Meta-learning Architectures: Collecting, Organizing and Exploiting Meta-knowledge

Joaquin Vanschoren

Department of Computer Science, K.U.Leuven, Leuven, Belgium

1 Introduction

While a valid intellectual challenge in its own right, meta-learning finds its real *raison d'être* in the practical support it offers Data Mining practitioners [20]. Indeed, the whole point of understanding how to learn in any given situation is to go out in the real world and learn as much as possible, from any source of data we encounter! However, almost any type of raw data will initially be very hard to learn from, and about 80% of the effort in discovering useful patterns lies in the clever preprocessing of data [47]. Thus, for machine learning to become a tool we can instantly apply in any given situation, or at least to get proper guidance when applying it, we need to build extended meta-learning systems that encompass the entire knowledge discovery process, from raw data to finished models, and that keep learning, keep accumulating meta-knowledge, every time they are presented with new problems.

The algorithm selection problem is thus widened into a *workflow creation* problem, in which an entire stream of different processes needs to be proposed to the end user. This entails that our collection of meta-knowledge must also be extended to characterize all those different processes.

In this chapter, we provide a survey of the various architectures that have been developed, or simply proposed, to build such extended meta-learning systems. They all consist of integrated repositories of meta-knowledge on the KD process and leverage that information to propose useful workflows. Our main observation is that most of these systems are very different, and were seemingly developed independently from each other, without really capitalizing on the benefits of prior systems. By bringing these different architectures together and highlighting their strengths and weaknesses, we aim to reuse what we have learned, and we draw a roadmap towards a new generation of KD support systems.

Despite their differences, we can classify the KD systems in this chapter in the following groups, based on the way they leverage the obtained meta-data:

Expert systems In expert systems, experts are asked to express their reasoning when tackling a certain problem. This knowledge is then converted into a set of explicit rules, to be automatically triggered to guide future problems.

Meta-models The goal here is to automatically predict the usefulness of workflows based on prior experience. They contain a meta-model that is updated as more experience becomes available. It is the logical extension of meta-learning to the KD process.

Case-based reasoning In general terms, case-based reasoning (CBR) is the process of solving new problems based on the solutions of similar past problems. This is very similar to the way most humans with some KD experience would tackle the problem: remember similar prior cases and adapt what you did then to the new situation. To mimic this approach, a knowledge base is populated by previously successful workflows, annotated with meta-data. When a new problem presents itself, the system will retrieve the most similar cases, which can then be altered by the user to better fit her needs.

Planning All possible actions in a KD workflow are described as operations with preconditions and effects, and an AI planner is used to find the most interesting plans (workflows).

Querying In this case, meta-data is gathered and organized in such a way that users can ask any kind of question about the utility or general behavior of KD processes in a predefined query language, which will then be answered by the system based on the available meta-data. They open up the available meta-data to help users make informed decisions.

Orthogonal to this distinction, we can also characterize the various systems by the *type of meta-knowledge* they store, although in some cases, a combination of these sources is employed.

Expert knowledge Rules, models, heuristics or entire KD workflows are entered beforehand by experts, based on their own experience with certain learning approaches.

Experiments Here, the meta-data is purely empirical. They provide objective assessments of the performance of workflows or individual processes on certain problems.

Workflows Workflows are descriptions of the entire KD process, involving linear or graph-like sequences of all the employed preprocessing, transformation, modeling and postprocessing steps. They are often annotated with simple properties or qualitative assessments by users.

Ontologies An *ontology* is a formal representation of a set of concepts within a domain and the relationships between those concepts [9]. They provide a fixed vocabulary of data mining concepts, such as algorithms and their components, and describe how they relate to each other, e.g. what the role is of a specific component in an algorithm. They can be used to create unambiguous descriptions of meta-knowledge which can then be interpreted by many different systems to reason about the stored information.

While we cover a wide range of approaches, the landscape of all meta-learning and KD solutions is quite extensive. However, most of these simply facilitate access to KD techniques, sometimes offering wizard-like interfaces with some expert advice, but they essentially leave the intelligent selection of techniques as an exercise to the user. Here, we focus on those systems that introduce new ways of leveraging meta-knowledge to offer intelligent KD support. We also skip

KD systems that feature some type of knowledge base to monitor the current workflow composition, but that do not use that knowledge to advise on future problems, such as the INLEN system [44, 33, 34]. This overview partially overlaps with previous, more concise overviews of meta-learning systems for KD [7, 20]. Here, however, we provide a more in-depth discussion of their architectures, focussing on those aspects that can be reused to design future KD support systems.

In the remainder of this chapter, we will split our discussion in two: past and future. The past consists of all previously proposed solutions, even though some of these are still in active development and may be extended further. This will cover the following five sections, each corresponding to one of the five approaches outlined above. For each system, we consecutively discuss its architecture, the employed meta-knowledge, any meta-learning that is involved and finally its benefits and drawbacks. Finally, in Section 7, we provide a short summary of all approaches, before looking towards the future: we outline a platform for the development of future KD support systems aimed at bringing the best aspects of prior systems together.

2 Expert systems

2.1 Consultant-2

One of the first inroads into systematically gathering and using meta-knowledge about machine learning algorithms was Consultant-2 [14, 57]: an expert system developed to provide end-user guidance for the MLT machine learning toolbox [46, 37]. Although the system did not learn by itself from previous algorithm runs, it did identify a number of important meta-features of the data and the produced models, and used these to express rules about the applicability of algorithms.

Architecture A high-level overview of the architecture of Consultant-2 is shown in Figure 1. It was centered around a knowledge base that stored about 250 rules, hand-crafted by machine learning experts. The system interacted with the user through question-answer sessions in which the user was asked to provide information about the given data (e.g. the number of classes or whether it could be expected to be noisy) and the desired output (e.g. rules or a decision tree). Based on that information, the system then used the stored rules to calculate a score for each algorithm. The user could also go back on previous answers to see how that would have influenced the ranking. When the user selected an algorithm, the system would automatically run it, after which it would engage in a new question-answer session to assess whether the user was satisfied with the results. If not, the system would generate a list with possible parameter recommendations, again scored according to the stored heuristic rules.

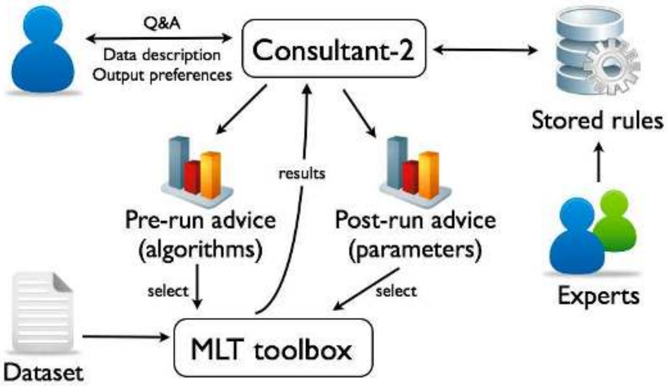


Fig. 1. The architecture of Consultant-2. Derived from Craw et al. [14]

Meta-knowledge The rules were always of the form $if(A_n)$ *then* B, in which A_n was a conjunction of properties of either the *task description* or the *produced models*, and B expressed some kind of action the system could perform. First, the task description included qualitative measures entered by the user, such as the task type, any available background knowledge and which output models were preferable, as well as quantitative measures such as the number of classes and the amount of noise. Second, the produced models where characterized by the amount of time and memory needed to build them and model-specific features such as the average path length and number of leaf nodes in decision trees, the number and average center distance of clusters and the number and significance of rules.[1] The resulting actions B could either adjust the scores for specific algorithms, propose ranges for certain parameter values, or transform the data (e.g. discretization) so as to suit the selected algorithm. Illustrations of these rules can be found in Sleeman at al. [57].

Meta-learning There is no real meta-learning process in Consultant-2, it just applied its predefined 'model' of the KD process. This process was divided into a number of smaller steps (selecting an algorithm, transforming the data, selecting parameters,...), each associated with a number of rules. It then cycles through that process, asking questions, executing the corresponding rules, and triggering the corresponding actions, until the user is satisfied with the result.

Discussion The expert advice in Consultant-2 has proven to be successful in a number of applications, and a new version has been proposed to also provide guidance for data preprocessing. Still, to the best of our knowledge, Consultant-3 has never been implemented. An obvious drawback of this approach is the fact that the heuristic rules are hand-crafted. This means that for every new

[1] These were used mostly to advise on the perspective algorithm's parameters.

algorithm, new rules have to be defined that differentiate it from all the existing algorithms. One must also keep in mind that MLT only had a limited set of 10 algorithms covering a wide range of tasks (e.g. classification, regression and clustering), making it quite feasible to select the correct algorithm based on the syntactic properties of the input data and the preferred output model. With the tens or hundreds of methods available today on classification alone, it might not be so straightforward to make a similar differentiation. Still, the idea of a database of meta-rules is valuable, though instead of manually defining them, they should be learned and refined automatically based on past algorithm runs. Such a system would also automatically adapt as new algorithms are added.

3 Meta-models

3.1 STABB and VBMS

The groundwork for many meta-learning systems were laid by two early precursors. STABB [61] was a system that basically tried to automatically match a learner's bias to the given data. It used an algorithm called LEX with a specific grammar. When an algorithm could not completely match the hidden concept, its bias (the grammar) or the structure of the dataset was changed until it could. VBMS [52] was a very simple meta-learning system that tried to select the best of three learning algorithms using only two meta-features: the number of training instances and the number of features.

3.2 The Data Mining Advisor (DMA)

The Data Mining Advisor (DMA) [19] is a web-based algorithm recommendation system that automatically generates rankings of classification algorithms according to user-specified objectives. It was developed as part of the METAL project [43].

Foundations: StatLog Much of the theoretical underpinning of the DMA was provided by the StatLog project [45], which was aimed to provide a large-scale, objective assessment of the strengths and weaknesses of the various classification approaches existing at the time. Its methodology is shown in Figure 2.

First, a wide range of datasets are characterized with a wide range of newly defined meta-features. Next, the algorithms are evaluated on these datasets. For each dataset, all algorithms whose error rate fell within a certain (algorithm-dependent) margin of that of the best algorithms were labeled *applicable*, while the others were labeled *non-applicable*. Finally, decision trees were built for each algorithm, predicting when it can be expected to be useful on a new dataset. The resulting rules, forming a rule base for algorithm selection, can be found in Michie et al. [45].

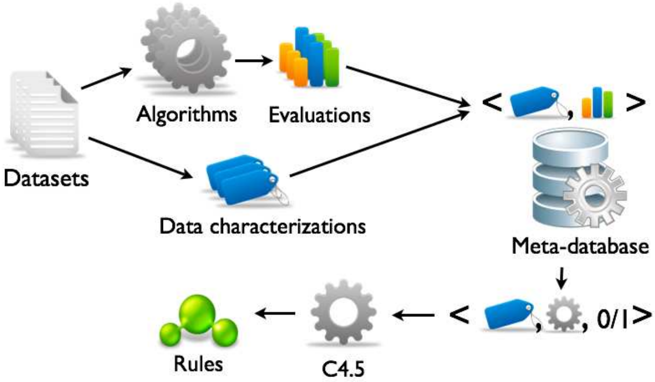


Fig. 2. The StatLog approach.

Architecture The architecture of the DMA is shown in Figure 3. First, the tool is trained by generating the necessary meta-data, just as in StatLog. However, instead of predicting whether an algorithm was applicable or not, it *ranked* all of them. New datasets can be uploaded via a web-interface, its meta-features automatically computed (below the dotted line in Figure 3). Bearing privacy issues in mind, the user can choose to keep the dataset, the data characterizations or both hidden from other users of the system. Next, the DMA will use the stored meta-data to predict the expected ranking of all algorithms on the new dataset. Finally, as a convenience to the user, one can also run a number of algorithms on the new dataset, after which the system returns their true ranking and performance. Figure 4 shows an example of the rankings generated by DMA. The meaning of the 'Predicted Score' is explained in Section 3.2.

Meta-knowledge While the METAL project discovered various new kinds of meta-features, these were not used in the DMA tool. In fact, only a set of 7 numerical StatLog-style characteristics was selected[2] for predicting the relative performance of the algorithms. This is most likely due to the employed meta-learner, which is very sensitive to the curse of dimensionality.

At its inception, the DMA tool was initialized with 67 datasets, mostly from the UCI repository. Since then, an additional 83 tasks were uploaded by users [19]. Furthermore, it operates on a set of 10 classification algorithms, shown in Figure 4, but only with default parameter settings, and evaluates them based on their predictive accuracy and speed.

Meta-learning DMA uses a k-nearest neighbors (kNN) approach as its meta-learner, with k=3 [8]. This is motivated by the desire to continuously add new meta-data without having to rebuild the meta-model every time.

[2] Some of them were slightly modified. For instance, instead of using the absolute number of symbolic attributes, DMA uses the *ratio* of symbolic attributes.

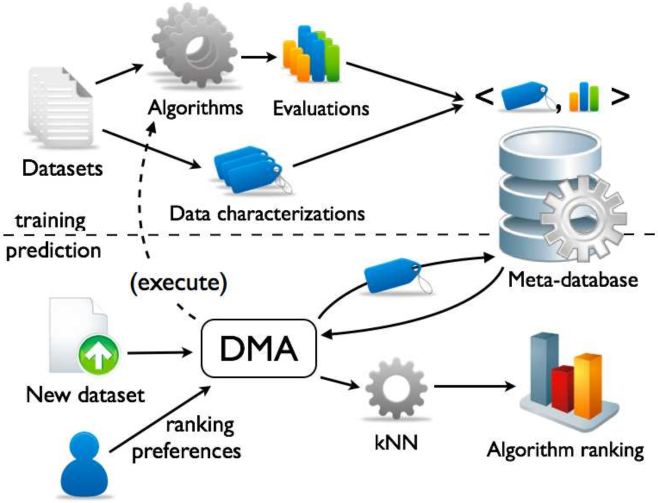
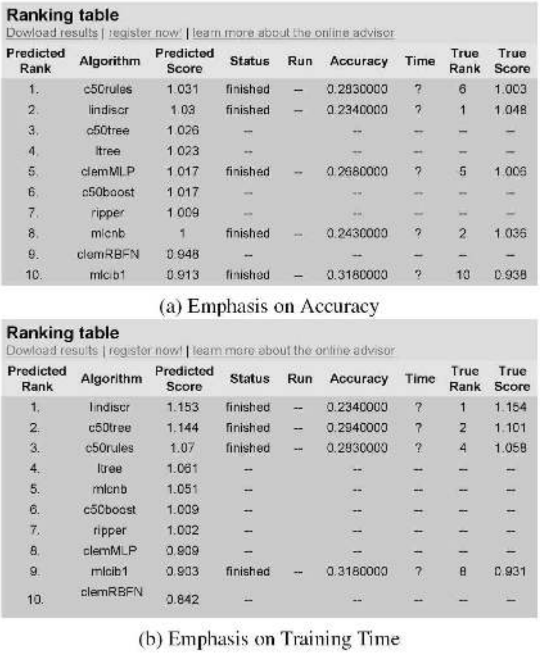


Fig. 3. The architecture of DMA. Adapted from Brazdil at al. [7].



Ranking table

Dowload results | register now! | learn more about the online advisor

Predicted Rank	Algorithm	Predicted Score	Status	Run	Accuracy	Time	True Rank	True Score
1.	c50rules	1.031	finished	–	0.2830000	?	6	1.003
2.	lindiscr	1.03	finished	–	0.2340000	?	1	1.048
3.	c50tree	1.026	–		–	–	–	–
4.	ltree	1.023	–		–	–	–	–
5.	clemMLP	1.017	finished	–	0.2680000	?	5	1.006
6.	c50boost	1.017	–		–	–	–	–
7.	ripper	1.009	–		–	–	–	–
8.	mlcnb	1	finished	–	0.2430000	?	2	1.036
9.	clemRBFN	0.948	–		–	–	–	–
10.	mlcib1	0.913	finished	–	0.3180000	?	10	0.938

(a) Emphasis on Accuracy

Ranking table

Dowload results | register now! | learn more about the online advisor

Predicted Rank	Algorithm	Predicted Score	Status	Run	Accuracy	Time	True Rank	True Score
1.	lindiscr	1.153	finished	–	0.2340000	?	1	1.154
2.	c50tree	1.144	finished	–	0.2940000	?	2	1.101
3.	c50rules	1.07	finished	–	0.2930000	?	4	1.058
4.	ltree	1.061	–		–	–	–	–
5.	mlcnb	1.051	–		–	–	–	–
6.	c50boost	1.009	–		–	–	–	–
7.	ripper	1.002	–		–	–	–	–
8.	clemMLP	0.909	–		–	–	–	–
9.	mlcib1	0.903	finished	–	0.3180000	?	8	0.931
10.	clemRBFN	0.842	–		–	–	–	–

(b) Emphasis on Training Time

Fig. 4. An illustration of the output of the DMA. The first ranking favors accurate algorithms, the second ranking favors fast ones. Taken from Giraud-Carrier [19].

A multi-criteria evaluation measure is defined (the *adjusted ratio of ratios*) which allows the user to add emphasis either on the predictive accuracy or the training time. This is done by defining the weight *AccD* of the training time ratio. This allows the user to trade *AccD*% accuracy for a 10-time increase/decrease in training time. The DMA interface offers 3 options: *AccD*=0.1 (focus on accuracy), *AccD*=10 (focus on speed) or *AccD*=1 (equal importance). The user can also opt not to use this measure and use the *efficiency measure* of Data Envelopment Analysis (DEA) instead [19]. The system then calculates a kind of average over the 3 most similar datasets for each algorithm (the 'predicted score' in Figure 4) according to which the ranking is established.

Discussion The DMA approach brings the benefits of meta-learning to a larger audience by automating most of the underlying steps and allowing the user to state some preferences. Moreover, it is able to continuously improve its predictions as it is given more problems (which are used to generate more meta-data). Unfortunately, it has a number of limitations that affect the practical usefulness of the returned rankings. First, it offers no advice about which preprocessing steps to perform or which parameter values might be useful, which both have a profound impact on the relative performance of learning algorithms. Furthermore, while the multi-criteria ranking is very useful, the system only records two basic evaluation metrics, which might be insufficient for some users. Finally, using kNN means no interpretable models are being built, making it a purely predictive system. Several alternatives to kNN have been proposed [50, 3] which may be useful in future incarnations of this type of system.

3.3 NOEMON

NOEMON [31, 32], shown in Figure 5, follows the approach of Aha [1] to compare algorithms two by two. Starting from a meta-database similar to the one used in DMA, but with histogram-representations of attribute-dependent features, the performance results on every combination of two algorithms are extracted. Next, the performance values are replaced with a statistical significance tests indicating when one algorithm significantly outperforms the other and the number of data meta-features is reduced using automatic feature selection. This data is then fed to a decision tree learner to build a model predicting when one algorithm will be superior or when they will tie on a new dataset. Finally, all such pairwise models are stored in a knowledge base.

At prediction-time, the system collects all models concerning a certain algorithm and counts the number of predicted wins/ties/losses against all other algorithms to produce the final score for each algorithm, which is then converted into a ranking.

4 Planning

The former two systems are still limited in that they only tackle the algorithm selection step. To generate advice on the composition of an entire workflow, we

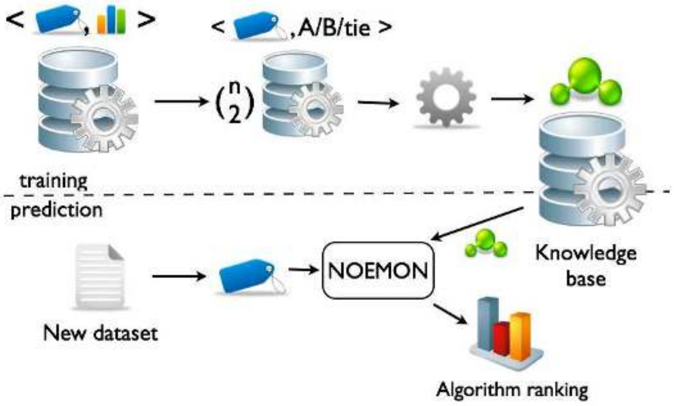


Fig. 5. NOEMON's architecture. Based on Kalousis and Theoharis [32].

need additional meta-data on the rest of the KD processes. One straightforward type of useful meta-data consists of the preconditions that need to be fulfilled before the process can be used, and the effect it has on the data it is given. As such, we can transform the workflow creation problem into a *planning* problem, aiming to find the best plan, the best sequence of actions (process applications), that arrives at our goal - a final model.

4.1 The Intelligent Discovery Electronic Assistant (IDEA)

A first such system is the Intelligent Discovery Electronic Assistant (IDEA) [5]. It regards preprocessing, modeling and postprocessing techniques as operators, and returns all plans (sequences of operations) that are possible for the given problem. It contains an ontology describing the preconditions and effects of each operator, as well as manually defined heuristics (e.g. the speed of an algorithm), which allows it to produce a ranking of all generated plans according to the user's objectives.

Architecture The architecture of IDEA is shown in Figure 6. First, the systems gathers information about the given task by characterizing the given dataset. Furthermore, the user is asked to provide additional metadata and to give weights to a number of heuristic functions such as model comprehensibility, accuracy and speed. Next, the planning component will use the operators that populate the ontology to generate all KD workflows that are valid in the user-described setting. These plans are then passed to a heuristic ranker, which will use the heuristics enlisted in the ontology to calculate a score congruent with the user's objectives (e.g. building a decision tree as fast as possible). Finally, this ranking is proposed to the user which may select a number of processes to be executed on the provided data. After the execution of a plan, the user is allowed to review the results and alter the given weights to obtain new rankings.

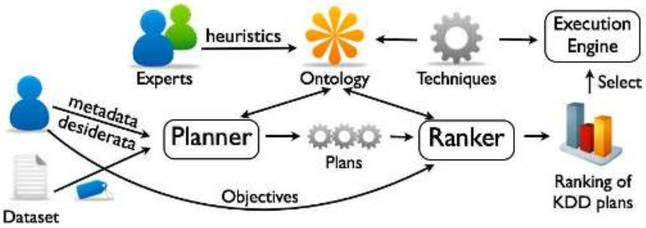


Fig. 6. The architecture of IDEA. Derived from Bernstein et al. [5].

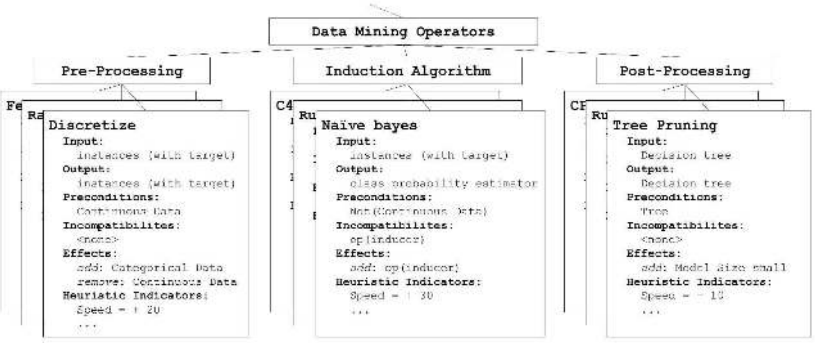


Fig. 7. Part of IDEA's ontology. Taken from Bernstein et al. [5].

For instance, the user might sacrifice speed in order to obtain a more accurate model. Finally, if useful partial workflows have been discovered, the system also allows to extend the ontology by adding them as new operators.

Meta-knowledge IDEA's meta-knowledge is all contained in its ontology. It first divides all operators into preprocessing, induction or postprocessing operators, and then further into subgroups. For instance, induction algorithms are subdivided into classifiers, class probability estimators and regressors. Each process that populates the ontology is then described further with a list of properties, shown in Figure 7. It includes the required input (e.g. a decision tree for a tree pruner), output (e.g. a model), preconditions (e.g. 'continuous data' for a discretizer), effects (e.g. 'removes continuous data', 'adds categorical data' for a discretizer), incompatibilities (e.g. not(continuous data) for naïve Bayes) and a list of manually defined heuristic estimators (e.g. relative speed). Additionally, the ontology also contains recurring partial plans in the form of composite operators.

Meta-learning Though there is no actual meta-learning involved, planning can be viewed as a search for the best-fitting plan given the dataset, just as learning is a search for the best hypothesis given the data. In the case of IDEA, this is achieved by exhaustively generating all possible KD plans, hoping to discover better, novel workflows that experts never considered before. The provided metadata constitute the initial state (e.g. 'numerical data') and the user's desiderata (e.g. 'model size small') make up the goal state.

Discussion This approach is very useful in that it can provide both experts and less-skilled users with KD solutions without having to know the details of the intermediate steps. The fact that new operators or partial workflows can be added to the ontology can also give rise to *network externalities*, where the work or expertise of one researcher can be used by others without having to know all the details. The ontology thus acts as a central repository of KD operators, although new operators can only be added manually, possibly requiring reimplementation.

The limitations of this approach lie first of all in the hand-crafted heuristics of the operators. Still, this could be remedied by linking it to a meta-database such as used in DMA and learning the heuristics from experimental data. Secondly, the current implementation only covers a small selection of KD operators. Together with the user-defined objectives, this might constrain the search space well enough to make the exhaustive search feasible, but it is unclear whether this would still be the case if large numbers of operators are introduced. A final remark is that most of the used techniques have parameters, whose effects are not included in the planning.

4.2 Global Learning Scheme (GLS)

The 'Global Learning Scheme' (GLS) [73, 71, 72] takes a multi-agent system approach to KDD planning. It creates an "organized society of KDD agents", in which each agent only does a small KDD task, controlled by a central planning component.

Architecture Figure 8 provides an overview of the system. It consists of a pool of agents, which are described very similarly to the operators in IDEA, also using an ontology to describe them formally. However, next to base-level agents, which basically envelop one DM technique each, there also exist *high-level agents*, one for each phase of the KD process, which instead point to a list of candidate agents that could be employed in that phase. The controlling 'meta-agent' (CMA) is the central controller of the system. It selects a number of agents (high-level agents at first) and sends them to the planning meta-agent (PMA). The PMA then creates a planning problem by transforming the dataset properties and user objectives into a world state description (WSD) for planning and by transforming the agents into planning operators. It then passes the problem to a STRIPS planner [17]. The returned plans are passed back to the CMA, which then launches new planning problems for each high-level agent. For instance, say the data has missing values, then the returned plan will contain a high-level missing value imputation agent: the CMA will then send a range of base-level agents to the

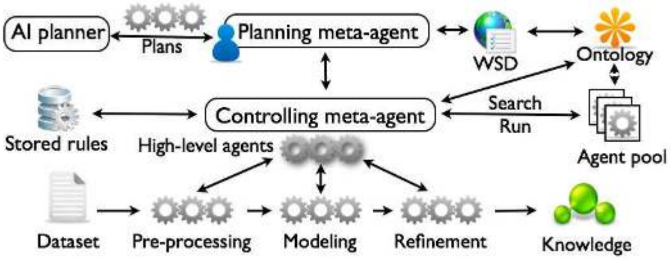


Fig. 8. The architecture of GLS. Adapted from Zhong et al. [72].

PMA to build a plan for filling in the missing values. The CMA also executes the resulting sub-plans, and calls for adaptations if they do not prove satisfactory.

Meta-knowledge GLS's meta-knowledge is similar to that of IDEA. The ontology description contains, for each agent, the data types of in- and output, preconditions and effects. In the case of a base-level agent, it also contains a KD process, otherwise, a list of candidate sub-agents. However, the same ontology also contains descriptors for the data throughout the KD process, such as the state of the data (e.g. raw, clean, selected), whether or not it represents a model and the type of that model (e.g. regression, clustering, rule). This information is used to describe the world state description. Finally, the CMA contains some static meta-rules to guide the selection of candidate agents.

Meta-learning In Zong et al. [72], the authors state that a meta-learning algorithm is used in the CMA to choose between several discretization agents or to combine their results. Unfortunately, details are missing. Still, even if the current system does not learn from previous runs (meaning that the CMA uses the same meta-rules every time), GLA's ability to track and adapt to changes performed by the user can definitely be regarded as a form of learning [7].

Discussion Given the fact that covering the entire KD process may give rise to sequences of tens, maybe hundreds of individual steps, the 'divide and conquer' approach of GLS seems a promising approach. In fact, it seems to mirror the way humans approach the KD process: identify a hierarchy of smaller subproblems, combine operators to solve them and adapt the partial solutions to each other until we converge to a working workflow. Unfortunately, to the best of our knowledge, there are no thorough evaluations of the system, and it remains a work in progress. It would be interesting to replace the fixed rules of the CMA with a meta-learning component to select the most promising agents, and to use IDEA-like heuristics to rank possible plans.

4.3 The Goal-Driven Learner (GDL)

The Goal-Driven Learner [3] [62], is a quite different planning approach, mainly because it operates on a higher level of abstraction. Its task is to satisfy a predefined *goal*, given a number of knowledge bases, a group of human experts, and a collection of KD tools. Each of these (including the experts) are described in a Learning Modeling Language (LML) which describes properties such as their *vocabulary* (e.g. the attribute names in a decision tree or the expert's field of expertise), *solutions* (e.g. classes predicted by the tree or medical treatments) or the *time* it takes them to provide a solution (which is obviously higher for an expert than for a decision tree). Its goal and subgoals are also described in LML, e.g. expressing that a simple model must be built within a given timeframe. The GDL then executes a search, using the 'distance' to each of the subgoals as a heuristic and applications of the KD tools as actions. During this process, it selects useful knowledge systems (e.g. based on their vocabulary), combines them (e.g. after conversion to Prolog clauses), applies KD tools (e.g. to build a new decision tree) or petitions an expert to provide extra information (e.g. if part of the vocabulary cannot be found in the available knowledge systems). It is a very useful approach in settings were many disparate knowledge sources are available, although the LML description is probably still too coarse to produce fine-grained KDD plans.

4.4 Complexity Controlled Exploration (CCE)

Complexity Controlled Exploration [22] is also a high-level system that consists of a number of *machine generators*, which generate KD workflows, and a search algorithm based on a measure for the *complexity* of the proposed KD workflows. While not performing planning per se, many of the generators could be planners such as IDEA and GLS. Alternatively, they may simply consist of a list of prior solutions, or *meta-schemes*: partial workflows in which certain operators are left blank to be filled in with a suitable operator later on (also see 'templates' in Section 6.1).

The algorithm asks all generators to propose new KD processes, and to rank them using an adapted complexity measure [39, 40] defined as $c(p) = [l(p) + log(t(p))] \cdot q(p)$, in which p is a *program* (a KD workflow), $l(p)$ the length of that program, $t(p)$ the estimated runtime of the program, and $q(p)$ the inverse of the *reliability* of the program, which reflects their usefulness in prior tasks. For instance, if a certain combination of a feature selection method and a classifier proved very useful in the past, it may obtain an improved reliability value. If the complexity cannot be estimated, it is approximated by taking the weighted average of past observations of the program on prior inputs. Note that a 'long'

[3] Goal-driven learning is also a more general AI concept, in which the aim is to use the overall goals of an intelligent system to make decisions about when learning should occur, what should be learned, and which learning strategies are appropriate in a given context.

program can still be less complex than a ‘short’ one, e.g. if a feature selection step heavily speeds up the runtime of a learning algorithm.

The CCE algorithm then takes the least complex solution from each generator, adds the least complex of those to a queue (until it is full), and updates the *complexity threshold*, the maximum of all proposed workflows. The workflows in the queue are then executed in parallel, but are aborted if they exceed the complexity threshold and are added, with an increased complexity value, to the ‘quarantine generator’, which may propose them again later on. The algorithm’s stopping criterion is set by the user: it may be the best solution in a given time, below a given complexity threshold, or any custom test criterion.

The CCE turns KD workflow selection into an adaptive process in which a number of other systems can be used as generators, and in which additional metadata can be added both in the generators (e.g. new meta-schemes) as well as in the complexity estimation procedure. Instead of proposing workflows and leaving it to the user to decide which ones to try next, it also runs them and returns the optimal solution. Its success is thus highly dependent on the availability of good generators, and on how well its complexity controlled exploration matches the choices that the user would make.

5 Case-based reasoning

Planning is especially useful when starting from scratch. However, if successful workflows were designed for very similar problems, we could simply reuse them.

5.1 CITRUS

CITRUS [16, 68] is built as an advisory component of Clementine, a well-known KDD suite. An overview is shown in Figure 9. It contains a knowledge base of available ‘processes’ (KD techniques) and ‘streams’ (sequences of processes), entered by experts and described with pre- and postconditions. To use it, the user has to provide an abstract task description, which is appended with simple data statistics, and choose between several modi of operation. In the first option, the user simply composes the entire KDD process (stream) by linking processes in Clementine’s editor, in which case CITRUS only checks the validity of the sequence. In the second option, case-based reasoning is used to propose the most similar of all known streams. In the third option, CITRUS assists the user in decomposing the task into smaller subtasks, down to the level of individual processes. While pre- and postconditions are used in this process, no planning is involved. Finally, the system also offers some level of algorithm selection assistance by eliminating those processes that violate any of the constraints.

5.2 ALT

ALT [41] is a case-based reasoning variation on the DMA approach. Next to StatLog-type meta-features, it adds a number of simple algorithm characteriza-

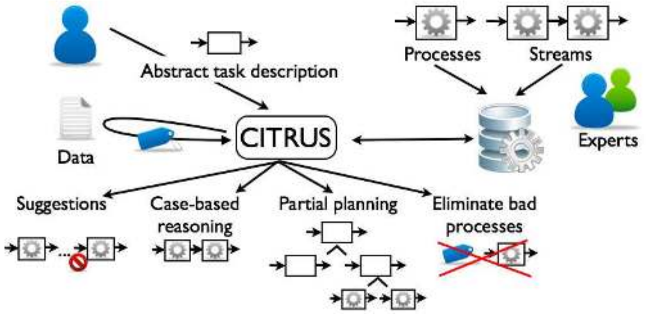


Fig. 9. The architecture of CITRUS. Derived from Wirth et al. [68].

tions. It has a meta-database of 'cases' consisting of data meta-features, algorithm meta-features and the performance of the algorithm. When faced with a new problem, the user can enter application restrictions beforehand (e.g. 'the algorithm should be fast and produce interpretable models'), and this information is then appended to the data meta-features of the new dataset, thus forming a new case. The system then makes a prediction based on the three most similar cases. The meta-data consists of 21 algorithms, 80 datasets, 16 StatLog data characteristics, and 4 algorithm characteristics.

The two previous approaches share a common shortcoming: the user still faces the difficult task of *adapting* it to her specific problem. The following systems try to alleviate this problem by offering additional guidance.

5.3 MiningMart

The MiningMart project [47] is designed to allow successful preprocessing workflows to be shared and reused, irrespective of how the data is stored. While most of the previously discussed systems expect a dataset in a certain format, MiningMart works directly on any SQL database of any size. The system performs the data preprocessing and transformation steps in the (local) database itself, either by firing SQL queries or by running a fixed set of efficient operators (able to run on very large databases) and storing the results in the database again. Successful workflows can be shared by describing them in an XML-based language, dubbed M4, and storing them in an online case-base. The case description includes the workflow's inherent structure as well as an ontological description of business concepts to facilitate searching for cases designed for certain goals or applications.

Architecture The architecture of the system is shown in Figure 10. To map uniformly described preprocessing workflows to the way data is stored in specific databases, it offers three levels of abstraction, each provided with graphical editors for the end-user. We discuss them according to the viewpoint of the users who wish to reuse a previously entered case. First, they use the business ontology to search for cases tailored to their specific domain. The online interface

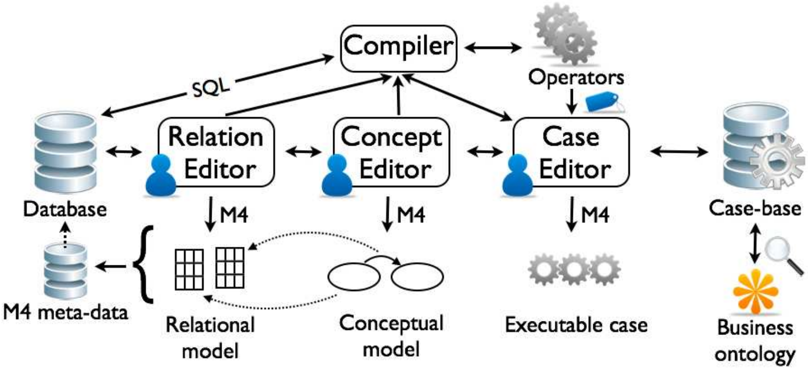


Fig. 10. MiningMart architecture. Derived from Morik and Scholz [47].

then returns a list of cases (workflows). Next, they can load a case into the *case editor* and adapt it to her specific application. This can be done on an abstract level, using abstract concepts and relations such as 'customer', 'product' and the relation 'buys'. They can each have a range of properties, such as 'name' and 'address', and can be edited in the *concept editor*. In the final phase, these concepts, relations and properties have to be mapped to tables, columns, or sets of columns in the database using the *relation editor*. All details of the entire process are expressed in the M4 language[4] and also stored in the database. Next, the *compiler* translates all preprocessing steps to SQL queries or calls to the operators used, and executes the case. The user can then adapt the case further (e.g. add a new preprocessing step or change parameter settings) to optimize its performance. When the case is finished, it can be annotated with an ontological description and uploaded to the case-base for future guidance.

Meta-knowledge MiningMart's meta-knowledge can be divided in two groups. First, there is the fine-grained, case-specific meta-data that covers all the meta-data entered by the user into the M4 description, such as database tables, concepts, and workflows. Second, there is more general meta-knowledge encoded in the business ontology, i.e. informal annotations of each case in terms of its goals and constraints, and in the description of the operators. Operators are stored in a hierarchy consisting of 17 'concept' operators (from selecting rows to adding columns with moving windows over time series), 4 feature selection operators, and 20 feature construction operators, such as filling in missing values, scaling, discretization and some learning algorithms used for preprocessing: decision trees, k-means and SVMs. The M4 schema also captures known preconditions and assertions of the operators, similar to the preconditions and effects of IDEA and GLS. Their goal is to guide the construction of valid preprocessing

[4] Examples of M4 workflows can be downloaded from the online case-base: http://mmart.cs.uni-dortmund.de/caseBase/index.html

sequences, as was done in CITRUS. The case-base is available online [47] and currently contains 6 fully-described cases.

Meta-learning While it is a CBR approach, there is no automatic recommendation of cases. The user can only browse the cases based on their properties manually.

Discussion The big benefit of the MiningMart approach is that users are not required to transform the data to a fixed, often representationally limited format, but can instead adapt workflows to the way the source data is actually being stored and manipulated. Moreover, a common language to describe KD processes would be highly useful to exchange workflows between many different KD environments. Pooling these descriptions would generate a rich collection of meta-data for meta-learning and automated KD support. M4 is certainly a step forward in the development of such a language.

Compared to IDEA, MiningMart focusses much more on the preprocessing phase, while the former has a wider scope, also covering model selection and postprocessing steps. Next, while both approaches describe their operators with pre- and postconditions, MiningMart only uses this information to guide the user, not for automatic planning. MiningMart could, in principle, be extended with an IDEA-style planning component, at least if the necessary meta-data can also be extracted straight from the database.

There are also some striking similarities between MiningMart and GLS. GLS's agents correspond to MiningMart's operators and its controller (CMA) corresponds to MiningMart's compiler. Both systems use a hierarchical description of agents/operators, which are all described with pre- and postconditions. The differences lie in the scope of operators (MiningMart focuses of preprocessing while GLS wants to cover the entire KD process), the database integration (MiningMart interfaces directly with databases while GLS requires the data to be prepared first) and in storing the meta-data of the workflows for future use, which MiningMart supports, but GLS doesn't.

5.4 The Hybrid Data Mining Assistant (HDMA)

The Hybrid Data Mining Assistant[5] (HDMA) [11–13] also tries to provide advice for the entire knowledge discovery process using ontological (expert) knowledge, as IDEA does, but it does not provide a ranking of complete processes. Instead, it provides the user with expert advice during every step of the discovery process, showing both the approach used in similar cases and more specific advice for the given problem triggered by ontological knowledge and expert rules.

[5] This is not the official name, we only use it here to facilitate our discussion.

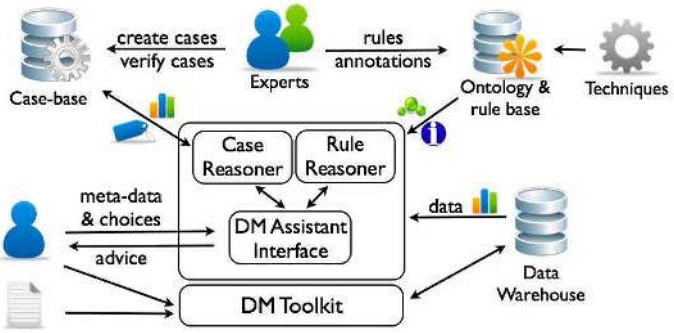


Fig. 11. The architecture of HDMA. Adapted from Charest et al. [13].

Architecture An overview of the system is provided in Figure 11. The provided advice is based on two stores of information. The first is a repository of KDD 'cases', detailing previous workflows, while the second is an ontology of concepts and techniques used in a typical KDD scenario and a number of rules concerning those techniques. Furthermore, the system is assumed to be associated with a DM toolkit for actually running and evaluating the techniques.

The user first provides a dataset, which is characterized partly by the system, partly by the user (see below). The given problem is then compared to all the stored cases and two scores are returned for each case, one based on similarity with the current case, the other based on the 'utility' of the case, based on scores provided by previous users. After the user has selected a case, the system starts cycling through five phases of the KDD process, as identified by the CRISP-DM [10] model. At each phase the user is provided with the possible techniques that can be used in that phase, the techniques that were used in the selected case, and a number of recommendations generated by applying the stored rules in the context of the current problem. The generated advice may complement, encourage, but also advice against the techniques used in the selected case. As such, the user is guided in adapting the selected case to the current problem.

Meta-knowledge In the case base, each case is described by 66 attributes. First, the given problem is described by 30 attributes, including StatLog-like meta-features, but also qualitative information entered by users, such as the type of business area (e.g. engineering, marketing, finance,...) or whether the data can be expected to contain outliers. Second, the proposed workflow is described by 31 attributes, such as the used preprocessing steps, how outliers were handled, which model was used, which evaluation method was employed and so on. Finally, 5 more attributes evaluate the outcome of the case, such as the level of satisfaction with the approach used in each step of the KD process. A number of seed cases were designed by experts, and additional cases can be added after evaluation.

The ontology, on the other hand, captures a taxonomy of DM concepts, most of which where elicited from CRISP-DM. A small part is shown in Figure 12.

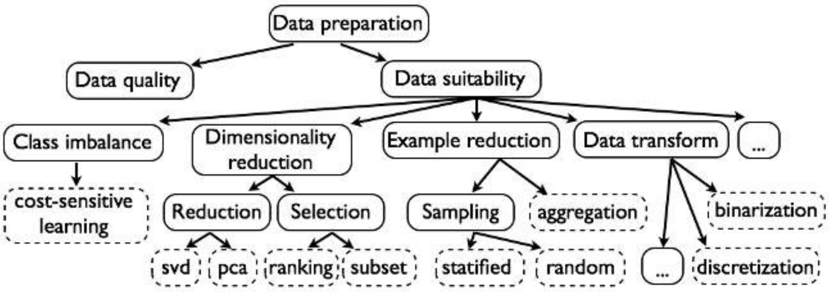


Fig. 12. Part of the HDMA ontology. Adapted from Charest et al. [13].

For instance, it shows that binarization and discretization are two data transformation functions, used to make the data more suitable to a learning algorithm, which is part of the data preparation phase of the CRISP-DM model. The dashed concepts are the ones for which exist specific KD-techniques (individuals). These individuals can be annotated with textual recommendations and heuristics, and also feature in expert rules that describe when they should be used, depending on the properties of the given problem. Some of these rules are heuristic (e.g. "use an *aggregation* technique if the *example count* is greater than 30000 and the data is of a *transactional* nature"), while others are not (e.g. "if you select the *Naive Bayes* technique, then a *nominal target* is required"). In total, the system contains 97 concepts, 58 properties, 63 individuals, 68 rules and 42 textual annotations. All knowledge is hand-crafted by experts and formalized in the OWL-DL ontology format [15] and the SWRL rule description language [28].

Meta-learning The only meta-learning occurring in this system is contained in the case based reasoning. It uses a kNN approach to select the most similar case based on the characterization of the new problem, using a feature-weighted, global similarity measure. The weight of each data characteristic is pre-set by experts.

Discussion HDMA is quite unique in that it leverages both declarative information, viz. concepts in the ontology and case descriptions, as well as procedural information in the form of rules. Because of the latter, it can be seen as a welcome extension of Consultant-2. It doesn't solve the problem of (semi-)automatically finding the right KD approach, but it provides practical advice to the user during every step of the KD process. Its biggest drawback is probably that it relies almost entirely on input from experts. Besides from the fact that the provided rules and heuristics may not be very accurate in some cases, this makes it hard to maintain the system. For every new technique (or just a variant of an existing one), new rules and concepts will have to be defined to allow the system to return proper advice. As with Consultant-2, some of these issues may be resolved by introducing more meta-learning into the system, e.g. advice on the proper weights for each data characteristic in the case based reasoning step, to update the heuristics in the rules and to find new rules based on experience. Finally, the case-based advice, while useful in the first few steps, may lose its utility in the later stages of the KD process. More specifically, say the user chooses a different preprocessing step as applied in the proposed case, then the subsequent stages of that case (e.g. model selection) may lose their applicability. A possible solution here would be to update the case selection after each step, using the partial solution to update the problem description.

5.5 NExT

NExT, the Next generation Experiment Toolbox [4] is an extension of the IDEA approach to case-based reasoning and to the area of *dynamic processes*, in which there is no guarantee that the proposed workflow will actually work, or even that the atomic tasks of which it consists will execute without fault. First, it contains a knowledge base of past workflows and uses case-based reasoning to retrieve the most similar ones. More often than not, only parts of these workflows will be useful, leaving holes which need to be filled with other operators. This is where the planning component comes in: using the preconditions and effects of all operators, and the starting conditions and goals of the KD problem, it will try to find new sequences of operators to fill in those holes.

However, much more can go wrong when reusing workflows. For instance, a procedure may have a parameter setting that was perfect for the previous setting, but completely wrong for the current one. Therefore, NExT has an ontology of possible problems related to workflow execution, including resolution strategies. Calling a planner is one such strategy, other strategies may entail removing operators, or even alerting the user to fix the problem manually.

Finally, it does not start over each time the workflow breaks. It records all the data processed up to the point where the workflow breaks, tries to resolve the issue, and then continues from that point on. As such, it also provides an online log of experimentation which can be shared with other researchers willing to reproduce the workflow.

NExT has only recently been introduced and thorough evaluations are still scarce. Nevertheless, its reuse of prior workflows and semi-automatic adaption of these workflows to new problems seems a very promising.

6 Querying

A final way to assist users is to automatically answer any kind of question they may have about the applicability, general utility or general behavior of KD processes, so that they can make informed decisions when creating or altering workflows. While the previous approaches only stored a selection of meta-data necessary for the given approach, we could collect a much larger collection of meta-data of possible interest to users, and organize all this data to allow the user to write queries in a predefined query language, which will then be answered by the system based on the available meta-data. While this approach is mostly useful for experts knowing how to ask the questions and interpret the results, frequently asked questions could be wrapped in a simpler interface for use by a wider audience.

6.1 Advanced Meta-Learning Architecture (AMLA)

One example of this approach is the Advanced Meta-Learning Architecture[6] (AMLA) [21]. It is a data mining toolbox architecture aimed at running ML algorithms more efficiently whilst inherently supporting meta-learning by collecting all meta-data from every component in the system and allowing the user to query or manipulate it.

Architecture An overview of the system is provided in Figure 13. It encapsulates all standard KD operations with modules which can then be used as building blocks for KD processes. Each module has a set of inputs and outputs, each of which is a 'model', i.e. an actual model (e.g. a decision tree) or a dataset. It also has a special 'configuration' input which supplies parameter settings, and a special 'results' output, which produces meta-data about the process. For instance, in the case the process encapsulates a decision tree learner, the input would be training data, the configuration would state its parameters, the output would provide a decision tree model and the results would state, for instance, the number of nodes in the tree. All results are represented as name-value pairs and stored in a repository, which collects all results generated by all modules. The modules are very fine grained. For instance, there are separate modules for testing a model against a dataset (which export the performance evaluations to the repository) and for sub-components of certain techniques, such as base-learners in ensembles, kernels in SVMs and so on. Modules often used together can be combined in 'schemes', which again have their own inputs, outputs, configurations and results. Schemes can also contain unspecified modules, to be filled in when used, in which case they are called templates. There also exist 'repeater' modules which repeat certain schemes many times, for instance for cross-validation. This compositionality allows to build arbitrarily complex KD workflows, quicker implementation of variants of existing algorithms, and a more efficient execution, as modules used many times only have to be loaded once. The result repository can be queried by writing short scripts to extract certain values, or to combine or filter the results of previous queries. Finally, 'commentators' can be written to perform frequently used queries, e.g. statistical significance tests, and store their results in the repository.

Meta-knowledge The stored meta-data consists of a large variety of name-value pairs collected from (and linked to) all previously used modules, templates and schemes. They can be of any type.

[6] This is again not the officially coined name of the system.

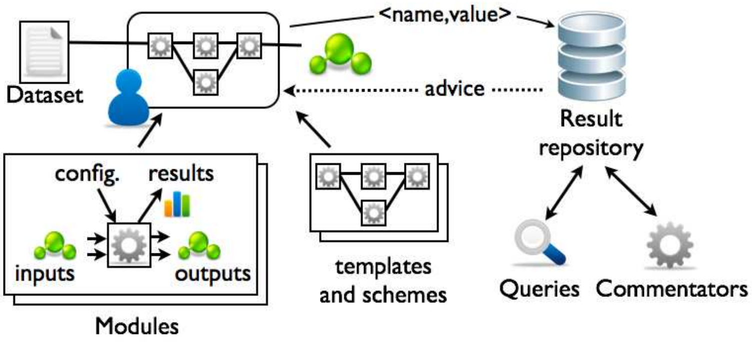


Fig. 13. The architecture of AMLA. Derived from Grabczewski and Jankowski[21].

Meta-learning Meta-learning is done manually by programming queries. The query's constraints can involve the modules having generated the meta-data or the type of properties (the name in the name-value pairs). As such, previously obtained meta-data can be extracted and recombined to generate new meta-knowledge. Secondly, templates with missing modules can also be completed by looking up which were the most successful completed templates in similar problems.

Discussion This is indeed a very fundamental approach to meta-learning, in the sense that it keeps track of all the meta-data generated during the design and execution of KD processes. On the other hand, each query has to be written as a small program that handles the name-value pairs, which might make it a bit harder to use, and the system is still very much under construction. For instance, at this stage of development, it is not entirely clear how the results obtained from different workflows can be compared against each other. It seems that many small queries are needed to answer such questions, and that a more structured repository might be required instead.

6.2 Experiment Databases

Experiment databases (ExpDBs) [6, 65, 67, 66, 64] provide another, although much broader, user-driven platform for the exploitation of meta-knowledge. They aim to collect the thousands of machine learning experiments that are executed every day by researchers and practitioners, and to organize them in a central repository to offer direct insight into the performance of various, state-of-the-art techniques under many different conditions. It offers an XML-based language to describe those experiments, dubbed ExpML, based on an ontology for data mining experimentation, called Exposé, and offers interfaces for DM toolboxes to automatically upload new experiments or download previous ones. The stored meta-data can be queried extensively using SQL queries, or mined to build models of algorithm performance providing insight into algorithm behavior or to predict their performance on certain datasets. While most of the previously discussed systems are designed to be predictive, ExpDBs are mainly designed for *declarative* meta-learning, offering insights into *why* algorithms work or fail on certain datasets, although it can easily be used for predictive goals as well.

Architecture A high-level view of the system is shown in Fig. 14. The five boxed components include the three components used to share and organize the meta-data: an ontology of domain concepts involved in running data mining experiments, a formal experiment description language (ExpML) and an experiment database to store and organize all experiments (ExpDB). In addition, two interfaces are defined: an application programming interface (API) to automatically import experiments from data mining software tools, and a query interface to browse the results of all stored experiments.

Interface First, to facilitate the automatic exchange of data mining experiments, an application programming interface (API) is provided that builds uniform, manipulable experiment instances out of all necessary details and exports them as ExpML descriptions. The top of Fig. 14 shows some of the input details: properties (indicated by tag symbols) of the involved components, from download urls to theoretical properties, as well as the results of the experiments: the models built and their evaluations. Experiments are stored in full detail, so that they can be reproduced at any time (at least if the dataset itself is publicly available).

Software agents such as data mining workbenches (shown on the right in Fig. 14) or custom algorithm implementations can simply call methods from the API to create new experiment instances, add the used algorithms, parameters, and all other details as well as the results, and then stream the completed experiments to online ExpDBs to be stored. A multi-tier approach can also be used: a personal database can collect preliminary experiments, after which a subset can be forwarded to lab-wide or community-wide databases.

ExpML The XML-based ExpML markup language aims to be an extensible, general language for data mining experiments, complementary to PMML[7], which allows to exchange predictive models, but not detailed experimental setups nor evaluations. Based on the Exposé ontology, the language defines various kinds of experiments, such as 'singular learner evaluations', which apply a learning algorithm with fixed parameter settings on a static dataset, and evaluate it using a specific performance estimation method (e.g. 10-fold cross validation) and a range of evaluation metrics (e.g. predictive accuracy). Experiments are described as workflows, with datasets as inputs and evaluations and/or models as outputs, and can contain sub-workflows of preprocessing techniques. Algorithms are handled as composite objects with parameters and components such as kernels, distance functions or base-learners. ExpML also differentiates between general

[7] See `http://www.dmg.org/pmml-v3-2.html`

algorithms (e.g. ‘decision trees’), versioned implementations (e.g. weka.J48) and applications (weka.J48 with fixed parameters). Finally, the context of sets of experiments can also be added, including conclusions, experimental designs, and the papers in which they are used so they can be easily looked up afterwards. Further details can be found in Vanschoren et al. [66], and on the ExpDB website.

Exposé The Exposé ontology provides a formal domain model that can be adapted and extended on a conceptual level, thus fostering collaboration between many researchers. Moreover, any conceptual extensions to the domain model can be translated consistently into updated or new ExpML definitions and database models, thus keeping them up to date with recent developments.

Exposé is built using concepts from several other data mining ontologies. First, OntoDM [49] is a general ontology for data mining which tries to relate various data mining subfields. It provides the top-level classes for Exposé, which also facilitates the extension of Exposé to other subfields covered by OntoDM. Second, EXPO [58] models scientific experiments in general, and provides the top-level classes for the parts involving experimental designs and setups. Finally, DMOP [26] models the internal structure of learning algorithms, providing detailed concepts for general algorithm definitions. Exposé unites these three ontologies and adds many more concepts regarding specific types of experiments, evaluation techniques, evaluation metrics, learning algorithms and their specific configurations in experiments.

ExpDB The ExpDB database model, also based on Exposé, is very fine-grained, so that queries can be written about any aspect of the experimental setup, evaluation results, or properties of involved components (e.g. dataset size). When submitted to an ExpDB, the experiments are automatically stored in a well-organized way, associating them with all other stored experiments and available meta-level descriptions, thus linking empirical data with all known theoretical properties of all involved components.

All further details, including detailed design guidelines, database models, ontologies and XML definitions, can be found on the ExpDB website, which can be found at `http://expdb.cs.kuleuven.be`.

It also hosts a working implementation in MySQL which can be queried online through two query interfaces: an online interface on the homepage itself and an open-source desktop application. Both allow to launch queries written in SQL, or composed in a graphical query interface, and can show the results in tables or graphical plots. The database has been extended several times based on user requests, and is frequently being visited by over 400 users.

Meta-knowledge The ExpML language is designed not only to upload new experiments, but also to add new definitions of any new component, such as a new algorithm or a new data characteristic. As such, the meta-knowledge contained in the database can be dynamically extended by the user to be up to date with new developments. Algorithms, datasets, preprocessing algorithms and evaluation methods are first of all described with all necessary details (e.g. version

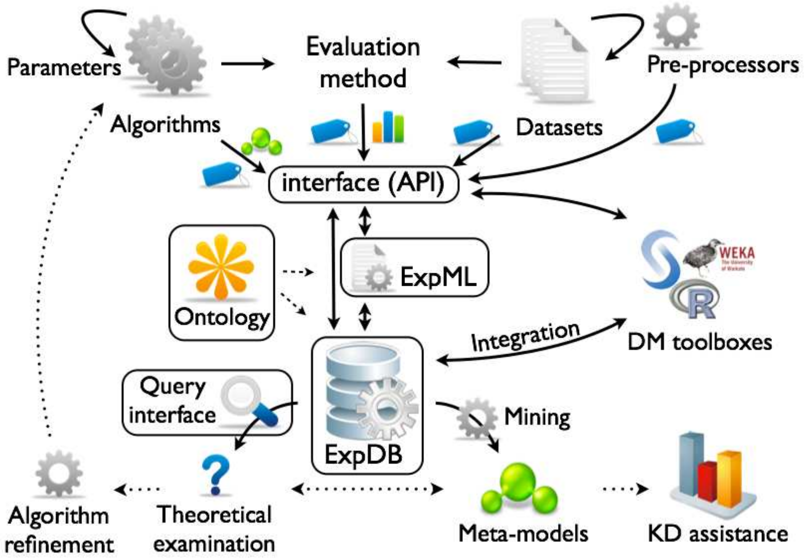


Fig. 14. The architecture of experiment databases.

numbers) to allow unique identification, as well as more informational descriptions of their use. Furthermore, they can be described with an arbitrary number of characterizations, and new ones can be added at any time, preferably with a description of how they are computed, and immediately be used to see whether they improve meta-learning predictions. Attribute-specific data characteristics can be stored as well. Parameter settings are also stored, and each parameter can be described with additional meta-data such as default values, suggested ranges and informal descriptions of how to tune them. Furthermore, an arbitrary number of performance evaluations can be added, including class-specific metrics such as AUROC, and entire contingency matrices. In principle, the produced models could also be stored, e.g. using the PMML format, although in the current version of the system only the predictions for all instances are stored[8]. At the time of writing, the database contained over 650,000 experiments on 67 classification and regression algorithms from WEKA [23], 149 different datasets from UCI [2], 2 data preprocessing techniques (feature selection and sampling), 40 data characteristics [29] and 10 algorithm characterizations [25]. In its current implementation, it covers mostly classification, and some regression tasks, naturally extending to more complex settings, such as kernel methods, ensembles or multi-target learning.

Meta-learning The bottom half of Figure 14 shows the different ways the stored meta-data can be used. First of all, any hypothesis about learning be-

[8] This allows computation of new evaluation metrics without rerunning experiments.

havior can be checked by translating it to an SQL query and interpreting the returned results. Adding and dropping constraints from the query provides an easy and effective means of thoroughly browsing the available meta-data. For instance, we could ask for the 'predictive accuracy' of all support vector machine 'SVM' implementations, and for the value of the 'gamma' parameter (kernel width) of the 'RBF' kernel, on the original (non-preprocessed) version of dataset 'letter'. The result, showing the effect of the gamma parameter, is shown in Fig. 15 (the curve with squares). Adding three more datasets to the query generates the other curves. Just as easily, we could have asked for the size of all datasets instead, or we could have selected a different parameter or any other kind of alteration to explore the meta-data further. SQL queries also allow for the meta-data to be rearranged or aggregated to answer very general questions directly, such as algorithm comparisons, rankings, tracking the effects of preprocessing techniques (e.g. learning curves), or building profiles of algorithms based on many experiments [6, 67]. An example of the latter is shown in Fig. 17: since a large number of bias-variance decomposition experiments are stored, we can write a query that reorganizes all the data and calculates the average percentage of bias error (as opposed to variance error) produced by each algorithm. Such algorithm profiling can be very useful for understanding the performance of learning algorithms.

As shown in Fig. 14, querying is but one of the possible meta-learning applications. First of all, insights into the behavior of algorithms could lead to algorithm refinement, thus closing the algorithm design loop. As a matter of fact, the results in Fig. 15 suggested a correlation between the useful range of the gamma parameter and the number of attributes (shown in parentheses), which led to a refinement of WEKA's SVM implementation [67]. Furthermore, instead of browsing the meta-data, we could also construct meta-models by downloading parts of the meta-data and modeling it. For instance, we could build decision trees predicting when one algorithm outperforms another, as shown in Fig. 16 for algorithms J48 and OneR, or how different parameters interact on certain datasets [63]. Finally, the meta-data can also be used to provide the necessary meta-data for other DM assistance tools, as shall be discussed in Section 7.

Discussion For researchers developing new algorithms, or practitioners trying to apply them, there are many benefits to sharing experiments. First, they make results reproducible and therefore verifiable. Furthermore, they serve as an ultimate reference, a 'map' of all known approaches, their properties, and results on how well they fared on previous problems. And last but not least, it allows previous experiments to be readily reused, which is especially useful when benchmarking new algorithms on commonly used datasets, but also makes larger, more generalizable studies much easier to perform. As such there is a real incentive to share experiments for further reuse, especially since integration in existing DM toolboxes is relatively easy. Such integration can also be used to automatically generate new experimental runs (using, for instance, active learning principles) based on a query or simply on a lack of experiments on certain

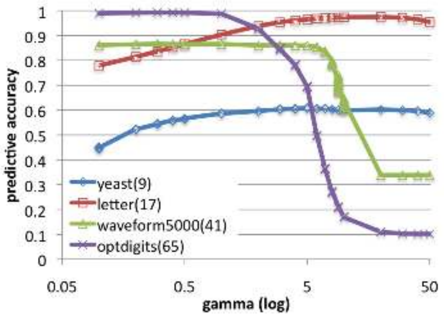


Fig. 15. The effect of parameter gamma of the RBF-kernel in SVMs.

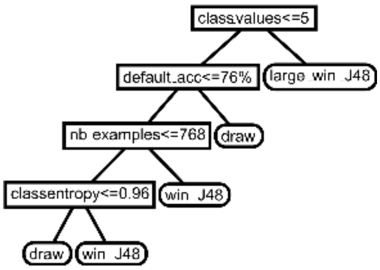


Fig. 16. A meta-decision tree predicting J48's superiority over OneR.

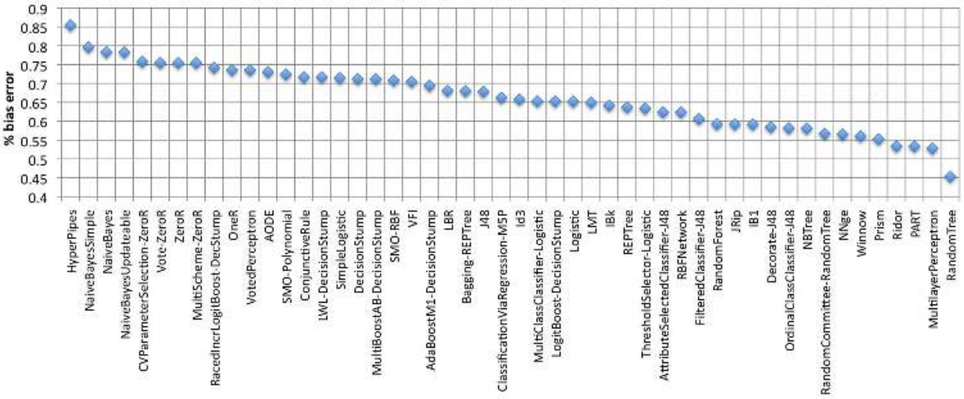


Fig. 17. The average percentage of bias-related error for each algorithm.

techniques. There are also numerous network effects in which advancements in learning algorithms, pre-processing techniques, meta-learning and other subfields all benefit from each other. For instance, results on a new preprocessing technique may be used directly in designing KD workflows, new data characteristics can highlight strengths and weaknesses of learning algorithms, and the large amounts of experiments can lead to better meta-learning predictions. Possible drawbacks are that, unless integrated in a toolbox, the system cannot execute algorithms or KD workflows on demand (although code or urls to executables are always stored) and that, while results received from toolboxes can be assumed to be accurate, results from individual users may have to be verified by referees.

Compared to DMA, one could say that DMA is a local, predictive system, while an ExpDB is a community-based, descriptive system. DMA also employed a database, but it was much simpler, consisting of a few large tables with many columns describing all different kinds of data characterizations. In ExpDBs, the database is necessarily very fine-grained to allow very flexible querying and to scale to millions of experiments from many different contributors. The main difference though is that, while DMA is a system designed for practical algorithm selection advice, ExpDBs offer a platform for any kind of meta-learning study and for the development of future meta-learning systems. It is also quite complementary to MiningMart. While ExpDBs do store preprocessing workflows, MiningMart is much more developed in that area. Finally, compared to AMLA, the repository in ExpDBs is much more structured, allowing much more advanced queries in standard SQL. On the other hand, while ExpDBs can be integrated in any toolbox, AMLA offers a more controlled approach which gathers meta-data from every component in a KD workflow.

7 A new platform for intelligent KD support

7.1 Summary

An overview of the previously discussed architectures is shown in Table 1. The columns represent consecutively the portion of the KD process covered, the system type, how it interacts with the user, what type of meta-information is stored, the data it has been trained on, which KD processes it considers, which evaluation metrics are covered, which meta-features are stored and which meta-learning techniques are used to induce new information, make predictions or otherwise advise the user.

As the table shows, and the systems' discussions have indicated, each system has its own strengths and weaknesses, and cover the KD process to various extents. Some algorithms, like MiningMart and DMA provide a lot of support and gather a lot of meta-information about a few, or only one KD step, while others try to cover a larger part of the entire process, but consider a smaller number of techniques or describe them with less information, usually provided by experts. All systems also expect very different things from their users. Some, especially CITRUS, GDL and AMLA, put the user (assumed to be an expert) firmly in the driver's seat, leaving every important decision to her. Others, like

Table 1. Comparison of meta-learning architectures

	KD step[a]				type	user interface	type meta-info	data scope	KD process scope	evaluation scope	meta-feature scope	meta-learning technique
Consultant-2			▨	▨	Expert system	Q&A sessions	heuristic expert rules	NA	10 algorithms from MLT	speed, model properties	simple statistics, prior knowledge	static meta-model
DMA			▨		meta-model	Webinterface: algo ranking	collection of experiments	67 datasets + 83 from users	10 classific. algorithms	accuracy and speed	7 modified StatLog features	kNN and ranking
NOEMON			▨		meta-model	Algorithm ranking	collection of decision trees	77 datasets (UCI)	3 classific. algorithms	accur., speed and memory	idem StatLog + histograms	Vote over models and ranking
IDEA		▨	▨	▨	Planning	Ranking of KD plans	ontology of KD operators with heuristics	NA	10 preproc., 6 ML algo's, 5 postproc.	accuracy, speed, model-properties	IOPE[b] + basic data properties	AI planning
GLS		▨	▨	▨	Planning	User reviews partial plans	ontology of KD operators + agent meta-rules	NA	7 preproc., 7 ML algo's, 2 postproc.	NA	NA	emergent agent behavior + AI planning
GDL		▨	▨	▨	Planning	May call on experts	LML descriptions	Knowledge bases	NA	NA	12 LML properties	Search
CCE		▨	▨	▨	Planning	Stop criterion	'reliability' values	NA	extensible	complexity	NA	Adaptive system
CITRUS		▨	▨	▨	CBR	Clementine editor	case base + process constraints	NA	Clementine processes	depends on operator	workflow description	CBR, checking constraints
ALT			▨		CBR	Returns 'best' algorithm	collection of cases (experiments)	80 datasets (UCI+more)	21 classific. algorithms	accuracy and speed	StatLog features + 4 algorithm feats	Case-based reasoning
MiningMart	▨	▨			CBR	Editors: create or adapt cases	collection of cases + business ontology	NA (any database)	41 preprocessors	NA	business description of each workflow	none (manual CBR)
HDMA		▨	▨	▨	CBR	Step-by-step, show cases + advice	set of KD cases + set of heuristic expert rules, ontological	NA	selection of WEKA algorithms	user ratings	30 data features, 31 solution features, 5 user ratings	Case-based reasoning
NExT		▨	▨	▨	CBR + Planning	Interaction to solve workflow issues	ontology of workflow issues and solutions	NA	NA	NA	IOPE[b] + workflow description	CBR + AI planning
AMLA		▨	▨	▨	Querying	Process editor + Query lang.	Name-value pairs from any component	NA	extensible	extensible	extensible	querying + fill in templates
ExpDBs		▨	▨	▨	Querying	Query interf. + meta-data up/download	searchable repository of experim's, algo's, data,... + ontology	extensible (150+ UCI datasets)	extensible (70+ WEKA algorithms)	extensible (50+ metrics)	extensible (50+ data and algorithm features)	querying and mining (any algorithm)

[a] The boxes stand consecutively for the Data Selection, Preprocessing+transformation, Data Mining, and Postprocessing step.

[b] IOPE=Inputs, Outputs, Preconditions and Effects of KD operators

Consultant-2, GLS, MiningMart, HDMA and NExT allow the user to interfere in the workflow creation process, often explicitly asking for input. Finally, the meta-model systems and IDEA almost completely automate this process, offering suggestions to users which they may adopt or ignore. Note that, with the exception of Consultant-2, none of the systems performing algorithm selection also predict appropriate parameters, unless they are part of a prior workflow.

A few systems obviously learned from each other. For instance, DMA, NOEMON and ALT learned from StatLog, NExT learned from IDEA and HDMA learned from prior CBR and expert system approaches. Still, most systems are radically different from each other, and there is no strong sense of convergence to a general platform for KD workflow generation.

7.2 Desiderata

We now look forward, striving to combine the best aspects of all prior systems:

Extensibility Every KD support system that only covers a limited number of techniques will at some point become obsolete. It is therefore important that new KD techniques can be added very easily. (GLS, AMLA, ExpDBs)

Integration Ideally, the system should be able to execute the workflow, instead of just offering an abstract description. Keeping extensibility in mind, it should also be able to call on some existing KD tools to execute processes, instead of reimplementing every KD process in a new environment. Indeed, as new types of algorithms are created, new data preprocessing methods are developed, new learning tasks come about, and even new evaluation procedures are introduced, it will be infeasible to re-implement this continuously expanding stream of learning approaches, or to run all the experiments necessary to learn from them. (Consultant-2, HDMA, ExpDBs)

Self-maintenance Systems should be able to update their own meta-knowledge as new data or new techniques become available. While experts are very useful to enrich the meta-knowledge with successful models and workflows, they cannot be expected to offer all the detailed meta-data needed to learn from previous KD episodes. (DMA, NOEMON, ALT, AMLA, ExpDBs)

Common language Effective KD support hinges on a large body of meta-knowledge. As more and more KD techniques are introduced, it becomes infeasible to locally run all the experiments needed to collect the necessary meta-data. It is therefore crucial to take a *community-based* approach, in which descriptions of KD applications and generated meta-data can be generated by, and shared with, the entire community. To achieve this, a common language should be used to make all the stored meta-data interchangeable. This could lead to a central repository for all meta-data, or a collection of different repositories interfacing with each other. (MiningMart, ExpDBs)

Ontologies Meta-knowledge should be stored in a way that makes it machine-interpretable, so that KD support tools can use it effectively. This is reflected by the use of ontologies in many of the discussed systems. Ideally, such an ontology should also establish a common vocabulary for the concepts and relations used in KD research, so that different KD systems can interact. This vocabulary could be the basis of the proposed common description language. (IDEA, GLS, MiningMart, HDMA, NExT, ExpDBs)

Meta-data organization The stored meta-data should also be stored in a way that facilitates manual querying by users. Indeed, we cannot foresee all the possible uses of meta-data beforehand, and should therefore enable the user to perform her own investigations. Moreover, when extending KD support tools with new ways of using meta-data, such a structured repository will drastically facilitate this extension. (AMLA, ExpDBs)

Workflow reuse and adaptation Since most KD applications focus on a limited number of tasks, it is very likely that there exist quite a few prior successful workflows that have been designed for that task. Any intelligent KD support system should therefore be able to return similar workflows, but also offer extensive support to adapt them to the new problem. For instance, if a prior workflow can only partially be reused, new solutions should be proposed to fill in the missing pieces. (MiningMart, HDMA, NExT)

Planning When no similar workflows exist, or when parts of workflows need to be redesigned, using the KD processes' preconditions and effects for planning is clearly a good approach, although care should be taken that the planning space is not prohibitively large. (IDEA,GLS,NExT)

Learning Last but not least, the system should get better as it gains more experience. This includes planning: over time, the system should be able to optimize its planning approach, e.g. by updating heuristic descriptions of the operators used, instead of generating the same plan every time it encounters the same problem. (DMA, NOEMON, GLS, all CBR approaches)

7.3 Architecture

These aspects can be combined in the KD support platform shown in Figure 18. It is based on both our analysis here and on the description of a proposed DM laboratory in Kalousis et al. [30].

A community-based approach This platform is set in a *community-based* environment, with many people using the same KD techniques. It could cover a very general domain, such as KD as a whole, or a more specific one, such as bio-technology, which may result in more specific types of meta-data and more specific ontologies.

Notice that first of all, a common language is used to exchange meta-data between the KD assistant and the tools with which it interacts. First, on the right, there are the many DM/KD toolboxes that implement a wide variety of KD processes. They exchange workflows, descriptions of algorithms and datasets, produced models and evaluations of the implemented techniques.

On the left, there are *web services.* In the last couple of years, there has been a strong movement away from locally implemented toolboxes and datasets, and towards KD processes and databases offered as online services. These *Service*

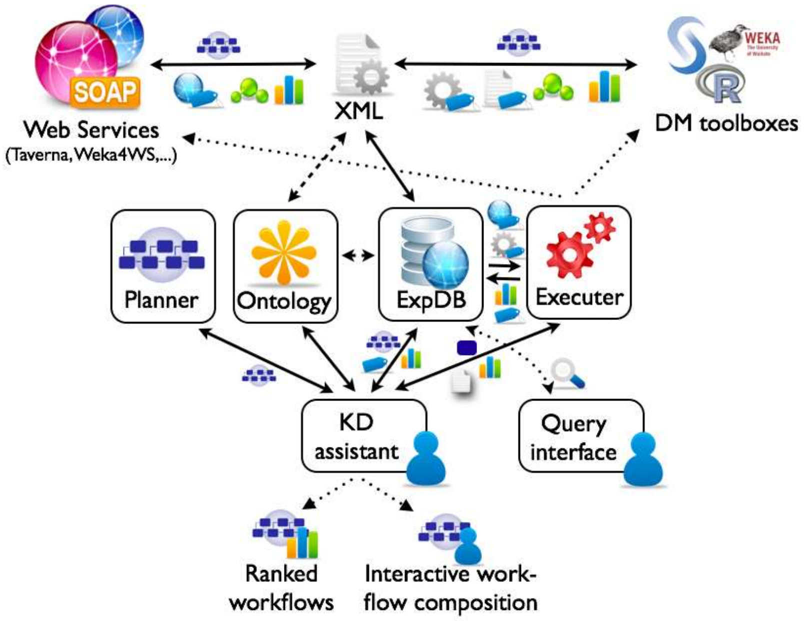


Fig. 18. A proposed platform for intelligent KD support.

Oriented Architectures (SOAs) [18] define standard interfaces and protocols that allow developers to encapsulate tools as services that clients can access without knowledge of, or control over, their internal workings. In the case of a database, this interface may offer to answer specific queries, e.g. a database of airplane flight may return the flight schedule for a specific plane. In the case of a learning algorithm, the interface may accept a dataset (or a database implemented as a web service) and return a decision tree.

While we did not explicitly include experts as a source of meta-knowledge, we assume that they will build workflows and models using the available web services and toolboxes.

When the KD assistant interacts with these services, it will exchange workflows, descriptions of the web services (where to find them and how to interact with them), produced models and evaluation results. Given the rise of web services, XML is a very likely candidate as the modeling language for this common language. Web services interact with each other using SOAP (Simple Object Access Protocol) messages, and describe their interface in WSDL (Web Services Description Language), both of which are described in XML. XML is also used by many KD/DM toolboxes to serialize their data and produced models.

A DM/KD ontology The vocabulary used for these descriptions should be described in a common ontology. Despite many proposed ontologies, there is of yet no consensus on an ontology for DM or KD, but we hope that such an ontology will arise over the coming years. Imagine the internet without HTML, and it is obvious that a common language for KD is essential for progress in this field. The ontology should also provide detailed descriptions of KD processes, such as their place in the hierarchy of different processes, the internal structure of composite learning algorithms, preconditions and effects of all known KD operators, and the parameters that need to be tuned in order to use them. Additional information can be added to extend the ontology to engender further KD support (such as the list of KD issues and solutions in NExT).

A meta-data repository All generated meta-data is automatically stored and organized in a central repository, such as an ExpDB. It collects all the details of all performed experiments, including the workflows used and the obtained results, thus offering a complete log of the experimentation which can be used to reproduce the submitted studies. Moreover, using the ontology, it automatically organizes all the data so it can be easily queried by expert users, allowing it to answer almost any kind of question about the properties and the performance of the used KD techniques, using the meta-data from many submitted studies. It serves as the long-term memory of the KD assistant, but also as that of the individual researcher and the community as a whole. It will be frequently polled by the KD assistant to extract the necessary meta-data, and should contain a query interface for manual investigations as well. The database itself can be wrapped as a web service, allowing automatic interaction with other tools and web services.

Planning and execution The KD assistant interacts with two more components: an AI planner used to solve any planning problems, and an executer component which runs the actually implemented KD algorithms in KD/DM toolboxes or web services to execute workflows, or perhaps to do other calculations such as computing meta-features if they are not implemented in the KD assistant itself. The executer polls the ExpDB to obtain the necessary descriptions and locations of the featured web services, algorithms or datasets.

As for the output generated by the KD assistant, we foresee two important types of advice (beyond manual querying), although surely many more kinds of advice are possible. The first is a ranked list of workflows produced by the KD assistant (even if this workflow only consists of a single learning algorithm). The second, possibly more useful approach is a semi-automatic interactive process in which the user actively participates during the creation of useful workflows. In this case, the KD assistant can be associated with a workflow editing tool (such as Taverna), and assist the users as they compose workflows, e.g. by checking the correctness of a workflow, by completing partial workflows using the planner, or by retrieving, adapting or repairing previously entered workflows.

7.4 Implementation

ExpDBs and ontologies While such a KD support system may still be some way off, recently, a great deal of work has been done that brings us a lot closer to realizing it.

First of all, repositories organizing all the generated meta-data can be built using the experiment databases discussed in Section 6.2. Its ontology and XML-based language for exchanging KD experiments can also be a good starting point. However, building such ontologies and languages should be a collaborative process, so we should also build upon some other recently proposed ontologies in DM, such as OntoDM [48, 49], DMOP [30, 26], eProPlan [35], KDDONTO [24] and KD ontology [69, 70].

Planning Concerning planning, several approaches have been outlined that translate the ontological descriptions of KD operators to a planning description based on the standard Planning Domain Description Language (PDDL) by using an *ontological reasoner* to query their KD ontologies before starting the actual planning process [36, 42, 56]. Other approaches integrate a reasoning engine directly in the planner, so that the planner can directly query the ontology when needed [35, 69, 70]. For instance, eProPlan[35] covers the preconditions and effects of all KD operators, expressed as rules in the Semantic Web Rule Language (SWRL) [28].

Klusch et al. [36] and Liu et al. [42] use a classical STRIPS planner to produce the planning, while Sirin et al. [56] and Kietz et al. [35] propose an Hierarchical Task Network (HTN) planning approach [55], in which each *task* has a set of associated *methods*, which decompose into a sequence of (sub)tasks and/or *operators* that, when executed in that order, achieve the given task. The main task is then recursively decomposed until we obtain a sequence of applicable operators. Somewhat similar to GLS, this divide-and-conquer approach seems an effective way to reduce the planning space.

Zakova et al. [70] uses an adaptation of the heuristic Fast-Forward (FF) planning system [27]. Moreover, it allows the completed workflows to be executed on the Orange DM platform, and vice-versa: workflows composed in Orange are automatically annotated with their KD ontology so that they can be used for ontology querying and reasoning. It does this by mapping their ontology to the ontology describing the Orange operators.

Finally, Kalousis et al. [30] propose a system that will combine planning and meta-learning. It contains a kernel-based, probabilistic meta-learner which dynamically adjusts transition probabilities between DM operators, conditioned on the current application task and data, user-specified performance criteria, quality scores of workflows applied in the past to similar tasks and data, and the users profile (based on quantified results from, and qualitative feedback on, her past DM experiments). Thus, as more workflows are stored as meta-knowledge, and more is known about the users building those workflows, it will learn to build workflows better fit to the user.

Web services The development of service oriented architectures for KD, also called *third-generation DM/KD*, has also gathered steam, helped by increasing support for building workflows of web services.

Taverna [53], for instance, is a system designed to help scientists compose executable workflows in which the components are web services, especially for biological in-silico experiments. Similarly, Triana [60] supports workflow execution in multiple environments, such as peer-to-peer (P2P) and the Grid. Discovery Net [54] and ADMIRE [38] are platforms that make it easier for algorithm designers to develop their algorithms as web services and Weka4WS [59] is a framework offering the algorithms in the WEKA toolkit as web services.

Finally, Orange4WS [51] is a *service-oriented KD* platform based on the Orange toolkit. It wraps existing web services as Orange workflow components, thus allowing to represent them, together with Orange's original components as graphical 'widgets' for manual workflow composition and execution. It also proposes a toolkit to wrap 'local' algorithms as web services.

8 Conclusions

In this chapter, we have provided a survey of the different solutions proposed to support the design of KD processes through the use of meta-knowledge and highlighted their strengths and weaknesses. We observed that most of these systems are very different, and were seemingly developed independently from each other, without really capitalizing on the benefits of prior systems. Learning from these prior architectures, we proposed a new, community-based platform for KD support that combines their best features, and showed that recent developments have brought us close to realizing it.

Acknowledgements

This research is supported by GOA 2003/08 "Inductive Knowledge Bases" and F.W.O.-Vlaanderen G.0108.06 "Foundations of Inductive Databases for Data Mining".

References

1. Aha, D.: Generalizing from case studies: A case study. Proceedings of the Ninth International Conference on Machine Learning pp. 1–10 (1992)
2. Asuncion, A., Newman, D.: Uci machine learning repository. University of California, School of Information and Computer Science (2007)
3. Bensusan, H., Giraud-Carrier, C.: Discovering task neighbourhoods through landmark learning performances. Lecture Notes in Computer Science **1910**, 136–137 (2000)
4. Bernstein, A., Daenzer, M.: The next system: Towards true dynamic adaptations of semantic web service compositions. Lecture Notes in Computer Science **4519**, 739–748 (2007)

5. Bernstein, A., Provost, F., Hill, S.: 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), 503–518 (2005)
6. Blockeel, H., Vanschoren, J.: Experiment databases: Towards an improved experimental methodology in machine learning. Lecture Notes in Computer Science **4702**, 6–17 (2007)
7. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer (2009)
8. Brazdil, P., Soares, C., Costa, J.P.D.: Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. Machine Learning **50**, 251–277 (2003)
9. Chandrasekaran, B., Josephson, J.: What are ontologies, and why do we need them? IEEE Intelligent systems **14**(1), 20–26 (1999)
10. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: Crisp-dm 1.0. a step-by-step data mining guide [http://www.crisp-dm.org] (1999)
11. Charest, M., Delisle, S.: Ontology-guided intelligent data mining assistance: Combining declarative and .... Proceedings of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing pp. 9–14 (2006)
12. Charest, M., Delisle, S., Cervantes, O., Shen, Y.: Intelligent data mining assistance via cbr and ontologies. Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA'06) (2006)
13. Charest, M., Delisle, S., Cervantes, O., Shen, Y.: Bridging the gap between data mining and decision support: A case-based reasoning and .... Intelligent Data Analysis **12**, 1–26 (2008)
14. Craw, S., Sleeman, D., Graner, N., Rissakis, M.: Consultant: Providing advice for the machine learning toolbox. Research and Development in Expert Systems IX: Proceedings of Expert Systems '92 pp. 5–23 (1992)
15. Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., orah L McGuinness, D., Patel-Schneider, P.F., Stein, L.A.: Web ontology language (owl) reference version 1.0. W3C Working Draft. Available at http://www.w3.org/TR/2003/WD-owl-ref-20030331. (2003)
16. Engels, R.: Planning tasks for knowledge discovery in databases; performing task-oriented user-guidance. Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'96) pp. 170–175 (1996)
17. Fikes, R., Nilsson, N.: Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence **2**, 189–208 (1971)
18. Foster, I.: Service-oriented science. Science **308**(5723), 814 (2005)
19. Giraud-Carrier, C.: The data mining advisor: meta-learning at the service of practitioners. Proceedings of the 4th International Conference on Machine Learning and Applications pp. 113–119 (2005)
20. Giraud-Carrier, C.: Metalearning-a tutorial. Tutorial at the 2008 International Conference on Machine Learning and Applications (ICMLA'08) (2008)
21. Grabczewski, K., Jankowski, N.: Versatile and efficient meta-learning architecture: Knowledge representation and .... IEEE Symposium on Computational Intelligence and Data Mining pp. 51–58 (2007)
22. Grabczewski, K., Jankowski, N.: Meta-learning with machine generators and complexity controlled exploration. Lecture Notes in Computer Science **5097**, 545–555 (2008)
23. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The weka data mining software: An update. SIGKDD Explorations **11**(1), 10–18 (2009)

24. Hidalgo, M., Menasalvas, E., Eibe, S.: Definition of a metadata schema for describing data preparation tasks. Proceedings of the ECML/PKDD09 Workshop on 3rd generation Data Mining (SoKD-09) pp. 64–75 (2009)
25. Hilario, M., Kalousis, A.: Building algorithm profiles for prior model selection in knowledge discovery systems. Engineering Intelligent Systems **8**(2) (2000)
26. Hilario, M., Kalousis, A., Nguyen, P., Woznica, A.: A data mining ontology for algorithm selection and meta-mining. Proceedings of the ECML/PKDD09 Workshop on 3rd generation Data Mining (SoKD-09) pp. 76–87 (2009)
27. Homann, J., Nebel, B.: The ff planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14**, 253–302 (2001)
28. Horrocks, I., Patel-Schneider, P., Boley, H.: Swrl: A semantic web rule language combining owl and ruleml. W3C Member submission, http://www.w3.org/Submissions/SWRL/ (2004)
29. Kalousis, A.: Algorithm selection via meta-learning. PhD Thesis. University of Geneve. (2002)
30. Kalousis, A., Bernstein, A., Hilario, M.: Meta-learning with kernels and similarity functions for planning of data mining workflows. ICML/COLT/UAI 2008 Planning to Learn Workshop (PlanLearn) pp. 23–28 (2008)
31. Kalousis, A., Hilario, M.: Model selection via meta-learning: a comparative study. International Journal on Artificial Intelligence Tools **10**(4), 525–554 (2001)
32. Kalousis, A., Theoharis, T.: Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. Intelligent Data Analysis **3**(4), 319–337 (1999)
33. Kaufman, K.: Inlen: a methodology and integrated system for knowledge discovery in databases. PhD Thesis. School of Information Technology and Engineering, George Mason University (1997)
34. Kaufman, K., Michalski, R.: Discovery planning: Multistrategy learning in data mining. Proceedings of the Fourth International Workshop on Multistrategy Learning pp. 14–20 (1998)
35. Kietz, J., Serban, F., Bernstein, A., Fischer, S.: Towards cooperative planning of data mining workflows. Proceedings of the Third Generation Data Mining Workshop at the 2009 European Conference on Machine Learning (ECML 2009) pp. 1–12 (2009)
36. Klusch, M., Gerber, A., Schmidt, M.: Semantic web service composition planning with owls-xplan. Proceedings of the First International AAAI Fall Symposium on Agents and the Semantic Web (2005)
37. Kodratoff, Y., Sleeman, D., Uszynski, M., Causse, K., Craw, S.: Building a machine learning toolbox. Enhancing the knowledge engineering process: contributions from ESPRIT pp. 81–108 (1992)
38. Le-Khac, N., Kechadi, M., Carthy, J.: Admire framework: Distributed data mining on data grid platforms. Proceedings of the 1st International Conference on Software and Data Technologies **2**, 67–72 (2006)
39. Levin, L.: Universal sequential search problems. Problemy Peredachi Informatsii **9**(3), 115–116 (1973)
40. Li, M., Vitányi, P.: An introduction to kolmogorov complexity and its applications. Text and Monographs in Computer Science, Springer (1993)
41. Lindner, G., Studer, R.: Ast: Support for algorithm selection with a cbr approach. Lecture Notes in Computer Science **1704**, 418–423 (1999)
42. Liu, Z., Ranganathan, A., Riabov, A.: A planning approach for message-oriented semantic web service composition. Proceedings of the National Conference on AI **5**(2), 1389–1394 (2007)

43. METAL: Metal: A meta-learning assistant for providing user support in machine learning and data mining. ESPRIT Framework IV LRT Reactive Project Nr. 26.357 (2001)
44. Michalski, R., Kerschberg, L., Kaufman, K.: Mining for knowledge in databases: The inlen architecture, initial implementation and first results. Journal of Intelligent Information Systems **1**(1), 85–113 (1992)
45. Michie, D., Spiegelhalter, D., Taylor, C.: Machine learning, neural and statistical classification. Ellis Horwood (1994)
46. MLT: Machine learning toolbox. Esprit Framework II Research Project Nr. 2154 (1993)
47. Morik, K., Scholz, M.: The miningmart approach to knowledge discovery in databases. Intelligent Technologies for Information Analysis pp. 47–65 (2004)
48. Panov, P., Dzeroski, S., Soldatova, L.: Ontodm: An ontology of data mining. Proceedings of the 2008 IEEE International Conference on Data MIning Workshops pp. 752–760 (2008)
49. Panov, P., Soldatova, L., Dzeroski, S.: Towards an ontology of data mining investigations. Lecture Notes in Artificial Intelligence **5808**, 257–271 (2009)
50. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. Proceedings of the Seventeenth International Conference on Machine Learning pp. 743–750 (2000)
51. Podpecan, V., Jursic, M., Zakova, M., Lavrac, N.: Towards a service-oriented knowledge discovery platform. Proceedings of the SoKD-09 International Workshop on Third Generation Data Mining at ECML PKDD 2009 pp. 25–38 (2009)
52. Rendell, L., Seshu, R., Tcheng, D.: Layered concept learning and dynamically-variable bias management. Proceedings of the 10th International Joint Conference on Artificial Intelligence pp. 308–314 (1987)
53. Roure, D.D., Goble, C., Stevens, R.: The design and realisation of the myexperiment virtual research environment for social sharing of workflows. Future Generation Computer Systems **25**, 561–567 (2009)
54. Rowe, A., Kalaitzopoulos, D., Osmond, M.: The discovery net system for high throughput bioinformatics. Bioinformatics **19**, 225–231 (2003)
55. Sacerdoti, E.: Planning in a hierarchy of abstraction spaces. Artificial intelligence **5**(2), 115–135 (1974)
56. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: Htn planning for web service composition using shop2. Journal of Web Semantics **1**(4), 377–396 (2004)
57. Sleeman, D., Rissakis, M., Craw, S., Graner, N., Sharma, S.: Consultant-2: Pre-and post-processing of machine learning applications. International journal of human-computer studies **43**(1), 43–63 (1995)
58. Soldatova, L., King, R.: An ontology of scientific experiments. Journal of the Royal Society Interface **3**(11), 795–803 (2006)
59. Talia, D., Trunfio, P., Verta, O.: Weka4ws: a wsrf-enabled weka toolkit for distributed data mining on grids. Lecture Notes in Computer Science **3721**, 309–320 (2005)
60. Taylor, I., Shields, M., Wang, I., Harrison, A.: The triana workflow environment: Architecture and applications. Workflows for e-Science. Springer. pp. 320–339 (2007)
61. Utgoff, P.: Shift of bias for inductive concept learning. Machine learning: An artificial intelligence approach. Volume II. Morgan Kaufmann. (1986)
62. Van Someren, M.: Towards automating goal-driven learning. Proceedings of the planning to learn workshop at the 18th European conference of machine learning (ECML 2007) pp. 42–52 (2007)

63. Vanschoren, J., Assche, A.V., Vens, C., Blockeel, H.: Meta-learning from experiment databases: An illustration. Proceedings of the 16th Annual Machine Learning Conference of Belgium and The Netherlands (Benelearn07) pp. 120–127 (2007)
64. Vanschoren, J., Blockeel, H.: A community-based platform for machine learning experimentation. Lecture Notes in Artificial Intelligence **5782**, 750–754 (2009)
65. Vanschoren, J., Blockeel, H., Pfahringer, B.: Experiment databases: Creating a new platform for meta-learning research. Proceedings of the ICML/UAI/COLT Joint Planning to Learn Workshop (PlanLearn08) pp. 10–15 (2008)
66. Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G.: Organizing the world's machine learning information. Communications in Computer and Information Science **17**, 693–708 (2008)
67. Vanschoren, J., Pfahringer, B., Holmes, G.: Learning from the past with experiment databases. Lecture Notes in Artificial Intelligence **5351**, 485–492 (2008)
68. Wirth, R., Shearer, C., Grimmer, U., Reinartz, T., Schlosser, J., Breitner, C., Engels, R., Lindner, G.: Towards process-oriented tool support for knowledge discovery in databases. Lecture Notes in Computer Science **1263**, 243–253 (1997)
69. Zakova, M., Kremen, P., Zelezny, F., Lavrac, N.: Planning to learn with a knowledge discovery ontology. Second planning to learn workshop at the joint ICML/COLT/UAI Conference pp. 29–34 (2008)
70. Záková, M., Podpecan, V., Zelezný, F., Lavrac, N.: Advancing data mining workflow construction: A framework and cases using the orange toolkit. Proceedings of the SoKD-09 International Workshop on Third Generation Data Mining at ECML PKDD 2009 pp. 39–51 (2009)
71. Zhong, N., Liu, C., Ohsuga, S.: Dynamically organizing kdd processes. International Journal of Pattern Recognition and Artificial Intelligence **15**(3), 451–473 (2001)
72. Zhong, N., Matsui, Y., Okuno, T., Liu, C.: Framework of a multi-agent kdd system. Lecture Notes in Computer Science **2412**, 337–346 (2002)
73. Zhong, N., Ohsuga, S.: The gls discovery system: its goal, architecture and current results. Lecture Notes in Computer Science **869**, 233–244 (1994)