Learning Classification Rules with Differential Evolution for High-Speed Data Stream Mining on GPUs

Conference Paper · July 2018

DOI: 10.1109/CEC.2018.8477961

2 authors:

Alberto Cano
Virginia Commonwealth University
74 PUBLICATIONS **611** CITATIONS
SEE PROFILE

Bartosz Krawczyk
Virginia Commonwealth University
170 PUBLICATIONS **1,510** CITATIONS
SEE PROFILE

Learning classification rules with differential evolution for high-speed data stream mining on GPUs

Alberto Cano
Department of Computer Science
Virginia Commonwealth University
Richmond, Virginia, USA, 23284
Email: acano@vcu.edu

Bartosz Krawczyk
Department of Computer Science
Virginia Commonwealth University
Richmond, Virginia, USA, 23284
Email: bkrawczyk@vcu.edu

***Abstract*—High-speed data streams are potentially infinite sequences of rapidly arriving instances that may be subject to concept drift phenomenon. Hence, dedicated learning algorithms must be able to update themselves with new data and provide an accurate prediction in a limited amount of time. This requirement was considered as prohibitive for using evolutionary algorithms for high-speed data stream mining. This paper introduces a massively parallel implementation on GPUs of a differential evolution algorithm for learning classification rules in the presence of concept drift. The proposal based on the $DE/rand-to-best/1/bin$ strategy takes advantage of up to four nested levels of parallelism to maximize the performance of the algorithm. Efficient GPU kernels parallelize the evolution of the populations, rules, conditional clauses, and evaluation on instances. The proposed method is evaluated on 25 data stream benchmarks considering different types of concept drifts. Results are compared with other publicly available streaming rule learners. Obtained results and their statistical analysis proves an excellent performance of the proposed classifier that offers improved predictive accuracy, model update time, decision time, and a compact rule set.**

I. Introduction

Data stream mining is among the most dynamically developing topics in machine learning, aiming at building learning models from an ordered sequence of instances that arrive over time [1]. In contrast to static environments, a learner only has access to the most recent instances and must update its model according to potential changes in the underlying data distribution. This phenomenon is known as concept drift [2]. High-speed data streams [3] involve flows of instances that arrive at a higher rate than they can be mined by traditional systems, and demand very fast or even real-time processing. A delay in the class prediction is not accepted (e.g. credit card fraud detection) and there is a small time budget to update the classification model. At the same time, we aim at obtaining the highest possible predictive accuracy under these time constraints.

Decision trees and rule-based classifiers are preferred for extracting interpretable classification models from data streams [4], [5]. Interpretability is highly valuable, as it allows to provide meaningful information about the decision being made and reflect drifts in a comprehensible way. However, it usually comes at the cost of losing accuracy as compared to other black-box techniques. Evolutionary algorithms are very popular stochastic optimization techniques for learning rule-based classification models in static environments. However, they are based on the improvement of a population of individuals along a number of generations, and consequently they demand significant runtime. This is the main reason why they have not been frequently used in data stream mining. Fortunately, recent advances in parallel computing using Graphic Processing Units (GPUs) have demonstrated to speed up evolutionary learners to the point in which now they can be employed for high-speed data streams satisfying the time constraints [6], [7].

This paper introduces a differential evolution algorithm for learning classification rules from high-speed data streams. It presents the implementation of a $DE/rand-to-best/1/bin$ strategy on GPUs to massively parallelize each of the stages of the algorithm, maximizing the occupancy of the GPU resources and the performance throughput. Evolutionary learners are well suited for drifting data streams, since the fitness function enforces individuals to evolve and adapt to reflect underlying changes in the data distribution. However, it is necessary to balance the exploitation and exploration of new solutions in case of different types of drifts. The proposal, named DE-Rules, is also extended using the Naïve Bayes rule to increase the confidence of the predictions by computing the posterior probabilities, similar to other rule algorithms for data streams. The experimental study compares the prequential accuracy, model update time, decision time, and number of rules (complexity of the model) with other popular rule-based algorithms. Results validated with non-parametric statistical analysis clearly show the better performance of our proposal, demonstrating the feasibility of GPU-based evolutionary computation for high-speed data stream mining.

The main contributions of this paper are:

- A GPU-parallel implementation of a differential evolution algorithm for learning rules from high-speed streams. Four levels of parallelism are deployed to maximize the performance with an efficient individual representation to maximize memory coalescence and throughput.
- An elitism and population restart strategy to balance the transfer of model learned through the stream while exploring new solutions to adapt in case of concept drift.

- An extension using the Naïve Bayes rule to improve the accuracy using the posterior probabilities.
- A thorough experimental study on a large set of stream benchmarks to evaluate and compare the performance with publicly available rule-based algorithms.

The manuscript is organized as follows. Section II presents the background in stream mining and differential evolution. Section III provides a description of the proposed differential evolution for learning classification rules from data streams on GPUs. Section IV presents the experimental study and discusses the results. Finally, Section V presents the conclusions.

II. Background

Data stream mining builds predictive models from a potentially infinite flow of instances that arrive over time [1]. There are a number of intrinsic issues associated to the nature of data streams [8]. First, a classifier does not have an access to all of the instances in a given time, but only to a block (chunk processing) or individual one (online processing) arriving over time. Second, high-speed data streams constrain the time a classifier may take to update the model and give a prediction, thus demanding real-time processing with minimum latency. Third, properties and distribution of data may change through time, leading to a phenomenon known as concept drift [9]. Therefore, classifiers must be capable of self-adaptation, in order to the learn new concepts and forget the old ones.

Approaches based on ensemble learning have demonstrated to work effectively to cope with the aforementioned issues [10], [11], as components may be added and removed to reflect the drift presence. However, few proposals have focused on the interpretability and comprehensibility of the streaming classifiers. Rule-based classifiers are characterized by a trade-off between accuracy and interpretability [12]. *Ferrer-Troyano et al.* [13] introduced online rules that discarded positive and negative instances when they are not their nearest neighbors. They also presented FACIL [14], in which rules are first being inducted and then generalized over incoming instances. *Kosina and Gama* [15], [16] introduced Very Fast Decision Rules (VFDR) for high-speed data streams. It learns ordered or unordered growing rule sets by the divide and conquer strategy. A rule is expanded with the literal that has the highest gain measure of the examples covered by the rule employing the Hoeffding bound. A second version was proposed to improve the accuracy by taking into account the posterior probability given the Bayes rule (VFDR$_{NB}$). The main advantage is the boost of the confidence of the prediction even though the classification rule might not be sufficiently robust and reliable. The main disadvantage of VFDR is the lack of a methodology to adapt to drifting data, which results in an uncontrolled growth of the number of rules. In such a scenario, VFDR keeps adding new rules to cover instances coming from the new distribution. Having a very large number rules hampers the comprehensibility of the rule set and prevents any expert to interpret and visualize the learned model. This issue was resolved in Adaptive Very Fast Decision Rules (AVFDR) which embeds an explicit drift detector into the learning process. The drift detector monitors the error rate and removes degraded rules, preventing the excessive growth of the rule set. *Stahl et al.* [17] introduced an evolving rule-based classifier (eRules) to abstain from an uncertain label prediction and dynamically update its parameters. This approach was improved by *Le et al.* [18] (G-eRules) to deal with continuous attributes using a Gaussian distribution.

Evolutionary algorithms have been widely used for learning classification rules in static data mining [19]. However, the computational complexity and long runtimes prevented their application to high-speed data streams up to date. Fortunately, their intrinsic parallel nature along with recent advances on parallel computing using GPUs opens the door to their usage in data stream mining [20], [6]. Evolutionary rule-based classifiers offer high potential for data streams, since their flexibility and evolutionary adaptability allows for modeling the dynamic changes in data properties. Genetic operators facilitate to add and drop conditions on the classification rules to reflect the variable relevance of features through time. Elitism guarantees to keep best solutions so far whereas population restart advocates for exploration of solutions in new search spaces, which is especially beneficial when a sudden drift occurs. Another crucial advantage of evolutionary rules for data streams is that their iterative learning can be stopped and best solutions so far can be returned anytime. This behavior is desirable when a limited amount of time is given for a model update or we want to obtain a prediction with minimum delay.

Genetic programing and differential evolution are popular evolutionary techniques for learning classification rules [21], [22], [23], [24], [25]. However, differential evolution has the advantage of being a much simpler and faster evolution model. Therefore, it is better suited for high-speed data streams. Differential evolution has demonstrated solid, robust, and fast performance in converging to high-quality solutions in a low number of evaluations. It learns rules in the form of IF *antecedent* THEN *class*, where the antecedent contains AND clauses covering the dataset attributes. A rule set joins many of these rules and can be seen as logically OR clauses, covering many areas of the search space to facilitate exploration.

Differential evolution (DE) is a stochastic population-based optimization algorithm based on the idea of using difference vectors to produce new individuals [26]. The population is randomly initialized to contain P individuals, where each individual $\bar{x}$ is represented as a numeric vector of G genes. Individuals are evaluated according to a fitness function $f(\bar{x})$ and evolved through a transformation strategy following the convention $DE/a/b/c$, where a is the base vector (best, rand, rand-to-best), b is the number of difference vectors involved (usually 1 or 2), and c denotes the crossover method (exp for exponential, bin for binomial). There are many strategies but $DE/rand-to-best/1/bin$ is known to provide a robust performance. Given a base vector individual $\bar{x}_i$, it selects two random individuals $\bar{x}_{r_1}$ and $\bar{x}_{r_2}$, all different to each other. Then it generates a new individual $\bar{x}'_i$ using the following transformation:

$$\bar{x}'_{i,j} = \bar{x}_{i,j} + F \cdot (best_j - \bar{x}_{i,j}) + F \cdot (\bar{x}_{r_1,j} - \bar{x}_{r_2,j}) \quad (1)$$

where F is a constant which controls the magnitude of vector differences and $best$ is the best individual in the population. The new individual takes for each gene j the value produced in the transformation with a uniform probability CR, otherwise it copies the original value of the gene j in the base vector. This mechanism balances crossover and mutation. The new individual is compared against its base vector and used as a new solution if it improves the fitness function. This process is repeated for a number of generations, in order to converge the population to better solutions.

III. Differential evolution on GPUs for learning classification rules on high-speed data streams

Differential evolution stochastically initializes and optimizes a population of individuals, here representing IF *antecedent* THEN *class* rules. The genetic material is recombined along a number of generations to produce new improved solutions evaluated according to a fitness function. Stochastic optimization does not guarantee to find the global optimum but it allows finding a suboptimal solution in a reasonable time. In this work, we employ the $DE/rand-to-best/1/bin$ strategy to evolve the antecedents of a population of classification rules for which the consequent (predicted class) is fixed.

Iterative rule learning [27] builds a rule-base classifier by aggregating rules one by one until a certain criteria is satisfied. The number of rules may be determined by the user or heuristically encoded in the algorithm. Iterative rule learning executes the evolutionary process multiple times to learn rules targeting different classes and in every iteration it selects the best solution to become a member of the rule-base. Iterative rule learning has the potential advantage of promoting the diversity in the exploration of search spaces among runs, thus preventing the overlapping of the rules coverage areas. This way, the rule set for a class includes diverse components which will improve the confidence of the class predictions. However, running sequentially the evolutionary process many times to obtain a number of classification rules per class is incompatible with keeping a short runtime and fast classifier update for high-speed data streams. Fortunately, all the runs are independent to each other and we may define a first high-level coarse-grained parallelism evolving many populations concurrently, each aiming to obtain a single classification rule for a given data class.

Once all iterative rule learning executions run concurrently, we may define a second level of parallelism for handling all the individuals in the population concurrently. Each step in the differential evolution algorithm (initialization, genetic operators, fitness function, and selection) can be applied to every individual independently. Moreover, the implementation of the initialization, genetic operators, fitness function, and selection can be parallelized even further by defining a third level of parallelism in which each gene is operated concurrently. Finally, the fitness function admits a fourth level of parallelism to check the coverage of each rule condition on each instance in the train data. These four nested levels of parallelism can be natively implemented on the massively parallel programming model of a GPU to maximize the concurrency and minimize the runtime, thus allowing to apply the differential evolution algorithm for high-speed data stream mining.

A. Rule encoding

Given a data stream DS with A attributes and C classes, we may define an individual $\bar{x}$ representing a classification rule as a numeric vector of $2 \times A$ real values corresponding to A conditional clauses logically joined by AND. This allows to represent a rule consisting in a number of clauses equal to the number of attributes, where each clause encodes the interval range $[minimum, maximum]$ for a given attribute domain. In contrast to [22], which also employ an active vector $\{0,1\}^A$ to specify whether or not each clause in a rule is active and taken into account, here the activation of a clause is implicit in the genotype. If the minimum is less or equal than the maximum, then the clause is computed as an IN operator, otherwise it is considered inactive. This simplifies the representation and allows to easily ignore attributes not relevant for the class prediction. Moreover, it also reduces the complexity of having to additionally optimize the active vector.

The representation of the population predicting a data class would be intuitively seen as a sequence of each of the individual's genotype clauses $\{[minimum, maximum]^A\}^P$. However, this format is not efficient when implementing parallelism on the GPU, since concurrent accesses of memory positions to each of the individuals' values do not fall in the same memory transaction, leading to uncoalesced memory accesses and consequently a huge performance penalty. On the contrary, it is much more efficient to gather together all the minimum and maximum values of the clauses respectively $\{[minimum^P, maximum^P]\}^A$ so that the coalescence of the memory accesses by the threads in the warp maximizes the throughput. Finally, to scale out to the many number of parallel evolutions for obtaining a number of rules per class, this population representation is subsequently repeated as many times as the number of rules per class and number of classes.

B. Initialization of the population

Differential evolution begins with a randomly initialized population. The three nested levels of parallelism allows to initialize simultaneously the population in every rule learning run, every individual in each population, and every gene in each individual. We may define a 3D grid of thread blocks representing (number of rules per class, number of classes, number of attributes) whose structure comprises a 1D thread block with as many threads as the population size. This way, threads in a warp (group of 32 threads being executed in a multiprocessor) access consecutive fully coalesced memory positions, and initialize the gene values with a unique and uniform random real value contained within the known min/max boundaries of each attribute in the stream. While this approach is the most efficient by providing the maximal coalescence and throughput, it demands significant memory resources to allocate for each gene its own random generator structure. Moreover, it is crucial to provide a unique seed to the random generator in each thread based on the absolute thread index to guarantee producing unique starting states and different random number sequences.

C. Genetic operators

$DE/rand-to-best/1/bin$ mutates a base vector individual mixing the genetic material with two other distinct individuals randomly selected and the best solution in the population, as described in Section II. Detailed description of rand-to-best binomial mutation and crossover is provided in [26] and here we focus on the parallel implementation. Similar to the initialization, we may define a 3D kernel to take advantage of the three nested levels of parallelism to produce the offspring for every population, rule, and conditional clause on the attributes. A given thread is responsible for the mutation and crossover of a single minimum/maximum bound condition, keeping memory accesses fully coalesced, scaling to any number of rules and clauses, and maximizing the throughput.

D. Fitness function

The fitness function evaluates the quality of the individuals generated by assigning a fitness value according to their capability of correctly classifying the train data. It is based on the count of the confusion matrix values: true positives (T_P), true negatives (T_N), false positives (F_P), and false negatives (F_N), aiming to maximize both the sensitivity and specificity of the rule. The product of these two is commonplace in classification to define the single-objective function shown in Equation 2. The trade-off between sensitivity and specificity is especially beneficial in the case on imbalanced data [28], when there are many more examples of one class than the others. Importantly, this function is robust to dynamic changes that may happen to the data classes distribution through time. Evolutionary learners are implicitly self-adapting to changes in the data stream, thus being highly suitable for handling concept drift.

$$fitness = \frac{T_P}{T_P + F_N} \times \frac{T_N}{T_N + F_P} \tag{2}$$

The computational complexity of the sequential implementation of the fitness function to run a single iteration of the evolutionary process is $\mathcal{O}(P \times N)$ where P is the population size and N is the number of train instances. The coverage of each rule must be matched with every data instance, checking whether the conditional clauses apply to the instance values. The function is run for every individual in every population per class for a number of generations. Therefore, a sequential implementation would prevent its application to high-speed data stream mining because new data instances would arrive before the model is evaluated and updated, leading to a bottleneck in the learning system.

Speeding up the fitness function of evolutionary rule learners on GPUs is a well-researched area [20], [29]. However, in most of the research works it is the only portion of the algorithm parallelized on the GPU, since it takes most of the runtime and requires copying the rules to the device every time they need to be evaluated. While this approach is good enough for static datasets, it fails to satisfy the time constrains for high-speed data streams. Therefore, in this work we are introducing a native implementation of all the algorithm on the GPU, avoiding any data transfer overheads when intercommunicating the CPU and GPU.

The GPU implementation of the fitness function requires four nested levels of parallelism to maximize the occupancy and scalability to bigger population and dataset dimensionality. A first kernel defines a 4D grid of thread blocks representing (number of instances, number of rules per class, number of classes, number of attributes) whose structure comprises a 1D thread block with as many threads as the population size. The kernel evaluates in parallel the coverage of all the conditional clauses of all the rules on all the instances, avoiding any kind of serialization, producing the values for the confusion matrix [30]. This way, the execution of threads in a thread block and a warp is both fully coalesced and not having any divergent execution path which would hamper the performance. Finally, a second kernel reduces in parallel the confusion matrix values and assigns the fitness to the individuals.

E. Selection and model update

After the offspring has been evaluated, differential evolution compares every trial vector (child) and its base vector (parent), selecting the best of the two as the individual for the next generation. When evolution has run for the number of generations predetermined, the best individual from each of the populations is returned to become a rule member of the classifier, which comprises a number of user-defined rules per class. The classifier is now valid to predict the class label for new arriving instances from the stream.

Updating the classifier to reflect changes when new instances arrive can be addressed using three main approaches. First, the classifier may be rebuild from scratch every time a new data chunk arrives. Second, the evolution with the same population as in the previous iteration may be resumed. Third, the population can be restarted, while keeping the best performing elitist individuals. The first option is no different than running differential evolution on static data independently. The main drawback is that no knowledge (in the form of a model) is transferred from one iteration to the following, loosing highly valuable information that was mined during the stream processing. It will take some time and a number of generations to rebuild the model to accurately classify new incoming data. This idea is contrary to the principle of adaptability inherently implicit in evolutionary algorithms. The second option transfers the exiting model to the new data, which is highly likely to perform accurately if the data distribution is similar to the historic. In case of a gradual or small drift, the evolutionary process will update the rule conditions to reflect the changes accordingly. However, the main drawback

Algorithm 1 DE-Rules algorithm for stream mining on GPUs

```
1:  $elitistRules \leftarrow \{\emptyset\}$
2:  $trainData \leftarrow \{\emptyset\}$
3:  while $stream.hasData()$ do
4:      $ruleBase \leftarrow \{\emptyset\}$
5:      $trainData \leftarrow stream.nextData()$
6:      $population \leftarrow initialize(P) \cup elitistRules$
7:      $evaluate(population, trainData)$
8:      $generation \leftarrow 0$
9:      while $generation < numberGenerations$ do
10:         $r_1 \neq r_2 \leftarrow random(population)$
11:         $\bar{x}'_{i,j} = \bar{x}_{i,j} + F \cdot (best_j - \bar{x}_{i,j}) + F \cdot (\bar{x}_{r_1,j} - \bar{x}_{r_2,j})$
12:         $evaluate(offspring\ \bar{x}', trainData)$
13:         $population \leftarrow best(population \cup offspring)$
14:         $generation \leftarrow generation + 1$
15:     end while
16:     $ruleBase \leftarrow bestRules(population)$
17:     $elitistRules \leftarrow ruleBase$
18: end while
```

is the inability of fast adaptation to the new distributions if a sudden concept drift occurred. This will result in a very tedious adaptation that will not be efficient nor effective in time. Finally, the third option combines the advantages of the two previous while overcoming their drawbacks, regardless of the absence or presence of drifts of any magnitude. The elitist individuals will inherit the previously known model whereas new randomly initialized individuals will explore the search space in case of a significant drift. Anyhow, the best alternative will quickly spread in a small number of generations. This is the most suitable option for self-adapting evolutionary rule learning fir high-speed data stream mining.

Algorithm 1 shows the high-level pseudo-code of DE-Rules for extracting classification rules from high-speed data streams on GPUs. All operations for every population, individual, class, and attribute are performed in parallel as described in the previous sections, taking advantage of the multi-level parallelism. Scalability to multiple GPUs is straightforward, since evolution of each of the populations is independent, they may be distributed into many devices without any overhead.

IV. Experiments

This section presents the experimental study to evaluate and compare the accuracy, runtime, and complexity of the classification rule mining algorithms for high-speed data streams.

A. Experimental setup

Table I summarizes the characteristics of the 25 data stream benchmarks considered in the experimental evaluation, comprising various aspects of concept drift. The selection includes real-world and artificially-generated data streams publicly available in the Massive Online Analysis (MOA) software [31], [32].

There are three publicly available rule-based classifiers for data streams in MOA with whom we are comparing our proposal DE-Rules. Table II lists the algorithms and their main parameters, which were set according to the recommended values reported by their authors. The proposed DE-Rules has been implemented in CUDA and integrated in MOA via Java JNI. Moreover, we propose a modification of DE-Rules taking into account the posterior probability given the Bayes rule, following the same methodology of VFDR$_{NB}$ [15]. Algorithm's

TABLE I
Data stream benchmarks characteristics.

Dataset	Instances	Atts	Classes	Properties
Powersupply	29,928	2	24	unknown drift
Shuttle	58,000	9	7	unknown drift
DJ-30	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
BNG-wine	1,000,000	13	3	unknown drift
LED	1,000,000	7	10	no drift
Hyperplane	1,000,000	10	4	no drift
SEA	1,000,000	3	2	no drift
Waveform	1,000,000	40	3	no drift
Waveform-noise	1,000,000	40	3	no drift, noise
Agrawal	1,000,000	9	2	no drift
AssetNegotiation	1,000,000	5	2	no drift
LED-drift-fast	1,000,000	7	10	gradual drift
LED-drift	1,000,000	7	10	gradual drift
Hyperplane-drift	1,000,000	10	4	gradual drift
SEA-drift	1,000,000	3	2	gradual drift
SEA-drift-sudden	1,000,000	3	2	sudden drift
Waveform-drift	1,000,000	40	3	gradual drift
Agrawal-drift-F1-F10	1,000,000	9	2	gradual drift
Agrawal-drift-F10-F1	1,000,000	9	2	gradual drift
Agrawal-drift-random	1,000,000	9	2	gradual drift
STAGGER-drift	1,000,000	3	2	gradual drift

TABLE II
Rule-based algorithms for data streams and their parameters.

Acronym	Algorithm's name	Parameters
G-eRules [18]	Gaussian Evolving set of Rules	slidingWindowSize: 500 minRuleTries: 10
VFDR [15]	Very Fast Decision Rules	supervised: true ordered: false splitConfidence: 1E-6
VFDR$_{NB}$ [15]	Very Fast Decision Rules with Naïve Bayes	supervised: true ordered: false splitConfidence: 1E-6
DE-Rules	Differential Evolution Rules	population: 64 generations: 50 rulesPerClass: 3 CR: 0.9 F: 0.8
DE-Rules$_{NB}$	Differential Evolution Rules with Naïve Bayes	population: 64 generations: 50 rulesPerClass: 3 CR: 0.9 F: 0.8

code along with detailed results for the experimental analysis are publicly available in the website 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 1080 Ti GPU on a Linux Centos 7 x86-64 system.

B. Results and discussion

Performance metrics for evaluating data stream classifiers include predictive power, model update time, decision time, and memory complexity. A stream mining algorithm must aim to strike balance among all of these criteria. Prequential accuracy is the most frequently used measure for evaluating predictive power in data stream mining [33]. In the context of high-speed data streams, algorithms are required not only to provide the best possible accuracy but also work under strict constraints of model update and decision times. Model update time informs us about the time it takes to rebuild and update the structure of the classifier every time new train instances arrive. The classifier must avoid queuing instances due to slow processing. Decision time reports the time needed by a given classifier to output a prediction for an instance. In many real-world problems, real-time predictions are required and no decision latency is permitted [34]. Moreover, rule learners must infer a sufficient number of rules capable of modeling data accurately without suffering the bloat effect, i.e., the uncontrollable growth of the number and complexity of rules, which would prevent experts from the advantages of interpretability and comprehensibility of rule-based classification.

Table III shows the prequential accuracy of the rule-based classifiers implemented in MOA compared to our proposed DE-Rules and its modification using the Naïve Bayes rule. Results are reported for each dataset, along with the average results for accuracy, Friedman rank, model update time (train time), decision time (test time), and memory complexity (RAM-Hours). G-eRules is shown to achieve the worst performance among all classifiers, returning unsatisfactory accuracy for vast majority of benchmarks. The Naïve Bayes rule applied to VFDR$_{NB}$ significantly improves the accuracy of VFDR, at the cost of increasing the decision time due to the probabilities computation. The base model of our proposal DE-Rules clearly shows to improve not only the results of VFDR but also VFDR$_{NB}$, both in terms of accuracy, model update time, decision time, and memory complexity. Consequently, our proposal demonstrates to learn more accurate rules, faster, and simpler than existing rule-based classifiers publicly available in MOA. The alternative of applying the Naïve Bayes rule to our proposal DE-Rules$_{NB}$ shows to improve even further the obtained prequential accuracy, achieving the best results in 12 out of 25 datasets. However, computations of posterior probabilities introduce a significant delay in decision time. This makes this extension as an optional one that should be selected based on the specifics of problem under consideration.

[1]Source code available in http://people.vcu.edu/∼acano/DE-Rules

TABLE III
ACCURACY COMPARISON OF RULE-BASED DATA STREAM CLASSIFIERS.

Dataset	G-eRules	VFDR	VFDR$_{NB}$	DE-Rules	DE-Rules$_{NB}$
Powersupply	4.45	4.17	**15.74**	14.80	15.20
Shuttle	78.91	88.40	96.06	**99.58**	96.58
DJ-30	4.76	67.71	73.36	84.49	**87.24**
CovType	67.79	60.32	75.58	76.15	**79.71**
IntelLabSensors	2.59	4.68	7.47	**98.43**	96.11
BNG-bridges	55.29	43.23	67.08	65.64	**70.07**
BNG-zoo	77.22	54.47	**91.48**	85.83	90.90
BNG-wine	40.13	76.95	**91.27**	85.35	90.48
LED	51.14	41.16	**73.75**	65.98	73.53
Hyperplane	49.95	78.63	**85.18**	77.80	83.29
SEA	59.21	80.49	85.02	77.93	**86.81**
Waveform	33.28	63.88	75.84	72.58	**79.61**
Waveform-noise	33.28	49.04	79.27	72.58	**79.61**
Agrawal	64.76	84.10	89.13	94.31	**94.44**
AssetNegotiation	78.10	67.82	68.91	75.94	**78.84**
LED-drift-fast	33.94	29.45	**52.81**	44.28	52.04
LED-drift	49.36	41.16	**73.75**	66.30	73.59
Hyperplane-drift	49.97	79.11	**85.56**	75.49	82.89
SEA-drift	64.98	81.56	85.17	79.18	**86.47**
SEA-drift-sudden	55.94	79.73	85.28	77.83	**85.58**
Waveform-drift	33.28	64.44	75.94	71.38	**79.23**
Agrawal-drift-F1-F10	63.00	74.00	76.59	**82.92**	82.38
Agrawal-drift-F10-F1	61.02	74.36	77.03	**83.03**	82.52
Agrawal-drift-random	75.61	71.54	73.09	**82.65**	81.65
STAGGER-drift	69.61	63.41	69.78	71.46	**71.96**
Avg. Accuracy	50.30	60.95	73.21	75.28	**79.23**
Friedman ranks	4.48	4.2	2.16	2.64	**1.52**
Avg. Train time (s)	0.159	0.477	0.473	**0.049**	0.050
Avg. Test time (s)	0.012	0.006	0.017	**0.001**	0.012
Avg. RAM-Hours	6.1E-5	7.1E-2	7.1E-2	**1.9E-5**	2.4E-5

TABLE IV
COMPARISON OF NUMBER OF RULES GENERATED.

Dataset	G-eRules	VFDR	VFDR$_{NB}$	DE-Rules	DE-Rules$_{NB}$
Powersupply	**21**	63	63	72	72
Shuttle	91	**3**	**3**	21	21
DJ-30	10781	**14**	**14**	90	90
CovType	692	53	53	**21**	**21**
IntelLabSensors	3071	**84**	**84**	174	174
BNG-bridges	2307	955	955	**18**	**18**
BNG-zoo	320	251	251	**21**	**21**
BNG-wine	**7**	224	224	9	9
LED	70	**28**	**28**	30	30
Hyperplane	17	75	75	**12**	**12**
SEA	136	385	385	**6**	**6**
Waveform	**4**	85	85	9	9
Waveform-noise	**4**	85	85	9	9
Agrawal	486	218	218	**6**	**6**
AssetNegotiation	244	301	301	**6**	**6**
LED-drift-fast	111	**26**	**26**	30	30
LED-drift	78	45	45	**30**	**30**
Hyperplane-drift	**6**	69	69	12	12
SEA-drift	83	385	385	**6**	**6**
SEA-drift-sudden	86	387	387	**6**	**6**
Waveform-drift	**4**	100	100	9	9
Agrawal-drift-F1-F10	320	545	545	**6**	**6**
Agrawal-drift-F10-F1	412	639	639	**6**	**6**
Agrawal-drift-random	308	635	635	**6**	**6**
STAGGER-drift	27	46	46	**6**	**6**
Avg. Rules	787	228	228	**25**	**25**

One of the disadvantages of the other rule-based learners is that their speed cannot be tuned easily, i.e., the model update will take a fixed amount of time and the process cannot be stopped if needed. On the contrary, DE-Rules allows to tune the population size and number of generations to iterate. On the one hand, smaller population sizes and number of generations will speed up the train time but it will likely decrease accuracy. On the other hand, larger values will slow the model update but accuracy may be increased. Moreover, DE-Rules model update may be stopped anytime, for example, in case of a limited time boundaries (critical in high-speed data streams), and a classification model will still be capable of handling incoming data (by using the best solution found until the learning process was terminated). This is a desirable and flexible mechanism capable of adapting to the user and problem needs.

Table IV collects information on the number of rules obtained by the examined algorithms over used benchmarks. DE-Rules obtains a user-defined number of rules per class (3 by default), which is very flexible so that the user can tune the simplicity and speed of the learning process. On the other hand, VFDR reports a significantly larger number of rules $\sim$10x on average than DE-Rules. G-eRules provides the largest number of rules, $\sim$3x on average than VFDR and $\sim$30x on average than DE-Rules. The massive number of rules can be explained by the inability of G-eRules and VFDR to adapt to concept drifts, when they happen the algorithms keep adding new rules to model the new data, but they lack of proper mechanisms to forget rules no longer valid in the new data distribution. A rule-based classifier comprising hundreds of rules cannot be claimed to be interpretable by humans. On the contrary, the main advantage of DE-Rules is its implicit ability to evolve and adapt existing rules to model data distribution drifts, while keeping compact rule set.

Table V shows the results of the Holm and Wilcoxon post-hoc tests for statistical comparison of multiple and pairwise algorithms, respectively, over multiple datasets [35]. Results having a p-value < 0.05 indicate statistical significant differences between DE-Rules$_{NB}$ and the compared algorithm. According to the Holm test of multiple comparisons, differences are found for all methods except VFDR$_{NB}$, whereas according to the Wilcoxon test of pairwise comparisons, differences are identified for all rule-based classifiers compared. The statistical analysis also demonstrates the better performance of the addition of the Naïve Bayes rule as compared to the base model.

Figure 1 shows the evolution of prequential accuracy over time on CovType dataset. In the early beginning, a small number of instances have been used for training and therefore the limited knowledge about the data distribution make it difficult for rule learners to model accurate rule boundaries. The Naïve Bayes rule demonstrates to significantly improve accuracy in the beginning using the limited information extracted from the posterior probabilities. As soon as more instances become available, rule algorithms learn better boundaries and increase their accuracy rapidly. However, it is noteworthy to point out the inability of VFDR of adapting to concept drifts, since a notable decrease in accuracy is observed over time, whereas DE-Rules$_{NB}$ shows its ability to adapt and evolve, maintaining a stable accuracy.

TABLE V
HOLM AND WILCOXON TESTS: DE-RULES$_{NB}$ VS OTHERS.

Algorithm	Holm p-value	Wilcoxon p-value
G-eRules	0.00000	5.96E-8
VFDR	0.00000	5.96E-8
VFDR$_{NB}$	0.15247	0.00417
DE-Rules	0.01221	1.62E-4

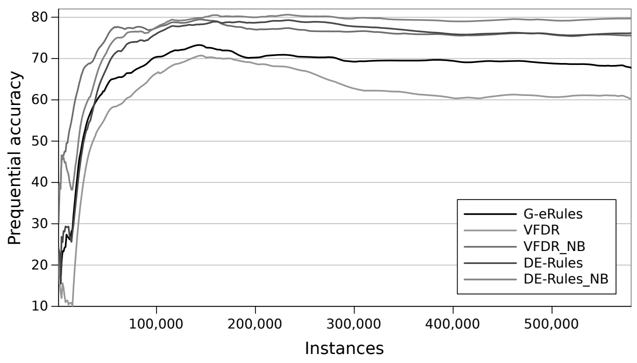

Fig. 1. Prequential accuracy on CovType dataset.

V. CONCLUSION

In this work we have introduced DE-Rules, a massively parallel implementation on GPUs of a differential evolution algorithm for learning classification rules from high-speed data streams. The differential evolution strategy based on $DE/rand-to-best/1/bin$ took advantage of up to four nested levels of parallelism to compute in parallel populations, individuals, genes, and evaluation on instances. The native implementation on GPUs minimized execution delays and provided the fastest performance. Efficient individual representation and kernel configuration allowed to maximize the memory coalescence, occupancy, and throughput. The elitism preservation and population restart strategy balanced the transfer of the existing knowledge represented in the learned model with the exploration of new solutions in case of concept drifts. Moreover, we have presented DE-Rules$_{NB}$, an extension of the proposal considering the Naïve Bayes posterior probabilities.

The differential evolution algorithm achieved better prequential accuracy than the other rule-based learners publicly available in MOA. Performance improvements were especially noteworthy in drifting data streams, showing the ability of DE-Rules of adapting and evolving according to changes in the data distribution and its properties. The extension with the Naïve Bayes rule improved the accuracy even further, similar as in reference VFDR. Despite being an evolutionary method and therefore being traditionally considered slow and inappropriate for high-speed data streams, the intrinsically parallel nature of the algorithms allowed for an efficient implementation on GPUs. This provided a model update time (train) up to 10x faster than VFDR. Moreover, the decision time (test) was 6x faster, making DE-Rules on GPUs highly suitable for high-speed data streams. However, when introducing the Naïve Bayes rule the decision time significantly increased whenever applied to any rule-based algorithm.

Finally, the parameters of DE-Rules facilitated to be adapted to any user needs regarding the number of rules (complexity of the model), and the population size and the number of generations (accuracy vs speed of learning). This way, the flexibility allowed to keep a simple yet accurate classification model in which a sufficient number of rules are interpretable by experts. These excellent results both in terms of accuracy and runtime demonstrated the feasibility of evolutionary rule learners on GPUs for high-speed data stream mining.

Acknowledgments

This research was supported by the VCU Research Support Funds, the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund, project TIN2017-83445-P.

References

[1] M. M. Gaber, "Advances in data stream mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 79–85, 2012.

[2] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–44, 2014.

[3] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, J. M. Benítez, and F. Herrera, "Nearest neighbor classification for high-speed big data streams using spark," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 10, pp. 2727–2739, 2017.

[4] A. Bifet, J. Zhang, W. Fan, C. He, J. Zhang, J. Qian, G. Holmes, and B. Pfahringer, "Extremely fast decision tree mining for evolving data streams," in *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1733–1742.

[5] A. Shaker, W. Heldt, and E. Hüllermeier, "Learning TSK fuzzy rules from data streams," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part II*, 2017, pp. 559–574.

[6] A. Cano, "A survey on graphic processing unit computing for large-scale data mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 1, p. e1232, 2018.

[7] A. Cano, A. Zafra, and S. Ventura, "A parallel genetic programming algorithm for classification," in *6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, vol. 6678 LNAI, no. PART 1, 2011, pp. 172–181.

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

[9] B. Krawczyk and A. Cano, "Online ensemble learning with abstaining classifiers for drifting and noisy data streams," *Applied Soft Computing*, 2018.

[10] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: a survey," *Information Fusion*, vol. 37, pp. 132 – 156, 2017.

[11] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren, "The online performance estimation framework: heterogeneous ensemble learning for data streams," *Machine Learning*, pp. 1–28, 2017.

[12] A. Cano, A. Zafra, and S. Ventura, "An interpretable classification rule mining algorithm," *Information Sciences*, vol. 240, pp. 1–20, 2013.

[13] F. J. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. R. Santos, "Incremental rule learning and border examples selection from numerical data streams," *Journal of Universal Computer Science*, vol. 11, no. 8, pp. 1426–1439, 2005.

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

[15] J. Gama and P. Kosina, "Learning decision rules from data streams," in *International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, pp. 1255–1260.

[16] P. Kosina and J. Gama, "Very fast decision rules for classification in data streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 168–202, 2015.

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

[18] T. Le, F. Stahl, J. B. Gomes, M. M. Gaber, and G. D. Fatta, "Computationally efficient rule-based classification for continuous streaming data," in *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 2014, pp. 21–34.

[19] A. A. Freitas, *Data mining and knowledge discovery with evolutionary algorithms*. Springer Science & Business Media, 2013.

[20] M. A. Franco and J. Bacardit, "Large-scale experimental evaluation of GPU strategies for evolutionary machine learning," *Information Sciences*, vol. 330, pp. 385–402, 2016.

[21] I. De Falco, "Differential evolution for automatic rule extraction from medical databases," *Applied Soft Computing*, vol. 13, no. 2, pp. 1265–1283, 2013.

[22] I. De Falco, U. Scafuri, and E. Tarantino, "A differential evolution approach for classification of multiple sclerosis lesions," in *IEEE Symposium on Computers and Communication*, 2016, pp. 141–146.

[23] E. Debie, K. Shafi, C. Lokan, and K. Merrick, "Investigating differential evolution based rule discovery in learning classifier systems," in *IEEE Symposium on Differential Evolution*, 2013, pp. 77–84.

[24] N. L. Tsakiridis, J. B. Theocharis, and G. C. Zalidis, "DECO3R: A Differential Evolution-based algorithm for generating compact Fuzzy Rule-based Classification Systems," *Knowledge-Based Systems*, vol. 105, pp. 160–174, 2016.

[25] A. Cano and B. Krawczyk, "Evolving rule-based classifiers with genetic programming on gpus for drifting data streams," *Pattern Recognition*, 2018.

[26] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[27] G. Venturini, "SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts," in *European Conference on Machine Learning*, 1993, pp. 280–296.

[28] B. Wang and J. Pineau, "Online bagging and boosting for imbalanced data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3353–3366, 2016.

[29] A. Cano, A. Zafra, and S. Ventura, "Parallel evaluation of Pittsburgh rule-based classifiers on GPUs," *Neurocomputing*, vol. 126, pp. 45–57, 2014.

[30] A. Cano and S. Ventura, "GPU-parallel Subtree Interpreter for Genetic Programming," in *Conference on Genetic and Evolutionary Computation*, 2014, pp. 887–894.

[31] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[32] A. Bifet, J. Read, G. Holmes, and B. Pfahringer, *Streaming Data Mining with Massive Online Analytics (MOA)*. World Scientific Publishing, 2018, ch. Chapter 1, pp. 1–25.

[33] J. Gama, R. Sebastião, and P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.

[34] A. Bifet, G. D. F. Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 59–68.

[35] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced non-parametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.