IEEE | IEEE Transactions on Cybernetics

Evolving Rule-Based Classifiers with Genetic Programming on GPUs for Drifting Data Streams

Journal:	*IEEE Transactions on Cybernetics*
Manuscript ID	CYB-E-2017-02-0291
Manuscript Type:	Regular Paper
Date Submitted by the Author:	26-Feb-2017
Complete List of Authors:	Cano, Alberto; Virginia Commonwealth University, Department of Computer Science Krawczyk, Bartosz; Wroclaw University of Technology, Department of Systems and Computer Networks
Keywords:	Machine learning, data streams, concept drift, genetic programming, rule-based classification, GPU, high-performance data mining

Evolving Rule-Based Classifiers with Genetic Programming on GPUs for Drifting Data Streams

Alberto Cano, *Member, IEEE,* Bartosz Krawczyk, *Member, IEEE,*

***Abstract*—Designing efficient algorithms for mining massive data streams has become one of the contemporary challenges for the machine learning community. Such models must display highest possible accuracy and ability to swiftly adapt to any kind of changes, while at the same time being characterized by low time and memory complexities. However, little attention has been paid to designing learning systems that will allow us to gain a better understanding of incoming data. There is but few proposals on how to design interpretable classifiers for drifting data streams, yet most of them are characterized by a significant trade-off between accuracy and interpretability. In this paper, we show that it is possible to have all of these desirable properties in one model. We introduce ERulesD^{2}S: Evolving Rule-based classifier for Drifting Data Streams. By using grammar-guided genetic programming, we are able to obtain accurate set of rules per class that are able to adapt to changes in the stream without a need for an explicit drift detector. Additionally, we augment our learning model with new proposals for rule propagation and data stream sampling, in order to maintain a balance between learning and forgetting of concepts. To improve efficiency of mining massive and non-stationary data, we implement ERulesD^{2}S on GPUs. We show that our method is highly adjustable to the users' needs by a flexible parameter tuning. A thorough experimental study proves that ERulesD^{2}S is able to efficiently adapt to any type of concept drift and outperform state-of-the-art rule-based classifiers, while using small number of rules. At the same time ERulesD^{2}S is highly competitive to other single and ensemble learners in terms of accuracy and computational complexity, while offering fully interpretable decision.**

***Index Terms*—Machine learning, data streams, concept drift, genetic programming, rule-based classification, GPU, high-performance data mining.**

I. Introduction

Machine learning and data mining became a vital part of moder problem-solving solutions. Many real-life applications require to handle data of such complexity, dimensionality and convolution that cannot be efficiently analyzed by humans. Additionally, in recent years we observe the phenomenon known as data flood. This is related to massive amounts of information being generated every minute, surpassing any storage or processing capabilities we have at our disposal. This has led to advent of big data era and problems connected to it, known as $5V's$. These are terms used to describe properties of such massive collections and include volume, velocity, variety, veracity, and value. This requires continuous development of new machine learning algorithms that would be capable of handling such problems [1] and to take advantage of modern computing architectures, including Graphic Processing Units (GPUs) [2] and computational clusters [3].

While volume can be seen as the most commonly discussed big data characteristic, one cannot forget the importance of remaining ones. Velocity becomes a predominant characteristic of many problems, as new instances are being continuously generated, causing need for algorithms that can process them and adapt to changes on-the-fly. This has led to the notion of data streams [4]. With this other challenges followed. Limitations in memory capacities mean that we cannot store the entire stream and need to work only with the most recent instances. Their high-speed nature forced the design of algorithms with low computational requirements and fast decision modes. Lack or latency of ground truth availability reduced the usefulness of fully supervised approaches. And finally, streams are characterized by a non-stationary nature and phenomenon known as concept drift, which may lead to shifts in data properties over time. This forces learning algorithms to be able to adapt to changes and recover their performance with the smallest possible time delay.

But what about the remaining $V's$? While there are some works devoted to addressing heterogeneous data sources (variety) or data uncertainty (veracity), there is little said so far regarding the value issue from the big data mining algorithms perspective. Most of works in this field focus on either boosting prediction performance, reducing computational complexity, or efficient implementations. But in many cases, especially from the business perspective, being able to understand, analyze and interpret reasons behind why a certain decision was made, is of at least the same value as making the correct decision itself [5]. Here, interpretable classifiers seem to offer good potential [6], [7], yet so far works on such models for massive data streams with concept drift were limited [8], [9], [10]. Additionally, they usually offer lower accuracy than other models, arguing that a trade-off between gains in interpretability and loss in predictive power is justified [9]. However, one can see how tempting it would be to have accuracy, high-speed adaptation, efficient implementation and full interpretability - all in one learning algorithm.

In this paper, we introduce ERulesD^{2}S: Evolving Rule-based classifier for Drifting Data Streams. Our approach combines efficient and flexible context-free grammar-guided genetic programming for inducting classification rules. We show that this allows to obtain high accuracy and adaptation rates to various types of concept drifts, while maintaining interpretability. Additionally, we show that ERulesD^{2}S can automatically adapt to concept drifts of any type and quickly recover its performance with no need for explicit drift detector.

A. Cano and B. Krawczyk are with the Department of Computer Science, Virginia Commonwealth University, Richmond, VA, 23284, USA; e-mail: {acano,bkrawczyk}@vcu.edu

Evolutionary computation seems as a promising direction for data stream mining [11]. However, evolving rule-based systems are characterized by a high computational complexity, making their usage prohibitive in data stream mining [12]. To address this issue, we introduce a highly efficient GPU implementation of our method that alleviates this problem. Furthermore, we propose novel rule diversification strategies for adapting to novel concepts, as well as fading sampling for smooth transitions between states.

The main contributions of this paper are as follow:

- ERulesD^{2}S: fast and evolving rule-based classifiers for drifting data streams that offers significant improvements over existing rule-based methods and comparable performance to other stream classifiers and ensembles.
- Usage of genetic programming for classifier adaptation with no need for explicit drift detection.
- Rule diversification and stream sampling strategies to allow for both fast adaptation and maintaining previously learned knowledge.
- Highly efficient implementation on GPUs for obtaining competitive runtimes on data streams.
- A thorough experimental study on a very large number of data stream benchmarks and comparison with state-of-the art algorithms (both single classifiers and ensembles) that prove high quality and performance of our proposal.

The rest of this manuscript is organized as follows. The next section presents the background in data stream mining field. Section III provides a detailed description of the proposed classification algorithm. Section IV presents the experimental results. Finally, Section V summarizes the concluding remarks.

II. Data stream mining

A data stream is a potentially unbounded and ordered sequence of instances that arrive over time [4]. This leads to a number of difficulties present only in this scenario that differ significantly from static ones. First of all, during the training of an initial classifier we do not have an access to all of instances. Instead they arrive over time one by one (online processing) of block by block (chunk-based processing). Furthermore, instances will arrive continuously and rapidly, thus creating a need for processing them within a limited amount of time to avoid queues and latencies. Data streams are assumed to be of potentially infinite size, thus making it impossible to store them in memory. This is connected with the fixed number of times each instance may be accessed (even up to using it only once in case of online learning) to reduce the memory and storage space being used. Due to the massive and high-speed nature of data the access to class labels is limited and it is often impossible to obtain them for every instance. Furthermore, access to the true labels may be additionally delayed, as in many real-life scenarios they became available after a long period, i.e., for credit approval could be 2-3 years. Finally, properties of data stream may be subject to changes over time, making continuous adaptation of the used learning model a necessity. This phenomenon is known as concept drift [13]. Presence of drift can affect the underlying properties of classes that the learning system aims to discover, thus reducing the relevance of used classifier as the change progresses. This may lead to such a deterioration of the quality of used model that it cannot be considered as a useful one anymore.

Let us assume that a data stream consists of a set of states $\mathbf{S} = \{\mathrm{S}_1, \mathrm{S}_2, \cdots, \mathrm{S}_n\}$, where S_i is generated by a distribution D_i. By a stationary data stream we will consider a sequence of instances characterized by a transition $\mathrm{S}_j \rightarrow \mathrm{S}_{j+1}$, where $\mathrm{D}_j = \mathrm{D}_{j+1}$. However, in most modern real-life problems the nature of data may evolve over time, leading to the presence of concept drift. Let us now discuss main views on the nature and types of concept drift.

- We may analyze drifts from the point of view of their influence on learned decision rules or classification boundaries. Here, we distinguish real and virtual drift. The former has an influence on decision boundaries (posterior probabilities) and additionally may impact unconditional probability density function. Therefore, it poses a challenge for the learning system and must be handled every time it appears. The latter type of drift holds no influence over decision boundaries, yet affects the conditional probability density functions. Therefore, it does not affect currently used classifier. Nevertheless, it still should be detected to understand the reason behind such a change in analyzed stream.
- We may categorize drifts according to the speed of changes taking place within the stream. Sudden concept drift is characterized by S_j being rapidly replaced by S_{j+1}, where $\mathrm{D}_j \neq \mathrm{D}_{j+1}$. Gradual concept drift can be considered as a transition phase where examples in S_{j+1} are generated by a mixture of D_j and D_{j+1} with their varying proportions. Incremental concept drift is characterized by a much slower ratio of changes, where the difference between D_j and D_{j+1} is not so significant.
- There is a special type of concept drift, known as recurring drift. In such a case it is possible that a concept from k-th previous iteration may reappear $\mathrm{D}_{j+1} = \mathrm{D}_{j-k}$. This may happen once or periodically.
- Two other types of drifts are connected with potential appearance of incorrect information in the stream. Blips are random changes in stream characteristics that should be ignored (may be seen as outliers). Noise are significant fluctuations in feature values or class labels, representing some corruption in received instances. Therefore, it must be properly filtered out during data stream mining operations.
- One may analyze the impact of the drift on the decision space [14]. Here, we may distinguish local and global drifts. Former ones affect only a small part of the stream, like a certain subset of classes. Latter ones have effect on the entire stream, which actually makes them easier to detect and handle.
- Finally, we must notice that in most real-life problems the nature of changes is far from well-defined and we must be able to deal with hybrid changes through time, known as mixed concept drift.

From this one may easily see that tackling concept drift is a major challenge for data stream mining. One may use one

of the following three solutions to manage the presence of changes:

- Rebuild the classifier whenever a new instance becomes available.
- Monitor the state of the stream, detect incoming change and rebuild the classifier when change becomes too severe.
- Use an adaptive method that will automatically adjust to any changes in the stream.

Obviously, the first approach is of prohibitive computational cost. Therefore, second and third approaches are widely used in this domain. Let us now present four main ways to tackle concept drift that are based on those principles: concept drift detectors, sliding windows, online learners and ensembles.

Concept drift detectors are external algorithms that measure the properties of stream over time and can be used to handle classifier updates. Characteristics measured by them usually include standard deviation [15] and instance distribution [16]. A change in these characteristics will indicate an appearance of concept drift. This is usually done in a two-step approach. When any deviations start to appear, a warning signal is being issued. This is an indicator to start buffering incoming instances and use them for training a new classifier. When severity of changes is above a given threshold, a detection signal is being outputted and old classifier is being replaced with a new one.

Sliding windows keep a batch of most recent and relevant examples in a dedicated buffer. Such a buffer is being used by the classifier and is being constantly updated with new instances [17]. Being of fixed size means that instances are bound to spend there a given amount of time (corresponding to the buffer size) and then being discarded. This allows to store the current state of the stream in the memory. Removal of old instances is achieved by either crisp cutting-off or weighting with applied forgetting factor. Main difficulty in efficient usage of sliding windows for drifting data streams lies in defining the size of the window. Too small window will become vulnerable to local overfitting, while too big one will store instances from mixed concepts. Therefore, recent works concentrate on dynamic adaptation of window size or using multiple windows [18].

Online learners exhibit an ability to continuously update their structure instance after instance. This allows them to adapt to changes in stream as soon as they appear. To be considered as an online learner, a given algorithm must fulfill a number of requirements. They include processing each instance at most once, working under time and memory constrains, and having a predictive accuracy not lower than a batch model trained on the same set of instances. Some of popular classifiers are actually able to work in online mode, e.g., Neural Networks or Naïve Bayes. However, there exist a plethora of methods modified to provide efficient online mode of operation [19].

Ensemble learners are becoming more and more popular for data stream mining [20]. Due to their compound structure [21] they offer at the same time good adaptability and improved predictive capabilities. There are two main approaches for forming ensembles from data streams - by changing line-up of the ensemble [22] or updating base classifiers [23]. In the former solution a new classifier is being trained on recently arrived data (usually collected in a form of chunk) and added to the ensemble. Pruning removes outdated classifiers, while a weighting scheme combines them with respect to their relevance to the current state of the stream. These algorithms may work with both incremental, online and static classifiers. Latter solutions use a pool of the same classifiers all the time, but update each component when new data become available. Here incremental or online learners are required.

Another difference between static and streaming problems lies in evaluation of classifiers. In static problems this is well-defined and usually one of many performance metrics related to predictive accuracy is being used. This however does not hold for data streams, as we must take into account the dynamic and drifting nature of instances, as well as requirements for real-time and high-throughput processing. Let us shortly discuss the correct metrics to be used for algorithms applied to data stream mining.

Predictive power is an obvious criterion measured in all learning systems. However, in data streams the relevance of instances is being reduced over time and by using a simple averaged measure we lose information on how a given classifier was able to adapt to changes. Therefore, prequential metrics are commonly used here. They give highest priority to the most recent examples and utilize a forgetting factor to reduce the impact of early stages of stream mining on the final metric. Prequential accuracy [24] is the most commonly used one.

Another vital criterion is memory consumption, due to the hardware limitations during processing potentially unbounded data stream [25]. Here both average memory usage and changes in its consumption over with time and specific operations taken by classifiers must be considered.

Update time gives us insight into the amount of time needed by a specific algorithm to accommodate new instances and update its structure. This part may be a bottleneck for many algorithms. We must assume that instances will arrive with high speed and frequency, therefore a classifier must be to process them before new ones will arrive to avoid queuing.

Decision time is another metric related to time complexity. It informs us how long a given classifiers needs to output a prediction for new instances. As the classification phase usually precedes the update phase, this may lead to another bottleneck for the stream data mining system. In many practical problem, a real-time response is required and thus no decision latency can be tolerated [26].

A well-designed algorithm for data stream mining must aim to strike a balance among all of these criteria.

III. ERulesD^{2}S algorithm

This section describes the genetic programming algorithm for learning classification rules from data streams, and its capability for adapting to concept drifts.

A. Individual representation

Individuals are represented by a genotype-phenotype pair. The genotype encodes a syntax tree, also known as derivation tree, generated through the production rules of a context-free grammar. The phenotype encodes the expression tree function resulting from the syntax tree, and it is represented in the form of a classification rule. Context-free grammars provide a formal definition of the syntactical restrictions in the generation of the classification rules. They provide high flexibility to generate linear, non-linear, and personalized functions. The use of grammars to generate genetic programming individuals is known as grammar-guided genetic programming, and it has been widely applied to data mining [27], [28]. Figure 1 shows the grammar employed to create the syntax trees using the $\langle S \rangle$ root symbol and the production rules. This grammar provides fast adaptability to both gradual and sudden concept drift, e.g. sudden concept inversion may be modeled by adding a NOT symbol at the root; surge of new concepts may be modeled by adding new branches to existing derivation combined by OR.

$$\langle S \rangle \rightarrow AND\ \langle S \rangle\ \langle Comparison \rangle$$
$$\langle S \rangle \rightarrow OR\ \langle S \rangle\ \langle Comparison \rangle$$
$$\langle S \rangle \rightarrow NOT\ \langle S \rangle$$
$$\langle Comparison \rangle \rightarrow \langle Operator \rangle\ \langle attribute \rangle\ \langle value \rangle$$
$$\langle Operator \rangle \rightarrow > \mid < \mid = \mid \neq$$
$$\langle Attribute \rangle \rightarrow random\ attribute\ in\ dataset$$
$$\langle Value \rangle \rightarrow random\ value\ within\ attribute's\ domain$$

Fig. 1: Context-free grammar to generate classification rules.

B. Genetic operators

Individuals are crossed and mutated to create new individuals along the evolutionary process. Crossover recombines the genetic information from parents expecting to produce synergist offspring. The crossover operator creates new syntax trees by mixing two random parent trees. The selective crossover chooses randomly with uniform probability a non-terminal symbol in each of the parents and swaps their subtrees creating two new offspring individuals. Nevertheless, in order to avoid bloating, individuals are not crossed if the resulting individual exceeds a maximum size predefined by the user.

Mutation maintains genetic diversity in the population by randomly altering an individual with a relatively low probability. The mutation operator randomly chooses a node (terminal or non-terminal) in the individual's syntax tree. If the node is terminal, the mutation will replace the current node with another compatible terminal. If the node is non-terminal, the subtree underneath will be replaced by a new randomly generated derivation using the grammar production rules. Similarly, the mutation operator guarantees that the offspring derivation size does not exceed the maximum size. Mutation is an essential mechanism for forgetting non-coding genetic information, e.g. removing tree branches modeling old concepts in the stream whereas introducing new mutated tree branches modeling new drifted concepts.

C. Iterative rule learning for data streams

Genetic programming algorithms usually follow the genetic cooperative-competitive learning approach, in which the rules for the classifier are selected among the best individuals obtained after running the evolutionary process once. However, this approach makes necessary to introduce complex mechanisms to maintain the diversity of the population in order to avoid that all individuals converge to the same area of search space. On the contrary, the iterative rule learning approach runs the evolutionary process multiple times, and every time it selects the single best individual to become a rule member of the classifier. Iterative rule learning has the advantage of preventing convergence of rules evolution in different runs, then providing diversity in the exploitation of search spaces.

ERulesD^{2}S follows the iterative rule learning approach. The algorithm obtains a number of user-parametrized classification rules per class to provide a diverse voting schema for test instances. Every iteration a population of individuals are evolved along a number of generations. This approach matches intrinsically the data streams natural flow. In every iteration, a new data chunk is received for training and updating the classifier. The algorithm adapts and evolves the classification rules to learn from data in the new chunk. It is essential to achieve a trade-off between maintaining the previously learned knowledge represented in the rules, and the adaptation to the data characteristics of the new chunk, possibly reflecting a concept drift. In order to achieve such balance, the population is reinitialized randomly in every iteration, but the single best individual from the previous iteration is maintained to preserve its genetic information. On the one hand, in a scenario without a concept drift, the elitist solution from the previous iteration will spread very quickly because it is already adapted to model the data, avoiding any accuracy penalty due to reinitialization. On the other hand, in a scenario with a concept drift, the new randomly generated individuals will provide new genetic diversity to quickly adapt the individuals to the drifted data characteristics. This way, in every iteration the population will comprise both the sufficient historical and new genetic material, adapting to the new chunk data and maintaining previous knowledge.

Figure 2 shows an exemplary classification rule learned for the data distribution in a given iteration. The antecedent of the rule is represented as an expression tree. In the next iteration, data has drifted in two ways: 1) the red class had a smooth transition in the y axes, and 2) new instances for the green class suddenly appeared in a unexpected area. The evolutionary algorithm evolves the classification rules to adapt them to the new data distribution by means of genetic operators and fitness selection. This illustrates the advantages of genetic programming to both preserve genetic information (subtree structures) that model existing knowledge as well as its flexibility to adapt and include new concepts.

Importantly, the difficulty of predicting each class may vary (e.g. some classes may comprise noisy, overlapped or sparse data). Therefore, the voting of the class prediction for each triggered rule for a test instance is weighted by the fitness of the rule.

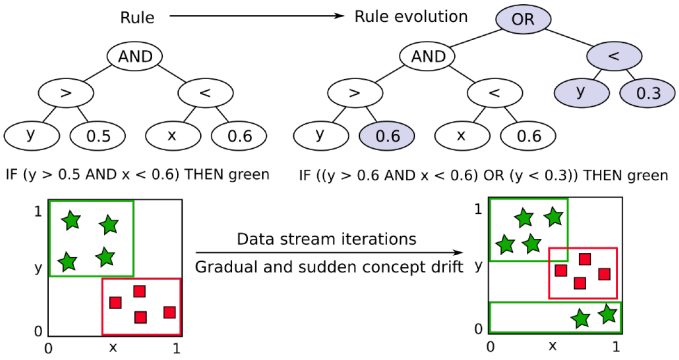

Fig. 2: Rule adaptation to concept drift.

D. Sampling sliding window

Sliding windows are one of the most popular forgetting strategies to manage removal of outdated instances. They allow the classifier to adapt and reflect concept drifts in the stream. Specifically, there are two common approaches to forgetting using sliding windows: fading and sampling. Fading employs a decay function that assigns a gradually smaller weight to each data instance in the window. A prediction error on older data will be less important than on recent data. However, it requires maintaining a large pool of ever-less important data through time, which demands ever-increasing computational resources. On the other hand, sampling keeps a fixed-size window by randomly selecting data samples. Sampling is capable of balancing smaller representativeness from older data while keeping constant computational costs through time. Specifically, ERulesD^{2}S employs a sampling sliding window for which the number of windows and the sampling factor are both user-parametrized, allowing to adapt forgetting to high-speed drifting streams. Figure 3 shows an exemplary behavior of the sampling sliding window under the presence of a smooth concept drift (rotation of data classes). The number of train examples is $ChunkSize \times \frac{samplingFactor^{windows}-1}{samplingFactor-1}$ keeping a constant computational complexity.

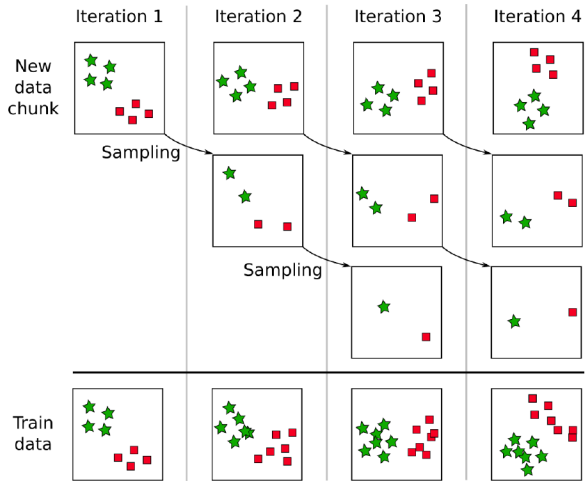

Fig. 3: Sampling sliding window (3 windows, 0.5 sampling).

E. Fitness function

The fitness function evaluates the quality of the individuals in the population. Specifically, it computes the true positives (T_P), true negatives (T_N), false positives (F_P), and false negatives (F_N) to maximize the sensitivity and specificity of the classification rules. The single-objective fitness function is defined as a linear combination of sensitivity and specificity. This function is commonplace in evolutionary algorithms for classification, and shows to perform well under the presence of imbalanced data [29].

$$fitness = \frac{T_P}{T_P + F_N} \times \frac{T_N}{T_N + F_P} \quad (1)$$

Since ERulesD^{2}S employs a sampling sliding window scheme, there is no weight assigned to the instances according to their age, thus saving computational costs.

F. Parallelization on GPUs

Evolutionary algorithms are time-consuming metaheuristics because in every generation the population of individuals must be evaluated according to the fitness function. In classification this problem is intensified due to the ever growing size of datasets, and requirement to evaluate every individual from the population on all of instances. The computational complexity of the fitness function is $\mathcal{O}(populationSize \times numberInstances)$. Indeed, more than 99% of the evolutionary algorithm's runtime is devoted to the fitness function in classification problems [30], [31], [32]. The high-computational costs prevents evolutionary methods to serve as learners for high-speed data streams. Fortunately, the fitness function is massively parallelizable both from the *population-parallel* and *data-parallel perspectives*. GPU's programming model matches naturally the massively parallel fitness function by defining a thread as the independent computation of a single individual on a single data instance. GPUs are capable of handling thousands of lightweight threads simultaneously [33], achieving speedups hundreds of times faster than the sequential implementation of the fitness function. GPU-implementations of classifiers demonstrated to become fast learners in high-speed data streams [34].

The cornerstone of the classification rules evaluator is the rule interpreter. Genetic programming individuals are created and manipulated using prefix notation. Rules are evaluated in single pass reading the symbols from the end to the beginning, performing push and pop operations on a stack. Traditional stack-based interpreters are limited due to the dependencies between the tree nodes, i.e., a parent node cannot be evaluated until its children nodes have been evaluated. However, *Cano* [35] proposed a interpreter for the GPU-parallel computation of independent subtree branches, increasing the interpreter performance to 100 billion genetic programming operations per second. This implementation of the interpreter is the one we use to speed up the evaluation of the classification rules for data streams, and it is scalable to multiple GPUs.

IV. Experimental study

In this section we present details of experiments conducted to evaluate the efficacy and usefulness of the proposed evolutionary rule-based classifier. We have designed an in-depth and thorough experimental study with three main goals in mind:

- How does various parameters of ERulesD^{2}S influence its accuracy and computational complexity, and is it possible to adapt it to specific needs of an end-user?
- Is ERulesD^{2}S able to deliver better accuracy than existing rule-based classifiers for data streams, while at the same time exhibiting comparable runtime and memory complexities?
- Is ERulesD^{2}S able to return performance comparable to state-of-the-art non-interpretable classifiers, including ensemble methods?

Answering these questions will allow us to analyze various aspects of our algorithm. Studying the influence of parameters on the performance is very important, as it offers insight into the stability and flexibility of the proposed classifier. Rule-base classifiers for data streams are natural reference methods for ERulesD^{2}S, as we want to evaluate its ability to induct rules from incoming stream of instances and to adapt them in the drift presence. Finally, existing rule-based classifiers are known to display inferior performance compared to other methods, claiming that they trade-off their performance for interpretability. Therefore, we want to evaluate if ERulesD^{2}S is able to bridge this gap. We designed three-part experiments that will allow us to answer these questions and practically validate proposed ERulesD^{2}S.

A. Datasets

In this study, we employed 26 data stream to evaluate the performance of examined algorithms. We have selected a diverse set of benchmarks reflecting various possible challenges, including real datasets and a variety of stream generators with different properties concerning speed and number of concept drifts. To the best of our knowledge this is one of the biggest experimental set-ups employed so far, as most papers in this domain are based on employing 8-16 benchmarks [36], [37]. Table I shows the characteristics of used datasets. It is important to mention here, that most of data generators, like RBF-based, are biased towards specific classifiers. Due to Gaussian-like distribution of instances, rule-based classifiers will never achieve satisfactory performance on these benchmarks. Therefore, we present them for the global perspective on the performance of our method (to identify both advantages and shortcomings), yet we direct the attention of the reader to analyzing the performance on more structured data streams (real-life ones or close to them).

TABLE I: Data stream benchmarks characteristics.

Dataset	Instances	Att	Classes	Drift
Electricity	45,312	8	2	unknown
Shuttle	58,000	9	7	no drift
DowJones	138,166	8	30	unknown
CovType	581,012	54	7	virtual
IntelLabSensors	2,313,153	6	58	unknown
RBF	1,000,000	10	2	blips
RBF-drift	1,000,000	20	4	gradual
RBF-drift-recurring	1,000,000	20	4	gradual recurring
Hyperplane	1,000,000	10	4	incremental
Hyperplane-faster	1,000,000	10	4	incremental
SEA-drift	1,000,000	3	2	sudden
SEA-drift-faster	1,000,000	3	2	sudden
LED	1,000,000	7	10	mixed
BNG-bridges	1,000,000	13	6	no drift
BNG-zoo	1,000,000	17	7	no drift
Waveform	1,000,000	40	3	gradual
Agrawal-F1-F4	1,000,000	9	2	sudden
Agrawal-F1-F10	1,000,000	9	2	sudden
Agrawal-F1-F10-faster	1,000,000	9	2	sudden
Agrawal-F10-F1	1,000,000	9	2	sudden
Agrawal-F3-F5-F7-F5-F3	1,000,000	9	2	sudden
Agrawal-F3-F5-F7-F3-F5-F7	1,000,000	9	2	sudden
Agrawal-F3-F5-recurring	1,000,000	9	2	sudden recurring
Agrawal-F9-F7-F3-F5-F4-F2	1,000,000	9	2	sudden
Agrawal-F8-F2-F7-F1-F3-F6	1,000,000	9	2	sudden
Agrawal-Random	1,000,000	9	2	sudden

B. Set-up

The experimental study evaluates and compares the performance of the proposed genetic programming rule classifier with 14 other data stream classifiers publicly available in the Massive Online Analysis (MOA) software [38]. They include two rule-based classifiers (using Very Fast Decision Rules approach), five other single models (using decision trees and distance-based methods), as well as seven ensembles.

Table II lists the algorithms compared and their main parameters, which were selected according to suggestions by authors and other studies in this area. The proposed ERulesD^{2}S has been implemented in the JCLEC software [39] and integrated in MOA. Algorithm's code along with detailed results for the experimental analysis are publicly available in[1] to facilitate the reproducibility of results and future comparisons. Experiments were run in an Intel i7-4790 CPU at 3.6GHz, 32 GB-DDR3 RAM, and NVIDIA GTX 980 GPU.

Let us now present in detail the framework used in all of following experiments:

- Methods were evaluated according to the following performance metrics: prequential accuracy, memory consumption, update time and classification time, allowing us to check their usefulness for evolving and high-speed data stream mining.
- We have used a chunk-based approach with train-then-test solution. Therefore, an incoming chunk of data as first used to measure performance of the classifier (according to the mentioned above metrics) and then to update it. We set window size $= 1000$ instances (default in MOA). Each experiment was repeated 10 times and we report averaged results over these runs.
- To assess the significance of the results, we conducted a rigorous statistical analysis [51]. We used Friedman ranking test with Bonferroni-Dunn and Shaffer post-hoc

[1] http://people.vcu.edu/%7Eacano/ERulesD2S/

TABLE II: Reference streaming classifiers and their parameters.

Acronym	Name	Parameters
VFDR [8]	Very Fast Decision Rules	supervised: true ordered: false splitConfidence: 1E-6
VFDR$_{NB}$ [8]	Very Fast Decision Rules with Naive Bayes	supervised: true ordered: false splitConfidence: 1E-6
AHT [40]	Adaptive Hoeffding Option Tree with Naive Bayes	paths: 10 splitConfidence: 0.01
NB [41]	Naive Bayes	none
KNNP [42]	KNN adaptive with PAW	neighbors: 10 limit: 1000
KNNPA [42]	KNN adaptive with PAW and ADWIN	neighbors: 10 limit: 1000
SCD [43]	Early drift detection	learner: HoeffdingTree
OBA [44]	Oza Bag Adwin	learner: HoeffdingTree ensembleSize: 10
LB [45]	Leveraging Bagging ADWIN	learner: HoeffdingTree ensembleSize: 10
SAE2 [46]	Social Adaptive Ensemble 2	learner: HoeffdingTree ensembleSize: 10
LNSE [47]	Learn^{++}.NSE	learner: NaiveBayes period: 250 ensembleSize: 15
DWM [48]	Dynamic Weighted Majority	learner: NaiveBayes
AWE [49]	Accuracy Weighted Ensemble	learner: HoeffdingTree ensembleSize: 10 ensembleBuffer: 30
OCB [50]	Online Coordinate Boosting	learner: HoeffdingTree ensembleSize: 10

tests for comparison of multiple algorithms over multiple datasets. For all of statistical tests the significance level was set to $\alpha = 0.05$.

C. Experiment 1: Selecting ERulesD^{2}S parameters

In our first experiment, we wanted to evaluate the sensitivity of ERulesD^{2}S to different parameter settings and examine how they will influence the accuracy and computational complexity. In case of genetic programming methods for static scenarios the role of proper parameter settings is of crucial importance and significantly affects the overall efficacy. It is only natural to speculate that this will carry on to the data stream mining scenario in an expanded form. Here, we care not only about the classification accuracy, but also on the computational complexity connected with adaptation to incoming instances and changes in the stream.

We have evaluated the influence of the following parameters on behavior of ERulesD^{2}S: population size, number of generations, number of rules per class, size of the window, and sampling factor. Let us now discuss the role of these parameters in learning from non-stationary data.

The population size will directly affect our ability to explore the search space for potential solution and adaptation properties of classification rules after new data becomes available. While a larger population may lead to better solution space coverage, it will also be connected with an increased computational cost for each new arriving data. Therefore, too big populations may become prohibitive in streaming scenarios.

TABLE III: Accuracy–Time parameter tuning for ERulesD^{2}S (averaged results over all data stream benchmarks).

Win	Sampl factor	Rules	Pop	Gen	Accuracy	Train Time	Test Time	RAM Hours
10	0.25	10	25	50	85.46	0.374	0.031	4.22E-4
10	0.5	10	25	50	85.74	0.365	0.030	2.39E-4
10	0.25	5	25	50	85.15	0.190	0.016	2.13E-4
10	0.5	5	25	50	85.28	0.185	0.015	1.20E-4
5	0.25	10	25	50	85.69	0.366	0.030	3.39E-4
5	0.5	10	25	50	85.65	0.364	0.030	2.30E-4
5	0.25	5	25	50	85.32	0.188	0.015	1.72E-4
5	**0.5**	**5**	**25**	**50**	**85.28**	**0.185**	**0.015**	**1.15E-4**
5	0.5	5	50	50	85.72	0.383	0.015	2.64E-4
5	0.5	5	15	25	83.76	0.072	0.015	4.65E-5
5	0.5	10	15	25	84.78	0.113	0.030	7.00E-5
5	0.5	3	25	50	85.33	0.185	0.009	1.17E-4

Number of generations will impact the ability to conduct both exploration and exploitation of the search space. While high number should potentially allow to find better solutions, the imposed increase in time complexity will be a significant limitation. We require a classifier that is able to quickly update itself with new instances and adapt to presence of concept drift, thus we need to keep the allowed iterations within reasonable bounds. Number of rules per class relates to the ability of a classifier to capture properties of given set of instances. It also influences the decision process, as majority voting among rules is implemented for predicting final class label of new instance. Higher number of rules will lead to ability of creating more complex decision boundaries and capturing atypical distributions of instances. On the other hand, this may lead to lack of diversity among evolved rules, as many of them may be very similar. As diversity is highly important for capturing concept drift (it allows to have differently specialized rules to anticipate potential changes), therefore we need to ensure it by setting a proper number of rules and using our diversification learning strategy (see section III-C for details). Next parameter informs how many data instances we keep in memory for our sliding window approach. With smaller windows, we are able to adapt to local changes. With larger windows, we keep previous contexts longer embedded in rule structure, thus allowing for better tracking the global progress of the stream. Finally, sampling factor will directly influence the forgetting ratio of ERulesD^{2}S, controlling how many instances from previous chunks are still being used to evolve classification rules. The influence of this parameter should be similar to the window size, as it also manages the ratio of adaptation to novel concepts.

We have conducted a thorough tuning of ERulesD^{2}S using 20 different parameter settings (selected via grid-search procedure) on 26 benchmarks datasets described in Table I. Due to space limitations, we present results for 12 best settings, averaged over all of datasets, in Table III.

It is highly interesting to observe that the different values of parameters do not hold a strong influence on the obtained accuracies over analyzed data streams. The highest variance in results is around 2%, which shows that our method displays

stable performance and is not highly sensitive to selected parameters. This is a very good characteristic, as it proves that even without time-consuming tuning or in-depth knowledge about domain, it is possible to efficiently use ERulesD^{2}S for data stream mining. This is very useful property, opening our method to variety of end-users. However, we should also analyze the accuracy-time trade-off. Here, one can see that although the variance in accuracy is low, the variance in time and memory consumption is much more significant. This points to the flexibility of our method, as with some tuning it is able to respond to specific needs of users. When dealing with high-speed data streams, ERulesD^{2}S may operate very fast at a small cost of accuracy loss. When accuracy is the priority, one may improve predictive power of ERulesD^{2}S, while still maintaining competitive time complexity.

For the experimental comparison with reference classifiers, we have selected the following parameter values (bold row in Table III): population size = 25, number of generations = 50, number of rules per class = 5, window size = 5, and sampling factor = 0.5. This setting offers the best balance in terms of accuracy and computational complexity.

D. Experiment 2: Comparison with rule-based classifiers

In our second experiment we compared ERulesD^{2}S with state-of-the-art Very Fast Decision Rules (VFDR) and its modification using Naive Bayes component [8]. These two methods are considered the most efficient rule-based classifiers for drifting data streams. Previous works with them showed that they are able to provide interpretable classification model that can quickly update its structure with new incoming instances and adapt to changes in the stream. VFDR claim to offer fast and accurate rule induction mechanism and high robustness to concept drifts. Therefore, they are a natural reference choice for our ERulesD^{2}S. Results over 26 data stream benchmarks with respect to accuracy, Friedman rank, memory and time complexities are reported in Table IV, while outcomes of Shaffer post-hoc test are given in Table V.

When analyzing the results one can clearly see that ERulesD^{2}S returns accuracy superior to standard VFDR method on all 26 benchmarks. A modified VFDR$_{NB}$ performs significantly better than VFDR, yet ERulesD^{2}S is still able to outperform it on 22 benchmarks. Additionally, on three benchmarks (Hyperplane, Hyperplane-faster, and BNG-zoo) the difference between these methods is less than 1% in favor of VFDR$_{NB}$. Only on LED benchmark the difference is bigger - 3.6%. The superiority of the proposed method is further verified by Friedman, Shaffer and Bonferroni-Dunn (see Figure 4) tests.

The main advantage of classification rules is the interpretability of the knowledge extracted. However, rules define a convex region in the input space in the form of axis-parallel hyper-rectangles, which limits their capability to model certain data distributions such as the random radial basis function (RBF) stream. Nevertheless, ERulesD^{2}S is shown to obtain much better accuracy than the other rule-based classifiers in the RBF datasets, especially under the presence of drifts. Additionally, VFDR tend to generate lower number of rules than the actual number of classes (e.g., for Shuttle or DowJones). This shows that in order to maximize the obtained accuracy they ignore minority classes, not covering them with any rule. ERulesD^{2}S always generates rules for each class, regardless of the imbalance ratio, and offers a balanced performance on all of them due to used skew-insensitive fitness function.

TABLE IV: Comparison between ERulesD^{2}S and state-of-the-art rule-based streaming classifiers.

Accuracy and number rules	VFDR		VFDR$_{NB}$		ERulesD^{2}S	
	Acc	Rules	Acc	Rules	Acc	Rules
Electricity	69.87	36	70.29	36	**77.75**	**10**
Shuttle	88.40	**3**	96.06	**3**	**99.77**	35
DowJones	67.71	**14**	73.36	**14**	**85.36**	150
CovType	60.32	53	75.58	53	**79.81**	**35**
IntelLabSensors	4.44	**84**	7.06	**84**	**98.75**	290
RBF	75.66	392	81.71	392	**83.34**	**10**
RBF-drift	65.42	93	88.35	93	**89.89**	**20**
RBF-drift-recurring	60.08	268	86.16	268	**87.33**	**20**
Hyperplane	78.89	69	**85.11**	69	84.64	**20**
Hyperplane-faster	78.63	75	**85.18**	75	84.63	**20**
SEA-drift	81.56	385	85.17	385	**85.89**	**10**
SEA-drift-faster	79.73	387	85.28	387	**86.27**	**10**
LED	29.45	**26**	**52.81**	**26**	49.26	50
BNG-bridges	43.23	955	67.08	955	**67.82**	**30**
BNG-zoo	54.47	251	**91.48**	251	90.90	**35**
Waveform	63.88	100	75.84	100	**80.72**	**15**
Agrawal-F1-F4	74.57	553	81.18	553	**89.04**	**10**
Agrawal-F1-F10	74.00	545	76.59	545	**88.90**	**10**
Agrawal-F1-F10-faster	71.63	601	73.66	601	**87.55**	**10**
Agrawal-F10-F1	74.36	639	77.03	639	**89.20**	**10**
Agrawal-F3-F5-F7-F5-F3	69.42	675	71.04	675	**86.71**	**10**
Agrawal-F3-F5-F7-F3-F5-F7	67.93	698	70.65	698	**87.74**	**10**
Agrawal-F3-F5-recurring	67.07	591	70.48	591	**86.05**	**10**
Agrawal-F9-F7-F3-F5-F4-F2	72.76	754	75.41	754	**87.38**	**10**
Agrawal-F8-F2-F7-F1-F3-F6	78.30	521	81.71	521	**91.31**	**10**
Agrawal-Random	71.54	635	73.09	635	**87.09**	**10**
Average	66.28	362	75.28	362	**85.12**	**33**
Friedman ranks	3.00		1.85		**1.15**	
Avg. Train time (s)	0.279		0.286		**0.260**	
Avg. Test time (s)	**0.010**		0.018		0.018	
Avg. RAM-Hours	3.0E-2		3.4E-2		**2.9E-4**	

TABLE V: Shaffer's test for comparison between ERulesD^{2}S and rule-based classifiers. Symbol '$>$' stands for situation in which ERulesD^{2}S is superior.

Hypothesis	p-value
ERulesD^{2}S vs. VFDR	$>$ (0.000000)
ERulesD^{2}S vs. VFDR$_{NB}$	$>$ (0.012555)

Let us take closer look on the performance of discussed methods on specific benchmarks. Figure 5 presents prequential and per-window accuracies, train time, and memory consumption for Agrawal-F1-F10 data stream. It is a sudden concept drift scenario, where we switch among 10 predefined functions. Therefore, previous concepts should be quickly forgotten and the restoration time (learning new concept with

high accuracy) should be as short as possible. While VFDR and VFDR$_{NB}$ have different accuracies, they both display almost identical recovery times. One can see that they slowly adapt to the new concept, increasing their competence chunk-by-chunk, and are usually never able to fully stabilize before a new drift occurs. ERulesD^{2}S on the other hand is able to recover almost instantly after each severe drift, using diverse set of rules and evolutionary adaptation to return to excellent performance in a manner of few data chunks. We also provide visualizations for Electricity (see Figure 6) and CovType (see Figure 7) data streams - they confirm the observed behavioral patterns of discussed methods.

Let us take a look on the number of rules generated by each method. While VFDR grow their rule-base according to the incoming data, ERulesD^{2}S keep a fixed number of rules per class (the number being its parameter). It is interesting to notice that VFDR will usually output lower number of rules than ERulesD^{2}S for streams with no or small drifts (e.g., Shuttle or LED), as ERulesD^{2}S rule-base is fixed per class. However, the main area of applicability of our method lies in drifting data streams. And here we can see that VFDR-based solutions tend to generate extremely high number of rules (e.g., 268 for RBF-drift-recurring or 639 for Agrawal-F10-F1), because they try to compensate for the drift by continuously adding new rules. Here the actual interpretability is being lost, as it is almost impossible to extract any useful knowledge from a set of hundreds of rules. At the same time ERulesD^{2}S always keeps a fixed and small number of rules, efficiently evolving them to adapt to appearing drifts and always maintaining excellent interpretability.

When comparing computational complexities of tested rule-based classifiers, we see that the proposed ERulesD^{2}S requires significantly lower amount of time and memory for updating itself with new instances than both version of VFDR, while displaying comparable decision times. This is an excellent outcome, as it proves that the proposed GPU-based implementation allows for very fast rule induction and learning from drifting data streams with evolutionary approach.

With this experimental study we were able to investigate the performance of ERulesD^{2}S and its relationship to state-of-the art rule-based classifiers in this domain. The proposed method takes advantage of efficient evolutionary rule induction, combined with mechanisms for assuring rule diversity and proper stream tracking via evolutionary adaptation and instance sampling. The complexity of this compound learning mechanism is alleviated via high-performance GPU implementation, leading to superior ERulesD^{2}S results in both terms of accuracy and speed. For detailed plots of each of used benchmark, please refer to the webpage associated with this paper[2].

E. Experiment 3: Comparison with other classifiers

In the previous experiment we showed that ERulesD^{2}S is able to outperform best rule-based classifier for data streams. Despite this, it is highly interesting to evaluate its performance in comparison with other non-interpretable stream classifiers. In static scenarios rule-based classifiers are known to often return inferior accuracy, claiming this as a trade-off for their interpretability. Our experiments show that this so far holds for data streams. when analyzing Bonferroni-Dunn test (see Figure 4) that compares all of tested 15 classifiers, we can see that both VFDR and VFDR$_{NB}$ return significantly lower predictive accuracy than reference methods. Therefore, user is forced to choose either a readable structure of classification rules or an improved accuracy.

In this experiment, we will evaluate the differences between ERulesD^{2}S and these non-interpretable classifiers. Results over 26 data stream benchmarks with respect to accuracy, Friedman rank, memory and time complexities are reported in Table VI, while outcomes of Shaffer post-hoc test are given in Table VII and of Bonferroni-Dunn test in Figure 4.

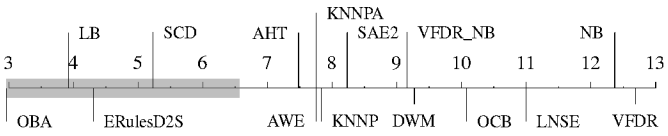

Fig. 4: Bonferroni-Dunn test for all examined classifiers.

Before we start a more in-depth discussion of results, let us state clearly that goal of this experimental study was not to show that ERulesD^{2}S are the best performing classifier for data streams. The goal here was to prove that ERulesD^{2}S can offer at least comparable performance to other classifiers, while at the same time benefiting from its interpretable nature. Therefore, we wanted to show that ERulesD^{2}S bridge a gap, no longer forcing a choice between obtaining either valuable and readable decision rules or high predictive accuracy.

By careful analysis of obtained results one may see that ERulesD^{2}S are highly comparable to all of examined 12 classifiers. This is especially important, as we had included 7 ensemble models considered to be among the most efficient ones for data stream classification [20]. Statistical analysis presented in Table VII and Figure 4 proves that ERulesD^{2}S never returns statistically significantly inferior results over multiple datasets. Lack of statistical differences between ERulesD^{2}S and most of reference methods is actually a favorable outcome. The simple fact that our rule-based classifier is able to return accuracy on pair (and in some cases even better) with ensemble classifiers shows how effective the proposed method is. Furthermore, many of the used reference methods have embedded explicit drift detectors (KNNPA, SCD, OBA, LB, LNSE), which leads to increased computational complexity, but offers no significant gains over ERulesD^{2}S in case of drifting streams. This showcases the ability of proposed GP-based learning to quickly and efficiently capture any changes in the stream and swiftly adapt to it.

This observation is backed-up by analysis of detailed behavior of selected methods in Figure 5 (we chose not to depict all of them for the sake of readability, for detailed plots of each of used benchmark data streams and examined classifiers please refer to the webpage associated with this paper[2]. It shows that ERulesD^{2}S display similar or even better recovery time that methods using external drift detectors. In this case adaptive methods without explicit drift handling (e.g., AHT)

[2]http://people.vcu.edu/%7Eacano/ERulesD2S/

TABLE VI: Comparison between ERulesD^{2}S and state-of-the art non-interpretable streaming classifiers.

Accuracy	AHT	NB	KNNP	KNNPA	SCD	OBA	LB	SAE2	LNSE	DWM	AWE	OCB	ERulesD^{2}S
Electricity	76.89	70.98	70.59	70.90	66.88	76.06	75.99	72.08	59.31	70.98	70.72	74.50	**77.75**
Shuttle	98.52	90.02	99.26	99.23	98.49	99.13	**99.80**	90.24	93.79	89.91	97.66	74.21	99.77
DowJones	69.39	61.35	78.79	78.79	**89.35**	80.82	84.21	66.19	2.74	61.35	21.39	6.03	85.36
CovType	86.22	60.04	87.89	87.80	59.56	84.26	**90.11**	76.21	69.97	71.96	80.03	71.29	79.81
IntelLabSensors	42.26	1.84	98.35	98.42	98.65	97.73	97.79	87.55	2.39	87.45	4.25	8.68	**98.75**
RBF	92.96	71.99	93.75	93.76	92.48	94.82	**95.10**	89.16	70.28	70.43	60.62	92.08	83.34
RBF-drift	95.60	60.83	98.94	**98.95**	94.92	98.11	98.76	89.04	77.53	79.98	83.80	47.46	89.89
RBF-drift-recurring	94.44	58.33	**98.43**	98.41	93.47	97.54	98.16	88.48	72.87	74.96	79.15	49.62	87.33
Hyperplane	85.36	77.25	84.27	84.24	87.64	**89.43**	89.28	83.04	86.51	86.76	88.94	87.59	84.64
Hyperplane-faster	85.34	77.23	84.27	84.24	84.33	**89.33**	89.23	83.01	86.50	86.76	88.91	87.80	84.63
SEA-drift	85.17	83.87	87.22	87.22	88.97	89.00	**89.46**	85.11	85.77	86.93	85.62	88.23	85.89
SEA-drift-faster	85.82	84.70	86.89	86.90	88.52	88.14	**89.18**	84.21	85.05	85.01	86.22	87.93	86.27
LED	52.26	51.70	44.38	44.33	52.71	**53.23**	51.31	48.53	40.54	50.22	51.70	12.56	49.26
BNG-bridges	68.27	**74.31**	70.46	70.46	56.01	51.48	43.12	44.19	68.54	70.22	46.33	19.49	67.82
BNG-zoo	92.47	**92.84**	92.67	92.67	89.63	92.35	87.04	84.61	90.23	91.74	69.77	58.67	90.90
Waveform	84.14	80.41	80.13	80.13	83.61	**85.52**	84.97	80.18	80.19	78.39	81.13	55.05	80.72
Agrawal-F1-F4	86.27	63.25	68.24	68.29	86.08	**90.93**	89.03	87.09	68.42	70.78	83.49	88.23	89.04
Agrawal-F1-F10	69.47	59.75	75.20	75.16	87.02	88.29	85.40	83.63	76.53	77.26	83.36	79.46	**88.90**
Agrawal-F1-F10-faster	68.94	59.72	74.35	74.40	84.41	85.90	82.56	83.16	75.24	76.89	82.06	78.10	**87.55**
Agrawal-F10-F1	76.59	59.12	75.08	75.10	87.63	86.42	84.21	82.35	76.48	76.98	83.47	64.30	**89.20**
Agrawal-F3-F5-F7-F5-F3	73.06	58.24	71.43	71.48	84.22	85.84	83.01	84.54	66.55	68.84	81.46	65.84	**86.71**
Agrawal-F3-F5-F7-F3-F5-F7	71.09	59.13	72.55	72.52	85.45	86.51	83.58	82.53	68.93	70.87	82.56	69.68	**87.74**
Agrawal-F3-F5-recurring	76.86	61.29	69.72	69.78	82.15	83.15	81.05	79.60	61.51	64.22	79.82	74.08	**86.05**
Agrawal-F9-F7-F3-F5-F4-F2	81.86	62.20	71.54	71.70	84.49	85.49	83.61	82.47	71.41	72.69	81.91	82.55	**87.38**
Agrawal-F8-F2-F7-F1-F3-F6	77.08	60.02	75.31	75.26	89.42	90.35	88.59	85.63	75.92	78.41	86.30	69.72	**91.31**
Agrawal-Random	68.27	57.68	73.06	73.05	81.76	82.88	81.60	80.36	76.50	77.46	80.67	74.74	**87.09**
Avg. Accuracy	78.64	65.31	80.11	80.12	83.76	**85.87**	84.85	80.12	68.83	76.06	73.90	64.15	85.12
Friedman ranks	7.04	10.83	7.33	7.25	4.92	**2.92**	3.77	7.85	9.92	8.50	7.33	9.19	4.15
Avg. Train time (s)	0.034	**0.011**	0.037	0.517	0.062	0.211	1.239	0.042	4.044	0.013	0.244	0.052	0.260
Avg. Test time (s)	0.010	0.007	0.624	0.482	**0.002**	0.012	0.015	0.016	0.248	0.011	0.025	0.015	0.018
Avg. RAM-Hours	4.7E-4	4.0E-6	2.6E-4	3.8E-4	4.4E-5	3.0E-3	3.8E-2	1.2E-4	3.2E-1	**1.1E-7**	4.5E-4	5.8E-4	2.9E-4

TABLE VII: Shaffer's test for comparison between ERulesD^{2}S and other classifiers. Symbol '$>$' stands for situation in which ERulesD^{2}S is superior and '$=$' for when there are no significant differences.

Hypothesis	p-value
ERulesD^{2}S vs. AHT	$>$ (0.007571)
ERulesD^{2}S vs. NB	$>$ (0.000000)
ERulesD^{2}S vs. KNNP	$>$ (0.003307)
ERulesD^{2}S vs. KNNPA	$>$ (0.004151)
ERulesD^{2}S vs. SCD	$=$ (0.476360)
ERulesD^{2}S vs. OBA	$=$ (0.254507)
ERulesD^{2}S vs. LB	$=$ (0.721777)
ERulesD^{2}S vs. SAE2	$>$ (0.000630)
ERulesD^{2}S vs. LNSE	$>$ (0.000000)
ERulesD^{2}S vs. DWM	$>$ (0.000057)
ERulesD^{2}S vs. AWE	$>$ (0.003307)
ERulesD^{2}S vs. OCB	$>$ (0.000003)

return unacceptable performance, as they accumulate outdated concepts, leading to incremental increase of error rates.

While the previous observation was already noted many times in data stream literature, let us now see how methods designed for concept drift handle stationary streams. Figure 7 shows accuracies for CovType stream benchmark. Here no severe drift is present, which allows AHT to display efficient performance as it incrementally learn the incoming concepts. However, usage of explicit drift detector leads to surprisingly inferior and highly unstable performance of SCD - one may easily see extreme variance in obtained accuracies over the stream progress. This can be contributed to false alarms caused by drift detector that forces a constant rebuilding of used classifier model and that prevents it from accumulating enough instances to properly capture the learned problem. ERulesD^{2}S once again exhibits performance and variance similar to best examined methods for this benchmark.

Those two previous observations lead to highly important conclusion. As mentioned before, currently we are limited to only a handful of real data stream benchmarks, while relying mainly on artificial generators. And while in such cases it is easy to generate a given type of drift, real streams are most likely to be much more complex. It is easy to anticipate presence of mixed concept drifts of varying characteristics. Appearance of real-life hybrid data streams that mix periods of drifts with periods of stationary characteristics is even more likely. Our study had showed that most popular methods are designed for specific problems in mind - ones performing well

Fig. 5: Plots of accuracy per window, prequential accuracy, memory consumption and update time of selected classifiers for AgrawalGenerator drifts from F1 to F10 data stream.

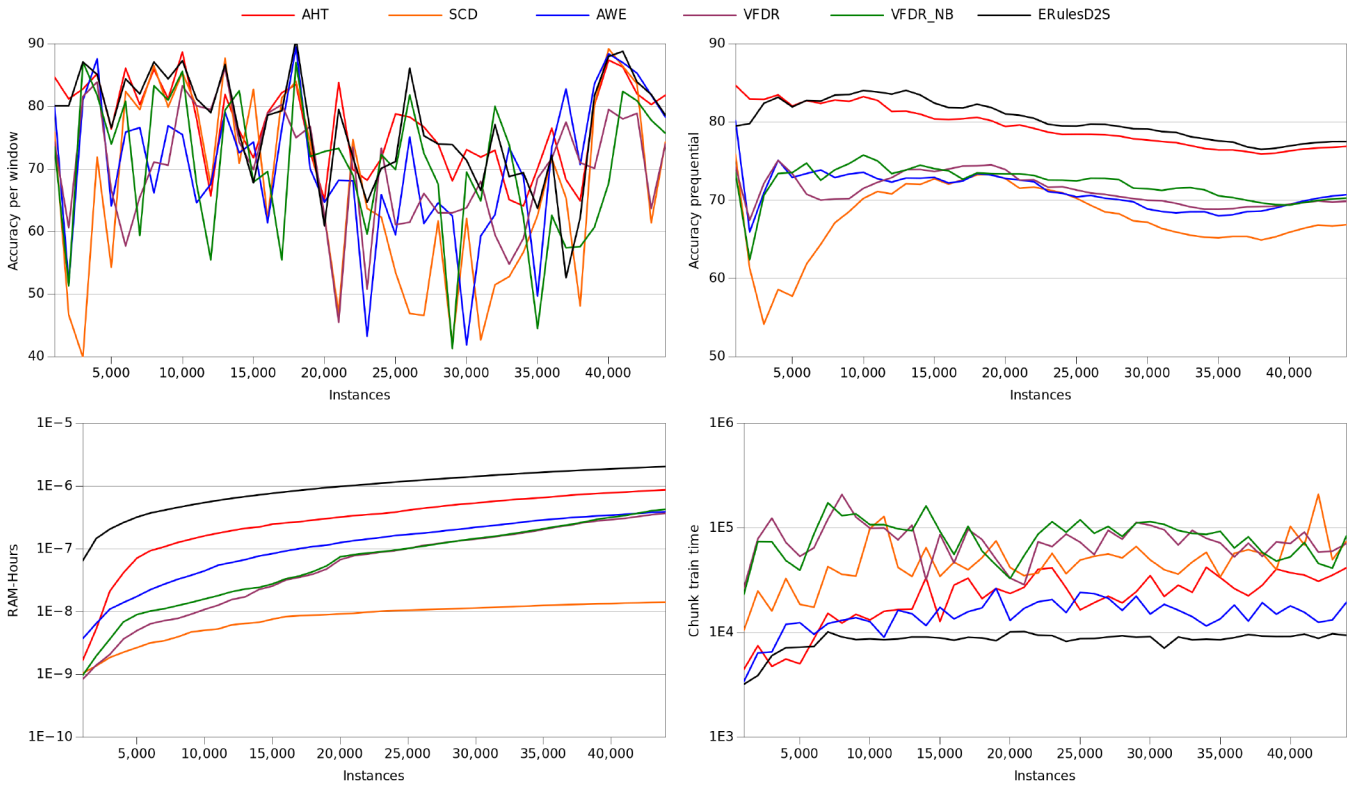

Fig. 6: Plots of accuracy per window, prequential accuracy, memory consumption and update time of selected classifiers for Electricity data stream.

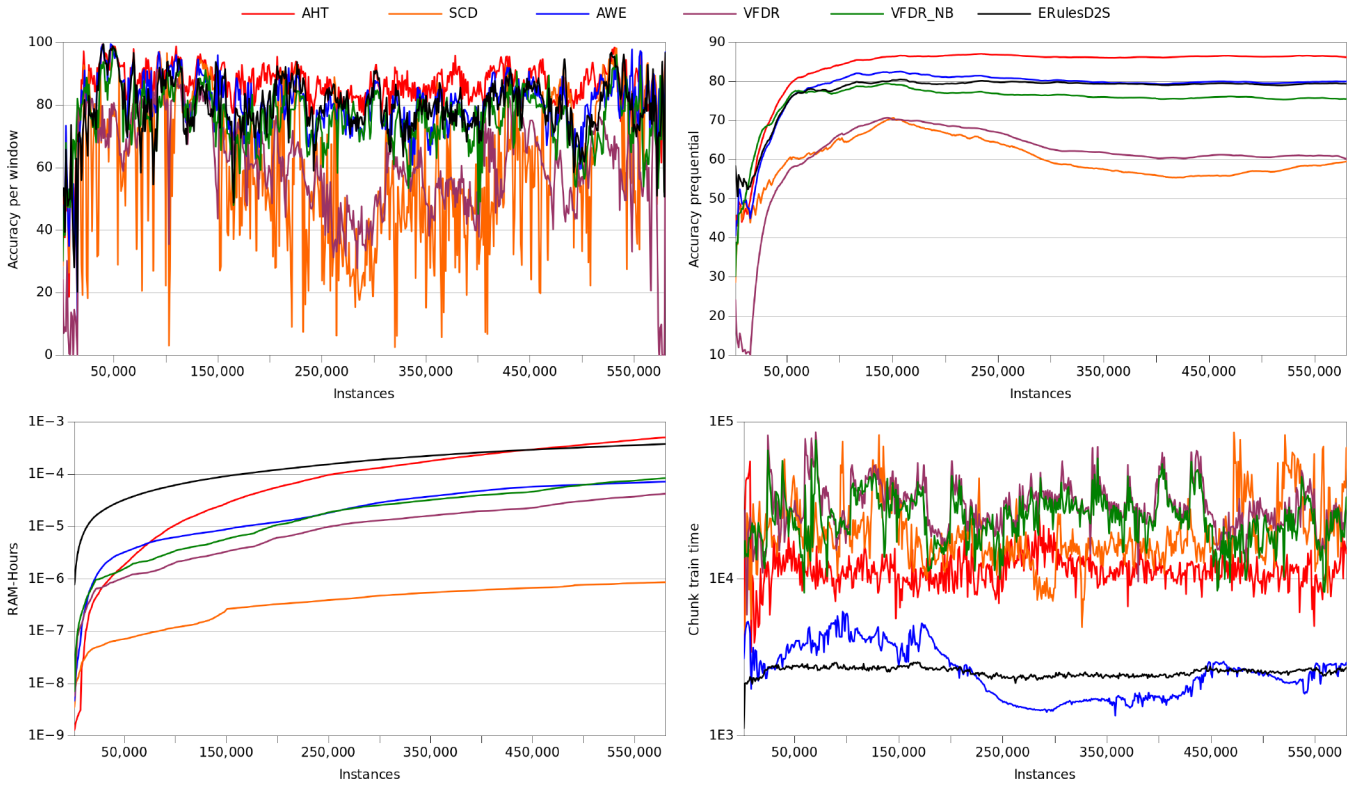

Fig. 7: Plots of accuracy per window, prequential accuracy, memory consumption and update time of selected classifiers for CovType data stream.

on drifting streams are likely to fail on stationary ones and vice versa. Therefore, in such real mixed scenarios they are likely to deliver inconsistent performance. Our experimental study showed that ERulesD^{2}S return consistent performance for various types of drifts as well as for stationary streams, being able to use evolutionary learning to adapt to any kind of scenario. Therefore, it will be able to efficiently handle such mixed and hybrid streams in real-life problems emerging in years to come. It can be backed-up by looking at performance of ERulesD^{2}S on DowJones and IntelLabSensors benchmarks. These are real-life streams with concept drift and high number of classes. We can see that ERulesD^{2}S displays an excellent performance in both cases, while many reference classifiers fail to achieve a satisfactory accuracy. This can be explained by the presence of mixed types of drifts within these datasets, high number of concepts to properly classify, as well as incremental nature of data, where new classes may appear over time and the definition of old ones may drastically change. ERulesD^{2}S is able to use its adaptive properties to efficiently tackle these difficulties embedded in real data streams.

When being compared to remaining classifiers, one can see that ERulesD^{2}S is displaying improved or equivalent performance to all of single classifiers, while at the same time offering interpretability, acceptable memory consumption and short training time (see associated website for details). Additionally, by analyzing time complexity plots one can see that ERulesD^{2}S always display highly stable training time, while other methods are characterized by a significant variance with respect to the incoming data. This additionally proves the usefulness of our classifier for real-life problems, where stable performance is usually required.

It is interesting to compare ERulesD^{2}S with ensemble approaches. Although they are considered to be among the most effective solutions for data stream mining, only OBA and LB are able to obtain better ranking performance. SAE2, LNSE, DWM, AWE and OCB return in many cases inferior accuracy when compared with our evolving rule-based classifier, despite utilizing a pool of diverse base learners and advanced classifier combination strategies. Additionally, one must note very high computational complexity displayed by LB and LSNE that make them impracticable for mining high-speed data streams.

This allows us to conclude that although our main aim was to propose an improvement over existing rule-based classifiers for drifting data streams, ERulesD^{2}S managed to achieve satisfactory performance in all of examined metrics that allows it be considered as a good all-purpose method for mining non-stationary streams.

V. Conclusions and future works

In this paper we have introduced ERulesD^{2}S: Evolving Rule-based classifier for Drifting Data Streams. It used a context-free grammar-guided genetic programming approach to evolve classification rules and adapt them to drifts in data streams without a need for an explicit drift detector. To allow for a more efficient adaptation, we augmented it with novel rule propagation between population and fading sampling for gradual forgetting of old concepts. As genetic programming methods tend to be computationally expensive and time-consuming, we decided for a highly effective ERulesD^{2}S implementation on GPUs.

A thorough experimental study compared ERulesD^{2}S with 15 state-of-the-art classifiers on 26 data streams. Results showed that our approach is able to tackle any type of concept drift, outperforming existing state-of-the-art rule-based classifiers for drifting streams, while returning comparable or even better performance than other single and ensemble learners. We have also showed that ERulesD^{2}S displays satisfactory runtime and memory consumption, while using a small number of rules for decision making process. Furthermore, we showed that the proposed method has very high flexibility and can be easily tuned by users according to their needs in terms of accuracy and runtime limitations.

We plan to develop ERulesD^{2}S further in our future works. We envision a high potential of adapting it to regression from data streams, multi-label and multi-instance non-stationary data, as well as active learning with limited instance labeling budget.

References

[1] X. Wu, X. Zhu, G. Wu, and W. Ding, "Data Mining with Big Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.
[2] D. Marron, A. Bifet, and G. D. F. Morales, "Random forests of very fast decision trees on GPU for mining evolving big data streams," in *21st European Conference on Artificial Intelligence*, 2014, pp. 615–620.
[3] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesús, J. M. Benítez, and F. Herrera, "Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380–409, 2014.
[4] M. M. Gaber, "Advances in data stream mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 79–85, 2012.
[5] H. G. Miller and P. Mork, "From data to decisions: A value chain for big data," *IT Professional*, vol. 15, no. 1, pp. 57–59, 2013.
[6] P. P. Angelov and X. Zhou, "Evolving fuzzy-rule-based classifiers from data streams," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 6, pp. 1462–1475, 2008.
[7] M. Pratama, S. G. Anavatti, P. P. Angelov, and E. Lughofer, "PANFIS: A novel incremental learning machine," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 55–68, 2014.
[8] J. Gama and P. Kosina, "Learning decision rules from data streams," in *International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, pp. 1255–1260.
[9] P. Kosina and J. Gama, "Very fast decision rules for classification in data streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 168–202, 2015.
[10] J. Duarte, J. Gama, and A. Bifet, "Adaptive model rules from high-speed data streams," *TKDD*, vol. 10, no. 3, pp. 30:1–30:22, 2016.
[11] M. Smith and V. Ciesielski, "Adapting to concept drift with genetic programming for classifying streaming data," 2016, pp. 5026–5033.
[12] M. I. Heywood, "Evolutionary model building under streaming data for classification tasks: opportunities and challenges," *Genetic Programming and Evolvable Machines*, vol. 16, no. 3, pp. 283–326, 2015.
[13] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 44:1–44:37, 2014.
[14] L. L. Minku, X. Yao, and A. P. White, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 730–742, 2009.
[15] J. Gama, P. Medas, G. Castillo, and P. P. Rodrigues, "Learning with drift detection," in *17th Brazilian Symposium on Artificial Intelligence*, 2004, pp. 286–295.
[16] P. Sobolewski and M. Woźniak, "Concept drift detection and model selection with simulated recurrence and ensembles of statistical detectors," *Journal of Universal Computer Science*, vol. 19, no. 4, pp. 462–483, 2013.

[17] M. Woźniak, "A hybrid decision tree training method using data streams," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 335–347, 2011.
[18] L. Du, Q. Song, and X. Jia, "Detecting concept drift: An information entropy based method using an adaptive sliding window," *Intelligent Data Analysis*, vol. 18, no. 3, pp. 337–364, 2014.
[19] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "The CART decision tree for mining data streams," *Information Sciences*, vol. 266, pp. 1–15, 2014.
[20] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: a survey," *Information Fusion*, vol. 37, pp. 132 – 156, 2017.
[21] M. Woźniak, M. Graña, and E. Corchado, "A survey of multiple classifier systems as hybrid systems," *Information Fusion*, vol. 16, pp. 3–17, 2014.
[22] Y. Sun, K. Tang, L. L. Minku, S. Wang, and X. Yao, "Online ensemble learning of data streams with gradually evolved classes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1532–1545, 2016.
[23] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619–633, 2012.
[24] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.
[25] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang, "Fast memory efficient local outlier detection in data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3246–3260, 2016.
[26] A. Bifet, G. D. F. Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 59–68.
[27] P. G. Espejo, S. Ventura, and F. Herrera, "A Survey on the Application of Genetic Programming to Classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 2, pp. 121–144, 2010.
[28] K. Nag and N. Pal, "A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification," *IEEE Transactions on Cybernetics*, vol. 46, no. 2, pp. 499–510, 2016.
[29] S. Khanchi, M. Heywood, and N. Zincir-Heywood, "On the impact of class imbalance in GP streaming classification with label budgets," in *19th European Conference on Genetic Programming*, vol. 9594 LNCS, 2016, pp. 35–50.
[30] A. Cano, A. Zafra, and S. Ventura, "Speeding up the evaluation phase of GP classification algorithms on GPUs," *Soft Computing*, vol. 16, no. 2, pp. 187–202, 2012.
[31] A. Cano, A. Zafra, and S. Ventura, "Parallel evaluation of Pittsburgh rule-based classifiers on GPUs," *Neurocomputing*, vol. 126, pp. 45–57, 2014.
[32] A. Cano, A. Zafra, and S. Ventura, "Speeding up multiple instance learning classification rules on GPUs," *Knowledge and Information Systems*, vol. 44, no. 1, pp. 127–145, 2015.
[33] D. Lam and D. Wunsch, "Unsupervised feature learning classification with radial basis function extreme learning machine using graphic processors," *IEEE Trans. Cybernetics*, vol. 47, no. 1, pp. 224–231, 2017.
[34] B. Krawczyk, "GPU-accelerated extreme learning machines for imbalanced data streams with concept drift," *Procedia Computer Science*, vol. 80, pp. 1692–1701, 2016.
[35] A. Cano and S. Ventura, "GPU-parallel Subtree Interpreter for Genetic Programming," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, 2014, pp. 887–894.
[36] D. Brzezinski and J. Stefanowski, "Combining block-based and online methods in learning ensembles from concept drifting data streams," *Information Sciences*, vol. 265, pp. 50–67, 2014.
[37] M. Pratama, G. Zhang, M. J. Er, and S. G. Anavatti, "An incremental type-2 meta-cognitive extreme learning machine," *IEEE Trans. Cybernetics*, vol. 47, no. 2, pp. 339–353, 2017.
[38] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.
[39] A. Cano, J. M. Luna, A. Zafra, and S. Ventura, "A Classification Module for Genetic Programming Algorithms in JCLEC," *Journal of Machine Learning Research*, vol. 16, pp. 491–494, 2015.
[40] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*, 2009, pp. 249–260.
[41] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *11th Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 338–345.
[42] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *28th ACM Symposium on Applied Computing*, 2013, pp. 801–806.
[43] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *4th International Workshop on Knowledge Discovery from Data Streams*, vol. 6, 2006, pp. 77–86.
[44] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 139–148.
[45] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *European Conference on Machine Learning and Knowledge Discovery in Databases*, 2010, pp. 135–150.
[46] H. M. Gomes and F. Enembreck, "SAE2: advances on the social adaptive ensemble classifier for data streams," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014, pp. 798–804.
[47] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.
[48] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.
[49] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 226–235.
[50] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin, "Online coordinate boosting," in *12th IEEE International Conference on Computer Vision*, 2009, pp. 1354–1361.
[51] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

Alberto Cano is an assistant professor in the Department of Computer Science, Virginia Commonwealth University, Richmond VA, USA, where he heads the High-Performance Data Mining Lab. He obtained his MSc and PhD degrees from the University of Granada, Spain, in 2011 and 2014 respectively. His research is focused on machine learning, data mining, general-purpose computing on graphics processing units, and evolutionary computation.

Bartosz Krawczyk is an assistant professor in the Department of Computer Science, Virginia Commonwealth University, Richmond VA, USA, where he heads the Machine Learning and Stream Mining Lab. He obtained his MSc and PhD degrees from Wroclaw University of Science and Technology, Wroclaw, Poland, in 2012 and 2015 respectively. His research is focused on machine learning, data streams, ensemble learning, class imbalance, one-class classifiers, and interdisciplinary applications of these methods. He has authored 35+ international journal papers and 80+ contributions to conferences. Dr Krawczyk was awarded with numerous prestigious awards for his scientific achievements like IEEE Richard Merwin Scholarship and IEEE Outstanding Leadership Award among others. He served as a Guest Editor in four journal special issues and as a chair of ten special session and workshops. He is a member of Program Committee for over 40 international conferences and a reviewer for 30 journals.