

Improving the Execution of KDD Workflows Generated by AI Planners

Susana Fernández, Rubén Suárez, Tomás de la Rosa, Javier Ortiz, Fernando Fernández, Daniel Borrajo¹
and David Manzano²

Abstract.

PDM is a distributed architecture for automating data mining (DM) and knowledge discovery processes (KDD) based on Artificial Intelligence (AI) Planning. A user easily defines a DM task through a graphical interface specifying the dataset, the DM goals and constraints, and the operations that could be used within the DM process. Then, the tool automatically obtains all the possible models that solve the task by combining the different KDD actions. The models are obtained by the execution of workflows in a DM engine. In turn these workflows are automatically generated using a state-of-the-art AI planner. Since the number of potential KDD workflows that solve a DM task can be huge, PDM allows the user to set up some quality criteria (accuracy, execution time, ...) to prune and rank the space of workflows. These criteria can be used when planning to guide the search process towards good workflows. This requires to model the effects that each KDD action has on the criteria. The first versions of the PDM Tool included estimations made by experts, and then by a machine learning module that improves them through the analysis of execution of the generated workflows in different datasets. This paper presents our current work on the PDM Tool for improving the estimations on those values, based on a more fine grained analysis of results.

1 Introduction

Data Mining (DM) and Knowledge Discovery in Databases (KDD) is a very dynamic research and development area that is progressing constantly. Recently, researchers are defining the third generation of DM and KDD systems. The first generation of DM systems support a single algorithm or a small collection of algorithms that are designed to mine attribute-valued data. Second generation systems are characterized by supporting high performance interfaces to databases and data warehouses and by providing increased scalability and functionality. And the emerging third generation systems should be able to mine distributed and highly heterogeneous data found across the network of computer systems and integrate efficiently with operational data/knowledge management and DM systems. This implies, among other things, the implementation of DM and KDD tools to enable the construction of KDD workflows (representing potentially repeatable sequences of DM and data integration steps) [18]. Such tools should be built on the basis of standard languages and available state-of-the-art technology.

At a high level, there are four main elements to define in the KDD process: the training data (obtained by selecting, pre-processing, and transforming the initial dataset), the model representation formalism, the learning algorithm, and how to evaluate the model. The number of combinations of those four elements is huge, since there are many different techniques appropriate for each phase, all of them with different parameter settings, which can be applied in different orders. Thus, the KDD process is sometimes seen as an expert process where DM engineers transform original data, execute different mining operators, evaluate the obtained models, and repeat this process until they are satisfied. The complexity of this combinatorial process suggests using automated approaches that are able to generate potential useful workflows and then execute them appropriately. Precisely, Automated Planning (AP) technology has become mature enough to be useful in applications that require selecting and sequencing actions [11], as it is the case of generating KDD workflows.

PDM is a tool for automatic planning of KDD workflows based on these ideas [8]. PDM describes KDD operations in terms of AP by defining a planning domain in PDDL (Planning Domain Definition Language) that is considered as the standard language in the planning community [9]. It uses another standard, the DM standard language PMML [12], to describe KDD tasks. It receives as input a KDD task, specified in a PMML file. The PMML file is automatically translated into a planning problem described in PDDL. So, any state-of-the-art planner can be used to generate a plan (or plans), i.e. the sequence of KDD actions that should be executed over the initial dataset to obtain the final model. Each plan is translated into a KDD workflow and it is executed by a machine learning engine. In our case, we employ one of the most used DM tools, WEKA [21]. In WEKA, workflows are described as files with a specific format, KFML, and datasets are described as ARFF (Attribute-Relation File Format) files. The results of the KDD process can be evaluated, and new plans may be requested to the planning system. An important characteristic of PDM is that it is a distributed architecture where each module can be executed in a different host. Thus, in combination to the use of standard languages, it makes it a modular architecture, so it is possible to substitute any of its components. For instance, we could change the DM tool or the planner. We would only have to adapt the translator of plans, to deal with the input requirements of the new DM tool.

PDM represents each common KDD operator as a planning action in the PDDL domain with its preconditions and effects. The effects include estimations on the variations of the desired mining results, as execution time, accuracy and errors, or comprehensibility. In a first approach, these variations were initially set to a value estimated by an expert. But, when compared with real values obtained by executing WEKA over different datasets, we saw that those estimated val-

¹ Universidad Carlos III de Madrid, Leganés, Spain, email: susana.fernandez@uc3m.es

² Ericsson Research Spain, Madrid, Spain, email:david.manzano.macho@ericsson.com

ues differ from the real ones. So, we enhanced the PDM architecture by integrating machine learning techniques to improve the KDD by planning process. However, the learning was performed considering the whole KDD process, i.e. PDM obtained the models and the total mining results for each training DM task without distinguishing between the particular effects due to each KDD action. In our current work that we describe in this paper, we are separating the execution of pre-processing actions from classification/regression actions, so that we can learn better predictive models of execution of KDD actions. In parallel, we are also improving the features used for learning the models.

The paper is distributed in the following way. Section 2 presents the overall PDM architecture. Section 3 the learning component as it is currently implemented, i.e. considering the whole KDD process. Section 4 describes some initial experiments on the original PDM architecture. Section 5 explains current work on the PDM tool. Section 6 presents the related work. And, the last section draws the conclusions and suggests future work.

2 The PDM Tool

This section summarizes the PDM Tool. More details of the architecture can be found in [8]. Figure 1 shows the PDM architecture. The PDM Tool is composed of four modules: Client, Control, Datamining and Planner; each one may be executed in a different computer connected through a network. We have used the Java RMI (Remote Method Invocation) technology that enables communication between different programs running JVM's (Java Virtual Machine). The planner incorporated in the architecture is SAYPHI [6] and the DM Tool is WEKA [21]. However other DM tools could have been used. Also, any other planner that supports fluents and optimization metrics may be used too.

The *Client* module offers a graphical interface that provides access to all the application functionalities. Using the interface, a user generates a PMML file from a high level description of the DM task. Then, the *Client* module sends the PMML description to the *Control* module. At the end of the execution, it receives the results in a file.

The *Control* module interconnects all modules. It performs some translations in order to provide the data to the rest of modules in the correct format and update the learned values in the PDDL problem file. The translations needed are: from PMML to a PDDL problem, `PMML2PDDL`; and from a PDDL plan to KFML, `Plan2KFML`. The input to the module is the DM task coded in the PMML file provided by the *Client* module, the dataset and a file with planning information provided by experts. First, the `PMML2PDDL` translator generates the PDDL problem file from the PMML file with the information provided by experts. Then, the module checks if there are learned information for the planning domain. If so, the PDDL problem file is updated with the learned information; otherwise the module does not modify the PDDL problem file and continues the execution using the expert knowledge.

Then, the planner is executed to solve the translated problem. The returned set of plans is translated to several KFML files. Finally, the DM Tool executes every plan in KFML format. The *result* is a compressed file containing a set of directories, one for each plan. Each directory contains the model generated by the DM Tool, the statistics related to the evaluation of the model, the plan generated by the planner, and the corresponding KDD workflow in KFML. Once the module has the information provided by the execution of all plans, it may perform the learning task in order to obtain more accurate values for the PDDL problem files for future executions.

The *Datamining* module permits the execution of DM tasks in the WEKA DM Tool through Knowledge Flow plans. It can obtain the model output and the statistics generated as a result of the Knowledge Flow execution. The inclusion or removal of ARFF files are managed by the user through the options offered in the user interface. The input to the module is a KFML file and the output is a compressed file that contains the model generated and the statistics related to the evaluation of the model.

The *Planning* module receives the PDDL problem file generated by the *Control* module and uses the PDDL domain file, which has been previously defined for the PDM architecture. The PDDL domain contains the generic actions that can be executed by a DM tool. Each action specifies:

- Arguments: the parameters assigned to the generic action.
- Preconditions: the facts that must be achieved previous to the action execution. For instance, a model needs to be generated by a learning algorithm previous to the model evaluation
- Effects: the facts achieved with the action execution
- Costs: the estimated variation of the desired mining results, as the accuracy or execution time. Action costs are part of the action effects and are computed as a function of available features of the dataset and constant values given by the expert.

Figure 2 shows the PDDL action used for representing the DM task that learns a model with the training data. In this example, the *exec-time* function depends on the number of dataset attributes, the number of instances and the constant `model-instance-exec-time`. This constant is particular to each model received as an action argument. Thus, adjusting these constant values an expert can reproduce the different estimated durations of DM tasks. Moreover, variations on other functions, (i.e., *unreadability* and *percentage-incorrect*) are also reproducible with its particular constant values. The objective of the PDM learning component, explained in the next section, is to learn these values based on the experience of WEKA knowledge flows executions.

After solving the PDDL problem, the *Planner* module returns a set of plans in XML format ready for the conversion to a KFML format. Currently, planning tasks are solved by the SAYPHI planner [6], but the architecture could use any other planner that supports fluents, conditional effects and metrics. We have used SAYPHI because it: i) supports an extensive subset of PDDL; and ii) incorporates several search algorithms able to deal with quality metrics.

3 The Learning Component

As defined above, the PDM architecture uses expert knowledge to define some important planning information, like the time required to build an specific model, or the estimated accuracy of the resulting model. Initially, these values are defined by an expert. However, those estimations can be far from correct values, since they are hard to define. Also, it can become difficult to provide those values under all possible alternative uses of the techniques, different orders in the execution of preprocessing techniques and the different domains.

The goal of the learning component is to automatically acquire all those estimations from the experience of previous data mining processes. The data flow of this component is described in Figure 3. The main steps of this flow are:

1. Gathering Data Mining Results: the goal is to gather data mining experience from previous data mining processes. All the information is stored in an ARFF file. For a given data mining process, the following information is stored:

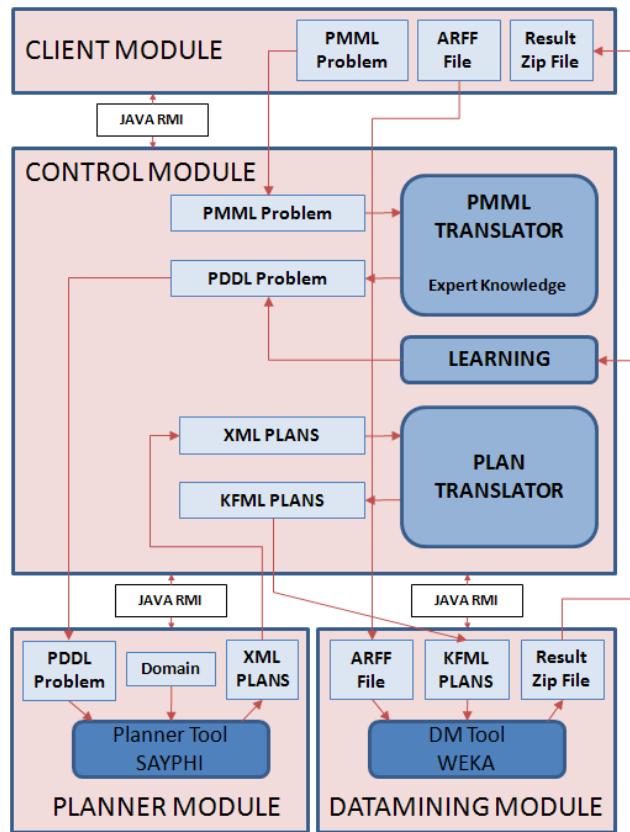


Figure 1. Overview of the PDM architecture.

```

(:action train-classification
 :parameters (?m - Model ?n - ModelName ?d - DataSet ?fi - FieldName ?t - TestMode)
 :precondition
 (and
  (implements ?m classification ?n)
  (is-field ?fi ?d)
  (is-class-field ?fi)
  (dataDictionaryDataField-otype ?fi categorical)
  (eval-on ?d ?t)
  (task-model-ready ?d))
 :effect
 (and
  (is-classification-model ?d ?n ?fi)
  (has-model-of classification)
  (not (preprocess-on ?d))
  (not (task-model-ready ?d))
  (increase (exec-time)
   (* (* (model-instance-exec-time ?n)
      (* (train-datasize-ratio ?t) (thousandsofInstances)))
      (* (dataDictionaryNumberOfFields) (dataDictionaryNumberOfFields))))
  (increase (unreadability) (model-instance-unreadability ?n))
  (increase (percentage-incorrect)
   (* (model-instance-percentage-incorrect ?n) (CorrectionFactor))))

```

Figure 2. An example of an action in the PDDL domain file used in PDM.

- Information about the dataset: number of instances of the dataset, number of attributes of the dataset, number of continuous attributes, etc.
- Information about the model to build: the type of model (association, clustering, general regression, neural network, tree, etc.), the algorithm used to learn the model (RBFNetwork, J48, etc.), the type of function (classification, regression, clustering), the learning parameters, etc.
- Information about the results obtained: type of evaluation (split, cross validation, training set), time to build the model, accuracy, mean squared error, etc.
- The plan that represents the data mining workflow, and that has

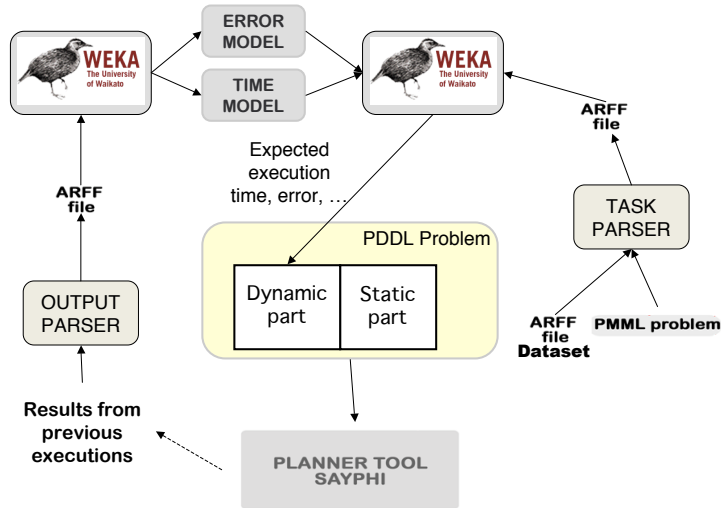


Figure 3. Learning Flow in the PDM Architecture.

been executed to obtain the model

2. Model generation: the information obtained in the previous step is used to learn prediction models. The functions to learn are time, accuracy and SME (in Figure 3, error and time models). These models can be generated with the WEKA tool, as shown in the figure.
3. Given a new dataset, a model type, a function, and a learning algorithm, and using the models generated in the previous step, we obtain a prediction of the learning time, accuracy and SME that will be obtained if we perform a new data mining process with such dataset, model type, function and learning algorithm. These estimations are included in the PDDL problem file, working out the value of the corresponding constant from the action cost formula. The updated values are then used when planning new data mining processes. Figure 4 shows an example of how the fluents of the dynamic part of the PDDL problem file are updated. In the figure, the exec-time of `treemodell` (i.e., a specific tree model corresponding to a learning algorithm with its parameters) is updated, among others.

The updated values in the example are not final estimations, but factors used in the computation of real estimations, as defined in the action cost formula of the `train-classification` operator of the PDDL domain file (see Figure 2). For instance, when we define `(= (model-instance-exec-time treemodell) 0.001)`, we do not mean that the execution time of learning a tree is 0.001, but that such time is computed as a function of the number of instances and attributes, weighted somehow with the learned factor 0.001.

There are two ways to update the PDDL problem file with these estimations: off-line and on-line. Off-line updates require to obtain information of many data mining processes, use the execution information to build the models, and employ these models to update

the PDDL problem file, which will stay fixed in the future. On-line updates assume that, while new data-mining processes are executed, new learning examples are obtained, so the models can be dynamically updated, and the PMML problem file is continuously updated. We will only show results for off-line updates in the following section.

4 Experiments

This section presents the experimental evaluation for the original learning-component of the PDM Tool. The experiments were designed to assess the level of improvement the current implementation of the PDM Tool (i.e. considering the whole KDD process and the original features) can achieve based on the experience acquired from working with past datasets. For the evaluation, we have used twenty datasets from the UCI KDD Archive³. For each dataset we built a PMML file with two filters (i.e., attribute selection and discretization) and six classification algorithms (i.e., J48, IBK, Simple-Logistics, Multi-layer Perceptron, RBF-Network and SMO). We kept WEKA default parameters for both filters and algorithms. In addition, the PDDL domain considers three evaluation methods: training-dataset, split and cross-validation; also with WEKA default parameters. As a result of different combinations, we got 72 different knowledge flows for each dataset. We focus this evaluation on planning for minimizing the execution time. In order to avoid bias of arbitrarily selected values for execution time factors (fluents representing the estimated values), we decided to use equal values for the DM learning algorithms. Thus, this is the baseline for comparison that represents no expertise in the selection of these values.

The regression models for cost (time) prediction are the results of the M5Rules algorithm [10]. The reason for selecting this particular algorithm is the understandability of the output, which allows us to

³ <http://archive.ics.uci.edu/ml/datasets.html>

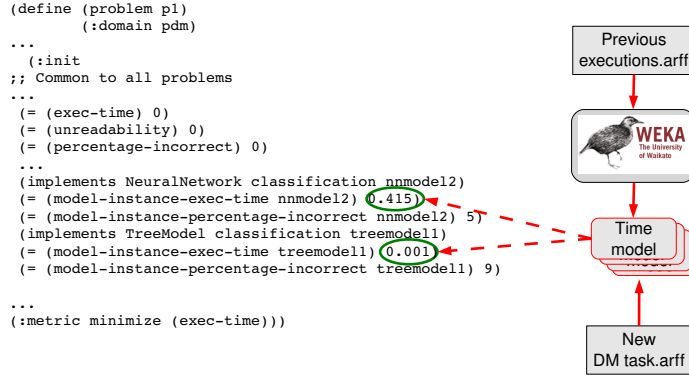


Figure 4. Example of how the expected DM results values are updated.

check the adequacy of the model. We have used the WEKA implementation of M5Rules.⁴ The model is queried with dataset attributes described in section 3, to obtain a time prediction for the execution time of each algorithm. Figure 5 shows an example of a rule in one of the learned models. Examining the models obtained so far we have observed that the learning algorithm, followed by the number of attributes are the most significant attributes for time prediction (as expected). The size of the rule sets is usually small (two or three rules).

We expect that the learning configuration will make estimations closer to the real execution time. Since we do not impose particular constraints in the PMML specification, the learning configuration will return the same number of plans (72) but in a different order due to the updated values. Therefore, the evaluation can be defined as the comparison of the ranking of plans generated by the planner and the real ranking.

With this experimental setup we have performed an off-line performance evaluation. The objective is to evaluate the performance of PDM that uses the learning component, after completely training the tool. We follow the leave-one-out strategy. For each evaluated dataset, we train the tool with the 19 other datasets.

For the evaluation we follow a ranking evaluation in the spirit of [19]. We establish the plan ranking for the set of plans $\{\pi_1, \dots, \pi_n\}$ (with $n = 72$ for this evaluation), using the plan cost function $c(\pi)$. Thus, plans are sorted in increasing order of cost (i.e., estimated execution time) where $c(\pi_i) \leq \dots \leq c(\pi_j)$. The real ranking is computed from the WEKA execution of the plans/knowledge-flows. If we define as $t(\pi)$ the time for the plan execution, the real ranking is computed from the relation $t(\pi_k) \leq \dots \leq t(\pi_l)$. As a result, the ranking position $R(\pi, f)$ is a function of a plan π and the function f for sorting all plans (f can be either t for real ranking order or c for planner ranking order). For instance, $R(\pi_1, c) = 10$ means that for plan π_1 , there are 9 plans that have lower estimated time. For any two plans π_1 and π_2 , if $R(\pi_1, c) < R(\pi_2, c)$, we consider an order switch if $R(\pi_1, t) > R(\pi_2, t)$. The interpretation over an order switch is the following: we consider an error if a plan is considered a better choice than another one by the planner, but in the real execution (with WEKA) it is actually the opposite. For the evaluation, we compute the following two measures:

- *Single ranking order switches* (Δ): It counts how many pairs (π_i, π_j) are ordered incorrectly when comparing estimated times against real times.
- *Weighted ranking order switches* ($W\Delta$): It multiplies each incorrectly ordered pair by the ranking distance as a measure of the error being committed.

These measures are normalized within the [0,1] interval, where 0 represents the perfect reverse ranking (all possible ordering errors) and 1 corresponds to the perfect ranking (no ordering errors). A random ordering criterion would obtain an average ranking score of 0.5.

4.1 Off-line Performance Results

Table 1 shows the result for the leave-one-out strategy. Some datasets for the no-learning configuration get values over the random ranking. This is mainly due to the influence that the evaluation methods have in the total execution time. For instance, if one uses the same algorithm, it is expected that a 10-fold cross-validation will last longer than a single training with a dataset split. The learning configuration improved the Δ measure in 18 out of 20 datasets, raising 11 percentage points in the average measure. The $W\Delta$ measure was improved in the same 18 datasets as well. The 14 percentage points of difference show that errors committed in order switches are less relevant with the learning configuration. Two datasets, *magicgamma* and *waveform-5000*, worsen both measures after learning. We have no explanation yet for this behaviour.

5 Current Work

This approach can be improved in two directions. The first one relates to the fact that the metric for which we learn the regression model is extracted from the execution of the whole plan, independently of the individual KDD actions that were executed, and its prediction is only used for the main DM task (classification/regression). Thus, we do not differentiate between using a couple of pre-processing techniques and not using them, and the resulting predictive model is a bit coarse. The second improvement we found out with the previous experiments relates to the input features for learning the predictive model. Therefore, our current goals are to have a more controlled

⁴ <http://weka.sourceforge.net/doc/weka/classifiers/rules/M5Rules.html>

```

Rule: 3
time =
  0.0038 * number-instances
  + 0.163 * number-attributes
  + 61.1754 * model-instance=GRMODEL2, SVMMODEL1, NNMODEL1, TREEMODEL2, LAZYMODEL1, NNMODEL3
  - 4.3018

```

Figure 5. Example of a rule in the M5Rules time prediction model.

Dataset	No-Learning		With Learning	
	Δ	$W\Delta$	Δ	$W\Delta$
arrhythmia	0.60	0.66	0.67	0.75
car	0.53	0.58	0.72	0.82
credit-a	0.49	0.52	0.59	0.65
diabetes	0.46	0.48	0.65	0.73
glass	0.51	0.54	0.69	0.78
haberman	0.42	0.43	0.67	0.76
hepatitis	0.41	0.41	0.52	0.55
hypothyroid	0.61	0.69	0.69	0.78
iris	0.48	0.51	0.69	0.79
kr-vs-kp	0.57	0.62	0.65	0.72
magicgamma	0.62	0.70	0.59	0.65
mfeat-fourier	0.59	0.65	0.79	0.89
mushroom	0.58	0.64	0.65	0.72
nursery	0.63	0.70	0.70	0.78
optdigits	0.61	0.68	0.71	0.82
page-blocks	0.63	0.70	0.71	0.8
waveform-5000	0.62	0.69	0.60	0.65
wine	0.47	0.52	0.69	0.79
yeast	0.58	0.64	0.61	0.65
zoo	0.44	0.45	0.51	0.55
Average	0.54	0.59	0.65	0.73

Table 1. Measures for ranking order switches between estimated and real execution time in the evaluated datasets.

execution of the plans, and the use of a better set of features when learning the criteria models. Both goals aim at improving the learning step of those models, so that we have better predictive models of the behaviour of KDD actions in different datasets. In relation to the first goal, we are changing the execution of the whole plan (KDD workflow), followed by an evaluation of the results, by an execution step by step of the plan. We differentiate between “plan actions” (the actions in the domain model used by the planner) and “execution actions” (specific KDD tasks that need to be performed to consider a plan action executed). Once the planner has generated a set of plans, each of the resulting plans is divided into its plan actions. Every plan action is passed to an execution module that performs three steps: map each plan action into one or more execution actions, execute each of these execution actions and obtain its results (e.g. execution time, accuracy). The results of applying the corresponding execution actions are aggregated to compose the corresponding plan action cost. This controlled execution environment allows PDM to establish how actions are executed, not repeating actions that have already been executed in previous plans (for example loading a dataset, or applying the same sequence of filters), and providing fine-grained information about action costs.

In relation to the second goal, we have incorporated more information about the datasets, such as the meta-learning characteristics described in [5]. These features are added to the previous ones that were used to learn the model of the corresponding plan action cost estimation (execution time, accuracy, ... models). In plan cost esti-

mation this current approach has two main advantages over previous work: a better characterization of preprocessing actions based in the dataset characteristics where they are applied, and a characterization of an algorithm expected time and accuracy, independently from previous preprocessing actions. Examples of new features are:

- Mean standard deviation of numeric features
- Average coefficient of variation of numeric features
- Average skewness of numeric features
- Average kurtosis of numeric features
- Average normalized entropy of nominal features.
- Average of mutual information of class and attributes of nominal features
- Equivalent number of attributes of nominal features
- Noise-signal ratio of nominal features

Currently, global meta-characteristics are considered, because the representation of the domain is propositional. We are also considering posing the learning problem in the relational setting, so that we can also consider features relevant to individual attributes.

6 Related Work

The work presented in this paper can be framed into three different fields. First, it is a tool based on AI planning for assisting data mining tasks. There have been already several works on automating the data mining process through planning techniques [1, 2, 17]. The closest to our work is the second one where authors present the Intelligent Discovery Assistant (IDA). It lists potentially useful DM operations, as well as a ranking of those. They represent common DM operations in an ontology instead of in a traditional planning domain. The ontology represents a hierarchy of operations in a planning style, including preconditions, effects and costs. IDA implements its own search algorithm using that ontology and it returns a list of all possible solutions by finding all plans that achieve the goals, but without using any state-of-the-art planner. Similarly to us, each DM operator includes estimations of the effects on each goal, as accuracy or execution time, and the returned plans try to optimize them. They can dynamically include new DM operators with their corresponding estimations that influence the ranking, but they cannot automatically change the estimated values as our learning module does. Also, since they use their “ad hc” planning system, they cannot benefit from the advances on this field, as we can do by using the PDDL standard.

Second, the learning module of the PDM tool applies machine learning techniques for improving future data mining episodes. One could see it as a form of meta-learning [3]. Meta-learning studies methods that exploit meta-knowledge to obtain efficient models and solutions by adapting machine learning techniques. Apart from the techniques necessary to build meta-learning systems, it involves addressing some tasks that are also important for our work. For example, what the typical properties of datasets (or meta-features) are that

have the strongest impact on learning [15] or how to profit from the repetitive use of a predictive model over similar tasks [20]. The description of a dataset in terms of its meta-features appeared for the first time within the framework of the European StatLog project [16], whose purpose was the comparison of learning algorithms. The European METAL project [7] extended StatLog to cover more learning algorithms and more datasets, and investigated a number of other meta-features. Both projects sought to map meta-features to either a best performing algorithm or to a ranking of algorithms [4]. Another attempt to characterize meta-data through an analysis of meta-features is reported in [5]. In our case, we map datasets features to predictions on the benefits of using one learning algorithm in terms of different metrics, as execution time, accuracy or model understandability.

And third, from a planning perspective, we learn the values of some fluents defined in the domain by combining planning and execution. This is similar to the work reported in [14] where they automatically model the duration of actions execution as relational regression trees learned from observing plan executions. Those models are incorporated into the planning domain actions in the form of probabilistic conditional effects. In our case, we do not model probabilistic knowledge, but we learn multiple concepts in parallel. Also, they used relational models, while we learn propositional ones. The research reported in [13] explores the area of learning costs of actions from execution traces as well. They used predictive features of the environment to create situation-dependent costs for the arcs in the topological map used by a path planner to create routes for a robot. These costs are represented as learned regression trees.

7 Conclusions

This paper presents current work on the use of machine learning techniques to improve the KDD process of the PDM architecture. PDM uses automated planning techniques to help the data mining process. Previous work on PDM applied a learning technique to model the effect of selecting specific DM classification/regression actions to analyze a given dataset. However, those models were obtained from values resulting of applying KDD plans that incorporated not only the specific classification or regression techniques, but also a sequence of pre-processing techniques. In our current work, we are separating the execution of pre-processing actions from classification/regression actions, so that we can learn better predictive models of execution of KDD actions. We are also improving the features used for learning those models. In the future, we would like to provide the user with a mixed-initiative tool, so that the user can guide the KDD steps towards preferred techniques, or selecting or pruning KDD actions.

7.1 Acknowledgements

This work has been partially supported by the Spanish MICINN under projects TIN2008-06701-C03-03, TRA-2009-008, the regional projects CCG08-UC3M/TIC-4141 and the Automated User Knowledge Building (AUKB) project funded by Ericsson Research Spain.

References

- [1] Robert S. Amant and Paul R. Cohen, 'Evaluation of a semi-autonomous assistant for exploratory data analysis', in *Proc. of the First Intl. Conf. on Autonomous Agents*, pp. 355–362. ACM Press, (1997).
- [2] Abraham Bernstein, Foster Provost, and Shawndra Hill, 'Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification', *IEEE Transactions on Knowledge and Data Engineering*, **17**(4), (2005).
- [3] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and R. Vilalta, *Metalearning: Applications to Data Mining*, Cognitive Technologies, Springer, January 2009.
- [4] Pavel Brazdil, Carlos Soares, and Joaquim P. Costa, 'Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (March 2003). ISI, DBLP.
- [5] Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli, 'Meta-data: Characterization of input features for meta-learning', in *MDAI*, pp. 457–468, (2005).
- [6] Tomás De la Rosa, Angel García-Olaya, and Daniel Borrajo, 'Using cases utility for heuristic planning improvement', in *Case-Based Reasoning Research and Development: Proceedings of the 7th International Conference on Case-Based Reasoning*, pp. 137–148, Belfast, Northern Ireland, UK, (August 2007). Springer Verlag.
- [7] Metal: A meta-learning assistant for providing user support in machine learning and data mining, 1998-2001.
- [8] Susana Fernández, Fernando Fernández, Alexis Sánchez, Tomás de la Rosa, Javier Ortiz, Daniel Borrajo, and David Manzano, 'On compiling data mining tasks to pddl', in *Proceedings of International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS'09*, Thessaloniki (Greece), (September 2009).
- [9] M. Fox and D. Long, 'PDDL2.1: An extension to PDDL for expressing temporal planning domains', *Journal of Artificial Intelligence Research*, 61–124, (2003).
- [10] Mark Hall Geoffrey Holmes and Eibe Frank, 'Generating rule sets from model trees', *Advanced Topics in Artificial Intelligence*, **1747**, 1–12, (1999).
- [11] Malik Ghallab, Dana Nau, and Paolo Traverso, *Automated Planning - Theory and Practice*, Morgan Kaufmann, San Francisco, CA 94111, 2004.
- [12] Alex Guazzelli, Michael Zeller, Wen-Ching Lin, and Graham Williams, 'PMML: An Open Standard for Sharing Models', *The R Journal*, **1**(1), 60–65, (May 2009).
- [13] Karen Zita Haigh and Manuela M. Veloso, 'Learning situation-dependent costs: Improving planning from probabilistic robot execution', in *In Proceedings of the Second International Conference on Autonomous Agents*, pp. 231–238. AAAI Press, (1998).
- [14] Jesús Lanchas, Sergio Jiménez, Fernando Fernández, and Daniel Borrajo, 'Learning action durations from executions', in *Proceedings of the ICAPS'07 Workshop on Planning and Learning*, Providence, Rhode Island (USA), (2007).
- [15] Jun Won Lee and Christophe G. Giraud-Carrier, 'New insights into learning algorithms and datasets', in *ICMLA*, pp. 135–140, (2008).
- [16] *Machine Learning, Neural and Statistical Classification*, eds., Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, Ellis Horwood, Upper Saddle River, NJ, USA, 1994.
- [17] Katharina Morik and Martin Scholz, 'The miningmart approach to knowledge discovery in databases', in *In Ning Zhong and Jiming Liu, editors, Intelligent Technologies for Information Analysis*, pp. 47–65. Springer, (2003).
- [18] Vid Podpecán, Nada Lavrač, Joost N. Kok, and Jeroen de Bruin, eds. *SoKD-09, "Third Generation Data Mining: Towards Service-oriented Knowledge Discovery"*, *International Workshop on Third Generation Data Mining at ECML PKDD 2009*, Bled, Slovenia, September 2009.
- [19] Saharon Rosset, Claudia Perlich, and Bianca Zadrozny, 'Ranking-based evaluation of regression models', *Knowledge and Information Systems*, **12**(3), 331–353, (2007).
- [20] R. Vilalta, Christophe Giraud-Carrier, Pavel Brazdil, and Carlos Soares, 'Using meta-learning to support data mining', *International Journal of Computer Science and Applications*, **1**(1), 31–45, (2004).
- [21] Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, 2005.