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

Article *in* Pattern Recognition · March 2019

DOI: 10.1016/j.patcog.2018.10.024

CITATION
1

READS
179

2 authors:

Alberto Cano
Virginia Commonwealth University
74 PUBLICATIONS **629** CITATIONS

SEE PROFILE

Bartosz Krawczyk
Virginia Commonwealth University
171 PUBLICATIONS **1,558** CITATIONS

SEE PROFILE

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

Alberto Cano[a], Bartosz Krawczyk[a]

[a]*Department of Computer Science, Virginia Commonwealth University, Richmond, VA, USA*

Abstract

Designing efficient algorithms for mining massive high-speed 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 are 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 sets 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 parallelized on GPUs. A thorough experimental study on 30 datasets 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 classification rules. Additionally, we show that ERulesD^{2}S can scale-up efficiently to high-dimensional data streams, while offering very fast update and classification times. Finally, we present the learning capabilities of ERulesD^{2}S for sparsely labeled data streams.

Keywords: Machine learning, data streams, concept drift, genetic programming, rule-based classification, GPU, high-performance data mining

1. Introduction

Machine learning and data mining became a vital part of modern problem-solving solutions. Many real-world applications require to handle data of such complexity and dimensionality that cannot be efficiently analyzed by humans. In recent years we observe the phenomenon known

Email addresses: acano@vcu.edu (Alberto Cano), bkrawczyk@vcu.edu (Bartosz Krawczyk)

as data flood, which is related to massive amounts of information being generated every minute, surpassing any storage or processing capabilities we have at our disposal. Consequently, this has led to the advent of the 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. Big data 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, 3] and computational clusters [4].

While volume can be seen as the most commonly discussed big data characteristic, one cannot forget the importance of the remaining properties. 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. The continuous flow of instances has led to the notion of data streams [5, 6, 7], which introduce new computational challenges. 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 forces 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 [8, 9, 10]. Concept drift 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 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 [11]. Here, interpretable classifiers seem to offer good potential [12, 13], yet so far works on such models for massive data streams with concept drift were limited [14, 15, 16, 17, 18]. 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 [17, 19, 20]. 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 present ERulesD^{2}S: Evolving Rule-based classifier for Drifting Data Streams. Our approach employs a flexible context-free grammar for inducting classification rules evolved with genetic programming. We show that our proposal obtains high accuracy and adaptation rates to various types of concept drifts, while maintaining the interpretability and a small number of rules. 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. Evolutionary computation seems as a promising direction for mining massive high-speed data stream [21]. However, evolutionary rule-based classifiers are characterized by a high computational complexity, making their usage prohibitive in data stream mining [22]. To address and resolve this issue, we introduce a highly efficient GPU-parallel implementation of our method that alleviates the runtime problem of genetic programming. 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:

- ERulesD^{2}S: fast and evolving rule-based classifier for drifting data streams that offers significant improvements over existing rule-based methods and comparable performance to other data stream classifiers and ensembles.
- Exploit of genetic programming for automatic rule adaptation to stream changes with no need for explicit drift detection.
- Rule diversification and stream sampling strategies to allow for both fast adaptation and maintaining previously learned knowledge.
- Efficient implementation on GPUs for obtaining competitive runtimes on data streams.
- Excellent scalability for high-dimensional data streams with thousands of features, while maintaining interpretable models, rapid update, and classification times.
- Capabilities for learning from partially labeled data streams with very limited access to ground truth.
- A thorough experimental study carried out on a very large number of data stream benchmarks, offering a comparison with state-of-the art algorithms (both rule-based, single classifiers and ensembles) that prove the high quality and performance of our proposal.

The rest of the manuscript is organized as follows. The next section presents the background in data stream mining. Section 3 provides a detailed description of the proposed genetic programming algorithm for learning classification rules from data streams. Section 4 presents the experimental study and discusses the results. Finally, Section 5 collects the concluding remarks.

2. Data stream mining

A data stream is a potentially unbounded and ordered sequence of instances that arrive over time [23, 24]. The stream flow leads to a number of difficulties present only in data stream mining that differ significantly from static ones. First of all, during the training of an initial classifier we do not have access to all of instances. Instead they arrive over time one by one (online processing) or 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, properties of data stream may be subject to concept drift [25], making continuous adaptation of the used learning model a necessity. 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 [26, 27, 28].

A data stream is a sequence $< S_1, S_2, ..., S_n, ... >$, where each element S_j is a set of instances (or a single instance in case of online learning), where each of them is independent and randomly

generated according to some stationary probability distribution D_j. 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, many real-life problems may be subject to various types of concept drift:

- Drifts may be analyzed 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.

- Drifts can be categorized 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 $\mathrm{D}_{j+1} = \mathrm{D}_{j-k}$, once or periodically.

- Two other types of drifts are connected with potential appearance of incorrect information in the stream: blips and noise. Blips are random changes in stream characteristics that should be ignored (may be seen as outliers). Noise represents significant fluctuations in feature values or class labels, representing some corruption in received instances.

- Finally, we must notice that in most real-world problems the nature of changes is far from being well-defined or known, and we must be able to deal with hybrid changes through time, known as mixed concept drift.

Considering the aforementioned problems, one may easily see that tackling concept drift is a major challenge for data stream mining. Thus, we may use one of the following three solutions. First, rebuild the classifier whenever a new instance becomes available, which implies a prohibitive computational cost. Second, monitor the state of the stream, detect incoming change and rebuild the classifier when change becomes too severe. Third, use an adaptive method that will automatically adjust to any changes in the stream. 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 and the accuracy of the classifier over time, which can be used to trigger classifier updates [29]. Characteristics measured by them usually include standard deviation [30] and instance distribution [31]. A change in these characteristics will indicate an appearance of concept drift.

Sliding windows keep a batch of most recent and relevant examples in a dedicated buffer. The buffer is being used by the classifier and is being constantly updated with new instances [32]. 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.

Online learners exhibit an ability to continuously update their structure instance after instance. This ability 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 [33, 34, 35].

Incremental and online rule-based classifiers have gained attention of the data stream mining community, offering effective and interpretable decisions. One of the first approaches for incremental learning of decision rules was FLORA group of algorithms [36]. They are based on continuous adaptation of nominal feature conditions for positive, negative and boundary rules. A sliding window was used to simulate forgetting. Ferrer-Troyano et al. [37] proposed online rule-based system that discarded positive and negative examples from a rule when they are not their nearest neighbors. The same authors developed FACIL [38], where rules are expressed as intervals over numerical attributes defining a hyper-rectangle. Specific rules are first being inducted and then generalized over incoming instances. Very Fast Decision Rules [14] and Adaptive Very Fast Decision Rules [17] construct ordered or unordered set of rules that are being expanded using divide and conquer approach. Pruning was applied to adapt to concept drift in incoming instances. Stahl et al. [15] proposed eRules, an evolving rule-based classifier that was able to abstain from an uncertain label prediction. Parameters of this classifier were dynamically updated with the progress of data stream, allowing to handle changing feature space. This approach was later developed into G-eRules by Le et al. [16] in order to deal with continuous attributes, where the Gaussian distribution is being used as a heuristic for rule induction on continuous attributes. RILL [39] was introduced as a drift-oriented rule-based system that uses bottom-up rule generalization based on nearest rules, while conducting extensive pruning of the rule set. Expressiveness of decision rules was proposed as an evaluation criterion in [20], where authors compared interpretability of existing models and showed how to handle uncertainty in rule-based decision via abstaining.

Finally, ensemble learners are becoming more and more popular for data stream mining [40]. Due to their compound structure [41] 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 [42] or updating base classifiers [43].

Performance metrics for evaluating classifiers differ between static and streaming problems. In static problems metrics are well-defined and usually related to predictive accuracy, area under the curve, or Cohen's kappa. However, for data streams we must take into account the dynamic and drifting nature of instances, as well as requirements for real-time and high-throughput processing. A well-designed stream mining algorithm must aim to strike a balance among all of these criteria.

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 exam-

ples and utilize a forgetting factor to reduce the impact of early stages of stream mining on the final metric. Prequential accuracy [44] is the most commonly used one. Memory consumption is another vital criterion, due to the hardware limitations during processing potentially unbounded data stream [45]. 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. Model update 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 classifier needs to output a prediction for new instances. As the classification phase usually precedes the update phase, which may lead to another bottleneck for the stream data mining system. In many practical problem, real-time response is required and thus no decision latency can be tolerated [46].

3. ERulesD^{2}S algorithm

This section describes the proposed genetic programming algorithm for learning classification rules from data streams, and its ability for adapting to concept drifts. First, the individual representation for classification rules is introduced. Second, the genetic operators for evolving the rules are detailed. Third, the iterative rule learning evolutionary model is described. Fourth, the sampling sliding window methodology is presented. Finally, the fitness function and its parallel implementation on GPUs is described.

3.1. Individual representation

Genetic programming evolves a population of individuals 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 IF-THEN 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 any combination of 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 [47, 48]. Figure 1 shows the context-free grammar employed to create the rules. The grammar G is defined by a tuple (V_N, V_T, P, S) where V_N represents the non-terminal symbols, V_T the alphabet of terminal symbols, P the production rules, and S the root symbol. A production rule is defined as $\alpha \rightarrow \beta$ where $\alpha \in V_N$ and $\beta \in \{V_T \cup V_N\}^*$. The generation of individuals is a stochastic process in which the production rules are randomly selected to produce an initial set of classification rules with varied length (limited by a maximum number of derivations). Production rules are applied obtaining a syntax tree (derivation tree) for each individual, in which internal nodes contain only non-terminal symbols, whereas leaves nodes contain only terminals. During the evolution process, genetic operators will employ the grammar to update the components of the classification rule. The 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, and old concepts may be forgotten by removing the respective branch.

$$
\begin{aligned}
&G = (V_N, V_T, P, S)\\
&V_N = \{Comparison,\ Operator,\ Attribute,\ Value\}\\
&V_T = \{AND,\ OR,\ NOT,\ <,\ >,\ =,\ \neq,\ attributes,\ values\}\\
&P\ =\ \{\langle S\rangle \rightarrow AND\ \langle S\rangle\ \langle Comparison\rangle\\
&\qquad \langle S\rangle \rightarrow OR\ \langle S\rangle\ \langle Comparison\rangle\\
&\qquad \langle S\rangle \rightarrow NOT\ \langle S\rangle\\
&\qquad \langle Comparison\rangle \rightarrow \langle Operator\rangle\ \langle Attribute\rangle\ \langle Value\rangle\\
&\qquad \langle Operator\rangle \rightarrow > \mid < \mid = \mid \neq\\
&\qquad \langle Attribute\rangle \rightarrow random\ attribute\ in\ dataset's\ features\\
&\qquad \langle Value\rangle \rightarrow random\ value\ within\ attribute's\ valid\ domain\\
&\}
\end{aligned}
$$

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

3.2. Genetic operators

Individuals are crossed and mutated to create new solutions along the evolutionary process. Crossover recombines the genetic information from parents in hopes of producing offspring with improved fitness. The crossover operator creates new syntax trees by mixing two random branches from the 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 [49].

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 and introducing new useful information. Therefore, removing tree branches from the classification rules modeling old concepts will be useful to forget no longer interesting knowledge which is not represented in the current data stream. On the other hand, introducing new mutated tree branches into the classification rule will be useful to model new drifted concepts. Thus, genetic programming rules evolve dynamically according to their adaptation to the data streams evolution [50, 51].

3.3. Iterative rule learning for drifting 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 [52]. 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 the convergence of rules evolution in different runs, then providing diversity in the exploitation of search spaces. Consequently, the rule set for a class includes diverse components which will improve the quality of the predictions.

Algorithm 1 ERulesD^{2}S algorithm for mining data streams.

```
1: elitistRules ← {∅}
2: trainData ← {∅}
3: while stream.hasData() do
4:    ruleBase ← {∅}
5:    dataChunk ← stream.nextData()
6:    trainData ← samplingSlidingWindow(trainData ∪ dataChunk, fadingFactor)
7:    iteration ← 0
8:    while iteration < numberRules do
9:       population ← initializePopulation(populationSize) ∪ elitistRules
10:      evaluate(population, trainData)
11:      generation ← 0
12:      while generation < numberGenerations do
13:         parents ← parentSelector(population)
14:         crossed ← crossover(parents)
15:         offspring ← mutator(crossed)
16:         evaluate(offspring, trainData)
17:         population ← selectBest(population ∪ offspring, populationSize)
18:         generation ← generation + 1
19:      end while
20:      ruleBase ← ruleBase ∪ bestRule(population)
21:      iteration ← iteration + 1
22:   end while
23:   elitistRules ← ruleBase
24: end while
```

ERulesD^{2}S follows the iterative rule learning approach and its pseudo-code is introduced in Algorithm 1. The algorithm learns a number of user-parametrized classification rules per class to provide a diverse voting schema for instances. In every iteration, a population of individuals are evolved along a number of generations. This approach matches intrinsically the data streams 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 encoding the model learned in previous iterations. 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 same data distribution, 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 genetic diversity to quickly adapt 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.

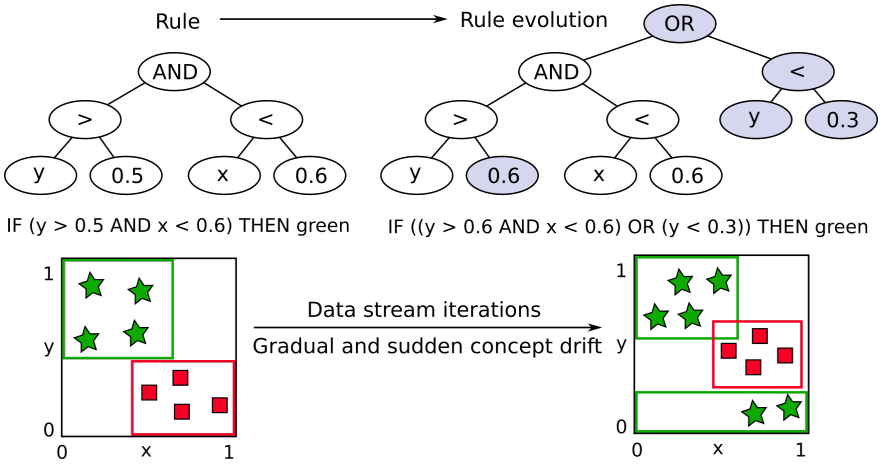

Figure 2: Rule adaptation to concept drift.

Figure 2 shows an example of a classification rule learned for the data 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 new 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. Particularly, thanks to the mutation operator, it is possible to introduce a new branch in the expression tree to model a new area covered by the classification rule, and adapt existing attribute–value comparisons to fit the drifted data distribution. 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 of the data classes 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. This way, we minimize the possibility of inaccurate rules with low fitness values to raise false positives.

3.4. Sampling sliding window

Sliding windows are one of the most popular forgetting strategies to manage removal of outdated instances [53]. They allow the classifier to adapt and reflect concept drifts in the data stream. Specifically, there are two common approaches to forgetting using sliding windows: fading and sampling [54]. 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. This is a desired property since we can predict the amount of time the classifier update is going to take, and this time is constant for every iteration. Specifically, ERulesD^{2}S employs a sampling sliding window for which the number of windows and the sampling factor are both user-parametrized. This flexible configuration allows to adapt sampling and forgetting to high-speed drifting streams. Figure 3 shows an example of the behavior of the sampling sliding window under

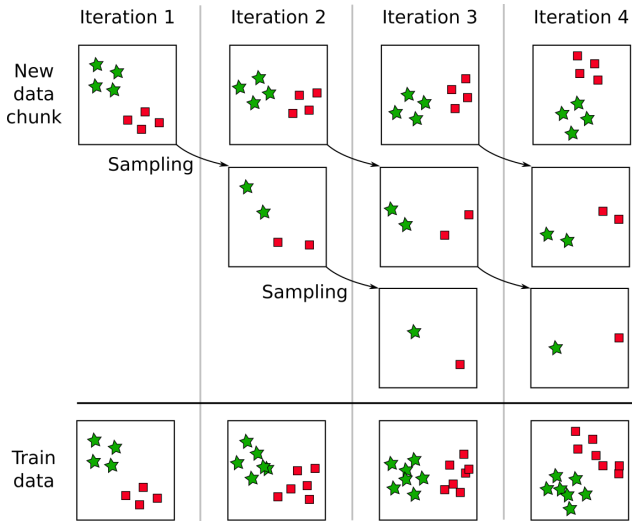

Figure 3: Sampling sliding window (3 windows, 0.5 sampling factor).

the presence of a smooth concept drift (rotation of data classes). The number of train examples is $ChunkSize \times \frac{samplingFactor^{windows}-1}{samplingFactor-1}$, then keeping a constant computational complexity.

3.5. Fitness function

The fitness function evaluates the quality of the individuals in the population by checking the coverage of the classification rule across the data instances in the train window, then comparing the predicted and ground-truth data class. Specifically, it computes the confusion matrix for the true positives (T_P), true negatives (T_N), false positives (F_P), and false negatives (F_N) to maximize the sensitivity (true positive rate) and specificity (true negative rate) of the classification rules. The single-objective fitness function is defined as the product of sensitivity and specificity. This function is commonplace in evolutionary algorithms for classification, and shows to perform well under the presence of imbalanced data [55]. Robustness of online learners for imbalanced data streams is a new challenging area where data classes distribution may shift through time [56].

$$fitness = \frac{T_P}{T_P + F_N} \times \frac{T_N}{T_N + F_P} \tag{1}$$

Since ERulesD^{2}S employs a sampling sliding window scheme, there is no need to weight the instances according to their relative age, thus saving the computational costs of age penalty calculation for each instance in every evaluation and iteration. The computational complexity of the fitness function is $\mathcal{O}(P \times N)$ where P is the population size and N is the number of instances.

3.6. 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. Indeed, more than 99% of the evolutionary algorithm's runtime in classification problems is devoted to the fitness function computation [57, 58, 59]. The high-computational costs prevent sequential implementations of evolutionary methods to serve as learners for high-speed data streams [60, 61]. Fortunately, the fitness function is massively parallelizable both from the *population-parallel* and *data-parallel*

perspectives. The former refers to the evaluation of each individual in the population in parallel, since there are no dependencies among rules. The latter refers to the coverage of a rule for each data instance, which can be also computed in parallel to obtain the confusion matrix. 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. Therefore, the computational complexity of the fitness function on a GPU can be reduced to $\mathcal{O}(\frac{P \times N}{NThreads})$, having a potentially unlimited number of threads issued by the kernel function. GPUs are capable of handling thousands and millions of lightweight threads simultaneously [62], achieving speedups hundreds of times faster than the sequential implementation of the fitness function. GPU-implementations of classifiers demonstrated to become very fast learners in high-speed data streams [63, 64].

The cornerstone of the classification rules fitness function evaluator is the rules 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 [65]. 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* [66] 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.

4. 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 five main goals in mind:

- Comparison with streaming rule-based learners. 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?
- Comparison with other streaming learners. Is ERulesD^{2}S able to return performance comparable to state-of-the-art non-interpretable classifiers, including ensemble methods? How methods behave under the presence of noise and concept drift?
- Robustness of ERulesD^{2}S. How do various parameters of our classifier influence its accuracy and computational complexity, and is it possible to adapt it to specific needs of an end-user?
- Scalability of ERulesD^{2}S. Is our classifier able to efficiently handle high-dimensional and high-speed data streams, while maintaining its predictive and runtime capabilities?
- Requirement for an access to class labels. Can ERulesD^{2}S effectively update itself and adapt to changes in partially labeled data streams?

Answering these questions will allow us to analyze various aspects of our algorithm. Rule-base classifiers for data streams are natural reference methods for comparing 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. Existing rule-based classifiers are known to display inferior performance compared

to other black-box methods, claiming that rules trade-off their performance for interpretability. Therefore, we want to evaluate if ERulesD^{2}S is able to bridge the gap. Finally, studying the influence of parameters on the performance is very important, as it offers insight into the stability, robustness, and flexibility of the proposed classifier. We designed five-part experiments that will allow us to answer these questions and empirically validate proposed ERulesD^{2}S.

4.1. Datasets

The experimental study comprises 30 data streams to evaluate and compare the performance of 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, number of concept drifts, and noise. To the best of our knowledge this is one of the biggest

Table 1: Data stream benchmarks characteristics.

Dataset	Instances	Atts	Classes	Properties
Electricity	45,312	8	2	unknown drift
Shuttle	58,000	9	7	unknown drift
DowJones	138,166	8	30	unknown drift
CovType	581,012	54	7	unknown drift
IntelLabSensors	2,313,153	6	58	unknown drift
BNG-bridges	1,000,000	13	6	unknown drift
BNG-zoo	1,000,000	17	7	unknown drift
RBF	1,000,000	50	4	no drift
RBF-drift	1,000,000	20	4	blips drift
LED	1,000,000	7	10	no drift
Hyperplane	1,000,000	10	4	no drift
RandomTree	1,000,000	10	2	no drift
RandomTree-drift	1,000,000	10	2	gradual drift
SEA-drift	1,000,000	3	2	sudden drift
Waveform	1,000,000	40	3	no drift
STAGGER	1,000,000	3	2	no drift
Agrawal-F1-F10-drift	1,000,000	9	2	gradual drift
Agrawal-recurring-drift	1,000,000	9	2	recurring drift
Agrawal-F10-F1-drift	1,000,000	9	2	gradual drift
Agrawal-drift	1,000,000	9	2	sudden drift
Agrawal-random-drift	1,000,000	9	2	sudden drift
AssetNegotiation-F2	1,000,000	5	2	no drift
AssetNegotiation-F3	1,000,000	5	2	no drift
AssetNegotiation-F4	1,000,000	5	2	no drift
Sine	1,000,000	5	2	no drift
RBF-drift-noise	1,000,000	10	2	gradual drift, noise
Hyperplane-drift-noise	1,000,000	10	4	gradual drift, noise
SEA-drift-noise	1,000,000	3	2	sudden drift, noise
CovType-noise	1,000,000	54	7	unknown drift, noise
Waveform-drift-noise	1,000,000	40	3	mixed drift, noise

experimental setups employed so far, as most papers in data streams are based on 8-16 benchmarks [67, 68]. Table 1 shows the characteristics of used datasets. Noise datasets were generated applying a filter to add random noise to 45% of the attributes. 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).

4.2. Experimental set-up

The experimental study evaluates and compares the performance of the proposed ERulesD^{2}S genetic programming rule classifier with 16 other data stream classifiers publicly available in the Massive Online Analysis (MOA) software [69]. They include three rule-based classifiers, five other single models (using decision trees and distance-based methods), as well as eight ensembles.

Table 2 lists the algorithms and their main parameters, which were selected according to the recommended values reported by authors and other studies in this area. The proposed ERulesD^{2}S has been implemented in the JCLEC software [70] 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 3.6 GHz, 32 GB-DDR3 RAM, and NVIDIA GTX 980 GPU on a Linux Centos 7 system.

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 test-then-train solution. Therefore, an incoming chunk of data is 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.
- To assess the significance of the results, we conducted a rigorous statistical analysis [83]. We used Friedman ranking test with Bonferroni-Dunn, Holm, and Shaffer post-hoc tests for comparison of multiple algorithms over multiple datasets. For all of statistical tests the significance level was set to $\alpha = 0.05$.

[1] http://go.vcu.edu/ERulesD2S

Table 2: Algorithms for data stream mining and their parameters.

Acronym	Algorithm's name	Parameters
ERulesD^{2}S	Evolving Rule classifier for Drifting Data Streams	population: 25 generations: 50 rulesPerClass: 5 windows: 5 samplingFactor: 0.5
VFDR [14]	Very Fast Decision Rules	supervised: true ordered: false splitConfidence: 1E-6
VFDR$_{NB}$ [14]	Very Fast Decision Rules with Naive Bayes	supervised: true ordered: false splitConfidence: 1E-6
G-eRules [16]	Gaussian Evolving set of Rules	slidingWindowSize: 500 minRuleTries: 10
AHT [71]	Adaptive Hoeffding Option Tree with Naive Bayes	paths: 10 splitConfidence: 0.01
NB [72]	Naive Bayes	none
KNNP [73]	KNN adaptive with PAW	neighbors: 10 limit: 1000
KNNPA [73]	KNN adaptive with PAW and ADWIN	neighbors: 10 limit: 1000
SCD [74]	Early drift detection	learner: HoeffdingTree
OBA [75]	Oza Bag Adwin	learner: HoeffdingTree ensembleSize: 10
LB [76]	Leveraging Bagging ADWIN	learner: HoeffdingTree ensembleSize: 10
SAE2 [77]	Social Adaptive Ensemble 2	learner: HoeffdingTree ensembleSize: 10
LNSE [78]	Learn^{++}.NSE	learner: NaiveBayes period: 250 ensembleSize: 15
DWM [79]	Dynamic Weighted Majority	learner: NaiveBayes
OCB [80]	Online Coordinate Boosting	learner: HoeffdingTree ensembleSize: 10
AWE [81]	Accuracy Weighted Ensemble	learner: HoeffdingTree ensembleSize: 10 ensembleBuffer: 30
AUE2 [82]	Accuracy Updated Ensemble 2	learner: HoeffdingTree ensembleSize: 10

4.3. Experiment 1: Comparison with rule-based classifiers

In our first experiment we compared ERulesD^{2}S with state-of-the-art Very Fast Decision Rules (VFDR) [14], its modification using Naive Bayes component [14], and G-eRules [16], a rule-based classifier using the Gaussian distribution as the heuristic for building rule terms on continuous attributes. These three methods are considered the most efficient rule-based classifiers for drifting data streams. Previous works showed that they are able to provide an interpretable classification model that can quickly update its structure with new incoming instances and adapt to changes in the stream [15]. VFDR claims to offer fast and accurate rule induction mechanism and high robustness to concept drifts. Therefore, they are a natural reference for our ERulesD^{2}S. Results over 30 data stream benchmarks with respect to accuracy, Friedman rank, number of rules, number of conditional clauses, memory, and time complexities are reported in Table 3, while outcomes of Holm and Shaffer post-hoc tests are in Table 4.

When analyzing the results, one can clearly see that ERulesD^{2}S returns accuracy superior to standard VFDR and G-eRules methods on all 30 benchmarks. The modified VFDR$_{NB}$ performs significantly better than VFDR, yet ERulesD^{2}S is still able to outperform it on 26 benchmarks. Additionally, on three benchmarks (BNG-zoo, Hyperplane-drift-noise, and SEA-drift-noise) the difference between these methods is less than 1% in favor of VFDR$_{NB}$. Only on LED benchmark the difference is bigger - 2.88%. The superiority of the proposed method is further verified by Friedman, Holm, Shaffer, and Bonferroni-Dunn 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 observed to obtain much better accuracy than the other rule-based classifiers in the RBF datasets, especially under the presence of concept drifts. Additionally, VFDR tends 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 accuracy they ignore minority classes, not covering them with any rule, which would prevent experts to extract knowledge from these classes. Moreover, G-eRules shows unstable behavior by learning an excessive number of rules and conditions, which not only overfits the data, decreasing the accuracy, but it also drops the advantage of the interpretability. Finally, 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, including datasets with drift and noise.

Let us take closer look on the performance of discussed methods on specific benchmarks. Figure 8 presents prequential and per-window accuracies, train time, and memory consumption for Agrawal-F1-F10-drift 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 9) and CovType (see Figure 10) data streams - they confirm the observed behavioral patterns of discussed methods.

Table 3: Comparison between ERulesD^{2}S and state-of-the-art rule-based streaming classifiers.

Accuracy and number rules	VFDR		VFDR$_{NB}$		G-eRules		ERulesD^{2}S	
	Acc	Rules	Acc	Rules	Acc	Rules	Acc	Rules
Electricity	69.87	36	70.29	36	74.49	65	**76.77**	**10**
Shuttle	88.40	**3**	96.06	**3**	78.91	91	**99.77**	35
DowJones	67.71	**14**	73.36	**14**	4.76	10781	**85.26**	150
CovType	60.32	53	75.58	53	67.79	692	**79.81**	**35**
IntelLabSensors	4.68	**84**	7.47	**84**	2.59	3071	**98.75**	290
BNG-bridges	43.23	955	67.08	955	55.29	2307	**67.82**	**30**
BNG-zoo	54.47	251	**91.48**	251	77.22	320	90.90	**35**
RBF	80.63	392	95.63	392	34.71	**12**	**98.01**	20
RBF-drift	66.90	93	88.35	93	32.77	**13**	**89.89**	20
LED	41.16	**26**	**73.75**	**26**	51.14	70	70.87	50
Hyperplane	77.32	69	82.49	69	49.94	**11**	**84.13**	20
RandomTree	69.73	114	81.06	114	56.55	242	**87.00**	**10**
RandomTree-drift	72.35	54	78.98	54	29.22	410	**81.39**	**10**
SEA-drift	79.73	385	85.28	385	55.94	86	**86.27**	**10**
Waveform	63.88	100	75.84	100	33.28	**4**	**80.72**	15
STAGGER	99.91	**4**	**100**	**4**	**100**	5	**100**	10
Agrawal-F1-F10-drift	71.63	545	73.66	545	62.83	441	**87.55**	**10**
Agrawal-recurring-drift	67.93	675	70.65	675	54.77	378	**87.74**	**10**
Agrawal-F10-F1-drift	74.36	639	77.03	639	61.02	412	**89.20**	**10**
Agrawal-drift	73.74	754	77.54	754	66.12	570	**89.00**	**10**
Agrawal-random-drift	72.76	521	75.41	521	54.92	428	**87.38**	**10**
AssetNegotiation-F2	90.38	301	94.56	301	94.73	244	**94.89**	**10**
AssetNegotiation-F3	87.39	450	93.09	450	93.88	252	**94.81**	**10**
AssetNegotiation-F4	85.93	506	93.19	506	93.99	326	**94.69**	**10**
Sine	91.18	149	92.81	149	67.42	113	**93.56**	**10**
RBF-drift-noise	36.85	48	47.09	48	25.87	**7**	**47.41**	10
Hyperplane-drift-noise	54.11	32	**56.37**	32	49.98	**5**	55.99	20
SEA-drift-noise	51.77	152	**51.95**	152	51.12	22	51.83	**10**
CovType-noise	59.39	87	62.78	87	52.50	571	**64.90**	**35**
Waveform-drift-noise	41.47	55	45.56	55	33.27	**4**	**47.77**	15
Average accuracy	66.64 $\pm$ 2.52		75.15 $\pm$ 1.13		55.57 $\pm$ 0.81		**82.14** $\pm$ 0.25	
Friedman ranks	3.29		2.00		3.55		**1.16**	
Average number of rules	252 $\pm$ 67		252 $\pm$ 67		732 $\pm$ 17		**31** $\pm$ 0	
Average number of clauses	211 $\pm$ 115		211 $\pm$ 115		893 $\pm$ 51		**171** $\pm$ 10	
Avg. Train time (s)	0.691 $\pm$ 0.885		0.668 $\pm$ 0.881		**0.157** $\pm$ 0.043		0.373 $\pm$ 0.007	
Avg. Test time (s)	0.007 $\pm$ 0.003		0.017 $\pm$ 0.024		**0.011** $\pm$ 0.001		0.020 $\pm$ 0.001	
Avg. RAM-Hours	5.9E-2 $\pm$ 9.0E-2		6.4E-2 $\pm$ 8.3E-2		**5.8E-5** $\pm$ 1.1E-6		3.7E-4 $\pm$ 1.7E-6	

Table 4: Holm and Shaffer's tests for comparison between ERulesD^{2}S and rule-based classifiers. Symbol '$>$' stands for situation in which ERulesD^{2}S is statistically superior.

Hypothesis	Holm p-value	Shaffer p-value
ERulesD^{2}S vs. VFDR	$>$ (0.0000)	$>$ (0.0000)
ERulesD^{2}S vs. VFDR$_{NB}$	$>$ (0.0124)	$>$ (0.0124)
ERulesD^{2}S vs. G-eRules	$>$ (0.0000)	$>$ (0.0000)

Let us take a look on the number of rules generated by each method. While both VFDR and G-eRules grow their rule-base according to the incoming data, ERulesD^{2}S keeps 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. Moreover, G-eRules produces an excessive number of rules for DowJones, IntelLabSensors, and BNG-bridges datasets. 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 large number of rules, e.g., 268 for RBF-drift-recurring or 639 for Agrawal-F10-F1, or 441 in the case of G-eRules, 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 for appearing drifts, while maintaining a smaller number of conditions then better 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.

This experiment investigates the performance of ERulesD^{2}S and its relationship to state-of-the art rule-based classifiers in data stream mining. Comparison with decision trees is not considered here, since their interpretability and expressiveness are smaller as compared to rules [20]. 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 computational complexity of genetic programming 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 and method, please refer to the associated webpage[2].

The user can inspect and assess the decision rules at any time during the evolution of the algorithm and stream. For the sake of illustrating the interpretability of the rules obtained by the model, we extracted exemplary rules from the Electricity, IntelLabSensors, and Agrawal datasets, which are shown in Figures 4, 5, and 6 respectively. Interestingly, one may access the code for the Agrawal generator and verify that our rule learner is effectively capable of learning the data distribution, since the function F1 of the generator produces data for which the class is group A if age $<$ 40 or age $>$ 60, class group B otherwise. This shows the advantages of genetic programming for learning attribute–value comparisons on relevant features and selecting decision boundaries.

[2]http://go.vcu.edu/ERulesD2S

IF ((nswdemand < 0.208) **OR** (period < 0.196) **OR** (nswprice < 0.067)) **THEN** (class = DOWN)
IF (((period > 0.264) **OR** (nswdemand > 0.275)) **AND** (nswprice > 0.066)
AND (nswdemand > 0.184)) **THEN** (class = UP)

Figure 4: Rules for Electricity dataset: predict evolution of electricity prices DOWN or UP.

IF ((light > 51.846) **AND** (humidity < 18.175)) **THEN** (class = 4)
IF ((voltage > 2.602) **AND** (light > 116.648)) **THEN** (class = 46)
IF ((voltage < 2.041) **AND** (temperature < 119.644)) **THEN** (class = 54)

Figure 5: Rules for IntelLabSensors dataset: predict the sensor (out of 58) providing the measures.

IF ((age > 60.668) **OR** (age < 38.515)) **THEN** (class = group A)
IF ((age < 61.616) **AND** (age > 38.703)) **THEN** (class = group B)

Figure 6: Rules for Agrawal F1 generator: predict the group A or B.

4.4. Experiment 2: Comparison with other classifiers

In the previous experiment we showed that ERulesD^{2}S is able to outperform the best rule-based classifier for data streams. Moreover, 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, while offering a trade-off for their interpretability. Our experiments show that the trade-off holds for data streams. When analyzing Bonferroni-Dunn test (see Figure 7) that compares all tested 17 classifiers, we can see that both VFDR, VFDR$_{NB}$, and G-eRules 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 non-interpretable classifiers. Results over 30 data stream benchmarks with respect to accuracy, Friedman rank, memory and time complexities are reported in Table 5, while outcomes of Holm and Shaffer post-hoc tests are given in Table 6 and of Bonferroni-Dunn test in Figure 7.

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 bridges a gap, no longer forcing a choice between obtaining either valuable and readable decision rules or high predictive accuracy.

Analyzing the results one may see that ERulesD^{2}S is highly comparable to all classifiers. This is especially important, as we had included eight ensemble models, which are considered to be among the most efficient ones for data stream classification [40]. Statistical analysis presented in Table 6 and Figure 7 proves that ERulesD^{2}S never returns statistically significantly inferior results over multiple datasets. Lack of statistical differences between ERulesD^{2}S and most methods is actually a favorable outcome. The simple fact that our rule-based classifier is able to return accuracy on par (and in some cases even better) with ensemble classifiers shows how effective the proposed method is. Furthermore, many of the used 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. Therefore, ERulesD^{2}S demonstrates its ability to quickly and efficiently capture any changes in the stream and swiftly adapt to it.

Table 5: 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	OCB	AWE	AUE2	ERulesD^{2}S
Electricity	**76.89**	70.98	70.59	70.90	66.88	76.06	75.99	72.08	59.31	70.98	74.50	70.72	76.55	76.77
Shuttle	98.52	90.02	99.26	99.23	98.49	99.13	**99.80**	90.24	93.79	89.91	74.21	97.66	98.96	99.77
DowJones	69.39	61.35	78.79	78.79	**89.35**	80.82	84.21	66.19	2.74	61.35	6.03	21.39	18.55	85.26
CovType	86.22	60.04	87.89	87.80	59.56	84.26	**90.11**	76.21	69.97	71.96	71.29	80.03	86.68	79.81
IntelLabSensors	42.26	3.75	98.35	98.42	98.65	97.73	97.79	87.55	1.82	87.45	3.75	3.84	3.63	**98.75**
BNG-bridges	68.27	**74.31**	70.46	70.46	56.01	51.59	43.12	44.19	68.59	70.22	20.44	46.33	62.61	67.82
BNG-zoo	92.47	**92.84**	92.67	92.67	89.63	92.35	87.04	84.61	90.23	91.74	58.67	69.77	90.88	90.90
RBF	99.14	91.90	99.17	99.15	98.99	99.80	**99.90**	92.00	86.03	90.45	50.66	94.27	99.78	98.01
RBF-drift	95.60	60.83	98.94	**98.95**	94.92	98.11	98.76	89.04	77.53	79.98	47.46	83.80	97.61	89.89
LED	73.83	73.94	65.83	65.76	73.64	73.94	73.80	67.60	67.84	71.15	17.44	73.94	**73.95**	70.87
Hyperplane	81.56	66.28	71.34	71.27	81.94	87.49	87.00	81.37	81.83	85.70	84.94	86.62	**90.28**	84.13
RandomTree	94.22	72.02	59.78	59.78	94.46	95.48	**97.06**	85.64	62.21	69.17	63.21	80.34	95.24	87.00
RandomTree-drift	91.09	61.00	38.49	38.49	90.82	91.93	91.45	80.05	46.81	57.29	51.92	60.24	**93.50**	81.39
SEA-drift	85.82	84.70	86.89	86.90	88.52	88.14	**89.18**	84.21	85.05	85.01	87.93	86.22	88.36	86.27
Waveform	84.14	80.41	80.13	80.13	83.61	**85.52**	84.97	80.18	80.19	78.39	55.05	81.13	85.33	80.72
STAGGER	99.99	**100**	**100**	**100**	**100**	**100**	**100**	95.04	89.82	**100**	**100**	96.11	**100**	**100**
Agrawal-F1-F10-drift	68.94	59.72	74.35	74.40	84.41	85.90	82.56	83.16	75.24	76.89	78.10	82.06	87.41	**87.55**
Agrawal-recurring-drift	71.09	59.13	72.55	72.52	85.45	86.51	83.58	82.53	68.93	70.87	69.68	82.56	86.97	**87.74**
Agrawal-F10-F1-drift	76.59	59.12	75.08	75.10	87.63	86.42	84.21	82.35	76.48	76.98	64.30	83.47	88.58	**89.20**
Agrawal-drift	78.35	58.56	72.01	72.00	87.74	85.76	84.96	83.15	72.98	74.76	79.60	82.92	88.11	**89.00**
Agrawal-random-drift	81.86	62.20	71.54	71.70	84.49	85.49	83.61	82.47	71.41	72.69	82.55	81.91	84.67	**87.38**
AssetNegotiation-F2	94.87	92.36	88.92	88.90	94.87	**94.89**	94.76	90.36	92.70	92.11	94.69	94.84	94.88	**94.89**
AssetNegotiation-F3	94.79	92.20	85.82	85.89	94.78	**94.81**	94.65	90.27	90.94	91.56	94.60	94.46	94.80	**94.81**
AssetNegotiation-F4	94.68	92.53	87.92	87.92	94.68	**94.72**	94.47	90.21	91.48	92.06	94.48	94.43	94.51	94.69
Sine	99.43	83.34	92.29	92.28	99.31	99.58	**99.77**	93.48	83.22	81.93	99.02	92.25	99.53	93.56
RBF-drift-noise	50.91	24.35	49.74	49.74	50.49	53.27	**53.84**	45.31	33.17	42.85	24.35	44.87	52.89	47.41
Hyperplane-drift-noise	55.31	55.72	52.17	52.17	54.49	55.80	54.32	51.21	53.36	**56.48**	55.72	54.28	55.08	55.99
SEA-drift-noise	51.98	**52.35**	50.21	50.21	51.95	51.77	51.12	51.54	50.84	52.29	**52.35**	51.20	51.82	51.83
CovType-noise	70.32	61.07	64.42	64.19	58.89	69.57	**75.24**	60.77	44.08	63.24	60.51	67.30	71.86	64.90
Waveform-drift-noise	47.85	41.44	41.96	41.96	47.73	**49.50**	47.07	41.75	45.28	45.77	27.16	47.50	49.07	47.77
Avg. Accuracy	79.21 ± 1.50	67.95 ± 0.33	75.92 ± 0.80	75.92 ± 0.06	81.41 ± 0.66	**83.21** ± 0.17	82.81 ± 0.20	76.83 ± 0.34	67.13 ± 0.08	75.04 ± 0.31	61.49 ± 0.23	72.88 ± 0.22	78.96 ± 0.22	82.14 ± 0.25
Friedman ranks	6.20	9.78	8.93	8.97	5.98	**3.52**	4.72	9.63	11.30	9.02	9.92	8.33	4.15	4.55
Avg. Train time (s)	0.039 ± 0.097	**0.017** ± 0.001	0.035 ± 0.036	0.724 ± 0.381	0.068 ± 0.153	0.350 ± 0.500	1.551 ± 2.601	0.047 ± 0.062	6.192 ± 17.651	0.021 ± 0.036	0.091 ± 0.116	0.292 ± 0.331	0.117 ± 0.183	0.373 ± 0.011
Avg. Test time (s)	0.010 ± 0.014	0.009 ± 0.006	0.801 ± 0.368	0.691 ± 0.362	**0.002** ± 0.004	0.018 ± 0.026	0.019 ± 0.022	0.018 ± 0.024	0.454 ± 0.448	0.018 ± 0.039	0.028 ± 0.038	0.033 ± 0.047	0.022 ± 0.030	0.020 ± 0.002
Avg. RAM-Hours	1.0E-4 ± 5.0E-4	7.6E-5 ± 2.3E-8	5.1E-4 ± 4.8E-5	8.8E-4 ± 5.5E-5	9.6E-5 ± 3.7E-5	6.6E-3 ± 1.4E-3	6.6E-2 ± 2.4E-2	1.7E-4 ± 2.5E-5	5.7E-1 ± 1.4E-1	**2.3E-7** ± 1.1E-7	1.3E-3 ± 4.3E-4	4.3E-4 ± 1.1E-4	7.0E-4 ± 3.8E-4	1.8E-4 ± 5.2E-6

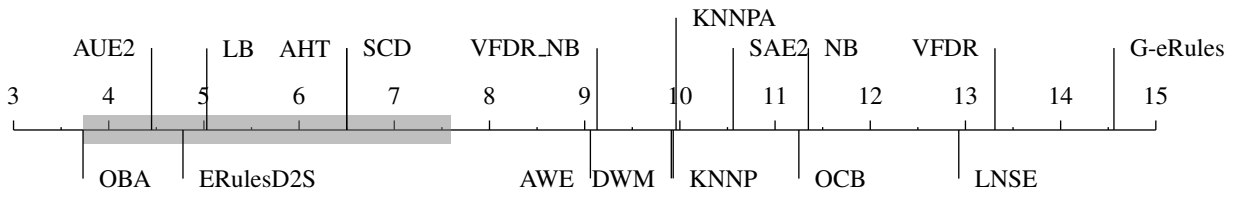

Figure 7: Bonferroni-Dunn test for all examined classifiers.

Table 6: Holm and Shaffer's tests for comparison between ERulesD^{2}S and other classifiers. Symbol '$>$' stands for situation in which ERulesD^{2}S is statistically superior and '$=$' for when there are no significant differences.

Hypothesis	Holm p-value	Shaffer p-value
ERulesD^{2}S vs. AHT	$=$ (0.1266)	$=$ (0.4899)
ERulesD^{2}S vs. NB	$>$ (0.0000	$>$ (0.0000)
ERulesD^{2}S vs. KNNP	$>$ (0.0000)	$>$ (0.0003)
ERulesD^{2}S vs. KNNPA	$>$ (0.0000)	$>$ (0.0003)
ERulesD^{2}S vs. SCD	$=$ (0.1845)	$=$ (0.6126)
ERulesD^{2}S vs. OBA	$=$ (0.3387)	$=$ (0.9251)
ERulesD^{2}S vs. LB	$=$ (0.8774)	$=$ (1.0000)
ERulesD^{2}S vs. SAE2	$>$ (0.0000)	$>$ (0.0000)
ERulesD^{2}S vs. LNSE	$>$ (0.0000)	$>$ (0.0000)
ERulesD^{2}S vs. DWM	$>$ (0.0000)	$>$ (0.0003)
ERulesD^{2}S vs. OCB	$>$ (0.0000)	$>$ (0.0000)
ERulesD^{2}S vs. AWE	$>$ (0.0005)	$>$ (0.0029)
ERulesD^{2}S vs. AUE2	$=$ (0.7111)	$=$ (1.0000)

ERulesD^{2}S superior performance is backed-up by analysis of detailed behavior of selected methods in Figure 8 (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 displays similar or even better recovery time that methods using external drift detectors [84]. In this case adaptive methods without explicit drift handling (e.g., AHT) 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 10 shows accuracies for CovType stream benchmark. Here no severe drift is present, which allows AHT to display efficient performance as it incrementally learns 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. The unstable behavior 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.

Those two previous observations lead to a 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 has showed that most popular methods are designed for specific problems in mind - ones performing well 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.

The experimental study showed that ERulesD^{2}S returns 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, which are real-life streams with concept drift and large number of classes. We can see that ERulesD^{2}S displays an excellent performance in both cases, while many reference classifiers fail to achieve satisfactory accuracy. This can be explained by the presence of mixed types of drifts within these datasets, large 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 the 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 AUE2 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.

Consequently, we can 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 to be considered as a good all-purpose method for mining non-stationary streams.

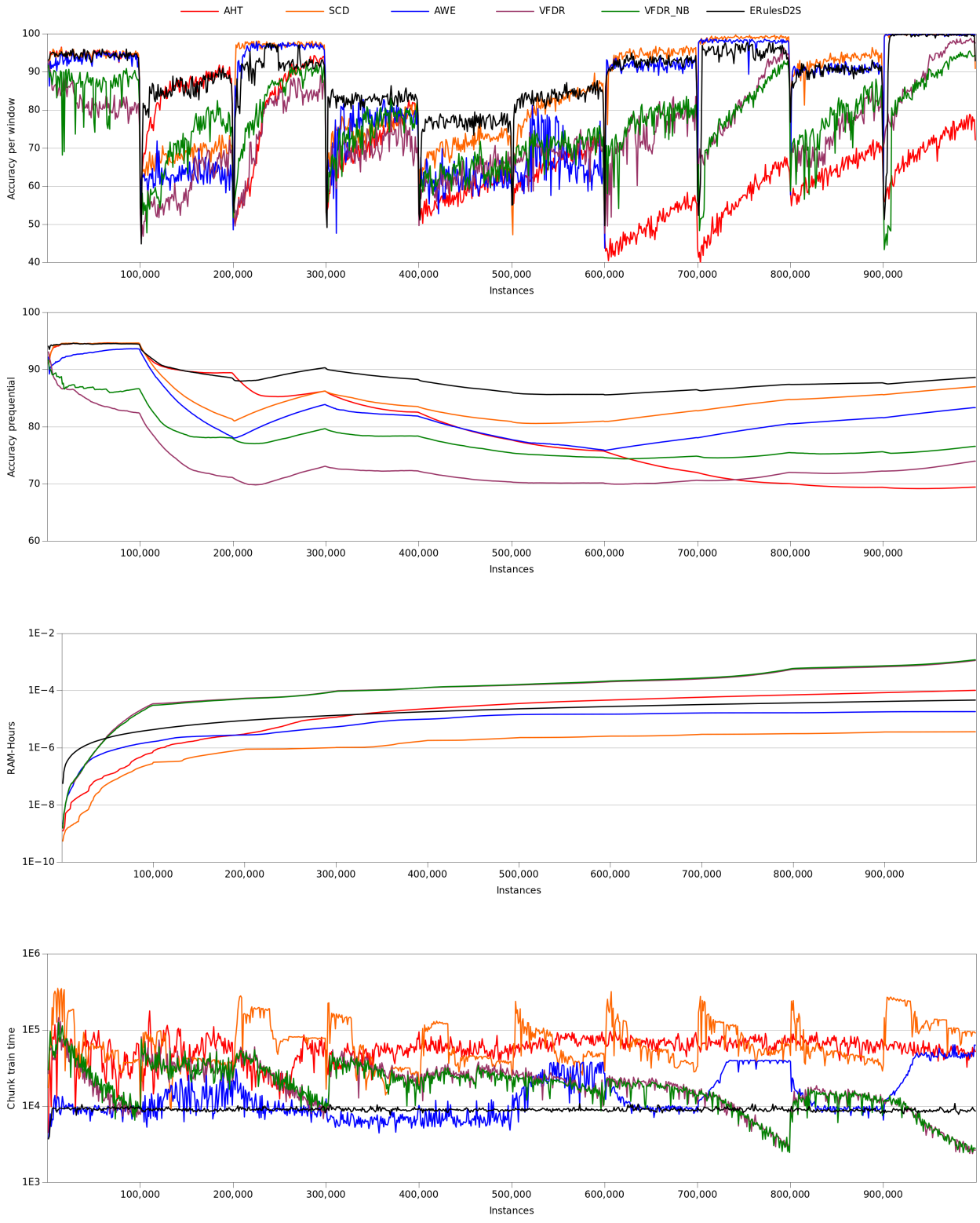

Figure 8: Plots of accuracy per window, prequential accuracy, memory consumption and update time of selected classifiers for AgrawalGenerator drifts from F1 to F10 data stream.

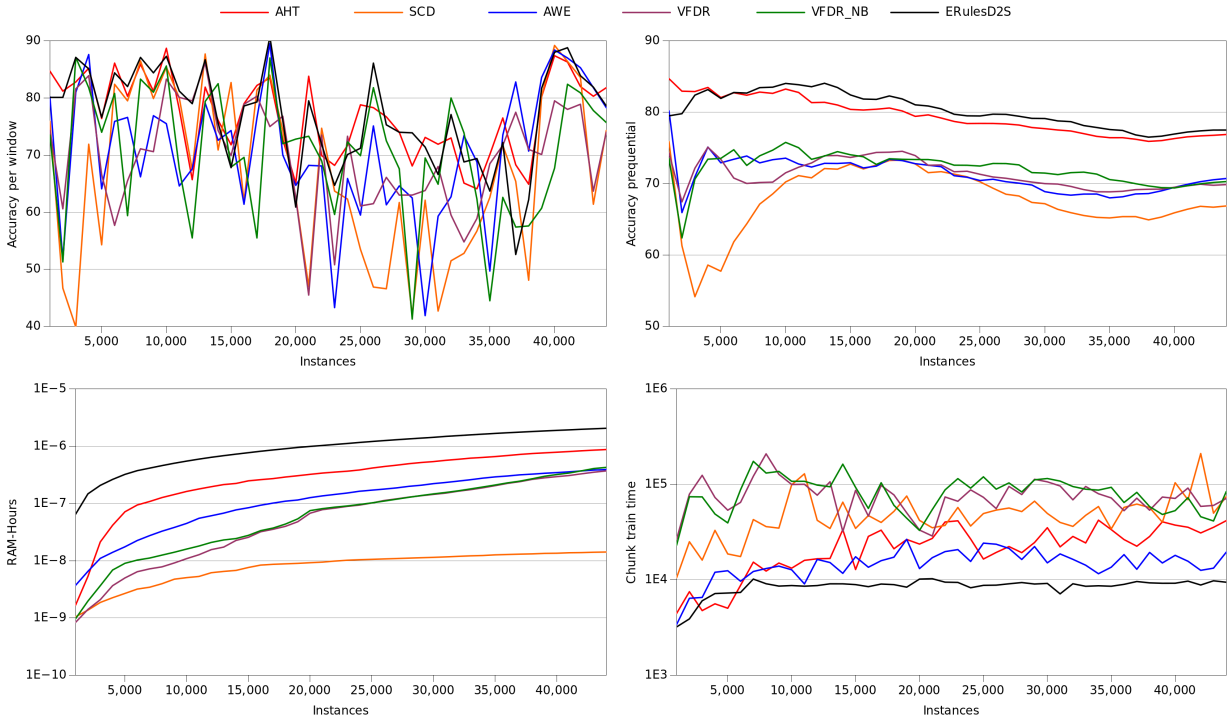

Figure 9: Plots of accuracy per window, prequential accuracy, memory consumption and update time of selected classifiers for Electricity data stream.

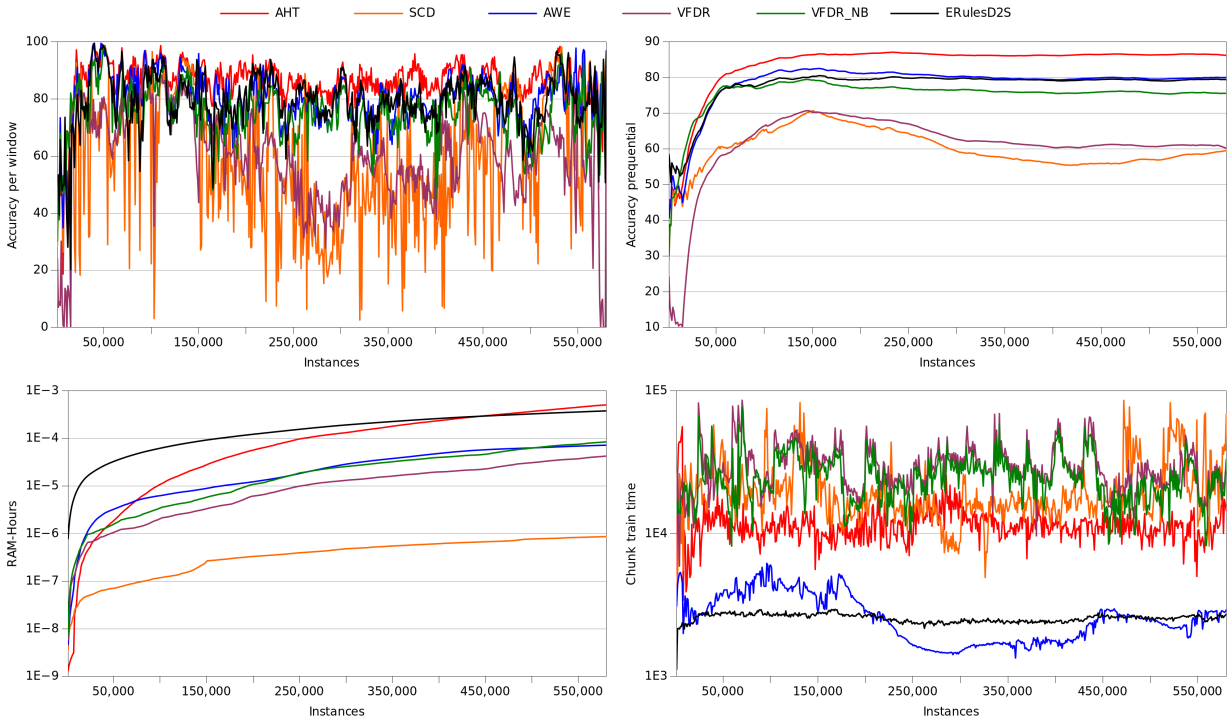

Figure 10: Plots of accuracy per window, prequential accuracy, memory consumption and update time of selected classifiers for CovType data stream.

4.5. Experiment 3: Stability, accuracy, and runtime of ERulesD^{2}S

In the third experiment we wanted to evaluate the stability and sensitivity of ERulesD^{2}S to different parameter settings and examine how they will influence the accuracy and computational complexity. In the 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 evaluated the influence of the following parameters on behavior of ERulesD^{2}S: population size, number of generations, number of rules per class, number of windows, 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 solutions by maximizing the 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. 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 the 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 3.3 for details). The number of windows controls how many data instances we keep in memory for our sliding and sampling 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 windows are still being used to evolve classification rules. The influence of this parameter should be similar to the number of windows, as it also manages the ratio of adaptation to novel concepts. We conducted a thorough sensitivity analysis of ERulesD^{2}S using 20 different parameter settings (selected via grid-search procedure) on all datasets from the experimental evaluation. Due to space limitations, we present in Table 7 the results for the 12 best settings averaged over all datasets.

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 1%, which shows that our method displays stable performance and is not highly sensitive to selected parameters. Robustness 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

Table 7: Accuracy–Time performance for ERulesD^{2}S (averaged results over all data stream benchmarks).

Windows	Sampling factor	Rules	Pop	Gen	Accuracy	Train Time	Test Time	RAM Hours
10	0.25	10	25	50	82.36	0.663	0.038	4.7E-4
10	0.5	10	25	50	82.41	0.656	0.036	3.2E-4
10	0.25	5	25	50	81.91	0.356	0.021	2.3E-4
10	0.5	5	25	50	82.08	0.350	0.018	1.6E-4
5	0.25	10	25	50	82.37	0.578	0.034	3.9E-4
5	0.5	10	25	50	82.59	0.596	0.034	2.8E-4
5	0.25	5	25	50	82.22	0.343	0.017	2.1E-4
5	**0.5**	**5**	**25**	**50**	**82.25**	**0.338**	**0.020**	**1.8E-4**
5	0.5	5	50	50	82.28	0.651	0.020	4.2E-4
5	0.5	5	15	25	81.66	0.208	0.017	6.4E-5
5	0.5	10	15	25	82.20	0.384	0.034	1.6E-4
5	0.5	3	25	50	81.38	0.199	0.010	7.4E-5

the variance in accuracy is low, the variance in time and memory consumption is much more significant. Therefore, our method proves to be flexible, 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. This configurability is an advantage not available in other data stream mining methods.

For the experimental comparison in with other classifiers, we selected the following parameters (bold row in Table 7): population size = 25, number of generations = 50, window size = 5, number of rules per class = 5, and sampling factor = 0.5. This setting offers the best balance in terms of accuracy and computational complexity trade-off.

4.6. Experiment 4: Scalability to high-dimensional data streams

Most of works in data stream mining use benchmarks characterized by a small number of features. However, in contemporary applications the streaming data is accompanied by a high dimensionality of the feature space. Curse of dimensionality is a well-known challenge for various machine learning algorithms and one can assume that it will take place also in the data stream mining scenario. Therefore, when developing new streaming classifiers, it is crucial to evaluate their robustness and scalability to increasing number of features.

In order to evaluate the scalability of ERulesD^{2}S, we have designed an experiment with four artificial data streams created using RBF, RBF-drift, Hyperplane-drift-noise, and RandomTree-drift generators. They follow the same parameters as described in Table 1, but we modify their dimensionality to include $\{10, 100, 1000, 10000\}$ features. This will allow us to gain an insight into the effects of increasing size of the feature space on effectiveness of examined classifiers, as well as on their scalability with respect to computational requirements.

Relationships between the prequential accuracy and increasing dimensionality of the feature space are depicted in Figure 11, where we independently compare ERulesD^{2}S with other rule-based classifiers, single classifiers, and ensembles. The obtained training (update) and testing times (seconds) per 1000 instances window are presented in Table 8.

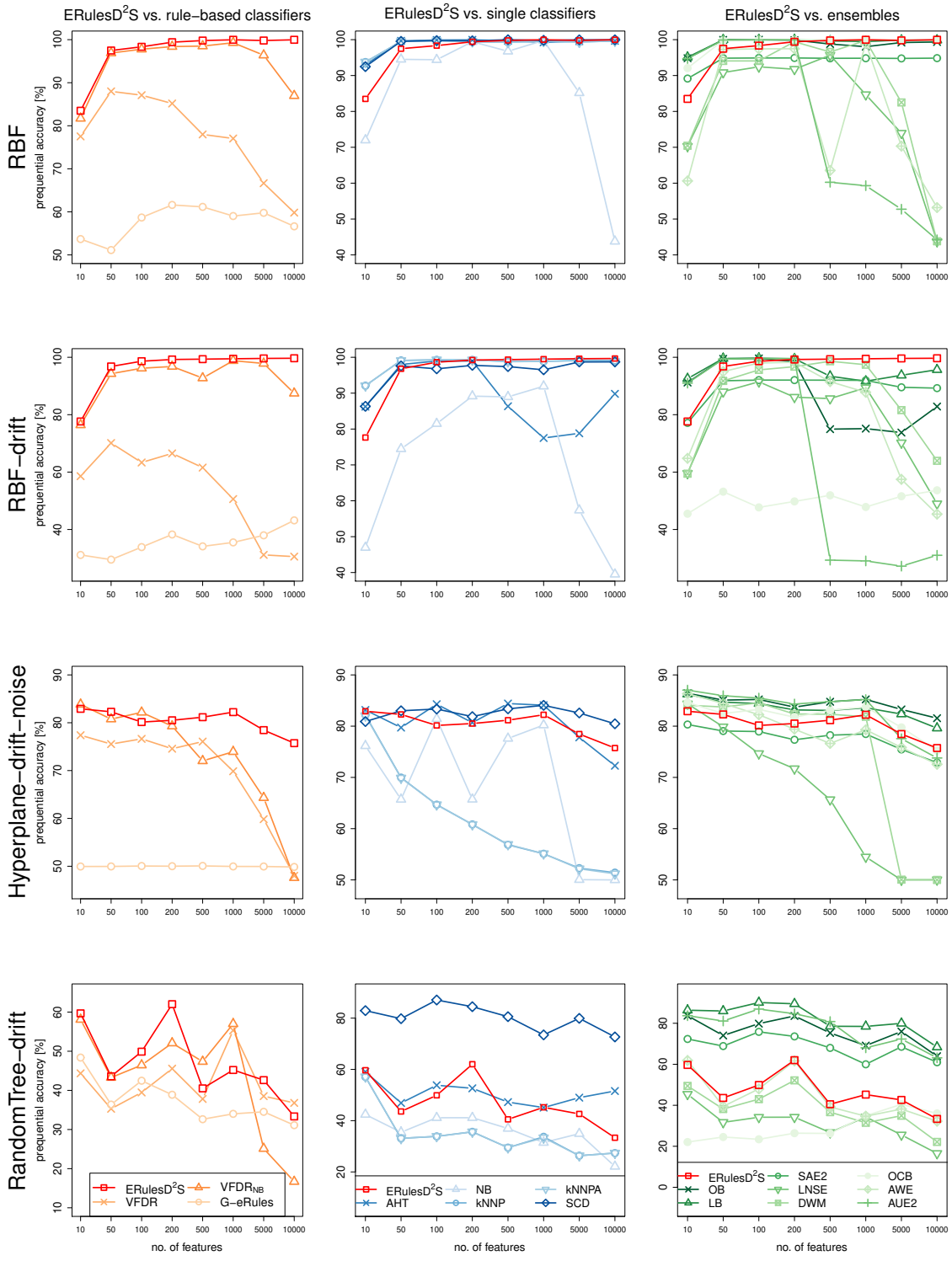

Figure 11: Relationship between prequential accuracy and increasing dimensionality of data streams for ERulesD^{2}S and reference classifiers.

Table 8: Training and testing times (s) per 1000 instances with respect to increasing dimensionality of data streams.

		RBF				RBF-drift				Hyperplane-drift-noise				RandomTree-drift			
	Features	10	100	1000	10000	10	100	1000	10000	10	100	1000	10000	10	100	1000	10000
AHT	Train	0.02	0.13	1.42	33.29	0.01	0.10	2.88	20.19	0.05	0.30	3.51	87.35	0.01	0.14	3.63	26.46
	Test	0.01	0.04	0.22	9.73	0.00	0.04	0.87	4.51	0.01	0.06	0.79	20.48	0.00	0.03	0.42	2.52
NB	Train	0.00	0.00	0.05	0.68	0.00	0.00	0.05	0.89	0.00	0.00	0.05	0.68	0.00	0.00	0.05	0.67
	Test	0.00	0.02	0.41	4.57	0.00	0.04	0.79	9.33	0.00	0.02	0.44	4.21	0.00	0.02	0.51	5.81
KNNP	Train	0.04	0.04	0.09	0.12	0.05	0.05	0.09	0.12	0.04	0.04	0.09	0.11	0.03	0.03	0.10	0.12
	Test	0.47	4.75	74.84	935.44	0.39	3.35	69.93	798.00	0.58	11.52	200.39	2426.03	0.48	9.72	198.37	2285.41
KNNPA	Train	0.39	3.12	78.32	923.04	0.51	4.87	68.25	827.83	0.60	11.35	215.24	2517.25	0.48	8.77	194.02	2278.97
	Test	0.34	3.08	78.71	897.83	0.48	4.84	68.54	819.24	0.56	11.23	215.39	2517.16	0.43	8.70	191.04	2251.41
SCD	Train	0.10	0.22	1.37	5.78	0.02	0.10	2.01	4.86	0.19	0.38	2.33	5.39	0.03	0.52	2.26	4.49
	Test	0.00	0.01	0.14	0.66	0.00	0.02	0.66	0.66	0.00	0.01	0.12	0.36	0.00	0.01	0.12	0.36
OBA	Train	0.68	1.43	8.62	35.87	0.21	0.95	12.19	36.66	0.20	3.25	17.77	40.21	0.63	3.94	18.38	26.08
	Test	0.02	0.07	1.39	8.94	0.03	0.12	2.15	8.40	0.01	0.11	1.54	4.91	0.03	0.09	1.68	2.95
LB	Train	5.11	4.55	17.76	36.20	1.83	3.97	18.25	35.37	1.63	9.01	60.40	116.28	3.77	8.83	30.97	47.43
	Test	0.02	0.07	1.39	4.76	0.02	0.05	0.64	2.47	0.02	0.13	2.06	3.10	0.02	0.06	0.94	1.64
SAE2	Train	0.06	0.28	2.39	20.34	0.02	0.56	10.08	35.19	0.06	0.43	10.20	70.88	0.03	0.13	1.95	25.67
	Test	0.02	0.10	0.91	4.87	0.01	0.27	4.89	16.13	0.02	0.12	3.13	24.47	0.01	0.03	0.52	4.28
LNSE	Train	4.14	38.20	352.65	1548.45	7.36	69.88	480.56	2111.25	4.04	36.70	348.69	1474.26	4.41	46.62	546.68	1645.37
	Test	0.04	2.87	350.28	0.00	0.07	17.18	478.31	55.95	0.08	1.13	342.89	0.00	0.03	1.04	0.00	0.00
DWM	Train	0.01	0.10	0.45	35.44	0.02	0.15	4.48	61.25	0.01	0.11	2.76	35.28	0.02	0.14	3.92	44.62
	Test	0.01	0.09	0.40	26.61	0.02	0.14	4.15	51.74	0.01	0.09	2.33	26.76	0.01	0.12	3.01	33.21
OCB	Train	0.07	0.42	10.71	61.25	0.11	0.80	16.44	181.09	0.07	0.48	11.42	114.43	0.10	0.67	13.14	173.52
	Test	0.02	0.18	4.89	18.08	0.04	0.37	6.84	77.03	0.01	0.09	2.29	28.80	0.03	0.18	3.68	12.88
AWE	Train	0.04	0.82	28.35	145.26	0.26	2.52	67.74	471.38	0.17	1.41	31.27	126.81	0.23	2.42	23.44	363.55
	Test	0.00	0.15	1.97	0.15	0.04	0.37	7.28	51.14	0.02	0.17	3.33	3.69	0.02	0.23	0.62	7.01
AUE2	Train	0.13	0.79	7.84	67.00	0.37	1.84	8.51	104.21	0.23	1.92	20.90	179.60	0.26	1.99	19.66	202.00
	Test	0.03	0.15	0.01	0.02	0.09	0.44	0.08	1.11	0.04	0.24	2.64	30.79	0.05	0.21	3.24	10.17
VFDR	Train	0.40	1.09	15.04	101.12	0.45	3.40	48.37	250.62	0.38	3.51	19.00	178.13	0.21	2.06	20.32	96.33
	Test	0.02	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$VFDR_{NB}$	Train	0.46	2.94	12.87	108.73	0.41	3.43	53.44	237.28	0.33	2.76	20.18	169.22	0.20	2.43	21.60	95.19
	Test	0.02	0.03	0.81	11.41	0.02	0.08	1.91	20.25	0.01	0.04	1.06	10.58	0.01	0.06	1.64	17.48
G-eRules	Train	0.01	0.11	3.43	76.44	0.02	0.22	5.29	45.11	0.02	0.21	3.65	40.70	0.11	1.36	24.59	266.81
	Test	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03
ERulesD^{2}S	Train	0.12	0.18	0.95	15.85	0.21	0.33	1.72	22.34	0.20	0.32	1.69	33.71	0.21	0.30	1.06	21.21
	Test	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03

For three benchmarks (RBF, RBF-drift, Hyperplane-noise-drift) the proposed ERulesD^{2}S offers a stable performance, regardless of the dimensionality of the feature space. While most of classifiers are destabilized by an increasing dimensionality, ERulesD^{2}S remains insensitive to such changes. This beneficial property can be contributed to the underlying genetic programming rule induction that performs an implicit feature selection. Only the most relevant features are used for inducting rules and thus we are able to efficiently avoid the curse of dimensionality. Reference classifiers do not have any coping mechanism and tend to be strongly affected by a variation in the number of dimensions. Therefore, for such scenarios they need to be coupled with appropriate feature selection methods [85]. However, this will lead to additional problems related to choosing a proper feature selection algorithm, as well as increasing the overall computational complexity of the entire data stream mining process.

It is interesting to observe that for RBF-based benchmarks, ERulesD^{2}S improves its performance when the number of features increases. In previous experiments, we have observed that ERulesD^{2}S returns inferior performance to many reference classifiers on RBF benchmarks (we used 10-dimensional ones). However, when the size of the feature space is increased to 50 fea-

tures and beyond, ERulesD^{2}S displays one of the best prequential accuracies among all examined methods. This can be explained by the fact that ERulesD^{2}S inducts enclosing rules (decision boundaries) per class. RBF benchmarks are hard to be properly described using such structures. With increasing dimensionality, ERulesD^{2}S can perform a much better description of such irregular instance distributions, thus increasing its generalization capability.

The only dataset for which ERulesD^{2}S did not returned a stable performance is RandomTree-drift. Such fluctuations in accuracy can be explained by a purely random nature of this generator, as rule-based classifiers have difficulties in capturing properties of such problems. This is one of few benchmarks in which ERulesD^{2}S did not returned a satisfactory performance in a low-dimensional case. Thus, this learning difficulty is propagated when the number of dimension increases, further augmenting the random nature of this benchmark stream.

When taking into account training and testing times, one can see that ERulesD^{2}S offers impressive speed for both updating itself and classifying new instances. Only SCD and G-eRules offer comparable time requirements. SCD tends to perform slightly worse than ERulesD^{2}S (see Table 5, avg. rank 5.98 vs. 4.55), but offers no interpretability. G-eRules are also an interpretable classifier, but display significantly inferior accuracy than ERulesD^{2}S. Remaining classifiers, especially ensemble-based, are characterized by a significant increase in their computational complexities with growing number of features. This makes their application prohibitive when number of features increase over 100, especially in high-speed real-world applications where little or no latency is allowed. Methods like LB, LNSE, or AUE2 despite their overall accuracy, cannot be used in such scenarios, while ERulesD^{2}S offers up to 350 times faster response.

Additionally, we would like to analyze the scalability of ERulesD^{2}S more in-depth. We selected RBF-drift generator and generated streams with dimensionality of feature space ranging between 10 and 10,000. The obtained results are depicted in Figure 12. One can see that ERulesD^{2}S can easily handle data streams with 10,000 features, while not excessing 16 seconds of update time per batch. Furthermore, this time is a combination of an actual update time and hardware overhead, caused by copying data into GPU's memory. Regardless of the number of features, ERulesD^{2}S is characterized by an excellent and stable accuracy. This allows us to conclude that proposed classifier is highly suitable for learning from high-dimensional and high-speed data streams.

As the final part of this experiment, we want to evaluate the impact of an increasing data stream dimensionality on rule-base classifiers. We are interested in seeing how the rule-base expands with the number of features, as well as what is the size of conditional clauses. Table 9 presents this information over four benchmarks with varying feature space sizes. Results of the number of rules and conditions for VFDR and VFDR$_{NB}$ are identical. Three reference rule-based classifiers show a high variance and complexity in both number and size of rules, which leads to reduced interpretability, low predictive accuracy, and increased computational complexity. ERulesD^{2}S always returns the same number of rules (set as its parameter) and works well with very small rule-base. At the same time, the produced rules are compact, which is indicated by a small number of conditional clauses being generated.

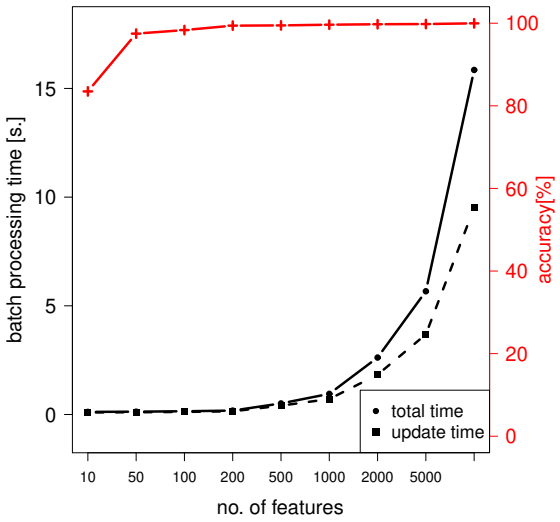

Figure 12: Scalability of ERulesD^{2}S for up to 10,000 features with respect to total update time and actual model training time. The difference between these times stands for copying new data into GPU's memory.

Table 9: Accuracy and complexity of the rule base for rule-based classifiers on high-dimensional datasets.

		Number of rules			Number of conditions			Accuracy			
Dataset	Atts	ERulesD^{2}S	VFDR	G-eRules	ERulesD^{2}S	VFDR	G-eRules	ERulesD^{2}S	VFDR	VFDR$_{NB}$	G-eRules
RBF	10	10	392	28	70	219	54	83.49	77.53	81.71	53.68
	100	10	142	34	70	503	67	98.34	87.10	97.78	58.69
	1000	10	342	55	74	884	109	99.97	77.04	99.28	59.03
	10000	10	368	78	69	2328	143	99.97	59.83	86.97	56.64
RBF-drift	10	20	184	94	140	45	185	77.63	58.58	76.42	31.13
	100	20	244	110	140	251	184	98.63	63.34	96.18	33.86
	1000	20	552	188	149	723	291	99.48	50.60	98.81	35.50
	10000	20	814	95	147	1282	190	99.66	30.55	87.45	43.18
HP-drift-n	10	20	73	21	88	106	42	82.91	77.39	83.83	49.95
	100	20	56	16	88	260	30	80.14	76.63	82.17	50.05
	1000	20	273	19	86	467	32	82.22	69.86	73.94	49.96
	10000	20	338	13	87	853	24	75.73	48.00	47.60	49.85
RT-drift	10	20	114	572	140	262	907	58.85	44.35	58.10	48.38
	100	20	107	397	139	365	1164	49.78	39.47	46.45	42.47
	1000	20	320	314	157	516	937	55.21	55.32	57.00	34.00
	10000	20	369	309	163	1356	1409	43.34	36.80	16.69	31.11

4.7. Experiment 5: Learning from partially labeled data streams

All previous experiments assumed a fully supervised learning scenario. However, in data stream mining one must take into an account the potential limited availability of ground truth (class labels) [86]. Supervised scenario assumes we have access to an oracle that provided class labels upon request. However, in real-life applications, one must employ an annotator (usually a domain expert) that will provide labeling capabilities. Due to massive amount of information arriving rapidly and continuously from data streams, this becomes impossible. First obstacle lies in cost, as it is very unlikely that a company will have budget for labeling millions of instances on regular basis. Second obstacle lies in availability of an annotator, as one cannot expect to get a non-stop coverage. The final obstacle lies in throughput and latency. A human annotator has physical limits on the number of queries to be processed in a given amount of time, and will not return labeling immediately upon seeing an instance. Furthermore, even if we deal with applications where class labels can be obtained at no cost by, e.g., observing the ongoing changes (like in weather or stock market predictions), we still need to cope with the fact that ground truth will not be available right away, due to the label latency.

Access to limited class labels has led to recent advances in active [87] and semi-supervised [88] learning from drifting data streams. Most of them work as simple plug-ins with any base classifier, but enhancing ERulesD^{2}S with these techniques is out of scope of this paper. Instead, we want to evaluate how ERulesD^{2}S can cope with partially labeled data streams. As we used labeled instances in genetic programming to evolve new rules, it is interesting to see how a limited access to such information will influence the evolutionary process.

For this purpose, we have selected four benchmarks: two real-life ones (CovType and IntelLabSensors) and two artificial ones (RBF-drift and RandomTree-drift). Additionally, to evaluate the joint effects of limited access to labeled instances and high-dimensional feature space, RBF-drift benchmark uses 500 features. Details of remaining benchmarks are identical to the ones in Table 1. We evaluate varying ratios of labeled instances in $\{1\%, 5\%, 10\%, 15\%, 20\%\}$, commonplace in semi-supervised learning. They have been selected at random from each batch, with stratified sampling among all classes. This simulates a behavior of any active or semi-supervised learning technique that will provide us with a sparse number of class labels. Obtained results for ERulesD^{2}S and reference classifiers are depicted in Figure 13.

Despite the intuition that the underlying evolutionary process may be strongly affected by a limited access to training instances, we observe a reverse situation. Not only ERulesD^{2}S offers stable performance over all examined sizes of the labeled training set, it also outperforms all other rule-based methods on three out of four benchmarks (the fourth one being RandomTree-drift, which poses difficulty to ERulesD^{2}S as discussed in previous subsections). With respect to the other algorithms, we offer at least a comparable performance, showing that we are capable of delievering similar or better accuracy than many single and ensemble classifiers when access to class labels is limited. This is especially visible for small number of labeled instances (1% and 5%), as well as for a challenging IntelLabSensor data stream. ERulesD^{2}S is able to utilize a small number of training instances in the most efficient way in order to induct effective classification rules. We can see that other rule-based classifiers return inferior performance on partially labeled data streams. Similar trend can be observed by ensembles. In a fully labeled scenario, many ensembles were able to outperform ERulesD^{2}S (what was to be expected). However, when access to class labels is limited, ERulesD^{2}S is able to utilize instances in a significantly more effective manner than most of

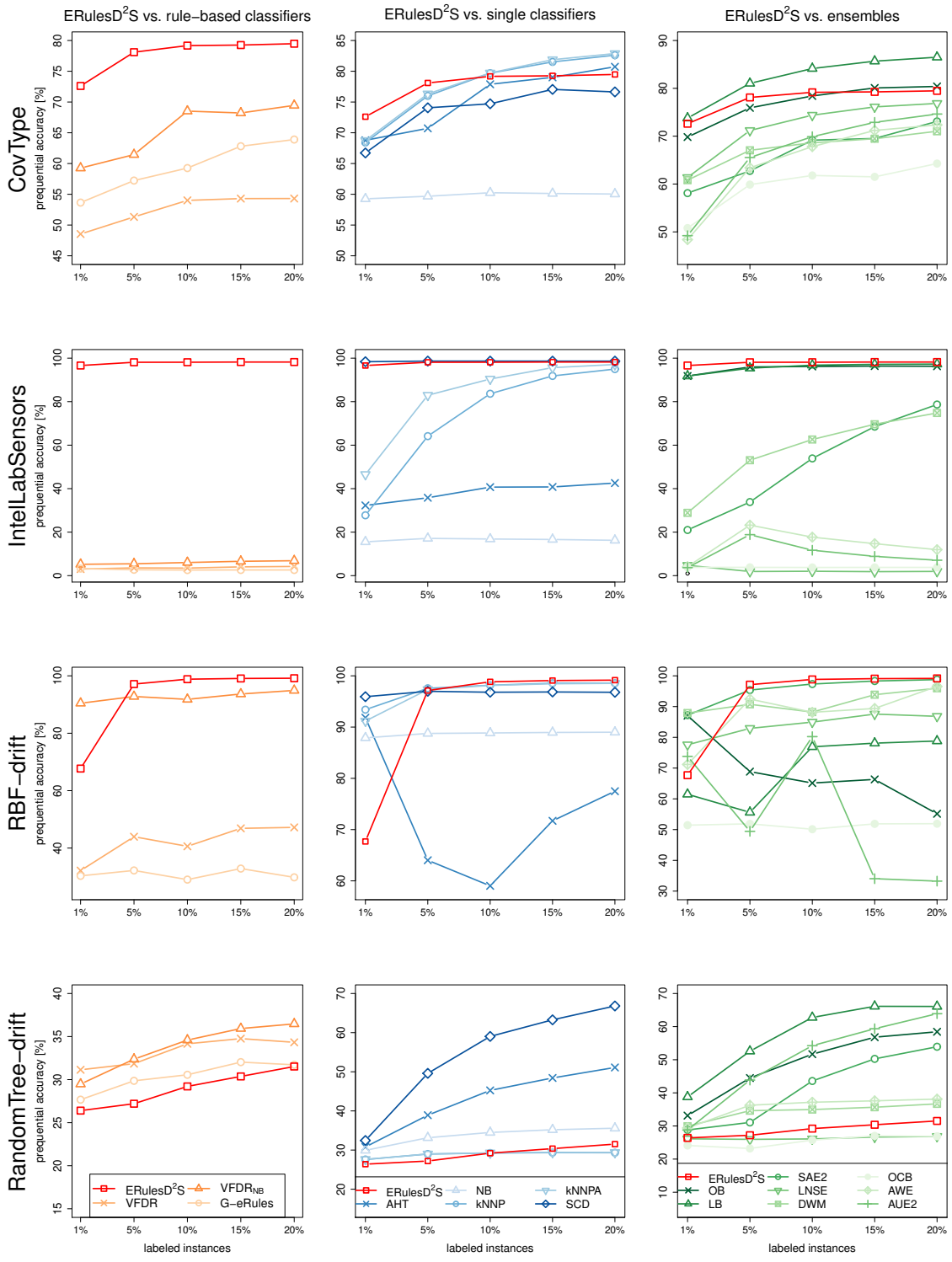

Figure 13: Relationship between prequential accuracy and percentage of labeled instances in partially labeled data streams for ERulesD^{2}S and reference classifiers.

ensemble methods, easily closing the gap in accuracy, and even outperforming many of ensemble learners.

In case of RBF-drift stream with 500 features, we may see that the proposed classifier displays both excellent accuracy and capabilities for learning from small training sets. Therefore, ERulesD^{2}S is insensitive to the phenomenon of simultaneous curse of dimensionality and small training sample size. We conclude that ERulesD^{2}S is an excellent choice for partially labeled data streams, even for extreme scenarios where access to ground truth is limited to as low as 1% of all instances.

5. Conclusions and future works

In this paper we have introduced ERulesD^{2}S: Evolving Rule-based classifier for Drifting Data Streams. It presented 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. Since 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 16 state-of-the-art classifiers on 30 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. Additionally, we showed that ERulesD^{2}S can scale-up efficiently to high-dimensional data streams, while offering very fast update and classification times. Finally, we presented the learning capabilities of ERulesD^{2}S for partially labeled data streams.

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.

Acknowledgments

This research was partially supported by the 2018 VCU Presidential Research Quest Fund and an Amazon AWS Machine Learning Research award.

References

[1] X. Wu, X. Zhu, G. Wu, W. Ding, Data Mining with Big Data, IEEE Trans. on Knowledge and Data Engineering 26 (1) (2014) 97–107.

[2] D. Marron, A. Bifet, 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, 615–620, 2014.

[3] A. Cano, A survey on graphic processing unit computing for large-scale data mining, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8 (1) (2018) e1232.

[4] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesús, J. M. Benítez, 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 4 (5) (2014) 380–409.

[5] J. Gama, Knowledge Discovery from Data Streams, Chapman & Hall/CRC, 2010.

[6] P. Angelov, D. P. Filev, N. Kasabov, Evolving Intelligent Systems: Methodology and Applications, Wiley-IEEE Press, 2010.

[7] M. Sayed-Mouchaweh, E. Lughofer, Learning in Non-Stationary Environments: Methods and Applications, Springer, 2012.

[8] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, T. Radauer, Drift detection in data stream classification without fully labelled instances, in: IEEE International Conference on Evolving and Adaptive Intelligent Systems, 1–8, 2015.

[9] D.-H. Tran, M. M. Gaber, K.-U. Sattler, Change Detection in Streaming Data in the Era of Big Data: Models and Issues, SIGKDD Explorations 16 (1) (2014) 30–38.

[10] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, K. Ghédira, Discussion and review on evolving data streams and concept drift adapting, Evolving Systems 9 (1) (2018) 1–23.

[11] H. G. Miller, P. Mork, From Data to Decisions: A Value Chain for Big Data, IT Professional 15 (1) (2013) 57–59.

[12] P. P. Angelov, X. Zhou, Evolving Fuzzy-Rule-Based Classifiers From Data Streams, IEEE Trans. on Fuzzy Systems 16 (6) (2008) 1462–1475.

[13] M. Pratama, S. G. Anavatti, P. P. Angelov, E. Lughofer, PANFIS: A Novel Incremental Learning Machine, IEEE Trans. on Neural Networks and Learning Systems 25 (1) (2014) 55–68.

[14] J. Gama, P. Kosina, Learning decision rules from data streams, in: International Joint Conference on Artificial Intelligence, vol. 22(1), 1255–1260, 2011.

[15] F. Stahl, M. M. Gaber, M. M. Salvador, eRules: A Modular Adaptive Classification Rule Learning Algorithm for Data Streams, in: 32nd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, 65–78, 2012.

[16] T. Le, F. Stahl, J. B. Gomes, M. M. Gaber, G. D. Fatta, Computationally Efficient Rule-Based Classification for Continuous Streaming Data, in: 34th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, 21–34, 2014.

[17] P. Kosina, J. Gama, Very fast decision rules for classification in data streams, Data Mining and Knowledge Discovery 29 (1) (2015) 168–202.

[18] J. Duarte, J. Gama, A. Bifet, Adaptive Model Rules From High-Speed Data Streams, ACM Trans. on Knowledge Discovery from Data 10 (3) (2016) 30:1–30:22.

[19] E. Lughofer, On-line assurance of interpretability criteria in evolving fuzzy systems Achievements, new concepts and open issues, Information Sciences 251 (2013) 22–46.

[20] T. Le, F. Stahl, M. M. Gaber, J. B. Gomes, G. D. Fatta, On expressiveness and uncertainty awareness in rule-based classification for data streams, Neurocomputing 265 (2017) 127–141.

[21] M. Smith, V. Ciesielski, Adapting to concept drift with genetic programming for classifying streaming data, in: IEEE Congress on Evolutionary Computation, 5026–5033, 2016.

[22] M. I. Heywood, Evolutionary model building under streaming data for classification tasks: opportunities and challenges, Genetic Programming and Evolvable Machines 16 (3) (2015) 283–326.

[23] M. M. Gaber, Advances in data stream mining, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (1) (2012) 79–85.

[24] E. Lughofer, On-line active learning: A new paradigm to improve practical useability of data stream modeling methods, Information Sciences 415 (2017) 356–376.

[25] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (4) (2014) 44:1–44:37.

[26] Z. Zhang, J. Zhou, Transfer estimation of evolving class priors in data stream classification, Pattern Recognition 43 (9) (2010) 3151–3161.

[27] T. T. T. Nguyen, T. T. Nguyen, A. W.-C. Liew, S.-L. Wang, Variational inference based bayes online classifiers with concept drift adaptation, Pattern Recognition 81 (2018) 280–293.

[28] H. L. Hammer, A. Yazidi, B. J. Oommen, On the classification of dynamical data streams using novel Anti-Bayesian techniques, Pattern Recognition 76 (2018) 108–124.

[29] A. Liu, J. Lu, F. Liu, G. Zhang, Accumulating regional density dissimilarity for concept drift detection in data streams, Pattern Recognition 76 (2018) 256–272.

[30] J. Gama, P. Medas, G. Castillo, P. P. Rodrigues, Learning with Drift Detection, in: 17th Brazilian Symposium on Artificial Intelligence, 286–295, 2004.

[31] P. Sobolewski, M. Woźniak, Concept Drift Detection and Model Selection with Simulated Recurrence and Ensembles of Statistical Detectors, Journal of Universal Computer Science 19 (4) (2013) 462–483.

[32] M. Woźniak, A hybrid decision tree training method using data streams, Knowledge and Information Systems 29 (2) (2011) 335–347.

[33] D. V. R. Oliveira, G. D. C. Cavalcanti, R. Sabourin, Online pruning of base classifiers for Dynamic Ensemble Selection, Pattern Recognition 72 (2017) 44–58.

[34] L. Rutkowski, M. Jaworski, L. Pietruczuk, P. Duda, The CART decision tree for mining data streams, Information Sciences 266 (2014) 1–15.

[35] X. Zeng, G. Li, Incremental partial least squares analysis of big streaming data, Pattern Recognition 47 (11) (2014) 3726–3735.

[36] G. Widmer, M. Kubat, Learning in the Presence of Concept Drift and Hidden Contexts, Machine Learning 23 (1) (1996) 69–101.

[37] F. J. Ferrer-Troyano, J. S. Aguilar-Ruiz, J. C. R. Santos, Incremental Rule Learning and Border Examples Selection from Numerical Data Streams, J. UCS 11 (8) (2005) 1426–1439.

[38] F. J. Ferrer-Troyano, J. S. Aguilar-Ruiz, J. C. R. Santos, Data streams classification by incremental rule learning with parameterized generalization, in: ACM Symposium on Applied Computing, 657–661, 2006.

[39] M. Deckert, J. Stefanowski, RILL: Algorithm for Learning Rules from Streaming Data with Concept Drift, in: International Symposium on Methodologies for Intelligent Systems, 20–29, 2014.

[40] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: a survey, Information Fusion 37 (2017) 132 – 156.

[41] M. Woźniak, M. Graña, E. Corchado, A survey of multiple classifier systems as hybrid systems, Information Fusion 16 (2014) 3–17.

[42] Y. Sun, K. Tang, L. L. Minku, S. Wang, X. Yao, Online Ensemble Learning of Data Streams with Gradually Evolved Classes, IEEE Trans. on Knowledge and Data Engineering 28 (6) (2016) 1532–1545.

[43] L. L. Minku, X. Yao, DDD: A New Ensemble Approach for Dealing with Concept Drift, IEEE Trans. on Knowledge and Data Engineering 24 (4) (2012) 619–633.

[44] J. Gama, R. Sebastião, P. P. Rodrigues, On evaluating stream learning algorithms, Machine Learning 90 (3) (2013) 317–346.

[45] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, X. Zhang, Fast Memory Efficient Local Outlier Detection in Data Streams, IEEE Trans. on Knowledge and Data Engineering 28 (12) (2016) 3246–3260.

[46] A. Bifet, G. D. F. Morales, J. Read, G. Holmes, B. Pfahringer, Efficient Online Evaluation of Big Data Stream Classifiers, in: 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 59–68, 2015.

[47] P. G. Espejo, S. Ventura, F. Herrera, A Survey on the Application of Genetic Programming to Classification, IEEE Trans. on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 40 (2) (2010) 121–144.

[48] K. Nag, N. Pal, A Multiobjective Genetic Programming-Based Ensemble for Simultaneous Feature Selection and Classification, IEEE Trans. on Cybernetics 46 (2) (2016) 499–510.

[49] S. Silva, E. Costa, Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories, Genetic Programming and Evolvable Machines 10 (2) (2009) 141–179.

[50] A. Shaker, E. Lughofer, Resolving global and local drifts in data stream regression using evolving rule-based models, in: IEEE Conference on Evolving and Adaptive Intelligent Systems, 9–16, 2013.

[51] E. Lughofer, P. P. Angelov, Handling drifts and shifts in on-line data streams with evolving fuzzy systems, Applied Soft Computing 11 (2) (2011) 2057–2068.

[52] M. O'Neill, L. Vanneschi, S. Gustafson, W. Banzhaf, Open issues in genetic programming, Genetic Programming and Evolvable Machines 11 (3) (2010) 339–363.

[53] L. Zhang, J. Lin, R. Karim, Sliding Window-Based Fault Detection From High-Dimensional Data Streams, IEEE Trans. Systems, Man, and Cybernetics: Systems 47 (2) (2017) 289–303.

[54] E. Lughofer, Efficient sample selection in data stream regression employing evolving generalized fuzzy models, in: IEEE International Conference on Fuzzy Systems, 1–9, 2015.

[55] S. Khanchi, M. Heywood, 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, 35–50, 2016.

[56] B. Wang, J. Pineau, Online bagging and boosting for imbalanced data streams, IEEE Trans. on Knowledge and Data Engineering 28 (12) (2016) 3353–3366.

[57] A. Cano, A. Zafra, S. Ventura, Speeding up the evaluation phase of GP classification algorithms on GPUs, Soft Computing 16 (2) (2012) 187–202.

[58] A. Cano, A. Zafra, S. Ventura, Parallel evaluation of Pittsburgh rule-based classifiers on GPUs, Neurocomputing 126 (2014) 45–57.

[59] A. Cano, A. Zafra, S. Ventura, Speeding up multiple instance learning classification rules on GPUs, Knowledge and Information Systems 44 (1) (2015) 127–145.

[60] E. Lughofer, M. Pratama, On-line Active Learning in Data Stream Regression using Uncertainty Sampling based on Evolving Generalized Fuzzy Models, IEEE Trans. on Fuzzy Systems 26 (1) (2018) 292–309.

[61] A. Cano, B. Krawczyk, Learning classification rules with differential evolution for high-speed data stream mining on GPUs, in: IEEE Congress on Evolutionary Computation, 197–204, 2018.

[62] D. Lam, D. Wunsch, Unsupervised Feature Learning Classification With Radial Basis Function Extreme Learning Machine Using Graphic Processors, IEEE Trans. Cybernetics 47 (1) (2017) 224–231.

[63] B. Krawczyk, GPU-Accelerated Extreme Learning Machines for Imbalanced Data Streams with Concept Drift, Procedia Computer Science 80 (2016) 1692–1701.

[64] A. Cuzzocrea, M. M. Gaber, A. M. Shiddiqi, Distributed Classification of Data Streams: An Adaptive Technique, in: 17th International Conference on Big Data Analytics and Knowledge Discovery, 296–309, 2015.

[65] D. M. Chitty, Faster GPU-based genetic programming using a two-dimensional stack, Soft Computing 21 (14) (2017) 3859–3878.

[66] A. Cano, S. Ventura, GPU-parallel Subtree Interpreter for Genetic Programming, in: Proceedings of the Conference on Genetic and Evolutionary Computation, 887–894, 2014.

[67] D. Brzezinski, J. Stefanowski, Combining block-based and online methods in learning ensembles from concept drifting data streams, Information Sciences 265 (2014) 50–67.

[68] M. Pratama, G. Zhang, M. J. Er, S. G. Anavatti, An Incremental Type-2 Meta-Cognitive Extreme Learning Machine, IEEE Trans. Cybernetics 47 (2) (2017) 339–353.

[69] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive Online Analysis, Journal of Machine Learning Research 11 (2010) 1601–1604.

[70] A. Cano, J. M. Luna, A. Zafra, S. Ventura, A Classification Module for Genetic Programming Algorithms in JCLEC, Journal of Machine Learning Research 16 (2015) 491–494.

[71] A. Bifet, R. Gavaldà, Adaptive learning from evolving data streams, in: International Symposium on Intelligent Data Analysis, 249–260, 2009.

[72] G. H. John, P. Langley, Estimating Continuous Distributions in Bayesian Classifiers, in: 11th Conference on Uncertainty in Artificial Intelligence, 338–345, 1995.

[73] A. Bifet, B. Pfahringer, J. Read, G. Holmes, Efficient Data Stream Classification via Probabilistic Adaptive Windows, in: 28th ACM Symposium on Applied Computing, 801–806, 2013.

[74] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, R. Morales-Bueno, Early drift detection method, in: 4th International Workshop on Knowledge Discovery from Data Streams, vol. 6, 77–86, 2006.

[75] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavaldà, New ensemble methods for evolving data streams, in: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 139–148, 2009.

[76] A. Bifet, G. Holmes, B. Pfahringer, Leveraging bagging for evolving data streams, in: European Conference on Machine Learning and Knowledge Discovery in Databases, 135–150, 2010.

[77] H. M. Gomes, F. Enembreck, SAE2: advances on the social adaptive ensemble classifier for data streams, in: 29th Annual ACM Symposium on Applied Computing, 798–804, 2014.

[78] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Trans. on Neural Networks 22 (10) (2011) 1517–1531.

[79] J. Z. Kolter, M. A. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts, Journal of Machine Learning Research 8 (2007) 2755–2790.

[80] R. Pelossof, M. Jones, I. Vovsha, C. Rudin, Online coordinate boosting, in: 12th IEEE International Conference on Computer Vision, 1354–1361, 2009.

[81] H. Wang, W. Fan, P. S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 226–235, 2003.

[82] D. Brzezinski, J. Stefanowski, Reacting to different types of concept drift: The accuracy updated ensemble algorithm, IEEE Trans. on Neural Networks and Learning Systems 25 (1) (2014) 81–94.

[83] S. García, A. Fernández, J. Luengo, 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 180 (10) (2010) 2044–2064.

[84] A. Shaker, E. Hüllermeier, Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study, Neurocomputing 150 (2015) 250–264.

[85] K. Yu, W. Ding, X. Wu, LOFS: A library of online streaming feature selection, Knowledge and Information Systems 113 (2016) 1–3.

[86] M. M. Masud, C. Woolam, J. Gao, L. Khan, J. Han, K. W. Hamlen, N. C. Oza, Facing the reality of data stream classification: coping with scarcity of labeled data, Knowledge and Information Systems 33 (1) (2012) 213–244.

[87] M. Woźniak, P. Ksieniewicz, B. Cyganek, A. Kasprzak, K. Walkowiak, Active Learning Classification of Drifted Streaming Data, in: International Conference on Computational Science, 1724–1733, 2016.

[88] M. J. Hosseini, A. Gholipour, H. Beigy, An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams, Knowledge and Information Systems 46 (3) (2016) 567–597.