IMAGE LICENSED BY INGRAM PUBLISHING

Bring Your Own Learner! A Cloud-Based, Data-Parallel Commons for Machine Learning

Abstract—We introduce FCUBE, a cloud-based framework that enables machine learning researchers to contribute their learners to its community-shared repository. FCUBE exploits data parallelism in lieu of algorithmic parallelization to allow its users to efficiently tackle large data problems automatically. It passes random subsets of data generated via resampling to multiple learners that it executes simultaneously and then it combines their model predictions with a simple fusion technique. It is an example of what we have named a *Bring Your Own Learner* model. It allows multiple machine learning researchers to contribute algorithms in a plug-and-play style. We contend that the *Bring Your Own Learner* model signals a design shift in cloud-based machine learning infrastructure because it is capable of executing anyone's supervised machine learning algorithm. We demonstrate FCUBE executing five different learners contributed by three different machine learning groups on a 100 node deployment on Amazon EC2. They collectively solve a publicly available classification problem trained with 11 million exemplars from the Higgs dataset.

Digital Object Identifier 10.1109/MCI.2014.2369892
Date of publication: 14 January 2015

I. Introduction

Technological advances in storage paired with the more frequent practice of large scale archiving of internet and business transaction data have resulted in an explosion in the number of massive datasets, even in domains previously unstudied using data driven approaches [1].

To learn predictive models from these massive datasets it has become convenient to develop platforms that scale and provide access to multiple machine learning algorithms and approaches.

1556-603X/15©2015IEEE

Ignacio Arnaldo, Kalyan Veeramachaneni, Andrew Song, and Una-May O'Reilly
Anyscale Learning For All (ALFA) Group,
Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institue of Technology,
Cambridge, Massachusetts, USA

Examples include [2], [3], and [4]. The design strategy for these platforms to date has been to parallelize popular machine learning algorithms to take advantage of large quantities of computational resources. The strategy ignores, however, the prolific nature of machine learning research, that is, new algorithms are always being designed and existing ones are continuously improved over time.

With these circumstances in mind, we have developed the FCUBE project. It aspires to an ambitious yet attainable vision: a machine learning commons, demonstrated with evolutionary computation-based learners. First of its kind, FCUBE creates a commons where machine learning researchers bring their approaches to be integrated into a large-scale, cloud-based, data-parallel framework. FCUBE thus exemplifies what we call a *Bring Your Own Learner* model.

Three different core technological components have enabled us to realize a *Bring Your Own Learner* model. First, virtualization improved the transferability and reuse of software applications by introducing virtual machines that can be set up with a complete execution environment (e.g. operating system, libraries, permissions, etc.). We use a virtual machine to encapsulate and integrate multiple learning approaches. Second, cloud computing provides anywhere, anytime access to scalable compute and storage, while it allows applications to be easily deployed to hundreds or even thousands of nodes. Third, our design embraces a data parallelization approach that is an inherently scalable and robust way to address big data.

The name FCUBE is coined from the three F's in *Factor, Filter and Fuse* (F^3). FCUBE exploits a data parallel approach that uses a randomly reduced (*factored*) subset of the data to simultaneously train each of a set of independent learners. This factoring strategy alleviates the cost of executing each individual learner on all the data and allows FCUBE to offer shorter waiting times. It offers two noteworthy additional advantages. First, on the particular problem, though it cannot be pre-determined, one learner is likely to be superior to the others. FCUBE relatively effortlessly identifies which one is the best. Second, by combining learners with different designs using *model fusion*, FCUBE offers an ensemble-based solution that is likely to outperform any individual one in terms of accuracy.

From the point of view of data scientists or other engineers needing a data driven model, it is possible to bring a dataset and a machine learning problem to FCUBE to be solved by its multiple learners simultaneously. The FCUBE project seamlessly connects learners with data and allows large data problems to be tackled efficiently upon cloud computing infrastructure. It makes it relatively effortless for a user to upload a dataset and, almost invisibly, harness cloud computing technologies, multiple learning algorithms and model representations to automatically generate massive multi-algorithm ensembles.

II. Contributions and Challenges

Our framework makes the following contributions:

A massive data-parallel approach: FCUBE is one of the first cloud-based, publicly available, data parallel solutions for large datasets [5]. The *Factor, Filter and Fuse* approach tackles large datasets and offers short waiting times.

Bring your own learner: FCUBE is the first standardized cloud-based machine learning platform where researchers can bring their supervised learning algorithms and straightforwardly plug them into the platform. It is designed to deploy any stand-alone learner as long as the learner is compliant with an undemanding input/output specification. Once added to the framework, the learner becomes a part of the community of learners, thus enabling researchers:

❏ to obtain automatic, *massive* deployment on the cloud and access to FCUBE's data factoring (via bootstrap aggregating) and variable factoring (via random subspace method),
❏ to become a part of an *ensemble* used for solving real world problems,
❏ to contrast and compare themselves with other approaches.

Bring your data and problem use case: FCUBE is envisioned to be used in the following scenarios:

❏ practitioners who have application specific problems based upon a large data set can use FCUBE with no requirement to be conversant in evolutionary computation or machine learning.
❏ researchers can organize a collaborative or competitive activity around a specific problem and its dataset.
❏ data scientists can organize data science camps around different pairs of problems and datasets using FCUBE as their machine learning engine.

Challenges

We addressed three primary challenges. Our first challenge involved defining standardized input/output interfaces for learners such that the *learning algorithm* and the cloud deployment layer were decoupled. Our goal was to enable integration in a *plug-and-play* fashion and enable the integration of learners coded in a variety of popular programming languages (Java, Python, C++, etc). We first validated FCUBE's learner interfaces by piloting them with a small group of researchers. They were good matches for the learners' software design and algorithmic logic. This allowed us to refine them to make them even more general.

Our second challenge required us to design a fusion technique that can work in classifier output space so that we could integrate heterogeneous learners. FCUBE allows a learner's classifier to either produce continuous values or produce a label. Continuous results are then standardized to labels by applying a decision rule. We implemented and evaluated a number of decision level fusion techniques and finally decided upon *logistic regression*.

Our third challenge involved management of data when a large number of learners are deployed in a data-parallel manner. It is a challenge to efficiently supply all learning algorithms with different subsets of the training data, i.e. factor the data. To support factoring, FCUBE implements a distributed data factoring service that is efficient but hidden from both FCUBE's contributors and users.

III. Related Work

Ensemble-based learning strategies have been accomplished previously in [6]–[8]. In [6], authors generate multiple models from the *same* learner by providing different subsets of data. These solutions are customized for a specific learner that the researchers use. The approaches are also developed to enable the learning of an ensemble on a specific compute cluster. However, frameworks capable to learn 100's of models on the cloud from a pool of heterogeneous learners are not readily available.

There have been efforts to enable evolutionary computation on parallel compute infrastructure. One such effort is EAsy Specification of Evolutionary Algorithms EASEA [9], where the authors claim that the user only needs to write some problem-related code. The framework is known to run on clusters and GPUs but to our knowledge does not run on the cloud. Another recent approach builds a framework called Distributed Evolutionary Algorithms in Python (DEAP) [10]. The platform is not as integrative as FCUBE because it was not designed for distribution on the cloud.

Several machine learning systems are being developed to run on the cloud and provide solutions for data science problems. Examples include [2], [3], and [4]. All these systems rely on learners provided (or assembled) by the system developers themselves and do not allow researchers to contribute to the core learning algorithm repository. We embrace an extensible approach that enables multiple researchers to contribute their learners believing that the result will be unique and diverse, ensemble learning systems.

Comparison frameworks are used to assess the performance of different applications, and can speed up their development and validation processes. One example is [11], a framework for comparing optimization algorithms in the context of logistics. However, most frameworks impose restrictions such as programming language or parallelization strategy among others. Our insight is to provide, through virtualization, large-scale resource access without the user needing to be concerned about managing it. In FCUBE, contributions of external developers are totally decoupled from the framework software layer. This enables the integration of stand-alone approaches in a plug-and-play manner. Moreover, the hardware specifications of the virtual machine where the framework will be executed can be customized.

IV. FCUBE Architecture

In this section, we give an overview of FCUBE's functionality and introduce its different components. As depicted in Fig. 1, domain users bring new problems and interface with an FCUBE Server to generate massive data-parallel ensembles. The FCUBE deployment server launches a number of cloud nodes with a variety of learners executed with different algorithm parameters. Nodes sample data randomly from the data server. Once learning is finished each learner provides the model back to the FCUBE server. The server then learns a meta-model (fused) using portion of data set aside for this, validates the model and outputs the final model to the domain user. The repository of learners is composed of the algorithms contributed by machine learning researchers.

A. FCUBE Components

1) *Learners:* FCUBE deploys stand-alone learners compliant with an input/output specification.
2) *FCUBE image:* A cloud image or snapshot contains all the learner executables and the logic that instantiates a learner.
3) *FCUBE Server:* An FCUBE server is responsible for deploying FCUBE instances, retrieving models, fusing models, and evaluating the resulting meta-model. The server can be instantiated from a publicly available software repository.

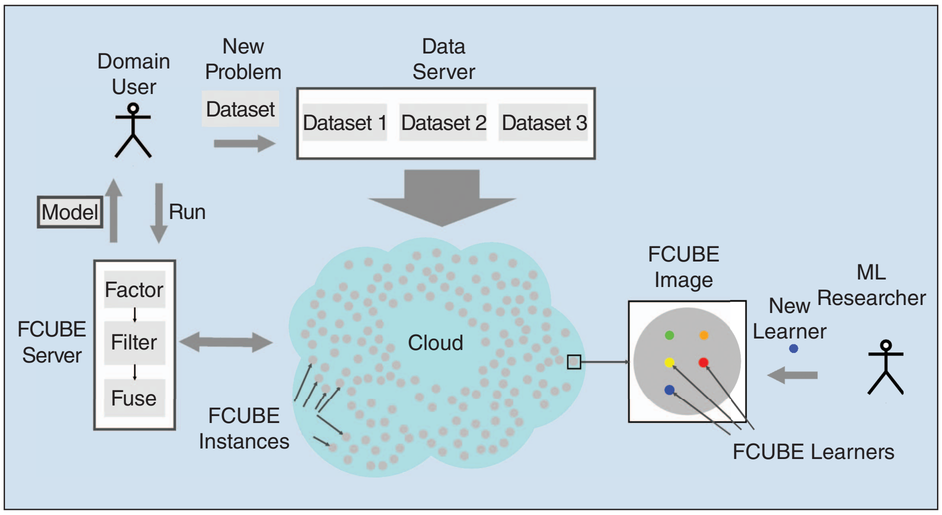

FIGURE 1 FCUBE commons' typical use case.

4) *FCUBE Data Server:* Data are stored in a remote storage accessible from FCUBE instances. Several architectural solutions are viable. The EC2 version of FCUBE uses Amazon's Simple Storage Service (S3). S3 is scalable both in size and throughput, eliminating the bottleneck introduced by the usage of traditional distributed file systems like NFS[1]. Additionally, S3 on Amazon AWS provides a drag-and-drop interface enabling external users to easily upload new datasets.

FCUBE exemplifies what we call a Bring Your Own Learner model. It creates a commons where machine learning researchers add their stand-alone methods to the framework in a plug-and-play manner.

B. FCUBE Setup

Given a dataset D, before deployment we generate the splits $D_{tr_1}, \ldots, D_{tr_n}, D_f$, and D_{te}, corresponding to training splits, fusion data, and test data respectively. The learners executed in the same run access the training splits stored on the data server. Splitting the training data reduces the network traffic (and therefore the cost) of data transfers from the data server to the FCUBE instances. The data employed for fusion training (D_f), and for testing (D_{te}) is accessed by the FCUBE server.

C. Deployment Sequence

The user only needs to interface with the FCUBE server. Four arguments need to be provided, namely number of FCUBE instances, learner name(s), duration of the learning process, and the *parameter options* file (see [5]). The deployment sequence depicted in Fig. 2a then takes place, but is hidden to the user.

1) The server wakes up the instances in batches and waits until the instances respond.
2) The server broadcasts the *parameter options* file to the instances.
3) The server remotely triggers FCUBE's factoring functionality and the execution of a learner.
4) At each instance, the FCUBE factoring service generates both a sample of the data and set of parameters.
5) The resulting *factored data* and *factored parameters* are passed as arguments to the specified learner and the learning process starts.

Once all the FCUBE instances have finished learning, the sequence depicted in Fig. 2b takes place.

1) The server retrieves the final model generated at each node.
2) The server employs three processes: *model output calculation, filtering* and *fusion* (see Section V).

Running on-demand instances introduces a delay in the execution since instances need to be booted up. We have observed that the time necessary to raise these instances varies greatly from a cloud environment to another. In particular CSAIL's OpenStack-based private cloud for development is far slower than Amazon EC2. In the latter, raising 100 instances in batches of 25 takes approximately 15 minutes. On the other hand, broadcasting the *parameter options* file to all the instances does not introduce any significant delay, since the size of the transferred file is at most a few kilobytes. The proposed architecture is highly scalable since the factoring and the majority of the learning process take place at the instances, and therefore is executed in parallel.

[1]The development version of FCUBE used in our OpenStack private cloud employs a dedicated NFS volume to store the data. While this solution is straightforward, in practice it limits the scalability of our system. In fact, we have experienced contention issues when many running instances try to mount and access the NFS volume concurrently.

V. FCUBE: Factor, Filter, Fuse

In this section we provide the detailed description of FCUBE's factoring, filtering, and fusion. We use the following notation:

Notation: $X_1 \ldots X_n \rightarrow$ *explanatory* variables, $P_i \rightarrow$ empirical *prior* for class $i, C_{ij} \rightarrow$ *cost* of declaring class i when it is class $j, P_{ij} \rightarrow$ probability of declaring class i given class $j, R \rightarrow$ *Bayesian* risk function $= \Sigma_{i,j} C_{ij} \times P_{ij}, \forall i \neq j$.

A. Factoring as a Service

Factoring is what we call the *generation* of multiple learners each with its own subset of data, explanatory variables, and parameters. This process is governed by the *parameters options* (see example in Table 1).

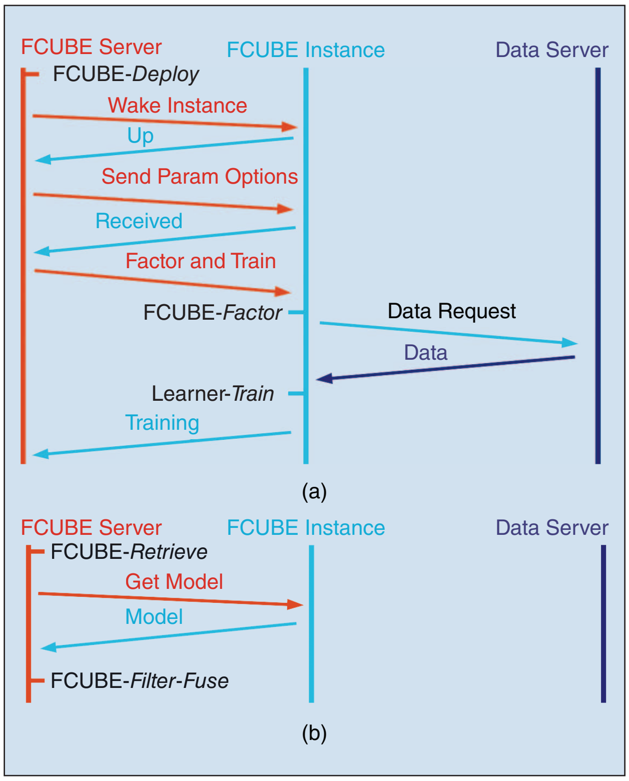

FIGURE 2 FCUBE flow: (a) deployment sequence and (b) model retrieval and fusion sequence.

TABLE 1 Parameters options file: an example of *data factoring* and *parameter factoring* configuration. For each exposed parameter, a default value and a range of choices can be specified.

	PARAMETER	TYPE	CHOICE TYPE	DEFAULT	CHOICES
DATA FACTORING	DATA	STRING	FIXED	FCUBE/TRAIN_FOLDER	–
	DATA SAMPLE RATE	FLOAT	DISCRETE	0.1	{0.1, 0.2}
	VARIABLE SAMPLE RATE	FLOAT	DISCRETE	1.0	{0.5, 0.75, 1}
PARAMETER FACTORING	FALSE NEGATIVE WEIGHT	FLOAT	RANGE	0.5	[0.4:0.01:0.6]
	MUTATION RATE	FLOAT	RANGE	0.1	[0.05:0.05:0.2]
	CROSSOVER RATE	FLOAT	RANGE	0.7	[0.5:0.05:0.85]
	POPULATION SIZE	INT	DISCRETE	1000	{1000, 5000}
	TOURNAMENT SIZE	INT	DISCRETE	2	{2, 7, 10}

Data factoring: The *parameters options* file provides the path to the remote data storage directory (NFS or S3 bucket name) where the training data are stored. FCUBE's factoring service randomly picks one training split and transfers it to the instance storage. Then, a stochastic sampling process takes place where both exemplars and explanatory variables are factored according to the *data_sample_rate* and *variable_sample_rate* parameters. The first specifies the ratio of exemplars that will be sampled from the data, while the second specifies the ratio of the explanatory variables. It is important to note that we keep track of the variables used for training in each of the instances. This information will be necessary later to evaluate the generated models on unseen data.

Parameter factoring: The user decides what *parameters* will be factored (parameters for which a value will be sampled stochastically from the possible ranges/choices) and what *parameters* will be set to their default value. As mentioned in the previous section each FCUBE instance has built-in factoring functionality that will parse the *parameters options* file and generate a configuration (properties) file for the learner during the deployment phase.

For further details and examples of data and parameter factoring, the reader is referred to FCUBE's website[2].

B. Filtering and Fusion

Once each node on the cloud has finished learning, the final model generated at each node is retrieved and used to build a meta-model. While data parallel approaches avoid the complexity of synchronization during the learning process, the fusion step is not without its own set of challenges. FCUBE provides a means of learning a fused model in the models' output space, that is, *predicted labels*. To this end, the following three processes take place at the FCUBE server: *models' output calculation, filtering* and *fusion*.

1) Calculate the models' outputs: In this step, we calculate the outputs (predicted labels) of the models retrieved from the FCUBE run when evaluated on the split set aside for fusion training D_f. To evaluate a model on unseen data, two steps are necessary.

1) First, we need to trim D_f such that the variables used for model evaluation match the variables used for training. To perform this operation, the information logged during the *data factoring* process is retrieved and the variables of D_f are sampled accordingly. Note that, although this process introduces a computational overhead, it also eliminates the need to impose variable labels or identifiers, and thus provides higher flexibility for the integration of learners in the system.
2) Once the data have been processed, the model is evaluated. As a result, we obtain a prediction or label for each exemplar in D_f.

This process is repeated for all the models retrieved from the FCUBE instances. Once all the models have been evaluated, the result is the matrix of model outputs L_f of n rows and $p+1$ columns, where n is the number of exemplars in D_f and p is the number of models retrieved from the FCUBE run. The $(p+1)$-th column corresponds to the true labels and is identical to the last column of D_f.

2) Filtering: The *filtering* process consists of discarding the models that have a performance lower than a pre-set baseline. The baseline performance for classification corresponds to the cost incurred by a naive classifier that always predicts the majority class:

$$\begin{cases} \text{baseline} = C_{01}, \text{if } P_1 > P_0 \\ \text{baseline} = C_{10}, \text{if } P_1 < P_0 \end{cases} \tag{1}$$

The split D_f set apart for fusion training is used to assess the cost (*empirical* Bayesian risk) of each classifier. Note that D_f is unseen by the learner during training. The cost is computed as a weighted sum of the false positive and false negative rates:

$$\text{cost} = C_{01}P_{01} + C_{10}P_{10} \tag{2}$$

All the classifiers exhibiting a lower cost than the baseline are used to build a *meta* model via training.

3) Fusion or Learning a meta-model: Fusion is the final step of the FCUBE flow. It generates a fused model and its test set performance metrics. Most classification approaches rely on *majority vote*. Using the majority vote has a major drawback: it does not weight the individual models' *accuracies* or the *correlations* between models. Our fusion method implicitly enables us to consider the *performance* of individual classifiers and the *dependence* between the classifiers. We proceed as follows:

Step 1: train the meta-model: We run logistic regression on the matrix of model outputs L_f, thus obtaining the *meta-model* θ.

Step 2: model evaluation on test data: We evaluate the filtered models on the testing set D_{te}. As a result, we obtain

[2]http://flexgp.github.io/FCUBE

matrix of model outputs L_{te} (one row for each exemplar in D_{te} and one column for each filtered classifier).

Step 3: applying the meta-model: We apply the meta-model θ to L_{te} and report its performance metrics.

VI. A Repository of Algorithms for Collaborative Learning

A. Bring Your Own Learner

FCUBE enables machine learning researchers to incorporate their stand-alone algorithms in the framework. To ease this process and incur minimal overhead on their part, we treat learning algorithms as black boxes totally decoupled from the code of the framework. Learners must, however, be compliant with a standardized specification, i.e. a predetermined list of input parameters and expected outputs. Learners need to provide functionality to accomplish two use cases: *train* and *predict*. These two interfaces are depicted in Fig. 3. In the first case, learners take as inputs a path to a dataset, the duration of the learning process, and a properties file containing any additional parameters (see [5] for details) and output a file containing exactly one model. The developer of the learner chooses the format of the output model file. Regarding the *predict* use case, we expect to obtain a file containing predictions given a data path, a model file, and the destination path for the predictions.

B. A Collaborative Platform for Learning

One of the appealing aspects of this platform is that it provides a scaling service to algorithm developers that might not have the infrastructure necessary to tackle large-scale problems. Another positive aspect is that it allows learner comparison on the basis of a fixed computational budget. However, the final goal of this project goes beyond that of providing a framework where collaborators can deploy their algorithms. Instead, we aim at uniting the efforts of worldwide developers of innovative learning algorithms to solve relevant problems of public domain.

With this idea in mind, we organized a collaborative Big Learning activity around FCUBE. The activity was part of the first edition of the EC for Big Data and Big Learning workshop held at GECCO 2014 [12]. Its call for participants invited collaborators to join a community that would solve large-scale problems of public interest with FCUBE. We received four learning algorithms from external collaborators. These algorithms were successfully added to the platform; over time, we expect the population of learners to keep increasing. The massive deployment of these learners with FCUBE enabled us to solve two problems based on large-scale datasets. Moreover, we showed that, by fusing the predictions of diverse algorithms, we can obtain better performance than that of individual algorithms alone. This evidence should convince others in the machine learning community to contribute their efforts. We hoped that the workshop would retrospectively become regarded as the kickstart toward *an automatic collaborative means of solving new problems*.

The key to the success of this initiative was that both the workload of collaborators and their interaction with the FCUBE team (a team within the ALFA group at MIT) was minimal. We restricted the role of collaborators to only adapt their learners FCUBE's standardized input/output specification (Fig. 3). The FCUBE team integrated the learners in the framework, performed all the factor, filter, fuse process and provided performance metrics. This is *Deployment as a Service (DaaS)* and is depicted in Fig. 4. Further details about the collaborative learning activity can be found in [12].

The activity allowed us to populate the repository of learners with the contributions from participants. In a future step, we will make a call for researchers of different domains to "bring their problem and dataset" and use FCUBE's learners to perform large-scale learning tasks. Researchers will greatly benefit from a learner factoring framework which gives them access to publicly available learners. They will be able to easily

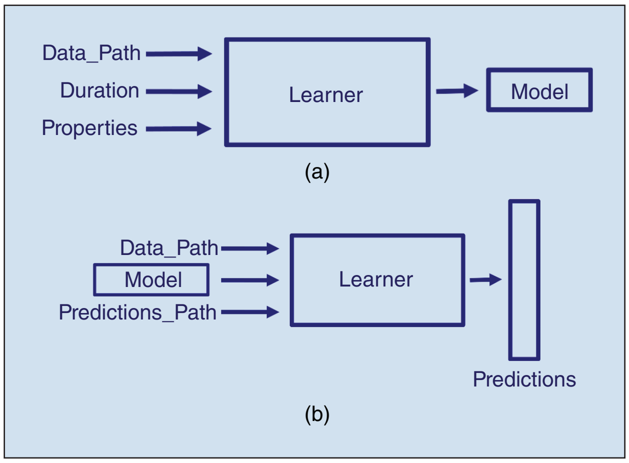

FIGURE 3 Learner black box specification for the (a) *train* use case and (b) *predict* use case.

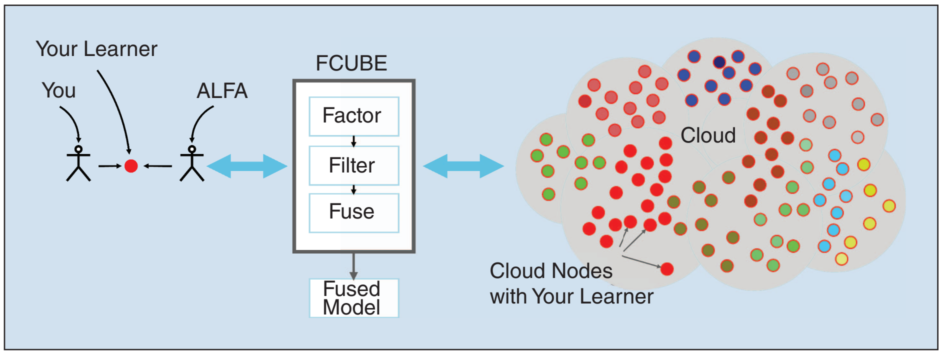

FIGURE 4 Collaborative Big Learning Activity taking place within the first edition of the EC for Big Data and Big Learning workshop, GECCO 2014. Participants simply provide stand-alone executables of their learners.

solve their problem by broadly exploring supervised machine learning algorithms. FCUBE provides them with a means of using the cloud's scalable and budget-flexible resources to meet a short term deadline or to speed up the process of obtaining a baseline result for their new problem.

C. Learning Algorithms

In this paper, we demonstrate the framework with five learners that are meant to be representative of the wide variety of learners existing in the EC community. Three of them, namely Rule List, Rule Tree, and GP Function, have been developed by the FCUBE team. Rule List and Rule Tree employ respectively a Genetic Algorithm and Genetic Programming to learn a rule-based classifier. On the other hand, GP Function performs a search in the space of discriminant functions with Genetic Programming. The remaining two algorithms were provided by external collaborators who kindly shared their software to test our framework before the learning activity period. In the following, we present these five learning algorithms.

1) Rule List Classifier: The Rule List classifier implements a Genetic Algorithm to search in the space of binary classifiers. Candidate solutions are lists containing one condition or rule for each explanatory variable of the problem.

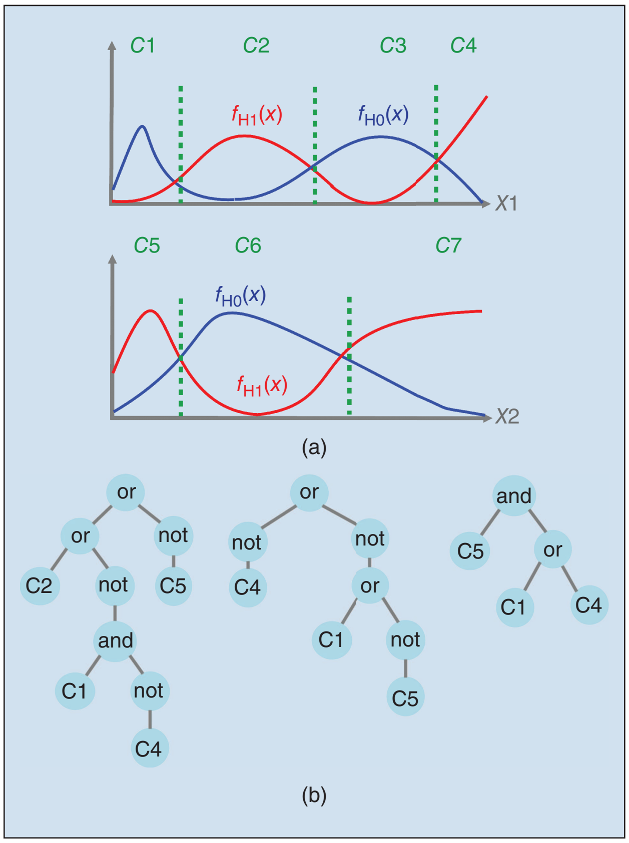

FIGURE 5 Rule Tree Classifier: preprocessing step and representation. (a) Obtaining the conditions for variables x_1 and x_2. In this example, four conditions (c_1, c_2, c_3, c_4) are retrieved for the first variable while the analysis of the second variable results in the three conditions c_5, c_6, c_7. (b) Examples of boolean rules coded with GP trees.

Representation and Initial Population: Individuals are defined by the tuple $I = \{R, A, S\}$ where

- ❑ R is a *list* containing one rule for each explanatory variable of the problem. Each rule is written as $x_i \leq c_i$ or $x_i > c_i$ where c_i is a constant and is in the range $c_i \in [x_i^{\min}, x_i^{\max}]$.
- ❑ A is a boolean vector such that the ith position determines whether or not the ith rule is applied to x_i.
- ❑ S is a strategy in $\{DNF, CNF, MV\}$ used to combine the outcomes of the *active rules* in the candidate solution.

DNF, CNF, and MV stand for *disjunctive* normal form, *conjunctive* normal form, and *majority* vote.

Each individual of the initial population is created as follows:

1) For each variable x_i, we choose randomly a value c_i from the interval $[x_i^{\min}, x_i^{\max}]$ and a relational operator in $\{<, >\}$ to build the initial condition $x_i < c_i$ or $x_i > c_i$. Then, we flip a coin to decide if the ith rule is *active*.
2) We select randomly one of three possible strategies, namely *disjunctive*, *conjunctive* normal form, or majority vote.

Evaluation: The prediction issued by a given individual depends on its voting strategy. In the CNF case, all the active rules need to be satisfied to emit a positive prediction. The majority vote strategy defines that at least half of the active conditions need to be satisfied to predict a positive class. Finally, a single satisfied condition triggers a positive prediction in the DNF case. We evaluate the individual for each exemplar of the dataset and compute its fitness as the Bayesian Risk introduced in (2).

2) Rule Tree Classifier: The training process of the Rule Tree classifier is divided into two steps. In a *preprocessing* step, a set of conditions in the form of $a \leq x_i \leq b$ are determined for each explanatory variable. In the second step, a Genetic Programming strategy is adopted to search in the space of boolean rules using the generated conditions as leaves of the GP trees.

Preprocessing Step: The conditions for each explanatory variable of the problem are obtained independently. Given a variable x, we proceed as follows:

1) The probability density function of the variable when conditioned on the two classes is estimated via nonparametric Kernel Density Estimation (KDE). Given a variable x and a class H_j, the probability density function of the distribution $p(x \mid H_j)$ at a point y is estimated as follows:

$$\hat{f}_{H_j}(y) = \frac{1}{n}\sum_{i=1}^{n} K_h(y - x_j(i)) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{y - x_j(i)}{h}\right) \quad (3)$$

where x_j is the subset of values of x with label H_j and n is the size of x_j. We employ the Gaussian basis function $K(u) = \left(1/\sqrt{2\pi}\right)e^{-\frac{1}{2}u^2}$ and set the bandwidth parameter h to $\left(4\hat{\sigma}^5/3n\right)^{\frac{1}{5}} \simeq 1.06\hat{\sigma}n^{-\frac{1}{5}}$, where $\hat{\sigma}$ is the standard deviation of x_j.

2) We then discretize the range of values of the variable x into m equal partitions and compute $\hat{f}_{H_0}(x)$ and $\hat{f}_{H_1}(x)$ at each of the m steps.
3) As depicted in Fig. 5a, a new condition in the form $a \leq x \leq b$ is created every time the estimated density functions $\hat{f}_{H_0}(x)$ and $\hat{f}_{H_1}(x)$ cross each other.

At the end of the preprocessing, we obtain a set of conditions $C = \{c_1, c_2, \ldots, c_k\}$, where each condition refers to a unique interval for one of the explanatory variables.

Representation and Initial Population: As illustrated in Fig. 5b, candidate solutions are Boolean expressions built with the set of operators $\{and, or, not\}$. These expressions are coded as GP trees where the leaves correspond to the conditions $C = \{c_1, c_2, \ldots, c_k\}$ retrieved in the preprocessing step. The population is initialized with a ramped half-and-half strategy.

Evaluation: The implemented multi-objective approach is based on Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) [13] and targets both *performance* and *complexity*. Classifier performance is calculated by evaluating (2). The *complexity* measure employed is the Subtree Complexity introduced in [14].

3) GP Function Classifier: The third binary classifier, first presented in [15], also implements a multi-objective Genetic Programming strategy based on NSGA-II. Given a set of explanatory variables $\bar{X}$, we search for a nonlinear function $y = f(\bar{X})$ such the distributions $p(f(\bar{X}) \mid H_0)$ and $p(f(\bar{X}) \mid H_1)$ are best separated. After learning the function, a threshold λ is used to build the decision rule

$$\hat{L}_i = \begin{Bmatrix} 1, \text{ if } y_i \geq \lambda \\ 0, \text{ if } y_i < \lambda \end{Bmatrix}$$

that determines whether a given output represents a class 0 or class 1 prediction. This approach is graphically depicted in Fig. 6a.

Representation and Initial Population: As in the Rule-Tree classifier, individuals are coded with GP trees and the population is initialized with a ramped half-and-half strategy. However, in this case the leaves of the trees are variables of the problem and individuals are numerical expressions built with the following set of operators:

$$\text{ops} = \begin{Bmatrix} +, -, /, *, \sin, \cos, \log, \\ \exp, \text{square}, \text{cube}, \text{sqrt} \end{Bmatrix} \quad (4)$$

Evaluation: Since the goal is to find the model with the highest discriminatory power between the two classes, the area under the ROC curve is used as fitness function to guide the search. Simultaneously, we minimize the complexity of the models to prevent bloating issues. Here in we present the steps involved in evaluating the fitness function.

Area under the ROC Curve: To compute the area under the ROC curve, we evaluate the model for each exemplar i of the dataset and vary the threshold λ in the decision rule

$$\hat{L}_i = \begin{Bmatrix} 1, \text{ if } y_i \geq \lambda \\ 0, \text{ if } y_i < \lambda \end{Bmatrix}.$$

We then evaluate the two errors for each threshold and compute the model's area under the ROC curve. We proceed as follows:

1) Evaluate the GP tree on the training exemplars resulting in $y_{1\ldots n}$.
2) Retrieve the $y_{\max}$ and $y_{\min}$ values for the model outputs $y_{1\ldots n}$.
3) Normalize the outputs of the model with the obtained boundaries.
4) Vary the threshold $\lambda \in [0;1]$ and apply the decision rule as above.
5) Obtain the False Positive and True Positive rates for each value of λ
6) Compute the area under the ROC curve with the obtained rates.

Complexity: As in the Rule Tree classifier, we employ the Subtree Complexity measure.

Threshold selection: The last step consists in identifying the threshold λ for the best model $f(\bar{X})$ such that the decision rule

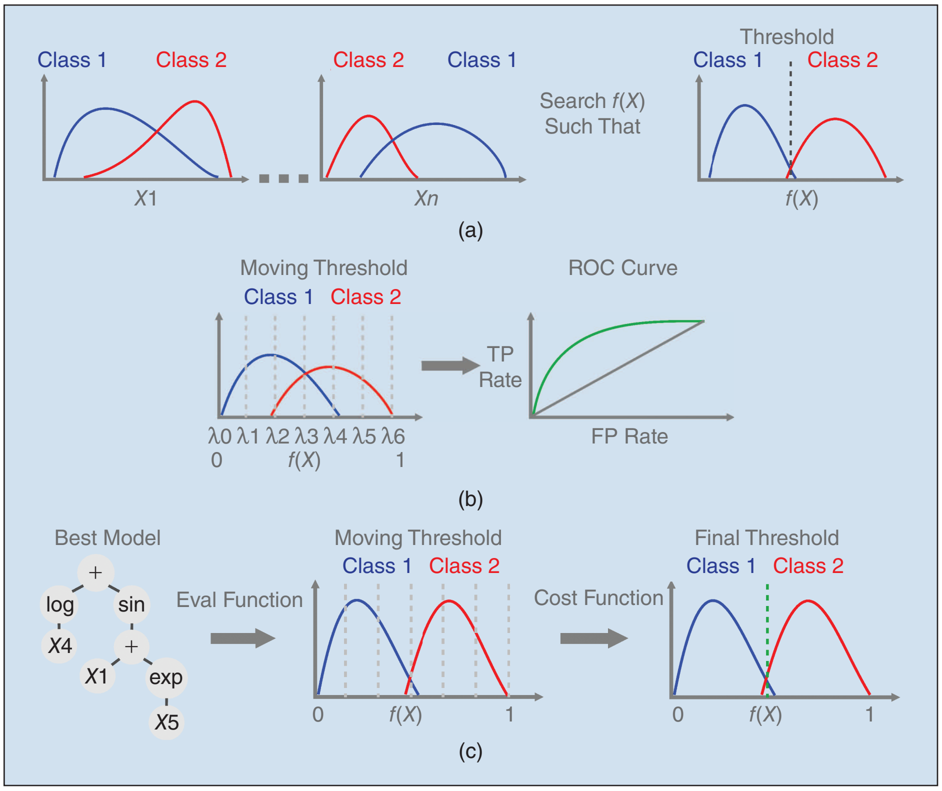

FIGURE 6 GP Function classifier: (a) search for the nonlinear function $f(\bar{X})$ such the distributions $p(f(\bar{X}) \mid H_0)$ and $p(f(\bar{X}) \mid H_1)$ are best separated. (b) GP Function evaluation: a moving threshold is used to compute the area under the ROC curve. (c) Threshold selection for the final model.

$$\hat{L}_i = \begin{Bmatrix} 1, \text{ if } y_i \geq \lambda \\ 0, \text{ if } y_i < \lambda \end{Bmatrix}$$

minimizes the weighted sum of the false positive and false negative rates as in Eq.(2). This is achieved by performing a grid search over λ. This process is carried out post-hoc after the classifier is trained.

4) Memetic Pittsburgh Learning Classifier System: Memetic Pittsburgh Learning Classifier System (MPLCS) is a learning algorithm based on the GAssist Pittsburgh Learning Classifier System. MPLCS evolves variable-length rule sets. This version of the algorithm incorporates a variety of efficiency enhancement mechanisms. In particular, the algorithm implements a memetic multi-parent crossover operator. The reader can find extensive documentation of this algorithm in [16]–[18].

5) Symbiotic Bid-Based Genetic Programming: Symbiotic Bid-Based Genetic Programming (SBB) is a learning algorithm that co-evolves a classifier population and a population of exemplars. Classifiers are grouped into *teams* (*ensembles*) that emit predictions according to a sophisticated bidding strategy. For an extensive description of this algorithm, the reader is referred to [19]–[22].

VII. Experimental Work

A. Higgs Dataset

We present our results on the Higgs dataset [23]. The Higgs dataset is a recently released binary classification problem with 11 million exemplars generated via Monte Carlo simulations. The goal is to differentiate cases where the Higgs boson is produced and cases corresponding to a background process. It contains 28 features, where the first 21 are kinematic properties measured by the particle detectors in the accelerator. The last seven features are higher level features proposed by domain experts, and are functions of the first 21. As in [23], the last 500,000 exemplars (D_{te}) are used for testing. The training set (D_{tr}) is in turn split into 10 folds such that the class balance is maintained. The first nine splits $D_{tr_1}, \ldots, D_{tr_9}$ are allocated in the Data Server and are accessed from the FCUBE instances and used for learning. The 10th split (D_f) must be available from the FCUBE server and is used for filtering and training the meta-model (fusion). Finally, the testing set D_{te} is used to test the performance of the retrieved meta-model. The characteristics of these splits are summarized in Table 2.

TABLE 2 Characteristics of the Higgs dataset and of the generated splits.

	$D_{tr_1} \ldots D_{tr_9}$	D_f	D_{te}	TOTAL
EXEMPLARS	1,050,000	1,050,000	500,000	11,000,000
FEATURES	28	28	28	28
NEGATIVE EXEMPLARS	47%	47%	47%	47%
POSITIVE EXEMPLARS	53%	53%	53%	53%
FUNCTION	TRAINING	FUSION TRAIN	TESTING	–
ACCESSED BY	FCUBE INSTANCES	FCUBE SERVER		–

B. Massive Data-Parallel Evolutionary Learning

We design the ensemble configuration of the FCUBE runs in the same way the participants of the collaborative activity proceeded. First, we assign a fixed computation budget to each of the learners, Rule List (RL), Rule Tree (RT), GP Function (GPF), Multiple Parents Pittsburgh Learning Classifier System (MPLCS), and Symbiotic Bid- Based Genetic Programming (SBBJ). In this case, each learner is assigned \$20 on Amazon EC2. Then, we choose a *data and parameter factoring strategy* and a combination of *number of instances, flavor,* and *running time* that fits into the budget:

❑ *Data and parameter factoring strategy:* Each of the deployed instances uses FCUBE's factoring service to generate a 1% split of the training data (105,000 exemplars) to learn from it. Note that all the explanatory variables of the problem are considered at each instance. When required as a parameter, the weight of false negative errors is set to 0.47 according to the class balance of the problem. The rest of the parameters of the learners are set to their default values.

❑ *Number of instances:* We deploy in parallel 100 instances of each of the five integrated learners: RL, RT, GPF, MPLCS, and SBBJ. As a result, we obtain 100 classifiers per algorithm, each trained with a different 1% of the data. With this strategy, we expect to cover a significant part of the training data in the learning process.

❑ *Flavor:* The *flavor* corresponds to the specifications of the virtual machines used in the cloud runs. The more computing power the virtual machine has, the more expensive it is. However, the price of the *flavor* does not increase linearly with the specifications of the virtual machine. In other words, virtual machines with higher computational power present a better compute/cost trade-off. With this idea in mind, we exploit the evaluation level parallelism of the learners Rule List, Rule Tree, and GP Function by executing them in a 4-threaded fashion. In accordance with this setup, we employ the compute-optimized *c3.xlarge flavor* that counts 4 virtual CPUs. The remaining learners, MPLCS and SBBJ, run in a single-threaded fashion. Therefore, we employ the cheaper single-core *m3.medium flavor* (\$0.070 per Hour).

❑ *Running time:* The running time is set according to the two previous parameters and the established budget. The \$20 budget allows to run 100 learners executed with the *c3.xlarge flavor* (\$0.210 per instance per hour) for 57 minutes while learners using the *m3.medium flavor* (\$0.070 per instance per hour) will run for 171 minutes.

The configuration of the runs is summarized in Table 3: a total of 500 learning algorithms are executed, and a total of 1,800 CPU hours have been necessary to generate the results presented in this paper.

C. Performance of Individual Learners

FCUBE exploits rapidly trained but weak classifiers with diverse behaviors to produce robust ensembles. As a first analysis of this diversity, we present in Fig. 7 the

TABLE 3 Configuration and cost of the five ensemble strategies deployed on Amazon EC2 with FCUBE.

	LEARNER	RULE LIST	RULE TREE	GPF	MPLCS	SBBJ	TOTAL
FACTORING CONFIGURATION	DATA SAMPLE RATE	1%	1%	1%	1%	1%	–
	VARIABLE SAMPLE RATE	100%	100%	100%	100%	100%	–
	FALSE NEGATIVE WEIGHT	0.47	0.47	0.47	–	–	–
ENSEMBLE CONFIGURATION	BUDGET	\$20	\$20	\$20	\$20	\$20	\$100
	INSTANCES	100	100	100	100	100	500
	FLAVOR	C3.XLARGE	C3.XLARGE	C3.XLARGE	M3.MEDIUM	M3.MEDIUM	–
	VIRTUAL CPUs	4	4	4	1	1	–
	COST PER HOUR	\$0.210	\$0.210	\$0.210	\$0.070	\$0.070	–
	TIME (MIN)	57	57	57	171	171	–
	VIRTUAL CPU HOURS	4×100	4×100	4×100	3×100	3×100	1800

false positive rate and the true positive rate of all the classifiers generated with the five learners. Two important observations are worth commenting. First, MPLCS and SBBJ generate classifiers covering a wide section of the TPR/FPR trade-off. On the other hand, RL, RT, and GPF are, broadly speaking, specialized in some region of the front. This is explained by the fact that RL, RT, and GPF receive the cost of the two errors as parameters of the learning process. As a result, the models generated by these algorithms are concentrated in the region of the trade-off that minimizes the established cost. Second, it is interesting to see that all the regions of the trade-off are covered by one or another learner. The Rule Tree learner is however clearly dominated by the other learners, and thus is not likely to contribute to build a collaborative solution. This analysis confirms that EC-based learners generate classifiers that exhibit a remarkable diversity.

To compare the performance of the different learners, we analyze the cost (Bayesian Risk) of the individual classifiers as in (2). The weight of the two errors is set according to the class balance of the problem with the goal of penalizing classifiers that do not perform well on the minority class. Thus, in this case, we set the weights of the false positive and false negative rates to 0.53 and 0.47 respectively. This analysis is useful to determine, for the considered problem, what are the most appropriate learners to use. Fig. 8a shows the testing set cost for the five different learners. Additionally, we plot the cost of a naive classifier that always predicts the majority class. Note that classifiers exhibiting a cost higher than the baseline with respect to the fusion set D_f have been discarded during the filtering process. We can see that all the learners improve on the baseline cost. GP Function learner generates classifiers that outperform those generated with Rule Tree and Rule List. SBBJ and MPLCS generate classifiers that cover a wide range of values, it is therefore hard to compare them against the other learners. The next step of the analysis consists in determining whether the observed diversity helps building ensembles that will improve the accuracy of the independent classifiers.

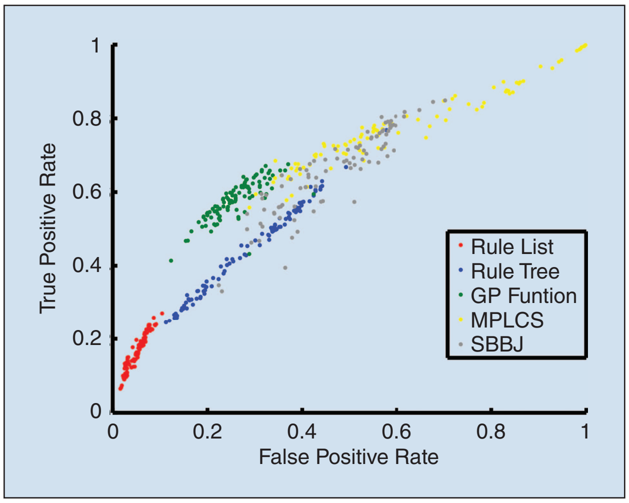

FIGURE 7 Trade-off between false positive rate and true positive rate of the 500 classifiers evaluated on the test set. We train 100 classifiers with each of the five learners integrated in the framework.

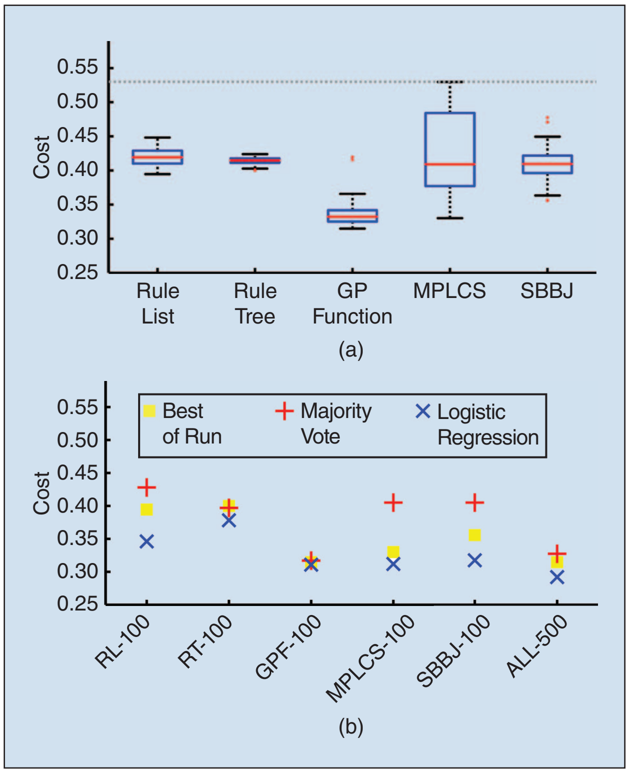

FIGURE 8 Cost or Bayesian Risk of (a) individual classifiers and (b) best model of the run and fused models obtained with Majority Vote and Logistic Regression. In all cases lower is better.

TABLE 4 Cost or Bayesian Risk of the best model of the run and fused models obtained with Majority Vote and Logistic Regression. Lower is better.

METHOD	RL-100	RT-100	GPF-100	MPLCS-100	SBBJ-100	ALL-500
BEST OF RUN	0.395	0.400	0.315	0.330	0.356	–
MV	0.428	0.397	0.317	0.405	0.405	0.3274
LOG REG	0.346	0.378	0.311	0.312	0.318	0.292

D. Performance of the Fused Models

For each learner, we build a meta-model that combines the predictions of the 100 models retrieved from the FCUBE run. Additionally, we obtain a fused model that combines all 500 models, i.e. that combines models obtained with all five learners. We employ two different methods for the fusion process: majority vote and logistic regression (see Section V-B).

The cost of the meta-models obtained with majority vote and logistic regression are reported in Table 4 and shown in Fig. 8b. Note that logistic regression provides a probability for each test case. We apply the decision rule

$$\hat{L}_i = \begin{Bmatrix} 1, \text{ if } y_i \geq 0.5 \\ 0, \text{ if } y_i < 0.5 \end{Bmatrix}$$

to determine whether a class 0 or class 1 prediction is emitted. For comparative purposes, we also show the cost of the best classifier of the run. Majority vote does not provide any benefit with respect to the best model of the run. On the other hand, the logistic regression fusion technique systematically reduces the cost of the best classifier of the run. This effect is remarkable in the case of MPLCS-100 and SBBJ-100 fused models since they are now as competitive as GPF-100. Training multi-algorithm ensembles is shown to be beneficial since the ALL-500 meta-model outperforms all the strategies. This verifies the hypothesis of the need of diversity and is an encouraging result to promote the growth of our repository of learners.

We compute the area under the curve (AUC) of the fused model trained via Logistic Regression. To this end, we employ a moving threshold λ in the decision rule

$$\hat{L}_i = \begin{Bmatrix} 1, \text{ if } y_i \geq \lambda \\ 0, \text{ if } y_i < \lambda \end{Bmatrix},$$

where y_i is the probability vector returned by the logistic regression algorithm. The obtained AUC is compared against the performance of Boosted Decision Trees (BDT), Neural Networks (NN), and Deep Neural Networks (DNN) reported in [23]. It is important to note that this comparison might not be accurate for BDT because precise details of the AUC calculation were not provided. Moreover, [23] does not specify the time required to train the classifiers. This comparative analysis is shown in Table 5. Within the methods tested in this work, the best AUC is achieved by the ensemble composed of all 500 classifiers (i.e., ALL-500). The strategies GPF-100, MPLCS-100, and SBBJ-100 present a similar AUC, and clearly outperform RL-100 and RT-100. When compared to the results presented in the cited work, we can see that our ensemble methods are inferior yet competitive against Boosted Decision Trees and Neural Networks. However, our results are clearly outperformed by Deep Neural Networks. The results presented in this work are nonetheless encouraging since we aimed at fast learning times: approximately one hour in the case of RL-100, RT-100, and GPF-100 and nearly three hours (with less computational capability) for MPLCS-100 and SBBJ-100. Moreover, the learners tested in these experiments are run with their default parameters, i.e. they have not been fine-tuned for the targeted problem.

E. Analysis of the Diversity of the Different Learning Algorithms

In the previous analysis, we have verified that considering different learning algorithms provides better accuracy. We now perform a deeper analysis of the predictions of the different models with the goal of determining which algorithms are complementary, in the sense that they produce classifiers that are competitive and yet emit dissimilar predictions. This information is extremely valuable since it will allow appropriate combinations of learners to be selected when new problems are fed into the framework.

We first analyze the correlation between classifiers. Fig. 9a shows the pairwise correlation coefficient ρ for all the 500 classifiers. This metric has been widely used to assess the diversity of ensemble classifiers [24] and takes into account the four possible combinations of hit/miss given two classifiers y_i and y_j. The coefficient $\rho_{i,j}$ is computed as follows:

$$\rho_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11}+N^{10})(N^{01}+N^{00})(N^{11}+N^{01})(N^{10}+N^{00})}} \quad (5)$$

where

- N^{11} is the number of examples classified correctly by both classifiers,
- N^{10} is the number of examples classified correctly by y_i but incorrectly by y_j,
- N^{01} is the number of examples classified incorrectly by y_i and correctly by y_j,
- N^{00} is the number of examples misclassified by both y_i and y_j.

Classifiers that tend to recognize the same exemplars are correlated, and thus depicted in yellow and red (positive values). On the other hand, negative values indicate that the two classifiers tend to

TABLE 5 Area under the curve of the methods reported in [23] (BDT, NN, and DN) and of the ensembles generated with the different evolutionary learners and fused with logistic regression (RL-100, RT-100, GPF-100, MPLCS-100, SBBJ-100, and ALL-500).

METHOD	BDT [23]	NN [23]	DNN [23]	RL-100	RT-100	GPF-100	MPLCS-100	SBBJ-100	ALL-500
AUC	0.810	0.816	0.885	0.697	0.670	0.746	0.751	0.747	0.779

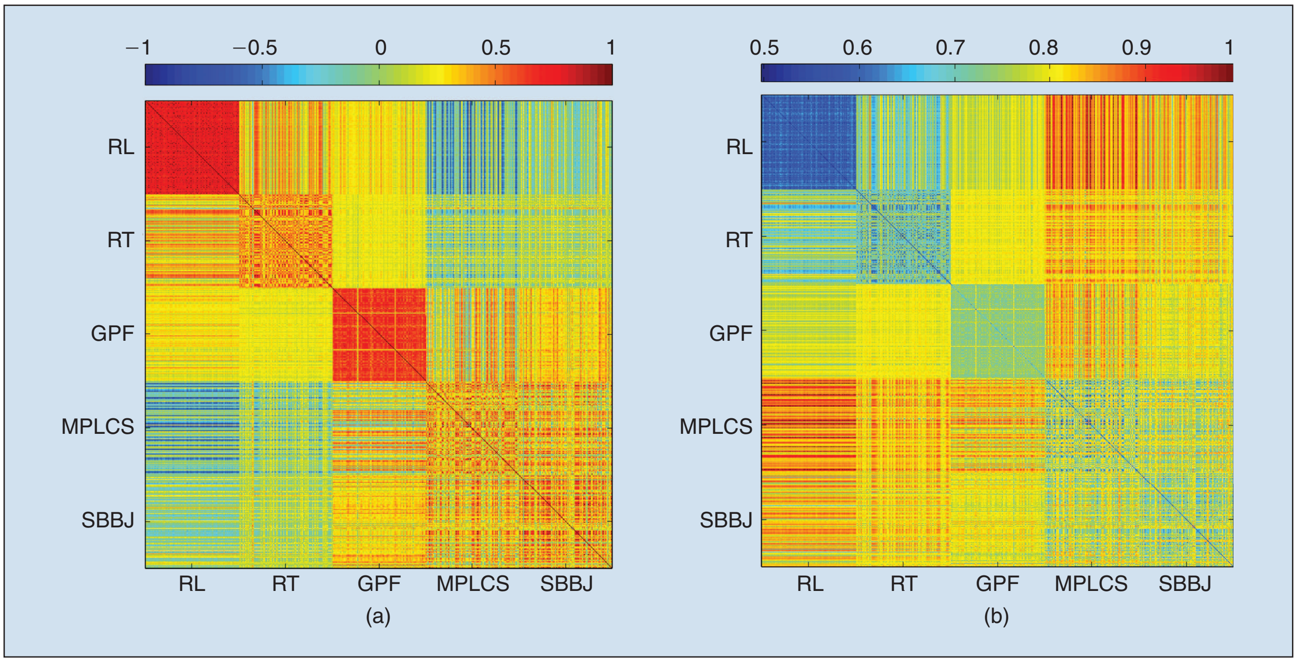

FIGURE 9 Analysis of the diversity of the classifier ensembles. (a) Pairwise correlation coefficient ρ between the 500 classifiers: positive values (yellow and red) indicate correlated predictions. (b) Pairwise complementarity of the 500 classifiers: close to red colors indicate that at least one of the classifiers of the pair tends to make the right prediction.

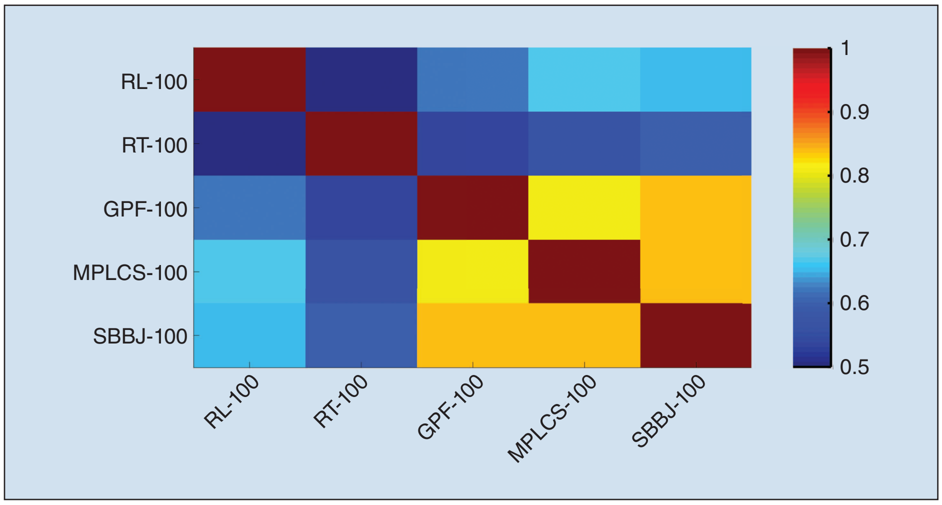

FIGURE 10 Correlation matrix of the five ensembles fused with logistic regression.

miss on different exemplars. It is possible to identify squared areas, each corresponding to 100 classifiers generated with the five different learners. The squared areas confirm the diversity obtained with different learners, even if the final accuracy of the trained classifiers is similar. As opposed to MPLCS and SBBJ, which present a high diversity, RL and GPF do not exhibit a great variance within their 100 classifiers. In these cases, it is not necessary to run many copies of the algorithm. However, it is also possible that the Higgs dataset does not present a high variance and, as a result, the retrieved classifiers are similar even when they are trained with only 1% of the data.

When we compare the correlation across learners, we observe that both RL and RT generate classifiers that are uncorrelated with the rest. This might be due to the fact that their performance is lower, and therefore we will exclude them going forward. GPF and SBBJ generate correlated classifiers, and thus should not be used together to generate ensembles. On the other hand, MPLCS seems to be different both from GPF and SBBJ, and thus is a solid candidate for multi-algorithm ensembles.

We introduce a different measure that allows us to quantify how complementary the pairs of classifiers are. We define *complementarity* as the ratio between the number of exemplars on which at least one of the classifiers is correct to the total number of exemplars. In other words, this measure represents the accuracy of the combined prediction of two classifiers given an *ideal* fusion method. This measure is given by:

$$\text{complementarity}\gamma_{i,j} = \frac{N^{01} + N^{10} + N^{11}}{N^{11} + N^{10} + N^{01} + N^{00}} \qquad (6)$$

The results of this analysis are shown in Fig. 9b. While RL exhibits a poor performance, its combinations with MPLCS and SBBJ seem to be promising. However, it is due to the fact that some of those classifiers tend to predict the majority class despite balanced data. This makes them unsuitable for fusion.

Therefore, we continue with only the top learners, i.e. GPF, MPLCS, and SBBJ. It is very interesting to observe that some combinations of GPF and MPLCS classifiers have a great potential, reaching values of up to 0.92.

We now study the similarities of the predictions of the five fused models, each built with models obtained with a single learning algorithm. In this case, we employ the probabilities returned by the logistic regression process rather than the predictions or labels. Since these probabilities are continuous values, we can generate the 5×5 correlation matrix, where each row corresponds to one of the five learners. The patterns observed in the analysis of individuals classifiers also take place in this case. We can see that RL-100 and RT-100 are highly uncorrelated with the other methods. SBBJ-100 is correlated to both GPF-100 and MPLCS-100. However, as observed in previous results, the correlation between GPF-100 and SBBJ-100 is lower, thus confirming the great potential of combining these two learners.

VIII. Conclusion

We have introduced FCUBE, a machine learning framework that harnesses cloud computing to solve largescale supervised learning problems via massive ensemble learning. It exploits the enhanced software transferability provided by virtualized cloud resources to automate the use of learning algorithms developed within the Evolutionary Computation community. The framework interfaces are carefully designed to enable machine learning researchers to easily import their learners and benefit from a massive data-parallel deployment of their algorithm with minimal overhead on their part. We refer to this concept as *Bring Your Own Learner (BYOL).*

We have demonstrated the framework by integrating five different learners and deploying them in a massive data-parallel fashion on Amazon EC2. We execute 100 runs per learner, each with 1% of the data, in as little as one hour. The employed algorithms are representative of evolutionary computation state-of-the-art. In particular, two of them are highly validated algorithms provided by external collaborators. We present results on a publicly available problem composed of 11 million exemplars based upon the Higgs dataset. The ensemble strategies adopted in this work are promising since we obtain a competitive performance while aiming at fast learning. We also analyze the remarkable diversity of the classifiers trained with the different evolutionary computation techniques. The goal of such analysis is to determine appropriate combinations of learners for future problems that the framework could encounter.

This project aspires to a commons where new large-scale problems can be uploaded and quickly solved thanks to a community-shared repository of learning algorithms and cloud computing. We encourage the machine learning community to contribute to the repository of learners and we offer the framework to all others who need a large-scale, supervised machine learning tool.

Acknowledgments

The authors would like to thank Dr. J. Bacardit, Dr. M. I. Heywood, and R. Smith for contributing their learning algorithms to the FCUBE framework.

References

[1] P. Huijse, P. Estevez, P. Protopapas, J. Principe, and P. Zegers, "Computational intelligence challenges and applications on large-scale astronomical time series databases," *IEEE Comput. Intell. Mag.*, vol. 9, no. 3, pp. 27–39, Aug. 2014.

[2] D. Harris, (2012, Aug. 14). How Oxdata wants to help everyone become data scientists. [Online]. Available: http://gigaom.com/2012/08/14/how-0xdata-wants-to-help-everyone-become-data-scientists/

[3] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A distributed machine-learning system," in *Proc. 6th Biennial Conf. Innovative Data Systems Research*, 2013.

[4] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new framework for parallel machine learning,"in *Proc. 26th Conf. Uncertainty Artificial Intelligence*, Catalina Island, CA, July 8-11, 2010.

[5] (2014). FCUBE. Project website. [Online]. Available: http://flexgp.github.io/FCUBE

[6] J. Bacardit, P. Widera, A. Márquez-Chamorro, F. Divina, J. S. Aguilar-Ruiz, and N. Krasnogor. (2012). Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features. *Bioinformatics* [Online]. *28(19)*, pp. 2441–2448. Available: http://bioinformatics.oxfordjournals.org/content/28/19/2441.abstract

[7] Y. Zhang and S. Bhattacharyya. (2004, June). Genetic programming in classifying large-scale data: An ensemble method. *Inf. Sci.* [Online]. *163(1–3)*, pp. 85–101. Available: http://dx.doi.org/10.1016/j.ins.2003.03.028

[8] N. Holden and A. A. Freitas. (2008, Oct.). Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation. *Soft Comput.* [Online]. *13(3)*, pp. 259–272. Available: http://dx.doi.org/10.1007/s00500-008-0321-0

[9] P. Collet, E. Lutton, M. Schoenauer, and J. Louchet, "Take it EASEA," in *Parallel Problem Solving From Nature PPSN VI* (Lecture Notes in Computer Science, vol. 1917), M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H.-P. Schwefel, Eds. Berlin Heidelberg, Germany: Springer, 2000, pp. 891–901.

[10] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Machine Learn. Res.*, vol. 13, pp. 2171–2175, July 2012.

[11] T. Weise, R. Chiong, J. Lassig, K. Tang, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking optimization algorithms: An open source framework for the traveling salesman problem," *IEEE Comput. Intell. Mag.*, vol. 9, no. 3, pp. 40–52, Aug. 2014.

[12] (2014). Big learning activity. in *Proc. EC Big Data Big Learning workshop Held Within GECCO Conf.*, [Online]. Available: http://flexgp.github.io/EC-for-Big-Learning

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[14] E. Vladislavleva, "Model-based problem solving through symbolic regression via pareto genetic programming," Ph.D. dissertation, Tilburg Univ., Tilburg, The Netherlands, 2008.

[15] I. Arnaldo, K. Veeramachaneni, and U. M. O'Reilly, "Building multiclass nonlinear classifiers with GPUs," in *Proc. Big Learning Workshop NIPS: Advances Algorithms Data Management*, 2013.

[16] J. Bacardit and N. Krasnogor. (2006). Smart crossover operator with multiple parents for a Pittsburgh learning classifier system. in *Proc. 8th Annu. Conf. Genetic Evolutionary Computation* [Online]. pp. 1441–1448. Available: http://doi.acm.org/10.1145/1143997.1144235

[17] M. Franco, N. Krasnogor, and J. Bacardit. (2013). GAssist vs. BioHEL: Critical assessment of two paradigms of genetics-based machine learning. in *Soft Computing* [Online]. *17(6)*, pp. 953–981. Available: http://dx.doi.org/10.1007/s00500-013-1016-8

[18] J. Bacardit and X. Llorá. (2013). Large-scale data mining using genetics-based machine learning. in *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* [Online]. *3(1)*, pp. 37–61. Available: http://dx.doi.org/10.1002/widm.1078

[19] P. Lichodzijewski and M. I. Heywood. (2008). Managing team-based problem solving with symbiotic bid-based genetic programming. in *Proc. Annu. Conf. Genetic Evolutionary Computation* [Online]. pp. 363–370. Available: http://doi.acm.org/10.1145/1389095.1389162

[20] J. A. Doucette, A. R. McIntyre, P. Lichodzijewski, and M. I. Heywood. (2012). Symbiotic coevolutionary genetic programming: A benchmarking study under large attribute spaces. *Genetic Program. Evolvable Machines* [Online]. *13(1)*, pp. 71–101, Available: http://dx.doi.org/10.1007/s10710-011-9151-4

[21] A. Atwater and M. I. Heywood. (2013). Benchmarking pareto archiving heuristics in the presence of concept drift: Diversity versus age. in *Proc. 15th Annu. Conf. Genetic Evolutionary Computation* [Online]. pp. 885–892. Available: http://doi.acm.org/10.1145/2463372.2463489

[22] R. Smith and M. Heywood. (2014). SBBJ. Project website. [Online]. Available: http://web.cs.dal.ca/ mheywood/Code/

[23] P. Baldi, P. Sadowski, and D. Whiteson. (2014, July). Searching for exotic particles in high-energy physics with deep learning. *Nature Commun.* [Online]. *5(4308)*, Available: http://dx.doi.org/10.1038/ncomms5308

[24] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learn.*, vol. 51, no. 2, pp. 181–207, 2003.