Meta-Modeling Execution Times of RapidMiner operators

Matija Piškorec[1], Matko Bošnjak[1,2], and Tomislav Šmuc[1]

[1]Ruđer Bošković Institute, Croatia
[2]Faculty of Engineering, University of Porto, Portugal

Abstract

Knowing the execution time of a computational model, especially when dealing with large data, is crucial in deciding whether the solution of the problem is attainable in acceptable time. In the case of data mining processes, typically both the time needed for model learning and model application could be of importance. We developed a meta-mining framework for execution time estimation of data mining algorithm built in RapidMiner. Operator execution time estimation is treated as a machine learning problem for which prediction models are built using execution times obtained by running algorithms on a set of predetermined datasets. With appropriate refitting this experimental methodology is applicable to any data mining environment. We present overall framework with modelling results for a subset of RapidMiner operators, and compare non-parametric distance measures based predictions with polynomial function fitting. Finally, integration of these models in the form of standalone RapidMiner extension is demonstrated and issues related to reliability, scalability and applicability for the overall workflow execution time modelling are discussed.

Keywords: meta-mining, data mining, execution time estimation, RapidMiner extension

1 Introduction

Execution time is one of the key elements of designing both scientific and commercial-level solutions. The magnitude of time cost frequently decides about which methods might be considered and which might be instantly discarded due to their infeasibility. Knowing the dependence of the algorithm's execution time on various data properties would greatly benefit users of data mining models.

One way to estimate resource consumption is by manual or automatic code analysis and using analytical methods of computational complexity theory [2, 11, 3]. However, this approach requires precise knowledge of the function modeling the resource consumption and renders this approach impractical for algorithms whose computational complexity cannot be clearly stated as a function of input data, i.e. optimization methods like Support Vector Machine (SVM).

In this paper we present experimentation and estimation methodology for predicting execution times of various RapidMiner[1] operators that we developed as a part of the e-LICO[2] project. We treated execution time estimation as a machine learning problem and calculated an estimation model for each algorithm. This approach was already demonstrated to be effective for general tasks in heterogeneous metacomputing environment [6] and for RapidMiner classifiers training time with parameter optimization [12]. These are the basic requirements we wanted our framework to satisfy:

1. **Portability/adaptability:** The framework should be easily adaptable to different execution environments and different data mining algorithms.

2. **Robustness and usability:** The framework should be robust and usable for algorithms with different characteristics.

3. **Speed:** Estimates should be generated instantaneously. This is especially important in a workflow environment where estimates for multiple operators have to be performed.

4. **Accuracy:** Estimates should be accurate enough for everyday assignments.

The outline of this work is the following. In section 2 we describe our experimentation methodology, from the datasets to the operators used. Section 3 describes two types of estimation models we use: one based on nonparametric distance measures and one based on simple polynomial function fitting. Finally, sections 4 and 5 describe prototype of our execution time extension for RapidMiner and illustrate its use on a simple use case.

2 Experimentation methodology

Our initial motivation was to investigate whether the machine learning approach can be applied to the execution time problem in the terms of reproducibility of results, scalability and portability. This is why we decided to

[1] http://rapid-i.com/
[2] http://elico.rapid-i.com/

concentrate on one particular and widely used data mining environment, in our case RapidMiner. This ensures our framework is easily accessible to a large community of data miners.

Not all operators contribute equally to the execution time. Therefore we decided to model a selection of classification and feature weighting operators because these typically carry most of the execution time overhead. Table 1 lists RapidMiner operators we modeled with our execution time framework.

Classification	Feature weighting
AutoMLP	Weight by Chi Squared Statistic
Data to Weights	Weight by Deviation
Decision Stump	Weight by Gini Index
Decision Tree	Weight by Information Gain
Default Model	Weight by Information Gain Ratio
Linear Discriminant Analysis	Weight by PCA
Naive Bayes	Weight by Relief
Random Forest	Weight by Rule
Random Tree	Weight by SVM
Regularized Discriminant Analysis	Weight by Uncertainty
Rule Induction	Weight by User Specification
Single Rule Induction Single Attribute	Weight by Value Average
SVM (LibSVM): linear and rbf kernel	

Table 1: RapidMiner operators we modeled with our execution time estimation framework and whose execution time models are available in our prototype extension.

Most of these operators have parameters that nontrivially influence their execution time but here we avoided their detailed modeling and modeled only the default parameters for two reasons. First, we wanted to keep the complexity of our models low. Second, we target our framework for inexperienced users who are are typically interested only in the behaviour with the default parameters.

However, for some algorithms, SVM being one of them, there are no sensible default parameters so it makes sense to investigate their influence on the execution time and incorporate them into the estimation model. This is why in the case of SVM we explicitly model two kernel types: linear and radial basis function kernels.

2.1 Meta-features selection

We concentrate on two dataset meta-features easily available in RapidMiner: number of examples (N) and number of attributes (M), although in principle

any number of meta-features can be used.

We also experimented with other meta-features more time consuming to obtain and not readily available in RapidMiner. These include information theoretic and statistical-based features from METAL[3] project [7], features used to describe categorical features (information theoretic) and continuous features (statistical-based) [9], geometrical and topological-based features [5, 4, 8], and landmarking features [10, 1].

For some of them we concluded that the additional time cost in obtaining those meta-features would increase the time analysis and in some cases overtake the execution time of some of the models, and for others we did not see substantial increase in accuracy of estimation. As an additional benefit, low number of meta-features allows intuitive visualization of estimation models, like the ones in Figures 2 and 4, that aids user experience.

2.2 Measuring execution time

Measurement of execution times is performed inside RapidMiner and automated through a dedicated workflow[4] whose diagram is showed on Figure 1. Each operator is trained on 200 random polynomial classification datasets with number of examples and number of attributes ranging randomly from 4 to 2000. Each measurement is performed five times and median time is taken as reference for that combination of examples and features.

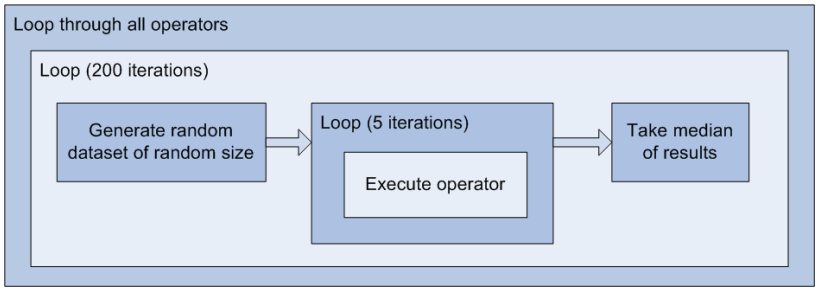

Figure 1: Diagram of the execution time measurement workflow.

2.3 Machine calibration

Because execution time depends on both the algorithm implementation and the machine overall performance capabilities we decided to introduce a *calibration* parameter that makes our estimation framework portable to new execution environments. This avoids the need to perform new set of execution

[3] http://www.metal-kdd.org/

[4] http://www.myexperiment.org/workflows/2905.html

time experiments each time the environment changes. In general, calibration procedure should provide reliable indicator for overall performance capabilities of the execution environment, and it should be time-limited so that it can be used even for online recalibration (e.g. on the same machine but different load conditions).

Our calibration procedure consists of benchmark RapidMiner workflow[5] that iterates some simple operator ("Naive Bayes" in our case) on dataset of fixed size and takes median of all execution times. This median time is compared to a reference time from a machine on which the experiments were performed and their ratio is used as a scaling factor for execution times on a new machine. We found this approach, where just one landmark feature is used to describe overall machine performance, robust enough for our needs, although more elaborate techniques with more features are also possible [12].

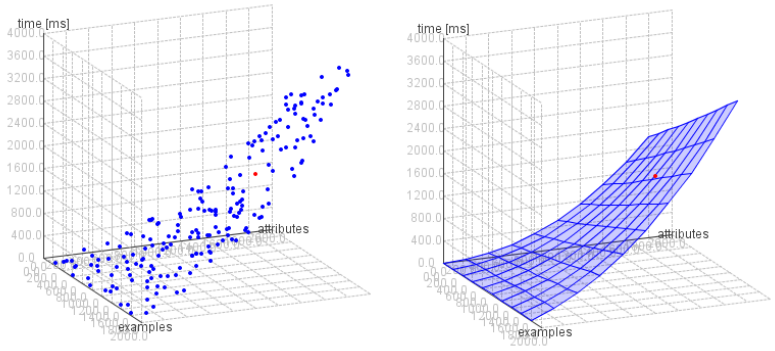

Figure 2: Graph representations of execution time models for Pager (left) and polynomial (right) models as visualized in our extension. This example is for "Weight by Information Gain" operator.

3 Estimation methodology

Having selected small number of features for our execution time estimation we decided to use two simple models as our estimators:

- **Pager:** A kNN-based algorithm for regression [13]. We use it here because it is parameterless and provides reliable and robust execution time estimations without the need for explicit statement of the regression equation. Our implementation is in Java.

[5] http://www.myexperiment.org/workflows/2903.html

- **Polynomial:** Cubic function fitting suitable for small number of meta-features. Our implementation is simple gradient descent optimised with `fminsearch` function in Matlab.

Figure 2 shows graphical representations of two estimation models in our extension: Pager as a collection of points corresponding to the 200 experimentation datasets and polynomial as surface plot in the meta-features space spanned by number of examples and number of attributes. Polynomial fitting gives stable and accurate estimations even in the range outside original experiments (extrapolation), as can be seen in Figure 3. On the other hand, Pager estimations are more stochastic. This is demonstrated on the graph on right side of Figure 5 in the simple workflow consisting of "Weight by Information Gain" and "SVM" with rbf kernel operators. In its default implementation Pager assumes linear extrapolation so it always underestimates execution times for larger datasets because they typically exhibit quadratic, cubic or even exponential relationship on the dataset size. This problem can probably be alleviated by working with the logarithm or square root of execution times.

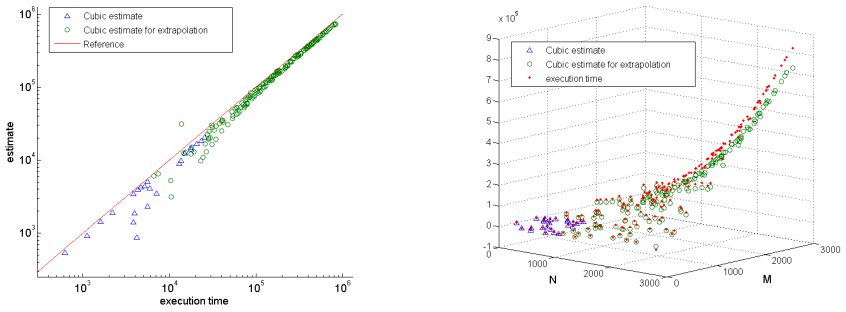

Figure 3: Extrapolation of estimated execution times to parameter ranges not used in experimentation for polynomial model. This example shows execution time estimates for a simple workflow consisting of "Weight by Information Gain" and "SVM" with rbf kernel operators where individual estimates for each operator are summed. Graphs show comparison between estimates and real values of execution times for test cases inside experimentation range (blue triangles) and outside experimentation range (green circles). Red points on the right are real execution times while red line on the left is a reference representing ideal estimation.

4 Execution time extension for RapidMiner

We implemented a RapidMiner extension that interactively displays execution time information for operators selected in the main process. The screenshot of the extension is shown in Figure 4 and interactive graphs of two estimation models in single and multiple selection modes on Figure 2.

As a distance-based estimator, Pager model is represented with pointwise data corresponding to the training datasets while cubic model is represented with a polynomial surface plot. Figure 2 shows both models as represented in our extension. When multiple operators are selected in the process window, the resulting execution time estimation is calculated as the simple sum of the individual estimations, assuming the same input dataset is applied to each operator. The resulting graphical representation of the model obtained in this way in Figure ?? shows estimates for the uniform grid of fictional datasets that serve as rough approximation of the execution time surface.

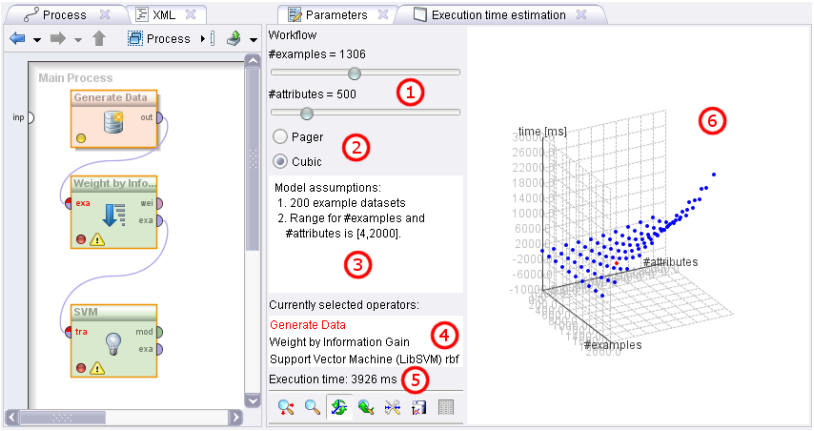

Figure 4: Screenshot of extension showing basic functionalities: (1) sliders for choosing number of attributes and examples for the input dataset, (2) prediction model (Pager or polynomial), (3) info box about for the estimation model, (4) list of currently selected operators and (5) their execution time estimate, (6) interactive display showing execution time surface in relation to the number of parameters and number of attributes.

Figure 4 shows graphical user interface of our extension inside RapidMiner. User can choose the parameters of the input dataset (number of examples and number of attributes) and estimation model ("Pager" or "Cubic" where cubic stands for polynomial) for operator selected in the process view. When multiple operators are selected they are listed and the names of the ones that

do not have an estimation model are colored red. Model representation in the form of interactive 3D plot is shown on the right. It includes a red reference point that locates resulting execution time. The numerical value of execution time estimate is also displayed.

5 Use case: simple classification workflow

Figure 5 shows simple use case consisting of data generation operator, feature weighting operator and classification operator. "Generate Data" operator generates random 200 polynomial classification datasets with number of examples and number of attributes randomly selected from the range $[4, 2000]$. Estimates are performed for feature weighting ("Weight by Information Gain") and classification ("SVM" with rbf kernel) operators individually and then summed to obtain the final estimate. The estimate is then compared with the true value of the execution time and the results are plotted on the right hand-side of Figure 5.

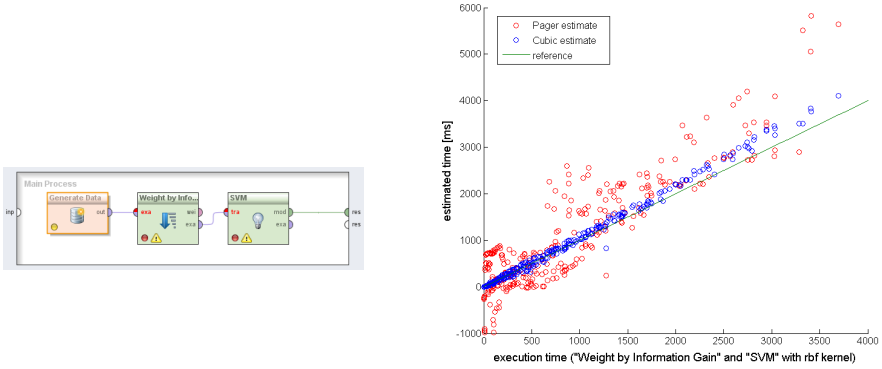

Figure 5: Simple use case consisting of feature weighting operator ("Weight by Information Gain") and classification operator ("SVM" with rbf kernel). On the right is the comparison of estimated and real execution times for estimation with Pager and polynomial model.

6 Conclusion and future work

Correctly dealing with the metadata propagation is one of the most pressing issues for increasing the quality of workflow estimates. Characteristics of datasets are changing during workflow execution: examples are sampled out

of the dataset, attributes are selected with various criteria. We expect that some of the cases could be dealt with efficiently before actual execution of the workflow. Unfortunately, some cases will probably stay elusive unless estimation is performed in real-time, during the workflow execution. This is the case when meta-features change according to the nontrivial criteria not available until the runtime, or criteria that is not computable from the meta-features themselves. For example, in sampling that selects examples with values *less or greater* than some prespecified values (because information on the number of such values is not encoded in the meta-features) or feature selection that selects best K features where K is a macro whose value is not defined until the runtime.

Parallel execution of operators is another issue that influences overall execution time and that becomes more important due to the advances in multiprocessor architecture. It manifests on the level of workflows as well as on the level of individual operators. We suppose pure machine learning approach will not be sufficient to solve this problem without the additional information on the algorithm implementations and inner workings of the execution environment (i.e. operating system).

Automatic collection of execution time data can potentially make experimentation methodology easier and more up-to-date with the current changes in the operator implementations. We suppose this could be done in a similar manner as collecting operator usage statistics that already exist in RapidMiner. Consolidating execution times from various machines and execution environments would probably prove as a fruitful future research direction.

Acknowledgments: This work is supported by the European community 7^{th} framework ICT-2007.4 (No 231519) "e-LICO: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science".

References

[1] S. D. Abdelmessih, F. Shafait, M. Reif, and M. Goldstein. Landmarking for meta-learning using rapidminer. In *RapidMiner Community Meeting and Conference*, September 2010.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and Stein C. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[3] J. Engblom, A. Ermedahl, M. Sjödin, J. Gustavsson, and H. Hansson. Towards industry strength worst-case execution time analysis, 1999.

[4] T. K. Ho, M. Basu, and M. Law. Measures of geometrical complexity in classification problems. *Data Complexity in Pattern Recognition*, pages 1–23, 2006.

[5] T. K. Ho and Basu. M. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):289–300, 2002.

[6] M. A. Iverson, F. Özgüner, and L. C. Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Transactions on Computers*, 48:1374–1379, 1999.

[7] C. Koepf, C. Taylor, and J. Keller. Meta-analysis: Data characterisation for classification and regression on a meta-level. In *In proceedings of International Symphosium on Data Mining and Statistics*, 2000.

[8] A. Orriols-Puig, N. Maci, and T. K. Ho. Documentation for the data complexity library in c++. Technical report, La Salle - Universitat Ramon Llull, 2010.

[9] Y. Peng, P.A. Flach, C. Soares, and P. Brazdil. Improved dataset characterisation for meta-learning. *Discovery Science*, pages 141–152, 2002.

[10] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *In proceedings of International Conference on Machine Learning*, 2000.

[11] P. Puschner and Ch. Koza. Calculating the maximum execution time of real-time programs. *Real-Time Systems*, 1(2):159–176, 1989.

[12] M. Reif, F. Shafait, and A. Dengel. Prediction of classifier training time including parameter optimization. In *Proceedings of the 34th Annual German Conference on Artificial Intelligence*, volume 7006, pages 260–271, 2011.

[13] H. Singh, A. Desai, and V. Pudi. Pager: Parameterless, accurate, generic, efficient knn-based regression. *Database and expert systems applications*, 6262:168–176, 2012.