Ensemble learning of runtime prediction models for gene-expression analysis workflows

David A. Monge, Matěj Holec, Filip Železný & Carlos García Garino

Cluster Computing
The Journal of Networks, Software Tools and Applications

ISSN 1386-7857

Cluster Comput
DOI 10.1007/s10586-015-0481-5

Springer

Cluster Comput
DOI 10.1007/s10586-015-0481-5

Ensemble learning of runtime prediction models for gene-expression analysis workflows

David A. Monge[1] · **Matěj Holec**[2] · **Filip Železný**[2] · **Carlos García Garino**[3]

Received: 17 December 2014 / Revised: 25 August 2015 / Accepted: 27 August 2015

Abstract The adequate management of scientific workflow applications strongly depends on the availability of accurate performance models of sub-tasks. Numerous approaches use machine learning to generate such models autonomously, thus alleviating the human effort associated to this process. However, these *standalone* models may lack robustness, leading to a decay on the quality of information provided to workflow systems on top. This paper presents a novel approach for learning ensemble prediction models of tasks runtime. The ensemble-learning method entitled bootstrap aggregating (bagging) is used to produce robust ensembles of M5P regression trees of better predictive performance than could be achieved by standalone models. Our approach has been tested on gene expression analysis workflows. The results show that the ensemble method leads to significant prediction-error reductions when compared with learned standalone models. This is the first initiative using ensemble learning for generating performance prediction models. These promising results encourage further research in this direction.

Reproducibility of Results The data used in this study and additional results are available in the following URL: http://damonge.wordpress.com/research/enslearn-clus2014.

✉ David A. Monge
dmonge@uncu.edu.ar

[1] ITIC Research Institute & Faculty of Exact and Natural Sciences, National University of Cuyo (UNCuyo), Mendoza, Argentina

[2] IDA Research Group, Czech Technical University in Prague, Prague, Czech Republic

[3] ITIC Research Institute & Faculty of Engineering, National University of Cuyo (UNCuyo), Mendoza, Argentina

Keywords Performance prediction · Ensemble learning · Data-intensive workflows · Gene expressions analysis experiments

1 Introduction

Workflow technology is intended to ease the development of applications by combining reusable software components permitting the development of large-scale applications by people with low or even null experience in programming languages. For this reason, workflow technology has been widely accepted in many scientific areas [20].

Scientific workflows usually describe large-scale data-intensive applications, whose execution is delegated to Workflow Management Systems (WMSs). This aspect is very important because users can take advantage of a huge computing power (i.e. clusters, grids or clouds) while abstracted from the particularities of the underlying infrastructure.

For managing the applications efficiently, WMSs rely on runtime estimates of tasks. This information is the basis for several processes like for example: task scheduling, fulfillment of quality of service (QoS) requirements, auto-scaling cloud infrastructures among others [3,5,11].

Most of the prediction methods used by WMSs were crafted for characterizing parallel applications [17]. Although such techniques provide accurate predictions, they require the supervision of an expert for constructing and tuning the prediction models. Such requirements invalidate one of the main advantages of workflow technology: *simplicity for the user*.

To cope with such limitation many authors applied machine-learning methods to generate runtime prediction models (semi-)automatically from historical data of previous task executions. The methods used focus on the construction

Published online: 09 September 2015

of a single model which may present a lack on robustness. Unlike these, *ensemble-learning* methods are capable of achieving a higher performance than standalone models due to the combination of multiple models for obtaining the predictions [13].

Following this line of thought, we have proposed a novel method for the autonomous generation of combined runtime prediction models derived using ensemble-learning methods. The final objective of our approach is the minimization of the human effort when generating the models without trading off the accuracy of predictions. This paper extends the ideas exposed in previous work [15] by providing an new experimental setting and deepening the analysis of results by applying adequate statistical tests.

The rest of this paper is organized as follows. In Sect. 2 we provide a review of application performance prediction strategies based on machine-learning methods. Section 3 presents our approach for learning runtime prediction models and also explains several machine-learning methods used in related work and a well known ensemble-learning method used for validating our approach. Section 4 describes a set of bioinformatics workflows and the experimental settings of this study. Section 5 presents and discusses the results obtained. Finally, conclusions and future work are given in Sect. 6.

2 Related work

The prediction of applications' performance has been studied since the genesis of parallel and distributed computing [2,17] due to its importance on the management of applications. Many of these techniques involve tedious tasks such as the construction of models by hand, benchmarking resources, profiling applications, etc. Among all the proposed techniques, machine-learning methods permit the derivation of models based on *historical data* (examples) in an automatic fashion. These methods represent an important advantage for workflow applications running on grids or clouds because models can be refined learning from new examples over time without requiring intervention of the user.

Prophesy [21] is a system for predicting the performance of applications in parallel and grid environments using historical data. Among the prediction strategies in Prophesy, (polynomial) *curve fitting* is used for constructing the models in an automated fashion. However, the authors state that the method is not suitable for scenarios with different system configurations.

Some of these strategies address the prediction issue using the k-Nearest Neighbors strategy [9,14]. Predictions are performed by first looking execution *examples* with similar features (e.g. examples with similar task parameters, processor speed, etc.) to the prediction query. Then, the execution times corresponding to the selected examples are averaged and returned as the prediction.

Authors like Ould-Ahmed-Vall et al. [16] used regression trees for modeling the performance of compute-intensive tasks and compared them with the performance of Artificial Neural Networks and Support Vector Regression. The authors used data obtained from a suite of benchmarks. Results reported by the authors correspond to a double-processor machine.

These techniques use statistical or machine-learning methods to estimate the performance of tasks on distributed computing environments. The techniques have been developed having in mind compute-intensive applications disregarding important information sources such the size or the structure of data, to say nothing of *data provenance* [4] (i.e. the origin and transformations suffered by the data during the execution of an application). In the context of scientific workflows, where data is becoming a first-class citizen [7,12], this information is fundamental for achieving accurate performance predictions.

A second aspect to remark is that these strategies rely on the use of a standalone model for performing the predictions. It is known that combining multiple models usually permits achieving a higher performance than using a unique model [13].

As the main contribution, this paper proposes a novel method for minimizing the intervention of a human expert to model the performance of tasks in the context of scientific workflows. The proposed approach relies on ensemble machine learning methods for generating models in a automatic fashion. Several sources of information like task parameters, hardware information, data characteristics and provenance information, are incorporated to maximize the accuracy of the models.

3 Learning performance models

This section describes a generic strategy for the autonomous generation of performance models (AGPM) for the prediction of workflow task runtime. Unlike other strategies, AGPM relies only on information accessible from the underlying workflow management system or from the definition of the application itself.

AGPM considers tasks as *black boxes*, which permits the modeling of software components whose code is unavailable or inaccessible. The user only needs to define the *meta-data* of tasks that might be important for modeling their performance. In this way, the process of performance modeling is focused on the parameters and data that affect the performance (user's empirical knowledge) and not in the particular process implemented by the tasks.

AGPM relies on machine-learning methods to model task's running time using information of workflow tasks parameters, data and dependencies as well as resource benchmark metrics. These methods permit the construction of the models and their readjustment as new performance data becomes available. In this manner, the required human effort to maintain the models is greatly reduced while keeping a high predictive accuracy. AGPM uses ensemble machine learning methods to construct a meta-model comprising multiple sub-models to achieve higher quality predictions.

3.1 AGPM learning process

AGPM drives a continuous learning process that comprises 4 stages: (i) execution of workflow tasks, (ii) performance data gathering, (iii) model learning, and (iv) tasks runtime prediction. These stages and their relationships are shown in Fig. 1.

These stages are repeated continuously throughout the execution of several applications. Each one of these cycles permits the adaptation of the models to new (unseen) execution examples improving the predictive accuracy of the models over time. The important aspect to note is that this *adaptive learning* process improves the accuracy of the prediction models without requiring human intervention more than the initial setup of the performance data to collect.

Stage 1: Workflow tasks execution This stage involves the execution and monitoring of tasks as well as the generation of the corresponding *execution logs*, which are later used in the following stages. This stage is carried out entirely by the WMS.

Stage 2: Performance-data gathering Consists in the harvesting of the necessary information for the further learning/refinement of the performance models. Execution logs are used to extract valuable information of tasks performance such as the parameters and the data used, provenance information and the characteristics of the resources where the tasks were executed. AGPM compiles all the information that can be gathered from the running workflow management system. The collected data is stored in separate datasets for each type of runnable task. Section 3.2 discusses in detail the representation of such performance data.

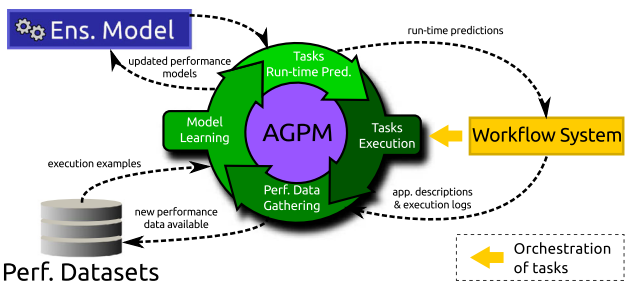

Fig. 1 Learning process carried out by AGPM

Stage 3: Model learning At this point of the process, the databases contain updated information of the last task execution. AGPM then learns a new model for each type of task following a two-step procedure consisting of (i) data pre-processing, and (ii) ensemble model learning. AGPM pre-processes the databases in order to prepare the data for the ensemble learning strategy. As a second step, multiple models are learned from the data and combined in order to perform future runtime predictions. Section 3.3 provides a deeper insight on the described process which is the central contribution of this paper.

Stage 4: Tasks runtime prediction Consists in the generation of runtime estimates for tasks using the models constructed on the previous stage. Runtime estimates are obtained considering the inputs of workflow tasks (i.e. parameters and data) and the characteristics of the resources which will eventually execute such tasks.

3.2 Performance-data representation

Performance data is stored separately for each type of task. The performance dataset for a task can be formally defined as a set $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_m^{i=1}$, where $x^{(i)}$ represents a column vector of features for the ith recorded *execution example* of a task, $y^{(i)}$ is the measured *runtime* for such execution (also known as *target*), and m is the total number of examples in the database.

Each feature vector $x = [x_1, x_2, \ldots, x_n]^T$ comprises three types of elements: (i) *task features*, which represent the inputs of the task, e.g. parameter values, data size, etc.; (ii) *provenance features*, describe previous processes that generated or modified the input data; and (iii) *resource features*, which model characteristics of the resource used on the execution of the task.

Task features This information includes the values of input parameters and characteristics of the data such as size, number of lines, registers or columns, etc.

Provenance features This type of features capture information of the data origin and the transformations produced by other tasks during the execution of the workflow. Such information can be easily extracted from the description of the workflow.

Resource features This kind of features describe the computing resources used in the tasks execution. These features can be obtained from the WMS. Such information is mainly provided by resource benchmarks. In general, WMSs provide such metrics and update them regularly.

3.3 Learning prediction models

Machine-learning methods are the core of our approach. This section briefly describes some of the traditional machine-learning techniques used in the state of the art to produce (standalone) prediction models. This section also discusses an ensemble-learning method entitled *bootstrap aggregating* (or simply bagging) used as mean to validate the hypothesis of this paper.

3.3.1 Preprocessing

Before constructing the models, the available performance data is normalized to avoid the dominance of some features of higher orders of magnitude in the construction of the models. To such end, each feature x_i is transformed into a new feature $\hat{x_i}$ computed as $\hat{x_i} = \frac{x_i - \mu}{\sigma}$, where μ and σ are the mean and the standard deviation of all the values for the feature x_i.

3.3.2 Standalone models

Our implementation of AGPM includes some well established machine-learning strategies used in previous work on performance prediction. The following paragraphs describe the essential concepts underlying such strategies.

Artificial neural networks (ANNs) These models emulate the operation of biological neural networks [24]. We specifically consider the *feed-forward* networks, which comprise a set of neurons (units) arranged in multiple layers. Units in one layer are connected only to units in the following layer. The last layer contains only one neuron whose output predicts the target value y. Networks used in this study comprise one hidden layer with $n/2$ hidden units, where n is the number of features in the input vector. The parameters of the network are a matrix $\Theta^{(1)} \in \mathbb{R}^{(n+1)\times(n/2)}$ and a column vector $\Theta^{(2)} \in \mathbb{R}^{n/2}$ that model the interactions between the units in different layers. Figure 2 shows an example neural network.

The activation (outputs) of the hidden units are computed as $a = \sigma(\bar{x} \cdot \Theta^{(1)})$, where $\bar{x}$ is an input vector x extended with a first component $x_0 = 1$ (bias unit), and $\sigma(z) = 1/(1+e^{-z})$, which is the *sigmoid* function. Activation of the hidden units are forward propagated to the next layer to produce the predicted value of the running time $y = \bar{a} \cdot \Theta^{(2)}$, where $\bar{a}$ is the activation vector a of the hidden layer extended with a bias unit $a_0 = 1$. Learning the model consists of learning the weights in the network, i.e. the values of $\Theta^{(1)}$ and $\Theta^{(2)}$. The *back-propagation* [24] algorithm is used to this end.

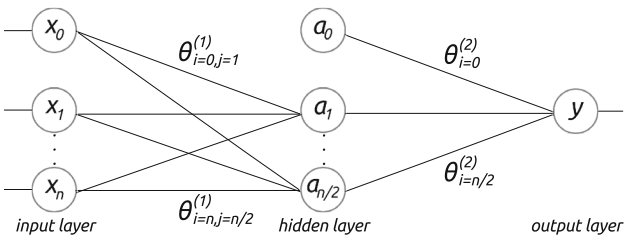

Fig. 2 Artificial neural network example. The network comprises one hidden layer and a single output unit y that provides the runtime prediction

k-Nearest neighbors (k-NN) In this strategy, training examples are stored verbatim. A distance function is used to determine those k examples of the training set that are closest (i.e., most similar) to an unknown test example [1]. The output of the method is the average of the target values (i.e. runtimes) corresponding to such k nearest examples. In this study we use the Euclidean distance, which is possible since all of our features are numeric. The distance between two examples $x^{(1)}$ and $x^{(2)}$ with components $x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)}$ and $x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)}$ is: $\sqrt{\sum_1^n (x_i^{(1)} - x_i^{(2)})^2}$.

Support vector regression (SVR) SVR [19] is an adaptation of the Support Vector Machines (SVM) classification strategy to deal with the prediction of numeric classes. Produced models can be expressed in terms of a few support vectors that best describe a prediction surface. The SVR model has the form $f(x) = \sum_{i \in SV} \alpha_i K(x^{(i)}, x) + b$, where SV is the set of support vectors and $K(x^{(i)}, x)$ is a kernel function that maps an example into a feature space of higher dimensions. α_i and b are model parameters determined by solving the following optimization problem:

$$\min_{\alpha, b, \xi_i, \xi_i^*} \frac{1}{2}\alpha^T \alpha + C \sum_{i=1}^{l} (\xi_i + \xi_i^*)$$

$$\text{s.t.}: \quad y - f(x_i) \leq \varepsilon + \xi_i$$
$$f(x_i) - y \leq \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* > 0,$$

where C is the model complexity parameter which penalizes the loss of training errors, ξ_i and ξ_i^* are slack variables that specify upper and lower bound training errors subject

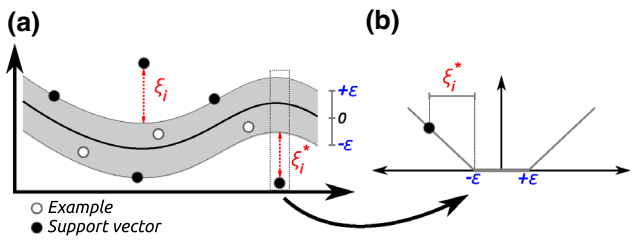

Fig. 3 SVR example. Left figure **a** represents the prediction function $f(x)$ and the right figure **b** represents the optimization function

to an error tolerance ε. To model non-linear functions of the running time we use a radial basis function (RBF) kernel:

$$k(x^{(i)}, x^{(j)}) = exp\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right) \quad (1)$$

with $\gamma = 0.01$. The values parameters of SVR were set to $C = 1$ and $\varepsilon = 0.001$. Figure 3 presents an example of the SVR optimization procedure.

M5P regression trees The M5 Prime (M5P) is an algorithm that permits the induction of decision trees whose leaves are associated to regression models [22]. M5P trees are a combination of decision trees and linear models in the tree leaves. The model is generated in three phases as follows.

First, a decision-tree is induced using the M5 algorithm [18]. The tree is constructed using a splitting criterion that minimizes the intra-subset variation in the output values down each branch. Given a node, the subset is divided by selecting the feature that maximizes the standard deviation reduction (SDR), which is computed as: $SDR = \sigma(\mathcal{D}) - \sum_i \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \times \sigma(\mathcal{D}_i)$, where $\sigma(\cdot)$ is the standard deviation, $\mathcal{D}$ is the set of examples that reach the node and the $\mathcal{D}_i$ are the sets that result from splitting the node on the selected feature. The procedure continues until the variation of the output values in a node is small or the number of examples is small.

Second, linear regression is used on each node to generate a model (regression hyperplane) considering only those features tested in the sub-tree below. The tree is pruned starting from the leaf nodes until the expected estimated error decreases [22].

Finally, a smoothing function is applied to avoid sharp discontinuities between the hyperplanes. The procedure starts from the leaf node to the root node smoothing the predicted value along the path. The smoothing of a prediction is made as $\tilde{p} = \frac{np+kq}{n+k}$, where p is the prediction passed from the node below, q is the prediction for the model associated to this node, n is the number of instances that reach the node below and k is a constant usually equal to 15. An example of M5P regression tree is given in Fig. 4.

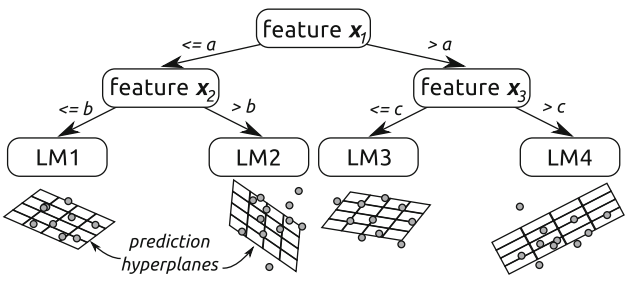

Fig. 4 Example of an M5P regression tree

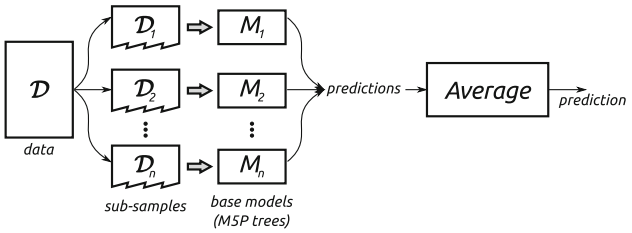

Fig. 5 Bagging process. The n sub-samples ($\mathcal{D}_i$) are used to construct the base models (M_i), which are M5P regression trees. Outputs of the base models are averaged to produce the performance prediction

3.3.3 Learning ensemble models

One of the main advantages of ensemble-learning methods is that they often produce predictions of better quality than those obtained by standalone models. For generating the models we use the bootstrap aggregating (bagging) technique [13]. This technique reduces the *variance* of the learning process as the expected error is derived from multiple training sets sub-sampled from the original set.

The bagging technique works as follows. For a given training dataset $\mathcal{D}$, n new training datasets ($\mathcal{D}_i$) of size m' are obtained by sampling the set $\mathcal{D}$ randomly with replacement. This means that, for generating each of the $\mathcal{D}_i$ sub-samples, some examples are removed and some of them are repeated. Each of the n samples are used to learn n different (base) models. The outputs of the n models are combined by averaging their predictions.

In this paper, the base models are learned using the M5P method discussed on Sect. 4. The selection of this method for learning the base model lies on that standalone M5P regression trees tend to have higher variance than the remaining methods. Therefore, bagging of M5P trees can produce an ensemble model with lower variance than the standalone trees and thus improve their performance. The entire process is illustrated in Fig. 5.

4 Experiments

To analyze the performance of the ensemble method we evaluated the predictive accuracy of standalone models generated using the reviewed methods and ensemble models learned with the bagging strategy.

4.1 Gene expression analysis workflows

For the purposes of this work we evaluated our approach on bioinformatics data-mining workflows, which perform a large-scale gene expression analysis experiment (GEAE). The goal of the experiment is to evaluate a novel classification

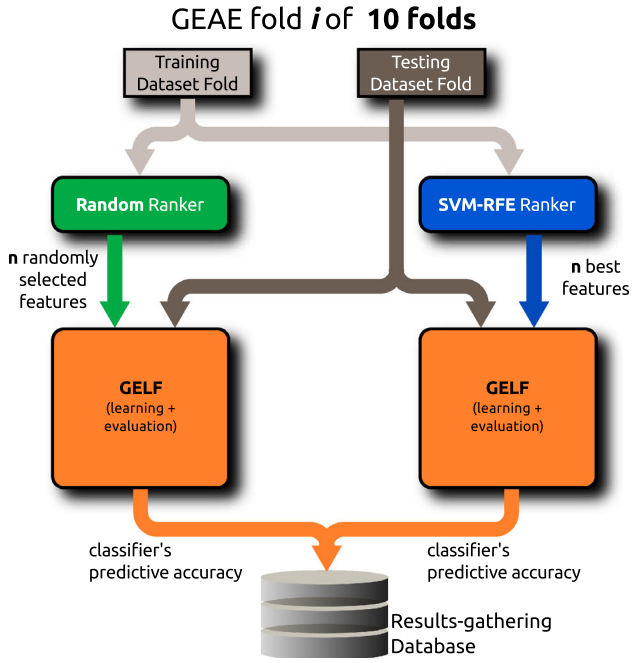

Fig. 6 GEAE workflows. One of the fold for a GEAE experiment operating on a given dataset. Each fold involves the execution of a Random ranker task, an SVM-RFE ranker tasks and two GELF tasks giving rise to 2 sub-experiments: *a* «Random, GELF», and *b* «SVM-RFE, GELF». For each dataset 10 folds are generated

algorithm (GELF) [8] on their respective ability to classify unseen gene expression samples[1].

The experiment comprises the execution of several workflows. Each of them processes one of 20 micro-array datasets used for the experiment using a tenfold cross-validation scheme. Figure 6 represents one fold of a GEAE workflow.

As can be seen from the figure each fold involves the execution of 3 types of tasks. Two of them are *ranker tasks* that perform a selection of genes in order to reduce the number of features for training the classifier. The first one uses recursive feature elimination using support vector machines (called SVM-RFE ranker), and the second one returns a random order of features (Random ranker). These ranker tasks precede the execution of the third type of tasks that consist on *learning and evaluating* the performance of the (GELF) classifier. GELF is a feature construction algorithm based on iterative improvement of the best solution obtained by the state-of-the-art approach [8].

Each workflow comprises 20 sub-experiments: both combinations of the GELF task with the rankers (Random and SVM-RFE) applied on the 10 dataset folds. As can be seen each workflow application consists of 40 tasks (i.e. 10 Random ranker executions, 10 SVM-RFE ranker executions and 20 GELF executions).

[1] This task of classification learning should not be confused with the learning task of runtime prediction.

Table 1 Computing infrastructure description

Characteristics	Resource type		
	Twister	Reloaded	Opteron
Proc. vendor	Intel	Intel	AMD
Proc. model	Core2 Duo	P4 HT	Opteron 242
Frequency	3.0 GHz	3.0 GHz	1.6 GHz
Memory	4 GB	1 GB	2 GB
JavaMFlops	962.23	281.05	400.76
KFlops	17.54E5	4.99E5	6.63E5
MIPS	4983.80	1465.42	2057.00
Quantity	10	12	4

4.2 Performance datasets

To collect the performance data, we measured the runtime of multiple instances of GELF tasks using 20 different micro-array datasets the 26 different computing resources described in Table 1. The infrastructure runs HTCondor[2] version 7.4.1 for administrating the tasks. JavaMFlops, KFlops and MIPS are the performance metrics for the resources provided by the SciMark2.,[3] Linpack[4] and Dhrystone [23] benchmarks respectively

For testing the applicability of our approach we evaluated the performance of the GEAE workflows using homogeneous (solely twister-type resources) and heterogeneous (all the resources) infrastructures. The execution of workflows was carried multiple times on each type of infrastructure to obtain the necessary data for learning the models. Execution logs generated were used to feed the performance databases for each type of task. Table 2 presents the features used for each execution example for the ranker tasks (Random and SVM-RFE) and the GELF tasks comprised in the GEAE workflows.

4.3 Preliminary analysis of tasks

For an initial characterization of tasks we present some statistic measures of tasks' runtime considering the execution examples in the performance datasets. Table 3 summarizes these measures considering mean and average runtime values, dispersion (standard deviation, minimum and maximum values) and shape of the distribution (skewness and kurtosis).

The following list describes the main observations derived from the analysis of the mentioned measures:

[2] HTCondor. http://research.cs.wisc.edu/htcondor/.

[3] SciMark2 benchmark. http://math.nist.gov/scimark2.

[4] Linpack benchmark. http://www.netlib.org/linpack.

Table 2 Features in the performance datasets for GEAE workflow tasks

Feature	Description
`dataset-id`	{1,2,…,20}
`predecessor`*	{Random, SVM-RFE}
`tr-size`	Size in bytes of the training set
`tr-rows`	No. of rows in the training set
`tr-columns`	No. of columns in the training set
`tt-size`*	Size in bytes of the training set
`tt-rows`*	No. of rows in the testing set
`tt-columns`*	No. of columns in the testing set
`java-mflops`	SciMark2 benchmark results
`kflops`	Linpack benchmark results
`mips`	Dhrystone benchmark results
`runtime`	Measured runtime

These are grouped (by single lines) into provenance, task and resource features. The last feature is the target variable (running time)
* Features that pertain only to GELF

Table 3 Tasks characterization according to various statistic measures including mean and median values, dispersion and shape of the distribution

Measures	Task		
	Random	SVM-RFE	GELF
Mean	12.70	29.20	3842.87
Median	11	26	3204
SD	7.06	16.79	2463.38
Min	4	9	907
Max	55	109	17889
Skewness	1.19	1.21	1.62
Kurtosis	1.75	1.67	3.31

1. *Mean and median* values indicate that GELF tasks have a considerable longer runtimes (in the order of 1 hour) than the the two ranker tasks (around 30 seconds).
2. *Standard deviation* and extreme values (*min* and *max*) indicate a high variability of tasks' runtime. This observation is specially notorious in the case of GELF, which presents very large maximum values (up to 5 hours of computation).
3. *Skewness* values (major than 1) indicate that the distributions are very right-skewed. This means that the three distributions are very asymmetric, with many of the execution examples concentrated on left of the mean and with extreme values to the right.
4. *Kurtosis* values (larger than 0) indicate that the distributions have very sharp peaks with values concentrated around the mean and thicker tails. Large kurtosis values mean high probability for extreme values.

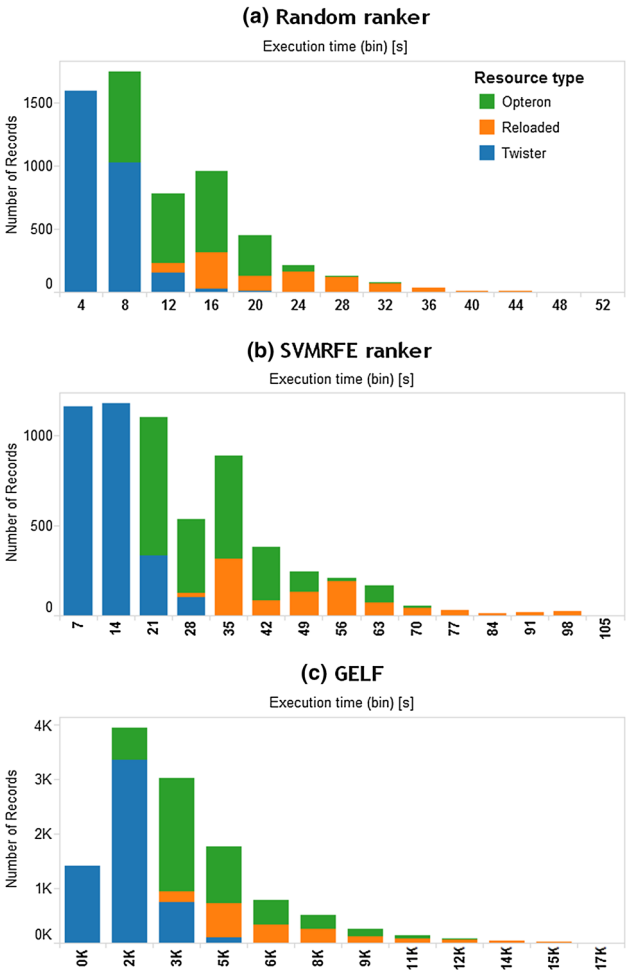

Fig. 7 Runtime histograms for the three analyzed tasks. Bars discriminate by colors the three types of resource used to generate the performance datasets

To complete this analysis, Fig. 7 presents runtime histograms for the three tasks differentiating by resource type. As said before it can be seen that the distributions are very right-skewed, having a large concentration of examples around the mean value and extreme values to the right. As we will see on Sect. 5.2, the skewness of these distributions will be reflected on the types of errors incurred by the models.

4.4 Experimental settings

GEAE workflows were executed several times to generate the required performance data to learn the models. For each task we obtained two datasets comprising 4000 execution examples each one. The first one, called *homogeneous*, contains running instances where only resources Twister type (see Table 1) were used. The second dataset, called *heterogeneous*, contains running instances corresponding to the use of all resource types. Table 2

Table 4 Parameters used for the learning methods

Method	Parameter/function	Value
k-NN	k: number of neighbors	3
	Distance function	Euclidean
M5P	Min. no. of instances per leaf	4
	Pruned	Yes
	Smoothed	Yes
ANN	Number of hidden layers	$n/2$
	Hidden units type	Sigmoid
	α: learning rate	0.3
	m: momentum	0.2
	Number of training epochs	500
SVR	Kernel function $K(x^{(i)}, x^{(j)})$	RBF
	γ: RBF's parameter, Eq. (1)	0.01
	C: model complexity	1
	ε: error tolerance	0.001
Bagging	bootstrap size	2800†
	n: number of base models	10
	Base learner	M5P

†2800 is the total number of examples in the training set

describes the features used to describe the task execution examples.

With these six datasets we trained models using all the learning methods discussed previously. From these datasets, 70 % of the (4000) original instances were used for *training* and the remaining 30 % for *validation*. For each dataset, this process was repeated 30 times using random sub-sampling without replacement to obtain 30 different pairs of training and validation sets.[5]

The implementations of all the methods used are provided by the Weka library [6] version 3.6.8. Table 4 provides a summary of the parameters used by the machine learning methods. These parameters except for k in k-NN are the default values in Weka. It is worth pointing out that it was deliberately decided to use the default parametrization for all the methods in order to resemble use cases of a user with no experience on machine learning or performance prediction.

For measuring the performance of each model we use the Relative Absolute Error (RAE), which is computed as $error = \frac{|p_1 - y_1| + \cdots + |p_m - y_m|}{|y_1 - \bar{y}| + \cdots + |y_m - \bar{y}|} \cdot 100\,\%$, where p_i and y_i represent the predicted and actual values respectively for the i^{th} example. $\bar{y}$ represents the mean value of the actual values and m is the number of testing examples. This metric measures the deviation of predictions with respect to the actual values.

[5] This sub-sampling process should not be confused with the sub-sampling carried out in bagging.

5 Results and analysis

Section 5.1 compares the performance of the discussed learning methods. A detailed analysis of predictions performed by the bagging strategy is presented further in Sect. 5.2.

5.1 Comparison of learning methods

Table 5 presents the prediction errors of each strategy for the homogeneous and heterogeneous environments. Highlighted values represent the minimum errors for each combination of task type and environment (scenario).

It can be seen that bagging is the best performing method considering all the scenarios. The best bagging results are evidenced for the GELF task, which presents median errors of 28.7 and 22.1 % for the homogeneous and heterogeneous environments respectively.

The highest errors are evidenced in the prediction of runtime for the Random ranker over the homogeneous environment which goes up to 46.4 %. The reason behind this observation is twofold. In the first place, the duration of the task does not depend on the size of the input data, which makes it difficult to characterize its performance. In the second place, Random ranker tasks running on resources of the homogeneous environment present a very short duration (median of 7 ms). This fact makes difficult the proper measuring of the runtime.

Another aspect to note is that in general, all the methods present a low dispersion on their performance, except for ANN which presents much higher deviations. These results can be mainly explained on the selection of parameters for ANN. The variability can be reduced by setting a higher number of training epochs. But, as said before, we decided to keep the default values assuming a setting by an inexperienced user.

It is worth pointing out that these results have been obtained using only benchmark information of the resources. It is expected that incorporating benchmark information of memory or disks will improve the quality of predictions. A second aspect ignored in this study is the effect of CPU load while executing the applications. In this work the CPU load was not measured.

Table 6 presents the error reductions in percentage points of the ensemble method in comparison with the best competitor strategy (indicated in *Versus* column). The d column represents the difference of median errors as $d = \text{median}_{competitor} - \text{median}_{bagging}$ in percent points. To determine if our results are statistically significant we used the Mann–Whitney U test [10] with a confidence level of $\alpha = 0.05$. This is a non-parametric test whose null hypothesis is that two populations are the same. In our case the significance of results can be asserted when the p value (the probability of obtaining the observed results if the null hypothesis is true) is

Table 5 Relative absolute errors for the homogeneous and heterogeneous environments

Task	Strategy	Prediction error—homogeneous env. (%)					Prediction error—heterogeneous env. (%)				
		Mean	Median	Min	Max	SD	Mean	Median	Min	Max	SD
Random	k-NN	47.96	48.13	44.23	51.31	1.72	20.96	20.95	19.23	22.62	1.07
	M5P	46.88	47.05	**43.75**	**50.11**	1.26	22.05	21.98	20.68	24.53	1.03
	ANN	55.83	52.42	45.40	102.42	11.89	29.28	25.36	19.86	54.39	10.09
	SVR	49.58	49.77	46.37	51.53	**1.21**	34.70	34.73	32.16	38.00	1.45
	Bagging	**46.63**	**46.41**	43.92	53.72	1.86	**19.96**	**19.94**	**18.08**	**21.63**	**0.86**
SVM-RFE	k-NN	28.08	28.01	25.77	30.66	1.12	11.08	11.11	10.11	12.48	0.53
	M5P	27.62	27.70	25.46	30.02	**0.95**	15.44	15.68	13.11	17.41	0.97
	ANN	37.48	34.24	27.04	74.97	11.24	18.50	16.26	12.52	55.45	8.07
	SVR	31.11	30.97	29.22	32.89	1.01	27.63	27.59	24.92	29.78	1.17
	Bagging	**26.54**	**26.68**	**24.80**	**28.26**	**0.95**	**10.31**	**10.33**	**9.46**	**11.23**	**0.50**
GELF	k-NN	44.55	44.44	42.49	46.87	1.12	31.24	31.49	28.91	33.22	1.14
	M5P	43.32	43.50	40.77	45.15	**1.07**	30.70	30.78	28.47	32.62	**0.98**
	ANN	57.21	54.18	46.59	83.49	10.38	38.19	35.97	30.74	65.44	6.64
	SVR	52.34	52.51	49.82	55.91	1.40	39.54	39.55	37.61	41.98	1.17
	Bagging	**28.85**	**28.72**	**27.13**	**31.36**	1.16	**22.36**	**22.06**	**20.57**	**31.48**	1.86

Smallest errors are highlighted in bold

Table 6 Error reductions of the ensemble method in comparison with the best competitor strategy (versus column), the difference of median errors (d column) and the significance of results according to the Mann–Whitney U test

Env.	Task	Versus	d	p
Homog.	Random	M5P	0.6	0.13
	SVM-RFE	M5P	1.0	$\mathbf{1.21 \times 10^{-4}}$
	GELF	M5P	14.8	$\mathbf{2.87 \times 10^{-11}}$
Heterog.	Random	M5P	1.0	$\mathbf{3.34 \times 10^{-9}}$
	SVM-RFE	k-NN	0.8	$\mathbf{2.23 \times 10^{-6}}$
	GELF	M5P	8.7	$\mathbf{2.73 \times 10^{-10}}$

less than α, i.e. when we discard the null hypothesis. The table also presents the p values in the last column. Those p values smaller than α are highlighted with bold font indicating the scenarios where bagging improvements are significant.

The table shows that error reductions are significant for 5 of the 6 studied scenarios (all except Random ranker task in the homogeneous environment). Results evidence that for the ranker tasks (in the heterogeneous environment) the error reductions are around 1 percentage point. For the GELF task we can observe higher error reductions of 8.7 percentage points in the heterogeneous environment and 14.8 percentage points in the homogeneous one.

5.2 Analysis of ensemble predictions

This section provides an analysis of the predictions performed by the bagging strategy. Figure 8 presents the predictions performed by the models learned for combination task-environment. Each sub-figure shows the actual runtimes of the tasks versus the values predicted by the ensemble method. Lines in the graphs work as a reference for comparison. They represent the performance of an hypothetical perfect predictor.

From the figure it can be seen that in the case of the heterogeneous environment, the points seem to be closer to the reference line. This indicates that the models achieved lower errors in the heterogeneous environment than in the homogeneous one.

An interesting pattern appears by comparing the predictions for the Random task on both type of environments. The arrangement of points for such task in the homogeneous environment (Fig. 8a) presents a similar shape to the subset of points in the bottom-left of Fig. 8b, which corresponds to the heterogeneous environment. Indeed, both sets of points correspond to execution examples on twister-type resources (see Table 1). Two more groups can be identified between 15 s and 25 s (Opteron type), and over 25 s (Reloaded type). For SVM- RFE and GELF tasks, the points are also grouped but sets are more overlapped than in the case of the RANDOM task for which the sets are clearly separated.

Another pattern shared by the six scenarios is that points present more dispersion as the duration of tasks is longer. It is also noticeable that in such long duration tasks, prediction errors tend to arrange below the reference line. This observation is explained by the fact that there are less examples related to larger execution times, which hinders the proper modeling.

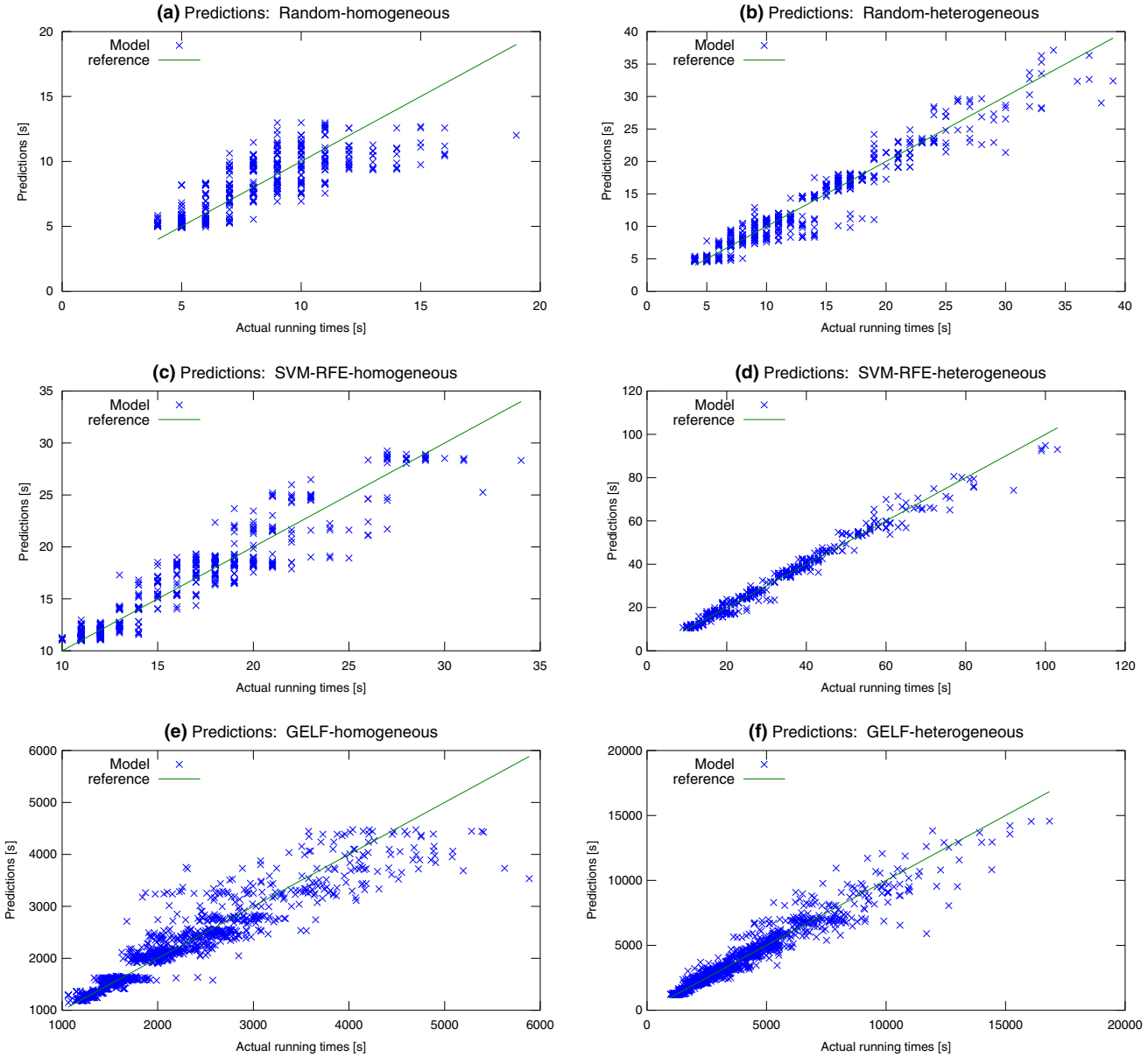

Fig. 8 Predictions for the models generated using the bagging strategy. *Points* represent the actual runtimes versus the predicted values for the execution examples in the validation. *Lines* on each graph represent the performance of an hypothetical perfect predictor

To complete this analysis, Fig. 9 presents the magnitude of Bagging errors for the six scenarios grouping instances by to actual task runtimes. Each graphic is divided in two parts: the upper half shows the mean over-estimation errors and the lower half show the mean under-estimation errors. Each bar also shows the number of instances for which the mean values were computed.

Green bars on the upper half of the graphics show over-estimation errors. We can observe that in general, such error types tend to disappear as the real runtime of tasks augments. Conversely, orange bars on the lower half of the graphics show the opposite behavior for under-estimation errors. The magnitude of such errors tend to augment considerably as the real runtime of tasks augments.

As we said in Sect. 4.3, execution instances present a very right-skewed distribution having most of them concentrated close to the small runtimes (median of the distribution) and fewer instances to the right (long tail). This means that there is more data for short running tasks that for long-running tasks. This fact impacts the on the performance of models in two ways.

First, for short-duration tasks the models present lower errors due to the availability of data, which permits a

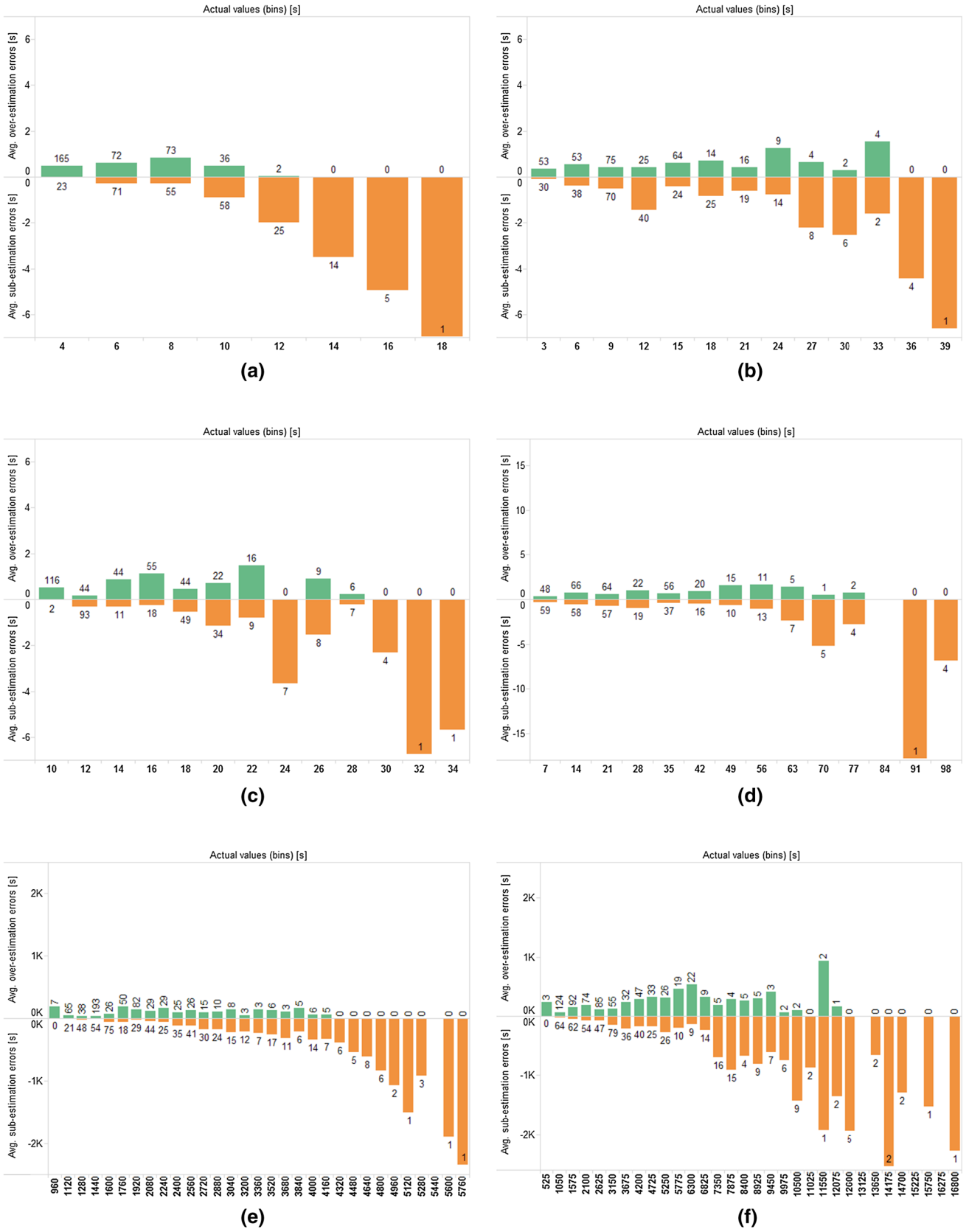

Fig. 9 Over-estimation and sub-estimation errors for the 6 scenarios. **a** Prediction errors: random-homogeneous, **b** prediction errors: random-heterogeneous, **c** prediction errors: SVM-RFE-homogeneous, **d** prediction errors: SVM-FRE-heterogeneous, **e** prediction errors: GELF-homogeneous, **f** prediction errors: GELF-heterogeneous

proper modeling of performance. Second, for long-running tasks, models show a bias towards the underestimation of predictions because of the availability of a larger corpus of short-duration instances. Incorporating more execution instances for long-running tasks during training would help on reducing such bias and improving the quality of predictions.

6 Concluding remarks

In this paper we proposed an approach for the autonomous learning of runtime prediction models suitable for scientific workflows executing on grids or clouds. The novelty of this approach lies on the utilization of an ensemble-learning method entitled *bootstrap aggregating* (bagging) to construct an *ensemble of models* using *data provenance* information and other sources of data available in workflow management systems.

The bagging method consists on the construction of several models using a base learning method on different subsamples of the original performance data. The base learner selected for this work is the M5P learning method for the induction of regression trees. Bagging (as other ensemble-learning methods) leads to reductions of the variance of models and thus to a reduction of the prediction errors.

The conducted experiments were designed for evaluating the performance of models learned using bagging in comparison with other widely used methods for learning standalone models. To this end, these machine-learning methods were applied on performance datasets obtained from the execution of real-world bioinformatics workflows for the analysis of gene expressions data over homogeneous and heterogeneous computing environments.

The results indicate that models learned using bagging outperform strategies that learn standalone models. The best results show significant margins of improvement with respect to their competitors. In the best case, ensemble models present error reductions ranging from 14.8 percentage points to 25.5 percentage points on the homogeneous environment, and ranging from 8.7 percentage points to 17.5 percentage points for the heterogeneous case.

It is worth to point out that this work is related to a large body of previous research on meta-learning, in which learning from features of algorithms and data is considered. The main differences are that (i) we also consider features of the computing environment, and that (ii) we predict runtime whereas in meta-learning, classification accuracy is usually the prediction target.

Undoubtedly, there is much more that can be investigated in relation with complex models for predicting the performance of data-intensive scientific workflows. This paper is an initial step towards such objective. Incorporating more complete information of the resources such as memory and disk benchmarks, characteristics of caches of multi-core architectures (latency, hits, etc.) as well as CPU load is one of the most important next steps to follow.

Acknowledgments This research is supported by the ANPCyT project No. PICT-2012-2731, and by the MINCyT project No. RC0904. MH and FZ were supported by the Czech Science Foundation project No. P202/12/2032. The financial support from SeCTyP-UNCuyo through project No. M004 is also gratefully acknowledged. DAM wants to thank CONICET for the granted fellowship. We also want to thank Alejandro Edera and Rubén Santos for their fruitful comments. Finally, the authors want to thank the anonymous reviewers for their valuable comments and suggestions that helped to improve the quality of this paper.

References

1. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. Mach. Learn. **6**(1), 37–66 (1991)
2. Allan, R.: Survey of HPC performance modelling and prediction tools. Tech. Rep. DL-TR-2010-006, Science and Technology Facilities Council, Great Britain (2010). http://epubs.cclrc.ac.uk/bitstream/5264/DLTR-2010-006
3. Chen, W., Deelman, E.: Partitioning and scheduling workflows across multiple sites with storage constraints. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, vol. 7204, pp. 11–20. Springer, Berlin (2012)
4. da Cruz, S., Campos, M., Mattoso, M.: Towards a taxonomy of provenance in scientific workflow management systems. In: 2009 World Conference on Services—I, pp. 259–266 (2009)
5. Genez, T., Bittencourt, L., Madeira, E.R.M.: Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels. In: Network Operations and Management Symposium (NOMS), 2012 IEEE, pp. 906–912 (2012)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. **11**, 10–18 (2009)
7. Hey, T., Tansley, S., Tolle, K. (eds.): The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research, Redmond, Washington (2009)
8. Holec, M., Klema, J., Železný, F., Tolar, J.: Comparative evaluation of set-level techniques in predictive classification of gene expression samples. BMC Bioinform. **13, Suppl. 10**(S15), 1–15 (2012)
9. Iverson, M., Ozguner, F., Potter, L.: Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. In: Heterogeneous Computing Workshop. (HCW '99) Proceedings of the Eighth, vol. 8, pp. 99–111. IEEE Computer Society, San Juan, PR (1999)
10. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. Ann. Math. Stat. **18**(1), 50–60 (1947)
11. Mao, M., Humphrey, M.: Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), pp. 67–78. IEEE (2013)
12. Marx, V.: Biology: the big challenges of big data. Nature **498**(7453), 255–260 (2013)
13. Mendes-Moreira, J.A., Soares, C., Jorge, A.M., Sousa, J.F.D.: Ensemble approaches for regression: a survey. ACM Comput. Surv. **45**(1), 10:1–10:40 (2012)

14. Monge, D.A., Bělohradský, J., García Garino, C., Železný, F.: A Performance Prediction Module for Workflow Scheduling. In: A.R. de Mendarozqueta et al. (ed.) 4th Symposium on High-Performance Computing in Latin America (HPCLatAm 2011), 40 JAIIO, vol. 4, pp. 130–144. Argentine Society of Informatics (SADIO), Córdoba (2011)
15. Monge, D.A., Holec, M., Železný, F., García Garino, C.: Ensemble learning of run-time prediction models for data-intensive scientific workflows. In: G.H. et al. (ed.) High Performance Computing, Communications in Computer and Information Science, vol. 485, pp. 83–97. Springer, Berlin (2014)
16. Ould-Ahmed-Vall, E., Woodlee, J., Yount, C., Doshi, K., Abraham, S.: Using model trees for computer architecture performance analysis of software applications. In: IEEE International Symposium on Performance Analysis of Systems Software, 2007. ISPASS 2007, pp. 116–125. IEEE Computer Society (2007)
17. Pllana, S., Brandic, I., Benkner, S.: A survey of the state of the art in performance modeling and prediction of parallel and distributed computing systems. Int. J. Comput. Intell. Res. **4**(1), 279–284 (2008)
18. Quinlan, J.: Learning with continuous classes. In: Proceedings of the 5th Australian joint Conference on Artificial Intelligence, pp. 343–348. World Scientific, Singapore (1992)
19. Smola, A., Schölkopf, B.: A tutorial on support vector regression. Stat. Comput. **14**(3), 199–222 (2004). doi:10.1023/B:STCO.0000035301.49549.88
20. Taylor, I., Deelman, E., Gannon, D., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids, 1st edn. Springer, London (2007)
21. Taylor, V., Wu, X., Stevens, R.: Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications. SIGMETRICS Perform. Eval. Rev. **30**, 13–18 (2003)
22. Wang, Y., Witten, I.: Induction of model trees for predicting continuous classes. In: Proceedings of the poster papers of the European Conference on Machine Learning. University of Economics, Faculty of Informatics and Statistics, Prague (1996)
23. Weicker, R.P.: Dhrystone: a synthetic systems programming benchmark. Commun. ACM **27**(10), 1013–1030 (1984)
24. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques, 3rd edn. Morgan, Kaufman (2011)

David A. Monge received the engineering degree on information systems from National Technological University, Mendoza, Argentina, in 2007 and received the Ph.D. degree in computer science at the National University of Central Buenos Aires, Tandil, Argentina in 2013. He is member of the ITIC Research Institute, National University of Cuyo, Argentina. His research interests include Machine Learning, Optimization, Workflow applications, Grid and Cloud Computing.

Matěj Holec recieved the Ph.D. degree at the Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic, in 2015. He was a member of the Intelligent Data Analysis Research Group at the Department of Computer Science, Czech Technical University in Prague where he worked on gene expression analysis using machine learning, in particular set-level classification of gene expression data. His main areas of interest are machine learning and knowledge discovery on massive data sets.

Filip Železný is professor of computer science at the Czech Technical University in Prague. Previously he was postdoctoral researcher at the University of Wisconsin-Madison and visiting professor at SUNY Binghamton. He is interested in machine learning, inductive logic programming and bioinformatics.

Carlos García Garino graduated in engineering at University of Buenos Aires, Argentina in 1978 and received the Ph.D. degree from the Polytechnic University of Catalonia, Barcelona, Spain in 1993. Currently he is Full Professor at the School of Engineering and Head of the ITIC Research Institute, National University of Cuyo, Argentina. His research interests include Computational Mechanics, Computer Networks and Distributed Computing. He has more than 50 papers published in scientific journals and proceedings of international conferences carried out in Argentina, Brazil, Chile, Canada, Spain, France, Portugal, Belgium and Japan.