Trends of Evolutionary Machine Learning to Address Big Data Mining

Sana Ben Hamida[1(✉)], Ghita Benjelloun[2], and Hmida Hmida[3]

[1] Université Paris Dauphine, PSL Research University, CNRS, UMR[7243], LAMSADE, Paris, France
Sana.mrabet@dauphine.psl.eu
[2] Université Paris Dauphine, PSL Research University, Paris, France
[3] Université de Tunis El Manar, Tunis, Tunisia

Abstract. Improving decisions by better mining the available data in an Information System is a common goal in many decision making environments. However, the complexity and the large size of the collected data in modern systems make this goal a challenge for mining methods. Evolutionary Data Mining Algorithms (EDMA), such as Genetic Programming (GP), are powerful meta-heuristics with an empirically proven efficiency on complex machine learning problems. They are expected to be applied to real-world big data tasks and applications in our daily life. Thus, they need, as all machine learning techniques, to be scaled to Big Data bases. This paper review some solutions that could be applied to help EDMA to deal with Big Data challenges. Two solutions are then selected and explained. The first one is based on the algorithmic manipulation involving the introduction of the active learning paradigm thanks to the active data sampling. The second is based on the processing manipulation involving horizontal scaling thanks to the processing distribution over networked nodes. This work explains how each solution is introduced to GP. As preliminary experiences, the extended GP is applied to solve two complex machine learning problem: the Higgs Boson classification problem and the Pulsar detection problem. Experimental results are then discussed and compared to value the efficiency of each solution.

Keywords: Big data mining · Machine learning · Genetic Programming · Horizontal parallelization · Active learning · Data sampling · Higgs Boson classification · Pulsar detection

1 Introduction

Modern processes are frequently monitored using information systems that record large amounts of information given rise to Big Data bases. For a decision making environment, better mining the available data is a challenge. It is the major issue for the current data mining and machine learning algorithms. Regardless the Big Data 3V (variety, velocity and volume), data mining requires powerful knowledge discovery techniques that are able to deal with the related

I. Saad et al. (Eds.): ICIKS 2021, LNBIP 425, pp. 85–99, 2021.
https://doi.org/10.1007/978-3-030-85977-0_7

challenges such complexity and learning cost. Thus, the use of evolutionary data mining procedures becomes a hot trend. Thanks to their global search in the solution space, evolutionary computation techniques help in the information retrieval from a voluminous pool of data in a better way compared to traditional retrieval techniques [10].

Genetic Programming (GP) [21], as a particular EDMA, is considered as a universal machine learning tool. Its paradigm has shown a great potential when applied to supervised learning, especially classification and regression problems [3,32]. However, like other machine learning techniques, GP computational overhead increases with large datasets and its efficiency is affected. Some improvements are needed to scale GP when learning from big datasets. The adaptation of EDMA, specially GP, for big data problems may require redesigning the algorithms and/or their inclusion in parallel environments. Both tracks are discussed in this paper.

An efficient strategy to redesign a machine learning technique to handle large data sets is to introduce a special learning paradigm or a special parallelization paradigm without parallel hardware. In the first case, the algorithm of the data mining method is modified in order to improve its efficiency. With the second case, a parallel data-intensive computing scenario is introduced to the method. This strategy has become very popular in the last few years thanks to the appearance of some open-source tools such as Hadoop/MapReduce.

As discussed in [25], learning paradigms could bring some solutions for big data mining. For example, Ensemble Learning can help alleviate the "Concept Drift" and "Curse of Modularity" issues associated with Big Data. Similarly, Local Learning can help alleviate "Data locality" and "Variance and Bias" issues. In this work, we are interested in the Active Learning paradigm. Active learning is implemented essentially with active data sampling. In previous works [15,17], we demonstrated how active sampling could be an efficient solution to learn from large data sets by decreasing the size of the training set. The idea is to introduce some components in the algorithm that allows it to select its training data according to the evolution of the learning process. Section 4 explains in details how an active sampling can be introduced in the GP engine.

The second strategy to scale an EA (Evolutionary Algorithm) to Big Data Mining is the parallelization in a distributed environment. Vertical parallelization (scaling up) paradigm was widely explored. It includes multicore CPUs, supercomputers, hardware acceleration including graphic processing units (GPUs) and field-programmable gate arrays (FPGAs) [25]. In this work, we are interested in the horizontal parallelization (scaling out) paradigm where processing is dispersed over networked nodes. This paradigm do not require detailed knowledge of the underlying hardware architecture. The most known framework implementing the horizontal parallelization is the MapReduce paradigm and its distributed file system [11], originally introduced by Google. However, MapReduce encounters difficulties when dealing with iterative algorithms. Several new frameworks have been proposed to provide better support for iterative processing such as

HaLoop [6] or Apache Spark[1]. It is this last framework that we propose to scale up GP to Big Data Mining with processing manipulation. Our goal is the implementation of GP in a distributed ecosystem by adapting existing libraries. In a previous work, we proposed a solution to port a known GP implementation to the Spark context in order to take the most of its proven potential [16]. The source code of this model is available at GitHub[2]. This implementation is reused in the present work.

The main purpose of this paper is to summarize the different strategies to address Big Data challenges with machine learning (Sect. 2). It then presents the horizontal parallelization and the active learning paradigm, two suited approaches for EDMA (Sects. 3 and 4). The general purpose and some implementation details of the proposed techniques in each strategy are then given and explained in details (Sects. 3 and 4). The efficiency of GP obtained with the different extensions are studied on the HIGGS classification problem and the pulsar detection problem in Sect. 5.

2 Some Approaches to Address Big Data Learning Cost

In modern Information Systems, the amount of stored data has reached a huge volume and their complexity has widely increased. New challenges had then arisen due to the data characteristics.

Approaches to countering the Big Data challenges for machine learning are quite diverse. They are summarized in [25] and classified according to the data analytics pipeline (see Fig. 1). Another review, published in [3], identifies methods and techniques to accelerate evolutionary algorithms when applied to learning tasks. Scaling approaches focus essentially on manipulating data, processing and algorithms. We summarize below the main categories of the different approaches and paradigms that can be developed to alleviate some issues associated with Big Data.

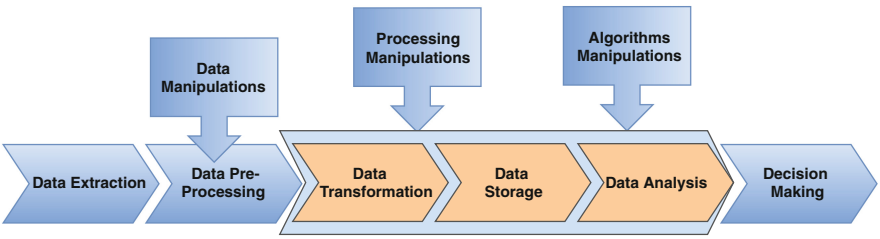

Fig. 1. Manipulations through Data Analytics pipeline [25].

[1] https://spark.apache.org.
[2] https://github.com/hhmida/gp-spark.

- Data manipulation: It is applied in the pre-processing phase before administering the data to the learning process. Its goal is to reduce the size of the learning data base, for example by applying techniques of sampling or feature selection. Sampling, in this case, is independent of the learning process (evolution process for GP).
- Processing manipulation: it relies on processing manipulations to handle the additional computational cost by vertical or horizontal scaling. It includes parallelization approaches using the parallel nature of some population based algorithms such as GP and EA. Parallelization can be done with:
  - Vertical scaling (scaling up): It is based on increasing resources for a single node. For example, hardware acceleration based on graphics processors called General Purpose Graphics Processing Units (GPGPU) and parallelization using multicore CPUs have been used in several jobs [14,23,27].
  - Horizontal scaling (scaling out): It refers to distributed systems where computations are deployed on a cluster of nodes [27]. This is the most common form of distribution with Big Data problems. Several frameworks have been put in place for two types of parallelization: by batch or flow-oriented. The most used are Hadoop/MapReduce [11] and Apache Spark [34].
- Algorithm Manipulation: it relies on algorithm manipulations in order to optimize its running time or to improve the learning quality. It involves the introduction of new machine learning paradigms or the adaptation of some existing machine learning paradigms. As new paradigms, the Online learning and Transfer learning are promising approaches. As existing paradigms that could be adapted to deal with Big Data, we find the Ensemble learning, the Local learning and the Active learning.

In the context of GP scaling, two paths are considered in this work: the acceleration of evaluations through the processing manipulation or the reduction of their number through the learning paradigm. In the first case, the GP engine is ported over a parallel framework to distribute the fitness computation. In the second case, the GP algorithm is extended with an active learning paradigm implemented with an active data sampling.

3 Scaling by Processing Manipulation: Horizontal Parallelization

The evolution of the Big Data ecosystem has allowed the development of new approaches and tools such as Hadoop MapReduce[3] and Apache Spark (see Footnote 1) that implement a new programming model over a distributed storage architecture. They are the de facto tools for any data intensive application. This section shows how an existing GP implementation is adapted to the Spark context.

[3] https://hadoop.apache.org.

3.1 Spark and MapReduce

MapReduce is a parallel programming model introduced by Dean et al. in [11] for Google. It was made popular with its Apache Hadoop implementation. The fundamental idea that gave rise to this model is to move computations to the data by reducing data traffic between the different nodes. MapReduce operates over a distributed file system.

It applies the "divide and rule" technique to break down a process into multiple tasks performed concurrently on different machines on the portions of data they store. These tasks are of two types: Map and Reduce, that are the functional programming origin. Despite the simplicity of this model, which has allowed to solve several large scale problems, it is not suited to iterative algorithms such as EDMA that are penalized by the large number of I/Os and network latency.

Apache Spark is one of many frameworks intended to neutralize the limitations of MapReduce while keeping its scalability, data locality and fault tolerance. The keystone of Spark is the Resilient Distributed Datasets (RDD) [34]. A RDD is a typical immutable parallel data structure that can be cached. These RDDs support two types of operations: transformations (*map. filter, join, ...*) that produce a modified RDD and actions (*count, reduce, ...*) that generate non-RDD-type results (an integer, a table, etc.) and require all the RDD partitions to be performed. Spark is up to 100 times faster than MapReduce with iterative algorithms (see Footnote 1).

3.2 Parallelizing GP over a Distributed Environment

The implementation of evolutionary algorithms over a distributed environment has taken up light since the emergence of the Big Data ecosystem. Several works have been published in this context. For example, Rong-Zhi Qi et al. [31] and Padaruru et al. [29] propose solutions for parallelizing the entire evolutionary process (fitness evaluation and genetic operators) with Spark. This solution has been applied to automatic software test game generation.

In Chávez et al. [7], the well-known EA library ECJ is modified in order to use MapReduce for fitness evaluations. This new tool is tested to resolve a face recognition problem over around 50 MB of data. Only time measure was considered in this work. Peralta et al. [30] applied the MapReduce model to implement an EA for the pre-processing of big datasets (up to 67 million records and attributes from 631 to 2000). It's a feature selection application where each map creates a vector of attributes on disjoint subsets of the original data base. The Reduce phase aggregates all the vectors obtained.

The implementation used in this paper, introduced in [16], is inspired by the works of Chavez et al. [7] Peralta et al. [30]. It is a solution for modifying an existing tool (DEAP) for the distribution of the training data base using the Spark engine for distributing GP evaluation. The GP loop with the different distribution steps are illustrated by the Algorithm 1. Steps 1, 4, 5, and 6 (blue lines) concern the distribution of the training data set for the population fitness computation. In step 1, we start by creating a RDD containing the training

set. Then, at each generation, a map transformation (step 4) is performed by sending individuals code to be executed (step 5) on RDD and then get results to compute each individual fitness in step 6. Afterwards, GP pursues its standard evolutionary steps.

Algorithm 1. GP + RDD

```
1: TrainingRDD ← parallelize the training set
2: g ← 0
3: for all generation g < g_max do
4:    map: serialize population and map it on the training set (trainingRDD)
5:    TrainingRDD ← compute distributed fitness
6:    Reduce : Compute final fitness (Reduce fitnessRDD)
7:    Update population fitness
8:    Select parents according to fitness
9:    Apply genetic operators and generate new population
10: end for
```

4 Scaling by the Learning Paradigm: Active Learning

Active Learning [2,8] could be defined as: 'any form of learning in which the learning program has some control over the inputs on which it trains.'

Active learning is implemented essentially with active sampling techniques. The goal of any sampling approach is first to reduce the original size of the training set, and thus the computational cost, and second to enhance the learner performance. Sampling training dataset has been first used to boost the learning process and to avoid over-fitting [20]. Later, it was introduced for Genetic learners as a strategy for handling large input databases. With active sampling, the training subset is changed periodically across the learning process. The data selection strategy is based on some dynamic criteria, such as random selection, weighted selection, incremental selection, etc. We distinguish one-level sampling methods using a single selection strategy and multi-level sampling (hierarchical sampling) methods using multiple sampling strategies associated in a hierarchical way.

4.1 One-Level Active Sampling

One-level sampling methods use a single selection strategy based on dynamic criteria. Records in the training subset S are selected before the application of the genetic operators each generation.

To select a training subset $\mathcal{S}$ from a data base $\mathcal{B}$, the simplest technique is to select randomly T_S records from B with a uniform probability. It is the basic approach for the Random Subset Selection method [13] (Sect. 4.1) and some variants like the Stochastic Sampling [28] and Fixed Random Selection [35]. Several other techniques use more sophisticated criteria in order to address some

learning difficulties like over-fitting or imbalanced data. For example, weighted sampling techniques [13] use information about the current state of the training data such as difficulty and the number of solved fitness cases. However data-topology based sampling techniques [15,24] use information about data topology in order to avoid to select similar fitness cases in the same training subset. Balanced sampling [19] and incremental sampling [22] techniques use information about the data classes to handle the problem of imbalanced data. For this work, we are interested in the random (RSS) and balanced (SBS) sampling methods. In this work, the training data base is divided on balanced blocks and SBS and RSS are applied the resulting blocks (Algorithm 2).

Random Sampling (RSS). The simplest method to choose fitness cases and build the training sample is random. The selection of fitness cases is based on a uniform probability among the training subset. This stochastic selection helps to reduce any bias within the full dataset on evolution. Random Subset Selection (RSS) is the first implementation given by Gathercole et al. [13]. In RSS, at each generation g, the probability of selecting any case i is equal to $P_i(g)$ such that:

$$\forall i : 1 \leq i \leq T_B, \quad P_i(g) = \frac{T_S}{T_B}. \tag{1}$$

where T_B is the size of the full dataset B and T_S is the target subset size. The sampled subset has a fluctuating size around T_S.

Balanced Sampling (SBS). The main purpose of balanced sampling is to overcome imbalance in the original data sets. The well known techniques in this category are those proposed by Hun et al. [19] aiming to improve classifiers accuracy by correcting the original dataset imbalance within majority and minority class instances. Some of these methods are based on the minority class size and thus reduce the number of instances. In this paper, we focus on the *Static Balanced Sampling (SBS)*. SBS is an active sampling method that selects cases with uniform probability from each class without replacement until obtaining a balanced subset. This subset contains an equal number of majority and minority class instances of the desired size.

4.2 Multi-level Active Sampling

Multi-level sampling combines several sampling algorithms applied at different levels. Its objective is to deal with large data sets that do not fit in the memory, and simultaneously provide the opportunity to find solutions with a greater generalization ability than those given by the one-level sampling techniques. The data subset selections at each level are independent. The usual schema is made up of three levels. The first one (level 0) consists in creating blocks with a given size from the original data set which are recorded in the hard disk. The remaining two levels are a combination of two active sampling methods. In the

Algorithm 2. GP + 2 Level Active Sampling

Parameters:
$\mathcal{B}$: learning data base
T_S: the subset size

1: Divide $\mathcal{B}$ into blocks B_b *(level 0)*
2: $\mathcal{S}(0) \leftarrow$ Select instances from B_b using RSS or SBS
3: $g \leftarrow 0$
4: **for all** generation $g < g_{max}$ **do**
5: Evaluate individuals against Training Subset $\mathcal{S}(g)$
6: Select parents according to fitness
7: Apply genetic operators and generate new population
8: Generate randomly new data subset $S(g+1)$ using RSS or SBS (level 1)
9: **end for**

present work, a balanced sampling is applied at levels 0 and 1 and a random sampling is applied at level 2.

Algorithm 3 details how the hierarchical sampling is added to the GP loop. At level 0, the data set $\mathcal{B}$ is first partitioned into blocks that are selected randomly for the sampling step. Then, at level 1, an intermediate sample is extracted from the current block using a balanced random selection SBS. Finally, at level 2, the training subset $\mathcal{S}$ is generated randomly from the intermediate sample.

Algorithm 3. GP + 3 Level Active Sampling

Parameters:
$\mathcal{B}$: learning data base
T_{l1}: level 1 maximum iterations
T_{l2}: level 2 maximum iterations

1: Divide $\mathcal{B}$ into blocks *(level 0)*
2: **for all** level 1 iterations $= T_{l1}$ **do**
3: Conduct Block Selection
4: Sample an Intermediate subset using SBS or RSS *(level 1)*
5: **for all** level 2 iterations $= T_{l2}$ **do**
6: $\mathcal{S}(g) \leftarrow$ Conduct Training Subset Selection using RSS *(level 2)*
7: Evaluate individuals against Training Subset $\mathcal{S}(g)$
8: Select parents according to fitness
9: Apply genetic operators and generate new population
10: **end for**
11: **end for**

In addition to the known GP parameters, two new parameters are used by the algorithm: T_{l1} and T_{l2}. T_{l1} and T_{l2} are respectively the number of GP iterations for the first-level sampling and for the second-level sampling. Thus, after each T_{l1} iterations, the level-one training data set is replaced with an other set with the SBS method. This set is used to generate the training sub set $\mathcal{S}$ at each T_{l2} iterations with the RSS approach.

5 Example of Application

This section presents an example of application of the proposed solutions in solving two real world problems. The aim is not to present a detailed experimental study but to give an example of the GP's behaviour when extended with an active sampling or a parallelization over Spark. For this purpose, we have chosen two applications: The Higg's Boson classification and the Pulsar detection problems.

5.1 Application Data Bases

The Higgs Data Base for Boson Detection. A Higgs or Z boson is a heavy state of matter resulting from a small fraction of the proton collisions at the Large Hadron Collider[5]. This heavy state quickly decays into more stable particles, so the intermediate states of matter are not observable by the detectors surrounding the point of collision. Highly faithful collisions are then simulated by the ATLAS Simulator [9] to reproduce the essential measurements provided by the detectors to reproduce the essential measurements provided by the detectors.

ATLAS experiment a portion of the simulated data to optimize the analysis of the Higgs Bosons by machine learning techniques. A subset of this data was presented as a challenge in 2014 [1] in the kaggle platform[4,5].

From the machine learning point of view, the problem can be formally cast into a binary classification problem. The task is to classify events as a signal (event of interest) or a background (event produced by already known processes). Baldi et al. [4] published for benchmarking machine-learning classification algorithms a big data set of simulated Higgs Bosons that contains 11 million simulated collision events and the 28 features. In this work, we propose to handle a subset of the published data to test th different GP implementations.

HTRU Data Set for Pulsar Detection. Pulsars are a rare type of Neutron star that produce radio emission detectable on Earth. They are of considerable scientific interest as probes of space-time, the inter-stellar medium, and states of matter [26]. As pulsars rotate, their emission beam sweeps across the sky, and when this crosses our line of sight, produces a detectable pattern of broadband radio emission. As pulsars rotate rapidly, this pattern repeats periodically. Thus pulsar search involves looking for periodic radio signals. The HTRU(2) dataset, is the publicly available data set[6] that describes a sample of pulsar candidates collected during the High Time Resolution Universe (HTRU) Survey. The goal of the classification process is to classify the given candidates as pulsars or non pulsars.

[4] https://www.kaggle.com/.

[5] See UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/datasets/HIGGS.

[6] https://archive.ics.uci.edu/ml/datasets/HTRU2.

Some Related Works on Higgs and HTRU Data Sets. The first machine learning technique applied to detect pulsars on the Lyon et al. HTRU data set is the Gaussian Hellinger very fast Decision Tree with a precision equal to 89.9% [26]. These results are improved in the study published in [33] where the precision and recall reach respectively 95% and 87.3% with the XG-Boost classifier and 96% and 87% with Random Forest classifier.

Otherwise, the Boson classification problem using the Higgs data set was first handled with Deep Neural Network (DNN) proposed by Baldi et al.[5]. Their method was compared the boosted decision tree and the Shallow Neural Network. They used a subset of Higgs data base of 2.6 million examples and 100K validation examples. They demonstrated how DNN can be trained on such data set with a high degree of accuracy. The whole Kaggle data set with 11 millions patterns has been studied in [16,18] where the best accuracy reach 66.93%.

5.2 Experimental Settings

Software Framework. DEAP (Distributed Evolutionary Algorithms in Python)[7] is presented as a rapid prototyping and testing framework [12] that supports parallelization and multiprocessing. It implements several Evolutionary Algorithms: Genetic Algorithm, Evolution Strategies and GP. The basic module contains objects and data structures frequently used in Evolutionary Computation. We decided to use this framework for the following reasons: (1) it implements standard GP with tree based representation (1), it is a Python package which is one of the 3 languages supported by Spark and (3) it is distributed ready. The third point means that DEAP is structured in a way that facilitates distribution of computing tasks.

Configurations and Performance Metrics. For each data set, five series of tests are performed with different configurations according to the sampling strategy or parallelization strategy, as follows:

- Standard GP: GP is run without active learning or parallelization.
- GP + Spark: GP is parallelized over Spark.
- GP + 2 Level Sampling: GP is extended with a two level sampling using SBS at level 0 and RSS at level 1.
- GP + 3 Level Sampling: GP is extended with a three level sampling using SBS at level 0 and 1 and RSS at level 2.

By the end of each run, the best individual based on the fitness function is evaluated on the test data set. Results are recorded in a confusion matrix from which accuracy is calculated according to the following formula.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ patterns}. \quad (2)$$

[7] https://github.com/deap/deap.

where *True Positives* and *True Negatives* are the numbers of exemplars correctly classified in respectively class 1 and 0. Experiments are performed on an Intel i7 (4 Core) workstation with 16 GB RAM running under *Windows* 64-bit Operating System.

The additional parameters needed for the different configurations are summarized in the Table 1. For these first experiments, the training size is limited 40000 for the case of Boson classification.

Table 1. Additional parameters for the GP runs.

Method	Parameter	Level 0	Level 1	Level 2
Standard GP and GP+Spark	Size	HIGGS: 40000 HTRU: 14000	– –	– –
GP+2 Level Sampling	Size Frequency	HIGGS: 40000 - HTRU:2000 50 g	5000 1	400 –
GP+3 Level Sampling	Size Frequency HTRU Frequency HIGGS	HIGGS:5000 - HTRU:2000 50 g 50 g	1000 5 8	200 2 4

GP Settings. The GP terminal set includes the features of data set benchmarks studied in this work. The GP function set includes basic arithmetic, comparison and logical operators reaching 17 functions. The objective is the minimisation of the classification error. The main GP parameters are summarized in Table 2.

Table 2. GP parameters.

Parameter	Value	Parameter	Value
Population size	200	Generations number	200
Crossover probability	0.5	Mutation probability	0.2
Tournament size	4	GP Tree depth	3

5.3 Results and Discussion

The results of these preliminary experiments are illustrated in Table 3 for the HTRU base and in Table 4 for the HIGGS data base. The recorded metrics are average and best accuracy and average computing time over 10 runs.

The key observation from the obtained results is that the proposed solutions are able not only to reduce the computation cost according to Standard GP results, but also to improve the performance of the classifiers. The reduction in the computational cost is about 50% with active learning and can be reduced

Table 3. Preliminary results on HTRU data set.

Test case	Avg accuracy	Computing time (en s)	Best accuracy
Standard GP	97%	5313,81	97,8%
GP+2L-Sampling	97,3%	764,183	**98,9%**
GP+3L-Sampling	97,2%	751,607	98,7%
GP+RDD	**97,4%**	**428,83**	98,2%

Table 4. Preliminary results on HIGGS data set.

Test case	Avg accuracy	Computing time (en s)	Best accuracy
Standard GP	57%	7918	59,2%
GP+2L-Sampling	53%	2220	56%
GP+3L-Sampling	52%	3417,89	55%
GP+RDD	**57%**	**1278,9**	**61%**

up to 10 times with GP+RDD in the case of HTRU. Likewise, the average and best accuracies are improved with parallelization and active learning in the majority of test cases. According to these first series of tests and considering the best measures, we can state that the extended GP is able to generate better results than the standard GP either for Higgs data set or HTRU data set.

Results for 1M Train Instances from Higgs Database. The GP behavior observed in the first experiments should remain unchanged with the increase of data size. To demonstrate it, some tests were carried out with a train sample of 1million instances from the Higgs data base and a test set of 200000 instances. Due to the limited computational capacity, only few runs are performed and the number of generations is limited to 25 for GP+Spark and to 100 for PG+2/3 L Sampling. The remaining GP parameters still unchanged.

GP+Spark, after 25 generations and about 5 h of computing time, the best test accuracy reach 57,15%. The parallelization of GP over Spark has not only allowed GP to be applied to a very large data set that is impossible to do with a Standard GP, but also to slightly improve the overall performance. For active learning, as for previous experiments, the training set is first divided into blocks of 100,000 instances (level 1). RSS is then applied on these blocks to generate samples with 10000 instances in the case of 2L sampling. For the 3L Sampling, SBS and RSS are applied respectively at level 1 and 2 where the sample sizes are equal of 10000 and 3000. The learning time reach 5h30mn, however the results are quite improved. The accuracy of the resulting classifiers is about 63% for 2L and 61% for 3L. This performance exceeds not only the first results in Table 4 but also some previous published results in the state of the art. However, it is not possible to compare the results of this paper with those of baldi et al. [5] since we use a smaller sample size.

Discussion. This paper presents some trends to address difficulties related to complexity and volume for big data mining with GP as an EDMA. Two solutions are explored, the active sampling with SBS and RSS and the horizontal parallelization with Spark, that are proved to be promising directions to improve classifiers' quality. Indeed, according to the first series of tests summarized above, and considering the best measures, we can state that the extended GP is able to generate better results than the standard GP either for Higgs data set or HTRU data set. Otherwise, when comparing these results with those of the state of the art, it is clear that GP, in the different explored configurations, is a competitive technique able to accomplish similar or better performance.

These preliminary experiments allow us to conclude that the solutions proposed in this work are effective and promising. However, additional experiments are needed to compare the different techniques according to a complete set of metrics such as the recall, precision and F2-score measures. Otherwise, further studies aim to better explore the different solutions and the different possibilities of combination.

6 Conclusion

With the incremental demand to analyze huge amounts of data, resulting from variant sources and generated at very high rates, researchers at different domains have studied the expansion of the existing data mining techniques to cope with the evolved nature of data. In this paper, we provide some trends to address complexity and large data size with evolutionary machine learning. Active sampling and parallelization over Spark are explained and we detailed how they can be implemented into GPs. A first experimental study to detect pulsar and Higgs Boson demonstrates how these extensions help GP to better deal with complex data. Further works aim to propose a framework implementing the different trends discussed in this paper with more comprehensive studies. In particular, implementing active sampling on top of Spark RDDs is an interesting path towards combining the discussed techniques in the same process. Another direction is studying the effect of cluster size on GP performance.

References

1. Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kegl, B., Rousseau, D.: Learning to discover: the Higgs Boson machine learning challenge (2014). http://higgsml.lal.in2p3.fr/documentation
2. Atlas, L.E., Cohn, D., Ladner, R.: Training connectionist networks with queries and selective sampling. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems 2, pp. 566–573. Morgan-Kaufmann (1990)
3. Bacardit, J., Llorà, X.: Large-scale data mining using genetics-based machine learning. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **3**(1), 37–61 (2013)
4. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nat. Commun. **5**, 1–9 (2014)

5. Baldi, P., Sadowski, P., Whiteson, D.: Enhanced Higgs Boson to $\tau+ \tau$- search with deep learning. Phys. Rev. Lett. **114**(11), 111–801 (2015)
6. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: Haloop: efficient iterative data processing on large clusters. Proc. VLDB Endow. **3**(1–2), 285–296 (2010)
7. Chávez, F., et al.: ECJ+HADOOP: an easy way to deploy massive runs of evolutionary algorithms. In: Squillero, G., Burelli, P. (eds.) EvoApplications 2016. LNCS, vol. 9598, pp. 91–106. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31153-1_7
8. Cohn, D., Atlas, L.E., Ladner, R., Waibel, A.: Improving generalization with active learning. Mach. Learn. **15**, 201–221 (1994)
9. ATLAS Collaboration: Dataset from the ATLAS Higgs Boson machine learning challenge 2014 (2014). http://opendata.cern.ch/record/328. https://doi.org/10.7483/OPENDATA.ATLAS.ZBP2.M5T8
10. Cummins, R., O'Riordan, C.: Evolved term-weighting schemes in information retrieval: an analysis of the solution space. Artif. Intell. Rev. **26**(1–2), 35–47 (2006). https://doi.org/10.1007/s10462-007-9034-5
11. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Brewer, E.A., Chen, P. (eds.) 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, 6–8 December 2004, pp. 137–150. USENIX Association (2004)
12. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. **13**, 2171–2175 (2012)
13. Gathercole, C., Ross, P.: Dynamic training subset selection for supervised learning in Genetic Programming. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) PPSN 1994. LNCS, vol. 866, pp. 312–321. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58484-6_275
14. Harding, S., Banzhaf, W.: Implementing cartesian genetic programming classifiers on graphics processing units using GPU. NET. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 463–470 (2011)
15. Hmida, H., Ben Hamida, S., Borgi, A., Rukoz, M.: Hierarchical data topology based selection for large scale learning. In: Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, 2016 International IEEE Conferences, pp. 1221–1226. IEEE (2016)
16. Hmida, H., Ben Hamida, S., Borgi, A., Rukoz, M.: Genetic programming over spark for Higgs Boson classification. In: Abramowicz, W., Corchuelo, R. (eds.) BIS 2019. LNBIP, vol. 353, pp. 300–312. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20485-3_23
17. Hmida, H., Hamida, S.B., Borgi, A., Rukoz, M.: Sampling methods in genetic programming learners from large datasets: a comparative study. In: Angelov, P., Manolopoulos, Y., Iliadis, L., Roy, A., Vellasco, M. (eds.) INNS 2016. AISC, vol. 529, pp. 50–60. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-47898-2_6
18. Hmida, H., Hamida, S.B., Borgi, A., Rukoz, M.: Scale genetic programming for large data sets: case of Higgs Bosons classification. Procedia Comput. Sci. **126**, 302–311 (2018). The 22nd International Conference, KES-2018
19. Hunt, R., Johnston, M., Browne, W., Zhang, M.: Sampling methods in genetic programming for classification with unbalanced data. In: Li, J. (ed.) AI 2010. LNCS (LNAI), vol. 6464, pp. 273–282. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17432-2_28

20. Iba, H.: Bagging, boosting, and bloating in genetic programming. In: Banzhaf, W., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference on GECCO-99, pp. 1053–1060. Morgan Kaufmann, San Francisco (1999)
21. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. Stat. Comput. **4**(2), 87–112 (1994). https://doi.org/10.1007/BF00175355
22. Kuscu, I.: Genetic programming and incremental approaches to solve supervised learning problems (1996)
23. Langdon, W.B.: Graphics processing units and genetic programming: an overview. Soft Comput. **15**(8), 1657–1669 (2011)
24. Lasarczyk, C.W.G., Dittrich, P., Banzhaf, W.: Dynamic subset selection based on a fitness case topology. Evol. Comput. **12**(2), 223–242 (2004). https://doi.org/10.1162/106365604773955157
25. L'Heureux, A., Grolinger, K., ElYamany, H.F., Capretz, M.A.M.: Machine learning with big data: challenges and approaches. IEEE Access **5**, 7776–7797 (2017). https://doi.org/10.1109/ACCESS.2017.2696365
26. Lyon, R.J., Stappers, B.W., Cooper, S., Brooke, J.M., Knowles, J.D.: Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. Mon. Not. R. Astron. Soc. **459**(1), 1104–1123 (2016). https://doi.org/10.1093/mnras/stw656
27. Maitre, O.: Genetic programming on GPGPU cards using EASEA. In: Tsutsui, S., Collet, P. (eds.) Massively Parallel Evolutionary Computation on GPGPUs. NCS, pp. 227–248. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37959-8_11
28. Nordin, P., Banzhaf, W.: An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. Adapt. Behav. **5**(2), 107–140 (1997). https://doi.org/10.1177/105971239700500201
29. Paduraru, C., Melemciuc, M., Stefanescu, A.: A distributed implementation using apache spark of a genetic algorithm applied to test data generation. In: Genetic and Evolutionary Computation Conference, 15–19 July, Companion Material Proceedings, pp. 1857–1863. ACM (2017)
30. Peralta, D., et al.: Evolutionary feature selection for big data classification: a mapreduce approach. Math. Prob. Eng. **2015**, 11 (2015)
31. Qi, R., Wang, Z., Li, S.: A parallel genetic algorithm based on spark for pairwise test suite generation. J. Comput. Sci. Technol. **31**(2), 417–427 (2016)
32. Vanneschi, L., Poli, R.: Genetic programming - introduction, applications, theory and open issues. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 709–739. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_24
33. Wang, Y., Pan, Z., Zheng, J., Qian, L., Li, M.: A hybrid ensemble method for pulsar candidate classification. Astrophys. Space Sci. **364**, 1–13 (2019). https://doi.org/10.1007/s10509-019-3602-4
34. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, 25–27 April, pp. 15–28. USENIX Association (2012)
35. Zhang, B.T., Joung, J.G.: Genetic programming with incremental data inheritance. In: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, USA, 13–17 July 1999, vol. 2, pp. 1217–1224. Morgan Kaufmann (1999). http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GP-460.pdf