, machine learning, data and web mining, Bayes compound theory, distributed algorithms, computer and networks security and teleinformatics. Prof. Wozniak has published over 140 papers, 2 books, and edited 3 books. He is member of editorial board of *Pattern Analysis and Applications* and guest editor of several international journals including, *Expert Systems*, *Information Fusion, Logic Journal of the IGPL, Neural Network World* and *International Journal of Communication Networks and Distributed Systems*. He serves on program committees of numerous international conferences. His works have been transitioned into commercial applications. Prof. Wozniak has involved in many research projects related to machine learning, computer networks and telemedicine. Moreover, he has been a consultant on several commercial projects for well-known Polish companies and for the Polish public administration. He is a senior member of IEEE (Computational Intelligence Society and Systems Man and Cybernetics Society) and IBS (International Biometric Society). For a more detailed profile of Prof. Wozniak see: http://www.kssk.pwr.wroc.pl/pracownicy/michal.wozniak-en.