Procedia Computer Science

Volume 80, 2016, Pages 1682–1691

ICCS 2016. The International Conference on Computational Science

Learning Decision Trees from Data Streams with Concept Drift

Dariusz Jankowski[1] and Konrad Jackowski[1] and Bogusław Cyganek[2]

[1]*Department of Systems and Computer Networks, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland*

[2]*AGH University of Science and Technology, Al. Mickiewicza 30, 30-059, Kraków, Poland*

dariusz.jankowski@pwr.edu.pl, konrad.jackowski@pwr.edu.pl, cyganek@agh.edu.pl

Abstract

This paper addresses a data mining task of classifying data stream with concept drift. The proposed algorithm, named Concept-adapting Evolutionary Algorithm For Decision Tree does not require any knowledge of the environment such as numbers and rates of drifts. The novelty of the approach is combining tree learner and evolutionary algorithm, where the decision tree is learned incrementally and all information is stored in an internal structure of the trees' population. The proposed algorithm is experimentally compared with state-of-the-art stream methods on several real live and synthetic datasets. Results indicate its high performance in term of accuracy and processing time.

Keywords: decision tree, adaptive learning, data stream, concept drift

1 Introduction

One of the major problems of contemporary data analysis is processing of large data volumes. Standard machine learning algorithms assume that data is available at the time of model training and that it has been generated from a static distribution. Thus, the common approach is to store the entire dataset in computer memory, and only after the training is completed a model can be used for prediction. Such processing is known as *offline learning* [9].

However, very often data comes in the form of continuous streams. In this case, traditional machine learning techniques fail- because storing large data volumes is impractical and infeasible. On the other hand, handling continuous data flow involves real-time online processing. Such online algorithms operate under the following assumptions: i) examples are processed only once, although the algorithm can remember data internally in "short term"; ii) memory used by an algorithm is limited; iii) work is done in a reasonable amount of time; iv) model can perform prediction at any point in time. Processing of large amounts of data coming in streams can be observed in such tasks as monitoring of production processes from sensor measurements, stock data exchange monitoring, computer network traffic analysis, and many more [11].

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.508

Another issue in learning from data streams is dynamically changing and non-stationary environment, which means that data distribution can change over time. This phenomenon is known as a *concept drift* [23].

For traditional classifiers the occurrence of a concept drift leads to a drop in classification accuracy. To remedy this problem, learning algorithms are endowed with suitable mechanism to adapt model to changes, as well as to distinguish drift from noise (i.e. adaptive to changes, but robust to noise).

In this paper a new algorithm named Concept-adapting Evolutionary Algorithm For Decision Tree (CEVOT) for data streams with concept drift is presented. The main idea and novelty is application of the Evolutionary Algorithm for incremental induction of a decision tree. In order to evaluate CEVOT, it is compared with the well-known and frequently used adaptive classifiers operating with real and synthetic data streams.

The rest of the paper is organized as follows. Section 2 discusses strategies related to detection and reaction on data drift. In Section 3 architecture and details of the CEVOT algorithm are provided. The last section experimental results are presented alongside with a discussion on validity of the proposed approach. Finally, Section 5 contains conclusions with directions of future research.

2 Related Work

We divide this section into two parts: research on offline algorithms and research on adaptive algorithms. Those parts include reference to our researches presented in Section 3.1 and Section 3.2 respectively.

There are many well-known decision-tree algorithms, such as ID3 [1], C4.5 [2] and CART [3]. In general, computation of the optimal model tree in a traditional top-down greedy way belongs to the group of NP-complete problems [16]. Therefore, traditional approaches are replaced for instance by induction of decision trees through evolutionary algorithms (EAs). In [1] binary decision trees are generated and individuals are encoded as trees. Then, each node is represented as a special tuple containing attribute, operator, threshold value and node type information. Similar approach is proposed in [28]. In [19] linear chromosomes (named caltrops) are used. Authors assume that each dataset attribute is Boolean and assigns the left/right child node when the attribute value is true/false. Smith [24] also designs binary decision trees that are coded through linear chromosomes. Presented there solution is dedicated for RNA search acceleration. Mentioned decision trees are the most common axis-parallel type (use of a single attribute to split each node). On the other hand, oblique trees use a linear or nonlinear combination of attributes. Examples of such oblique tree algorithms can be found in papers by Cantú-Paz and Kamath [6], as well as in the one by Krętowski [20]. However, these methods are dedicated to offline learning. In the next part we present a survey on adaptive base learners for data streams.

Adaptive base learners could be defined as algorithms which are able to dynamically adapt to new training data that can even contradict a learned concept. Depending on a chosen base classifier, adaptation can take one of the following forms:

- adaptation to the current decision model;
- adaptation to a summary of the data stream on which the model is based (windowing and weighting techniques);
- adaptation by ensemble techniques. Below we overview and map the related work. The overview is concentrated on a supervised learning in conditions of concept drift with particular emphasis on decision trees;

For handling concept drift many learning algorithms were used as a base models. One of the most popular and heavily studied is the decision tree. Domingos and Hulten proposed algorithm called a Very Fast Decision Tree (VFDT) [7]. VFDT implements node-splitting by a heuristic evaluation function in terms of the sufficient statistics and *Hoeffding bound* (HB).VFDT is capable of growing decision tree

from streaming data that are nearly equivalent to those built on a complete static pool of data. Because VFDT model works without any explicit detection of changes (blind adaptation) in paper [15] Hulten, Spencer and Domingos presented an improved version called a Concept- adapting Very Fast Decision Tree learner (CVFDT). CVFDT is able to adapt to concept-drift in streams, first growing alternate subtrees at each node of the decision tree, and then replacing the current subtree with the alternate, whenever the latter becomes more accurate. This is achieved by maintaining sufficient statistics on a time-window moving over the data stream.
Since development of the Hoeffding bounds, a number of modifications have been proposed. Bifet and Gavalda proposed the Hoeffding Window Tree (HWT) and the Hoeffding Adaptive Tree (HAT). HWT differs from CVFDT since it creates subtrees without waiting for a fixed number of instances (faster reaction for drift) and updates a subtree as soon as there is a benefit from building the new one., instead of using fixed-size sliding window to detect changes, HAT employs an adaptive window at each internal node. Another extension of VFDT is VFDTc, which is able to deal with numerical attributes, i.e. not only categorical ones [12]. The algorithm in each leaf stores counters for numerical values. Additionally, to improve performance authors proposed to add a local model in the leaves (i.e. a naïve Bayes).There are also other bounds – like the McDiarmid. In [21] authors proved that the Hoeffding's inequality is not suitable for soving the underlying problem.

Finally, let us list other learning algorithms adopted for concept drift. These are as follows:

- k-Nearest Neighbours (kNN) algorithm [2]. Authors modified the kNN for streaming environment and considered best value for *k* parameter. In [4] most recent examples is provided, as well as a forgetting window for examples updating is proposed;
- SVM version dedicated to learn on large dataset is presented in [27]. Algorithm uses a solution known as "Minimum Enclosing Balls". A balance between speed and precision can be obtained after parameters tuning. In the paper [8] further increase in performance is reported;
- Rule-based system handling data streams named FACIL is proposed in [26]. FACIL is an incremental rule learner with partial instance memory based on moderate generalization and example nearness;

An alternative idea is to create an ensemble of classifiers updating a set of classifiers created from previous chunks of data. One of the earliest proposition incorporating this idea is Streaming Ensamble Algorithm (SEA) [25].

Finally, many methods have been proposed for offline evolutionary tree induction. There are also many literature positions concentrated on incremental decision trees for data streams [10]. However there is lack of adaptive approaches for evolutionary tree learning.

3 Concept-adapting Evolutionary Algorithm For Decision Tree

We introduce a new decision tree algorithm for mining data streams in nonstationary environments called *Concept-adapting Evolutionary Algorithm for Decision Tree* (CEVOT). Proposed algorithm is extension of our batch (offline) algorithm EVO-Tree [17] (see section 3.1). In contrast to offline learning, CEVOT learns from the sliding windows without making any assumption about the nature or type of a drift nor on presence or lack of new concept classes. The novelty of the approach is combining tree learner and evolutionary algorithm, where the decision tree is learned incrementally and all information (knowledge) is stored in the internal structure of the population of trees. Moreover, this method has other advantages:

- a natural variable-length encoding structure is used, because the optimal size of a tree for a given data set is not known a priori;
- the initial algorithm allows for random generation of unbalanced trees of different sizes;
- the fitness function allows for simultaneous optimization of both, the accuracy and the tree size;

- all crossover and mutation operators are designed in such a way that as a result only correct individuals, i.e. decision trees, are created.

3.1 Background: The EVO-Tree Algorithm

EVO-Tree (Evolutionary Algorithm for Decision Tree Induction) [17] is a novel multi-objective evolutionary algorithm where two different objectives are aggregated and combined into one objective function (FF):

$$FF = \alpha_1 f_1 + \alpha_2 f_2 \tag{1}$$

where

$$f_1 = 1 - \frac{Total\ no.\ of\ Samples\ Correctly\ Classified\ in\ Training\ Set}{Total\ no.\ of\ samples\ in\ Training\ Set} \tag{2}$$

$$f_2 = \frac{Tree_{current\_depth}}{Tree_{target\_depth}} \tag{3}$$

The fitness function (1) is balanced between the number of correctly classified instances and size of the tree, weighted by the parameters α_1, α_2 which control the relative importance of the complexity term. Value of α_1, α_2 should be determined experimentally (by default $\alpha_2 = 1 - \alpha_1$). The component f_2 allows to obtain a desired size of a tree. The value of $\mathrm{Tree}_{\mathrm{target\_depth}}$ should be provided by the user by tuning it to a specific problem.

In this approach each individual generated by an evolutionary mechanism represents a binary tree. Each individual is stored in a breadth-first order as an implicit data structure, using for this purpose two arrays. All nominal data and class labels are mapped to an integer, so each component is a numeric (integer, real or null). Assuming that a node has an index *i*, its left child can be found at indices *2i* and the right child at *2i+1*, respectively. Terminal nodes store null values and class number. The root has always index one.

Genetic operators were designed in the following way: mutation may change test or class label in a node and crossover randomly chooses two nodes from two different trees (parents), then swaps subtrees rooted in those nodes.

3.2 CEVOT Algorithm Description

CEVOT inherits from EVO-Tree an evolutionary computation to process population of trees. The difference is its ability to handle data streams and gather knowledge.

The simplest and effective method of tracking concept drift is sliding window which keeps the most recent instances while older ones are dropped. CEVOT uses *fixed size sliding window* that takes a chunk of data of size *w* and retrains the model with the last *w* examples. Because nonstationary environment is considered, CEVOT algorithm will evolve with data. Whole process is done without any explicit detection of concept drift. It is known as a *blind* adaptation method [11].

Algorithm starts by random generation of an initial population. Nonetheless, this action takes place only when the first data chunk is received. For each subsequent chunks, CEVOT starts with population which remained from the previous run – this is a key feature which maintains "memory". The idea is that the algorithm adapts itself to data. With every incoming chunk, population of trees shall converge to a current state (i.e. concept) and improve their accuracy.

Learning under concept drift requires not only updating the model with new information, but also forgetting the outdated knowledge. The main limitation of the blind approaches is slow reaction to the concept drift in data. CEVOT forgets old concepts at a constant speed, independently of whether changes are happening or not (individuals selection process). To discard old information we implemented a special destructive mutation mechanism. It works as follows. Randomly selected internal nodes are converted to leaves and the sub-trees rooted at those nodes are pruned.

4 Experimental Results

Our objective is to present characteristics of CEVOT in terms of accuracy, memory usage and training time for varying frequency and level of the concept drift. We conducted an empirical study to examine the performance of the CEVOT algorithm which was implemented in the Matlab framework. The main parameters of the evolutionary computations were as follows: a number of generations per data chunk - 50, population size - 50, the crossover/mutation probability – 0.8/0.2, the selection method – stochastic uniform with elitism, the fitness functions with parameters $\alpha_1 = 0.95, \alpha_2 = 0.05$. The experiments were conducted on a computer with Intel i7-4770K 3.50 GHz with 16GB main memory, running Windows 8.1. As a references the following classifiers implemented in MOA framework were examined [19]: Hoeffding Adaptive Tree (HAT), Hoeffding Tree (HT), Naïve Bayes (NB), Accuracy Updated Ensemble (AUE) (based on HT), Accuracy Weighted Ensemble (AWE) (based on HT). All algorithms were tested using 5 real-life UCI datasets [3] and 11 synthetic datasets generated in the MOA framework [19]. A brief characteristic of used datasets is provided in Table 1.

Dataset	Instances	Attributes	Classes	Noise	Remarks	Source
MOALED Set1	100000	24	10	10%	No. of drifts - 5	MOA; desc. [5]
MOALED Set2	100000	24	10	10%	No. of drifts - 1	MOA; desc. [5]
MOHyperi Set1	100000	10	2	5%	No. of drifts - 5	MOA; desc. [15]
MOHyper Set2	100000	10	2	5%	No. of drifts - 1	MOA; desc. [15]
MORBF Set1	100000	10	5	10%	No. of drifts - 5	MOA; desc. [5]
MORBF Set2	100000	10	5	10%	No. of drifts - 10	MOA; desc. [5]
MOSEA Set1	100000	3	2	10%	Used func. 1	MOA; desc. [25]
MOSEA Set2	100000	3	2	10%	Used func. 5	MOA; desc. [25]
MOSTAGGER Set1	100000	3	2	10%	Used func. 3	MOA; desc. [22]
MOWaved Set1	100000	21	3	10%	No. of drifts - 5	MOA; desc. [13]
MOWaved Set1	100000	21	3	10%	No. of drifts - 10	MOA; desc. [13]
Airlines	539383	7	2	unknown	Unknown	http://stat-computing.org/ dataexpo/2009/
CovtypeNorm	581012	54	7	unknown	Unknown	desc. [3]
Powersupply	29928	24	2	unknown	Unknown	http://www.cse.fau.edu/ ~xqzhu/stream.html
Electricity	45000	7	2	unknown	Unknown	desc. [14]
KDDCUP99	494000	41	23	unknown	Unknown	http://sede.neurotech.com.br/ PAKDD2009/

Table 1: Datasets specification

Experiments have been executed by reading from a data stream in portions (*data chunks*). The procedure reads incoming instances, until they form a data chunk of size 1000. Each new chunk is first used to verify the existing model, then it updates the model. Finally it is removed to preserve memory. This approach is similar to the Test-then-Train method with the difference that it uses data chunks instead of single examples. It allows to measure training and testing times furthermore reducing the effect of accuracy obscuring.

4.1 Accuracy

The first set of experiments was devoted to measure performance of classification with different data streams. The average accuracies on all data chunks and average ranks achieved by tested algorithms are given in Table 2. The lower the rank is, the better it is.

To perform multiple comparisons of the classification algorithms, a statistical analysis of the results of experiments was performed. We used the non-parametric Friedman test with the Shaffer post-hoc test [18]. The null hypothesis states that there is no statistical difference between accuracy of the tested algorithms. If the null hypothesis is rejected then Shaffer post-hoc test is performed. It verifies whether there is a statistical difference between the classifiers (Table 3).

Our experiments lead to the following conclusions:

- Analysing the results contained in Table 2 we can see that the various classifiers have similar prediction results. Neither algorithm reached a clear advantage over the other, although the results of two classifiers such CEVOT and AUE should be highlighted. The highest Friedman rank was received by CEVOT, whereas the rank is quite close to the next AUE. Therefore, to draw final conclusion, we shall inspect result of post-hoc test.
- In Table 3 we can see that CEVOT performs significantly better than the Hoeffding Adaptive Tree, Accuracy Weighted Ensemble and Naïve Bayes. For the remaining algorithms the number of considered datasets is not sufficient to draw such a conclusion.

Dataset	CEVOT	AUE	HT	NB	AWE	HAT
Airlines	**66.88**	66.22	65.89	63.44	57.09	65.42
CovtypeNorm	78.39	**82.01**	77.20	62.91	77.09	75.79
elec2,data	73.58	**76.77**	75.54	68.19	68.23	73.20
kddcup99	98.90	**98.99**	98.89	98.78	98.82	98.67
MOALED_d5	**73.12**	72.82	72.34	72.89	72.91	72.04
MOALED	**72.96**	72.82	72.34	72.89	72.91	72.04
MOHyper_i5	74.17	87.28	87.69	**92.10**	91.56	86.61
MOHyper	74.24	87.28	87.69	**92.10**	91.56	86.61
MORBF_c5	**65.50**	50.09	63.30	52.14	52.31	65.26
MORBF	81.92	**86.17**	80.91	71.15	71.59	79.38
MOSEA_f4	87.78	87.69	87.39	**87.95**	87.46	86.89
MOSEA	87.41	**87.46**	86.63	87.21	86.77	86.93
MOSTAGGER_f3	99.00	99.00	99.00	99.00	99.00	99.00
MOWave_d1	80.29	**82.38**	80.27	79.68	80.15	79.10
MOWave_d5	80.66	**82.38**	80.27	79.68	80.15	79.10
Powersupply	**15.13**	14.60	15.00	15.00	14.79	14.90
Rank	***2.3438***	***2.7188***	***3.5***	***3.8125***	***3.8438***	***4.7812***

Table 2: Average accuracy on all data chunks achieved by tested algorithms and average ranks in Friedman test (last row)

No.	Algorithm	p-value
1	*CEVOT vs. HAT*	*0.000229*
2	*AUE vs. HAT*	*0.00182*
3	*AWE vs. CEVOT*	*0.023342*
4	*CEVOT vs. NB*	*0.026382*
5	HAT vs. HT	0.052737
6	CEVOT vs. HT	0.080449
7	AUE vs. AWE	0.088973
8	AUE vs. NB	0.09821

No.	Algorithm	p-value
9	AUE vs. NB	0.09821
10	HAT vs. NB	0.143027
11	AWE vs. HAT	0.156376
12	AUE vs. HT	0.237548
13	AUE vs. CEVOT	0.57075
14	AWE vs. HT	0.603272
15	HT vs. NB	0.636602

Table 3: Post-hoc Shaffer results ($\alpha = 0.05$). Only pairs of algorithm marked italic can confirm a statistically significant results.

4.2 Memory Usage and Time

Table 4 illustrate comparison of the memory usage and processing time among all datasets. We present average values obtained for all datasets.

	CEVOT	**AUE**	**AWE**	**HAT**	**HT**	**NB**
Memory [bajts]	85920* (***2148*****)	1056145	649462	127992	552107	**24360**
Training time [s]	14.41	0.55	0.89	0.07	0.02	**0.0017**

** population of trees, **single best tree model*

Table 4: Average algorithm memory usage and training time for all datasets from Table 1

In terms of memory consumption (Table 4), the following conclusions can be drawn:

- CEVOT creates the smallest model for prediction (a single tree classifier). This result is achieved through the optimal tree structure encoding and minimizing size of the tree in evolutionary algorithm. The exact description can be found in [17]. When considering the size of the entire population of individuals being processed, CEVOT took a second place.
- When analyzing the results for the CEVOT algorithm on the graphs, for both real and synthetic data (Figs. 1, 2), we can notice that despite presence of the data drift, the amount of memory consumed eventually reaches the stable value and further growth is not observed.
- It is worth mentioning that AUE and AWE requires much more memory than the remaining algorithms. This is due to the fact that both are ensembles of classifiers (i.e. they use multiple base algorithms which must be store and maintain in memory).

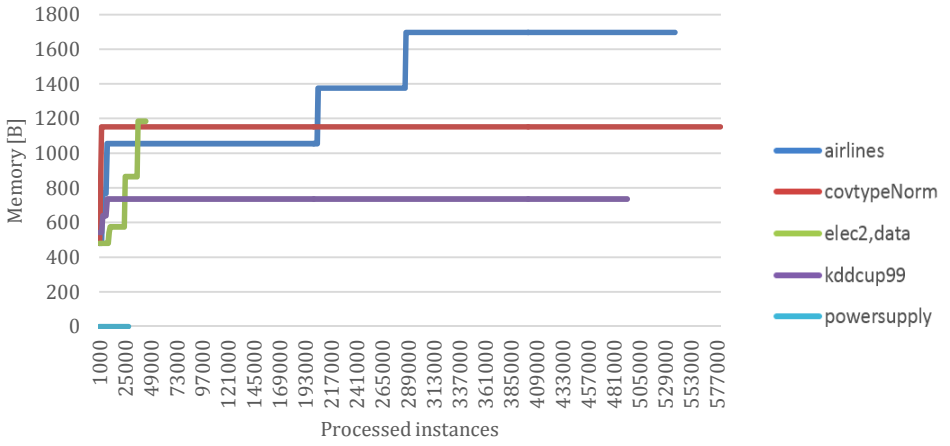

Figure 1: CEVOT memory usage on real datasets

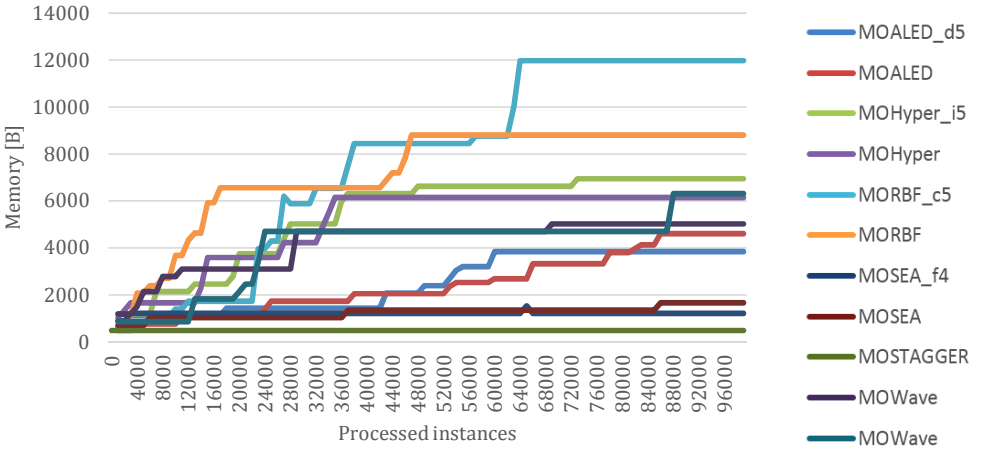

Figure 2: CEVOT memory usage on synthetic datasets

All classification algorithms were evaluated in terms of block training time (Table 4). Besides analysing the average values, we generated two graphical plots (Figs. 3, 4) for visual inspection of the CEVOT training time.

The conclusions are as follows:

- As can be seen in Table 4 the fastest classifiers were NB, HT and HAT, respectively. Unfortunately, these algorithms also have the lowest prediction performance. This fact can be explained by the simplicity of their models.
- Ensemble models received not much worse times than single classifier models (NB, HT, HAT) but AUE predicts much better than AWE. The details are given in Table 4.
- CEVOT turned out to be the slowest classifier. Evolutionary computation are very time-consuming processes. The positive aspect is the fact that most of real live datasets processing time was about five second (Fig. 3). Exception is the Powersupply dataset which heavily inflate mean result. In case of seven from eleven synthetic datasets processing time can be considered as stable (Fig. 4). For MORBF, MORBF_c5, MOHYPER_i5 and MOWAVE_d1 datasets we can observe a sudden increase of the processing time. It may be connected with growth of the model (tree) for a better adaptation to the data stream. After finding a new better (smaller) model, time can be reduced.

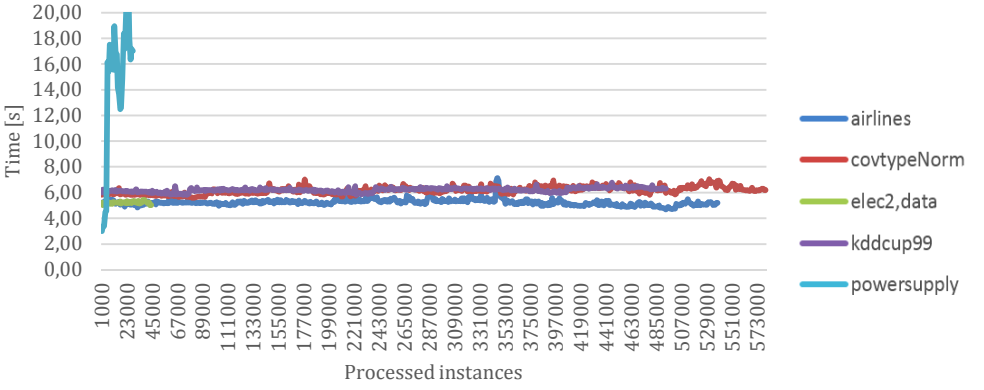

Figure 3: CEVOT training time on real datasets

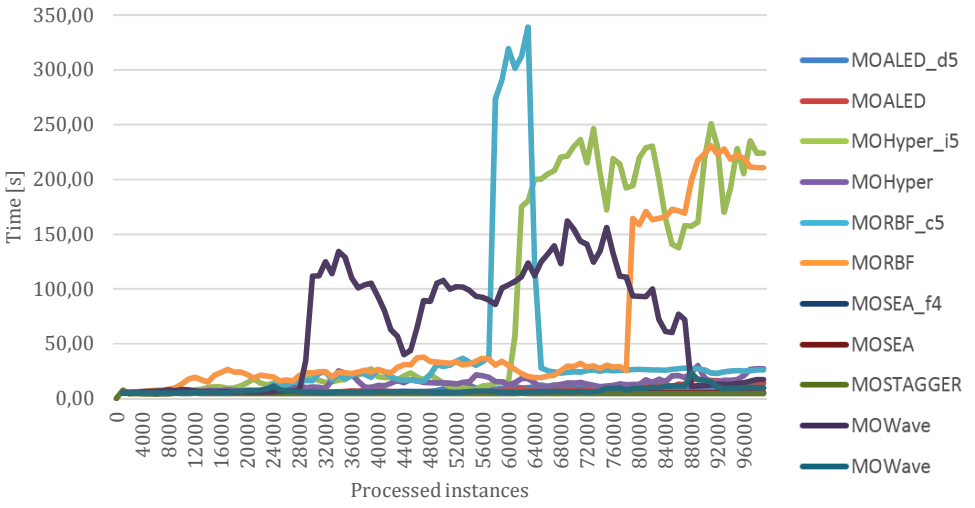

Figure 4: CEVOT training time on synthetic datasets

5 Conclusions

According to our knowledge, this article is the first work presenting an approach combining tree learner and evolutionary algorithm for incremental learning from data streams with the concept drift. Experimental analysis and comparison of accuracy and memory usage with state-of-the-art algorithms prove the efficiency of our approach. We plan to implement parallel version of the CEVOT to improve computation time. We also intend to extend the model for combining predictors from processed population (ensemble methods).

ACKNOWLEDGMENT

This work was partially supported by the Polish National Science Center under the grant no. DEC-2013/09/B/ST6/02264 and by the statutory funds of the Department of Systems and Computer Networks, Wroclaw University of Science and Technology (no. S50020). All computer experiments were carried out using computer equipment sponsored by ENGINE project (http://www.engine.edu.pl).

References

1. Aitkenhead, M.J.: A co-evolving decision tree classification method. Expert Syst. Appl. 34, 1, 18–25 (2008).
2. Alippi, C., Roveri, M.: Just-in-time adaptive classifiers in non-stationary conditions. Neural Networks, 2007. IJCNN 2007. Step 7, 1–6 (2007).
3. Bache, K., Lichman, M.: UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html, (2013).
4. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. Intell. Data Anal. 11, 627–650 (2007).
5. Breiman, L. et al.: Classification and Regression Trees. Wadsworth (1984).

6. Cantu-Paz, E., Kamath, C.: Using evolutionary algorithms to induce oblique decision trees. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1053–1060 (2000).
7. Domingos, P., Hulten, G.: Mining high-speed data streams. Proc. sixth ACM SIGKDD Int. Conf. Knowl. Discov. data Min. KDD 00. 00, 71–80 (2000).
8. Dong, J.X. et al.: Fast SVM training algorithm with decomposition on very large data sets. IEEE Trans. Pattern Anal. Mach. Intell. 27, 4, 603–618 (2005).
9. Duda, R.O. et al.: Pattern Classification. Wiley (2001).
10. Fisher, D., Schlimmer, J.: Models of Incremental Concept Learning: A coupled research proposal. (1988).
11. Gama, J. et al.: A survey on concept drift adaptation. ACM Comput. Surv. 46, 4, 1–37 (2014).
12. Gama, J. et al.: Accurate decision trees for mining high-speed data streams. Proc. ninth ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD ’03. 523 (2003).
13. Gama, J. et al.: Knowledge Discovery from Data Streams. In: Web Intelligence and Security - Advances in Data and Text Mining Techniques for Detecting and Preventing Terrorist Activities on the Web. pp. 125–138 (2010).
14. Harries, M.: SPLICE-2 Comparative Evaluation: Electricity Pricing. (1999).
15. Hulten, G. et al.: Mining time-changing data streams. Proc. seventh ACM SIGKDD Int. Conf. Knowl. Discov. data Min. KDD 01. 1, 97–106 (2001).
16. Hyafil, L., Rivest, R.L.: Constructing optimal binary decision trees is NP-complete. Inf. Process. Lett. 5, 1, 15–17 (1976).
17. Jankowski, D., Jackowski, K.: Evolutionary Algorithm for Decision Tree Induction. In: Saeed, K. and Snášel, V. (eds.) Computer Information Systems and Industrial Management. pp. 23–32 Springer Berlin Heidelberg (2014).
18. Japkowicz, N., Shah, M.: Evaluating Learning Algorithms: A Classification Perspective. Cambridge University Press (2011).
19. Kennedy, H.C. et al.: The contruction and evaluation of decision trees: A comparison of evolutionary and concept learning methods. Springer Berlin Heidelberg (1997).
20. Krętowski, M., Grześ, M.: Global induction of oblique decision trees: an evolutionary approach. Intell. Inf. Process. Web Min. 309–318 (2005).
21. Rutkowski, L. et al.: Decision Trees for Mining Data Streams Based on the McDiarmid’s Bound. IEEE Trans. Knowl. Data Eng. 25, 6, 1272–1279 (2013).
22. Schlimmer, J.C., Granger Jr, R.H.: Incremental learning from noisy data. Mach. Learn. 1, 3, 317–354 (1986).
23. Schlimmer, J.C., Granger, R.H.: Beyond Incremental Processing: Tracking Concept Drift. In: AAAI. pp. 502–507 (1986).
24. Smith, S.F.: RNA search acceleration with genetic algorithm generated decision trees. In: Proceedings - 7th International Conference on Machine Learning and Applications, ICMLA 2008. pp. 565–570 (2008).
25. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. Proc. seventh ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD ’01. 4, 377–382 (2001).
26. Troyano, F.J.F.: Incremental Rule Learning and Border Examples Selection from Numerical Data Streams. Computer (Long. Beach. Calif). 11, 8, 1426–1439 (2005).
27. Tsang, I.W. et al.: Core Vector Machines: Fast SVM Training on Very Large Data Sets. J. Mach. Learn. Res. 6, 363–392 (2005).
28. Zhao, Q., Shirasaka, M.: A Study on Evolutionary Design of Binary Decision Trees. In: Angeline, P.J. et al. (eds.) IEEE Congress on Evolutionary Computation (CEC 1999). pp. 1988–1993 IEEE Press (1999).