Ontology-Based Meta-Mining of Knowledge Discovery Workflows

Melanie Hilario, Phong Nguyen, Huyen Do,
Adam Woznica, Alexandros Kalousis

Artificial Intelligence Laboratory, University of Geneva

1 Abstract

This chapter describes a principled approach to meta-learning that has three distinctive features. First, whereas most previous work on meta-learning focused exclusively on the learning task, our approach applies meta-learning to the full knowledge discovery process and is thus more aptly referred to as meta-mining. Second, traditional meta-learning regards learning algorithms as black boxes and essentially correlates properties of their input (data) with the performance of their output (learned model). We propose to tear open the black box and analyse algorithms in terms of their core components, their underlying assumptions, the cost functions and optimization strategies they use, and the models and decision boundaries they generate. Third, to ground meta-mining on a declarative representation of the data mining (DM) process and its components, we built a DM ontology and knowledge base using the Web Ontology Language (OWL).

The Data Mining Optimization Ontology (DMOP, pronounced dee-mope)) provides a unified conceptual framework for analysing DM tasks, algorithms, models, datasets, workflows and performance metrics, as well as their relationships. The DM knowledge base uses concepts from DMOP to describe existing data mining algorithms and their implementations in major DM software packages. Meta-data collected from data mining experiments are also described in terms of concepts from the ontology and linked to algorithm and operator descriptions in the knowledge base; they are then stored in data mining experiment data bases to serve as training and evaluation data for the meta-miner.

These three features together lay the groundwork for what we call deep or semantic meta-mining, i.e., DM process or workflow mining that is driven simultaneously by meta-data and by the collective expertise of data miners embodied in the data mining ontology and knowledge base. In Section 2, we review the state of the art in the fields of meta-learning and data mining ontologies; at the same time, we motivate the need for ontology-based meta-mining and distinguish our approach from related work in these two areas. Section 3 gives a detailed description of DMOP, while Section 4 introduces a novel method for ontology-based discovery of generalized patterns from data mining workflows. Section 5 reports on proof-of-concept experiments conducted to gauge the efficacy of DMOP-based workflow mining, and Section 6 concludes.

2 State of the art and motivation

The work described in this chapter draws together two research streams that have remained independent so far—meta-learning and data mining ontology construction. This section reviews the state of the art in both areas and points out the novelty of our approach with respect to each.

2.1 From meta-learning to meta-mining

Meta-learning is learning to learn: in computer science, it is the application of machine learning techniques to meta-data describing past learning experience in order to modify some aspect of the learning process and improve the performance of the resulting model [29,3,13,78]. Meta-learning thus defined applies specifically to learning, which is only one—albeit the central—step in the data mining (or knowledge discovery) process[1]. The quality of mined knowledge depends as much on other steps such as data cleaning, data selection, feature extraction and selection, model pruning, and model aggregation. We still lack an understanding of how the different components of the data mining process interact; there are no clear guidelines except for high-level process models such as CRISP-DM [18]. Process-related issues, such as the composition of data mining operations and the need for a methodology of data mining, are among the ten data mining challenges discussed in [80].

In response to this challenge, a number of systems have been designed to provide user support throughout the different phases of the KD process (Serban et al., 2010). Most of them rely on a planning approach and produce workflows that are valid but not necessarily optimal with respect to a given cost function such as predictive accuracy. This is the case of the planner-based intelligent discovery assistant (IDA) implemented in the e-LICO project[2]. To allow the planner to select the most promising workflows from an often huge set of candidates, an ontology-based meta-learner mines records of past data mining experiments to extract models and patterns that will suggest which DM algorithms should be used together in order to achieve the best results for a given problem, data set and cost function. The e-LICO IDA therefore self-improves as a result of *meta-mining*, loosely defined as KD process-oriented meta-learning. Meta-mining extends the meta-learning approach to the full knowledge discovery process: in the same way that meta-learning is aimed at optimizing the results of learning, meta-mining optimizes the results of data mining by taking into account the interdependencies and interactions between the different process operations, and in particular

[1] We follow current usage in treating *data mining* and *knowledge discovery* as synonyms, using the terms *learning* or *modelling* to refer to what Fayyad et al. [25] called the data mining phase of the knowledge discovery process.

[2] http://www.e-lico.org

between learning and the different pre/post-processing steps. In this sense, meta-mining subsumes meta-learning and must address all the open issues regarding meta-learning.

Since it emerged as a distinct research area in machine learning, meta-learning has been cast as the problem of dynamically selecting or adjusting inductive bias [61,74,75,30,77]. There is a consensus that with no restrictions on the space of hypotheses to be explored by the learning algorithm and no preference criterion for comparing candidate hypotheses, then no inductive method can do better on average than random guessing. In short, without bias no learning is possible [51]; the so-called no-free-lunch theorem on supervised learning [79] and the conservation law for generalization performance [63] express basically the same idea. There are two types of bias: representational bias restricts the hypothesis space whereas procedural—aka search or preference—bias gives priority to certain hypotheses over others in this space. The most widely addressed meta-learning tasks—algorithm selection and model selection[3]—can be viewed as ways of selecting or adjusting these two types of bias. Algorithm selection is the choice of the appropriate algorithm for a given task, while model selection is the choice of the specific parameter settings that will produce relatively good performance for a given algorithm on a given task. Algorithm—or model class—selection amounts to adjusting representational bias and model selection to adjusting preference bias. Though there have been a number of studies on model selection [22,23,68,2], meta-learning research has focused predominantly on algorithm selection [73,67,41,43,47,69].

The algorithm selection problem has its origins outside machine learning [66]. In 1976 a seminal paper by John Rice [62] proposed a formal model comprising four components: a problem space $\mathcal{X}$ or collection of problem instances describable in terms of features defined in feature space $\mathcal{F}$, an algorithm space $\mathcal{A}$ or set of algorithms considered to address problems in $\mathcal{X}$, and a performance space $\mathcal{P}$ representing metrics of algorithm efficacy in solving a problem. Algorithm selection can then be formulated as follows: Given a problem $x \in \mathcal{X}$ characterized by $f(x) \in \mathcal{F}$, find an algorithm $\alpha \in \mathcal{A}$ via the selection mapping $S(f(x))$ such that the performance mapping $p(\alpha(x)) \in \mathcal{P}$ is maximized. A schematic diagram of the abstract model is given in Fig. 1.

In Rice's model, selection mapping from problem space $\mathcal{X}$ onto algorithm space $\mathcal{A}$ is based solely on features $f \in \mathcal{F}$ over the problem instances. In machine learning terms, the choice of the appropriate induction algorithm is conditioned solely on the characteristics of the learning problem and data. Strangely, meta-learning research has independently abided by the same restriction from its inception to the present. From early meta-learning attempts [61,12] to more recent investigations, the dominant trend relies almost exclusively on meta-data describing the characteristics of base-level data sets used in learning, and the goal of meta-learning has even been defined restrictively as learning a mapping from

[3] We take algorithm/model selection to include task variants and extensions such as algorithm/model ranking and algorithm/model combination.

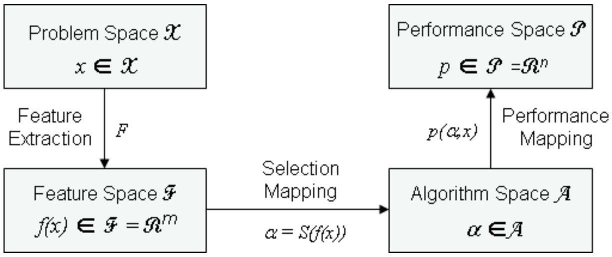

Problem Space $\mathcal{X}$
$x \in \mathcal{X}$
Feature Extraction
F
Feature Space $\mathcal{F}$
$f(x) \in \mathcal{F} = \mathfrak{R}^m$
Selection Mapping
$\alpha = S(f(x))$
Algorithm Space $\mathcal{A}$
$\alpha \in \mathcal{A}$
$p(\alpha,x)$
Performance Mapping
Performance Space $\mathcal{P}$
$p \in \mathcal{P} = \mathfrak{R}^n$

Fig. 1. Rice's model of the algorithm selection problem. Adapted from [62,66]

dataset characteristics to algorithm performance [3]. Researchers have come up with an abundant harvest of such characteristics, in particular statistical and information-theoretic properties of training data [46,36,42,72,16]. A more recent research avenue, dubbed landmarking, characterizes data sets in terms of the predictive performance attained by simple learning algorithms when applied to them [57,8,27,48]; yet another approach describes data sets based on features of the models that were learned from them [8,55]. In all cases, however, the goal is to discover mappings from data set characteristics to learning algorithms viewed essentially as black boxes.

Thus far there has been no attempt to correlate dataset and algorithm characteristics, in other words to understand which aspects of a given algorithm explain its expected performance given the features of the data to be modelled. As a consequence, current meta-learners cannot generalize over algorithms as they do over data sets. To illustrate this problem, suppose that three algorithms are observed to achieve equivalent performance on a collection of datasets representing a task family. Meta-learning would yield three disjunctive rules with identical conditions and distinct recommendations. There would be no way of characterizing in more abstract terms the class of algorithms that would perform well on the given task domain. In short, no amount of meta-learning would reap fresh insights into the commonalities underlying the disconcerting variety of algorithms.

To overcome this difficulty, we propose to extend the Rice framework and pry open the black box of algorithms [37]. To be able to differentiate similar algorithms as well as detect deeper commonalities among apparently unrelated ones, we propose to characterize them in terms of components such as the model structure built, the objective functions and search strategies used, or the type of data partitions produced. This compositional approach is expected to have two far-reaching consequences. Through a systematic analysis of all the ingredients that constitute an algorithm's inductive bias, meta-learning systems (and data miners in the first instance) will be able to infer not only which algorithms work for specific data/task classes but—more importantly—why. In the long term, they should be able to operationalize the insights thus gained in order to combine algorithms purposefully and perhaps design new algorithms. This novel approach to algorithm selection is not limited to the induction phase; it should be applicable to other data and model processing tasks that require search in the space of candidate algorithms. The proposed approach will also be adapted to model selection, i.e., finding the specific parameter setting that will allow a given algorithm to achieve acceptable performance on a given task. This will require an extensive study of the parameters involved in a given class of algorithms, their role in the learning process or their impact on the expected results (e.g., on the complexity of the learned model for induction algorithms), and their formalization in the data mining ontology.

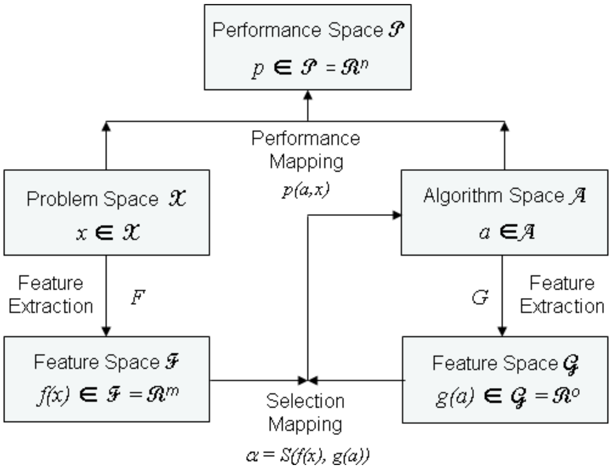

Performance Space $\mathcal{P}$
$p \in \mathcal{P} = \Re^n$
Performance Mapping
$p(a,x)$
Problem Space $\mathcal{X}$
$x \in \mathcal{X}$
Algorithm Space $\mathcal{A}$
$a \in \mathcal{A}$
Feature Extraction
F
G
Feature Extraction
Feature Space $\mathcal{F}$
$f(x) \in \mathcal{F} = \Re^m$
Feature Space $\mathcal{G}$
$g(a) \in \mathcal{G} = \Re^o$
Selection Mapping
$\alpha = S(f(x), g(a))$

Fig. 2. Proposed model for algorithm selection

The proposed revision of Rice's model for algorithm selection is visualized in Fig. 2. It includes an additional feature space $\mathcal{G}$ representing the space of features extracted to characterize algorithms; selection mapping is now a function of both problem and algorithm features. The revised problem formulation now is: Given a problem $x \in \mathcal{X}$ characterized by $f(x) \in \mathcal{F}$ and algorithms $a \in \mathcal{A}$ characterized by $g(a) \in \mathcal{G}$, find an algorithm $\alpha \in \mathcal{A}$ via the selection mapping $S(f(x), g(a))$ such that the performance mapping $p(a(x)) \in \mathcal{P}$ is maximized.

2.2 Data Mining Ontologies

An ontology is a structure $\mathcal{O} := (\mathcal{C}, \leq_C, \mathcal{R}, \sigma, \leq_R, IR)$ consisting of a set of concepts $\mathcal{C}$ and a set of relations $\mathcal{R}$, a partial order $\leq_C$ on $\mathcal{C}$, called concept hierarchy or taxonomy, a function $\sigma : \mathcal{R} \rightarrow C \times C$ called signature, a partial order $\leq_R$ on $\mathcal{R}$ called relation hierarchy, and a set IR of inference rules expressed in a logical language $\mathcal{L}$ [39]. Before the coming of age of ontological engineering as a distinct research area, there had been early attempts at a systematic description of machine learning and data mining processes. CAMLET [71] used a rudimentary ontology of learning tasks and algorithms to support the automatic composition and revision of inductive processes. While CAMLET focused on model building, the MiningMart project [52] shifted the focus to the preprocessing phase. The metadata used in MiningMart was condensed into a small ontology whose primary purpose was to allow for the reuse of stored data mining cases. Operator chains, mainly for preprocessing, were described at both the abstract and executable levels to facilitate maintenance of the case base and adaptation of retrieved cases. The taxonomy of data mining operators underlying the IDEA system [9] had a broader scope in the sense that it covered that preprocessing, induction and postprocessing phase of the knowledge discovery process. It had an explicit representation of operator preconditions and effects and was used by an AI-style planner to generate all valid workflows for a given application task. However, unlike CAMLET and MiningMart, where the assembled operator sequences were executed and later revised and reused, IDEA did not go beyond the simple enumeration of valid DM process plans.

The advent of ontology languages and tools for the Semantic Web gave rise to a new generation of data mining ontologies, the majority of which are aimed at the construction of workflows for knowledge discovery. Among these, DAMON [17] and GridMiner Assistant (GMA) [14] focus more specifically on the development of distributed KDD applications on the Grid. DAMON describes available data mining software, their underlying methods and associated constraints in order to enable semantic search for appropriate DM resources and tools. GMA's data mining ontology, written in OWL, is based on industry standards like the CRISP-DM process model [18] and the Predictive Model Markup Language [32]. The ontology is used to support interactive workflow design: GMA first backward-chains from the initial goal/task to compose an abstract task sequence, eliciting user preferences as needed (e.g., to select the preferred type of model). In the second phase, it forward-chains along this sequence to fill in task parameters, either by reasoning from preconditions and effects given in the ontology or by getting user input.

Other ontologies for DM workflow construction are KDDONTO [20], KD Ontology [82] and DMWF [44]. KDDONTO provides knowledge of data mining algorithms required by KDDComposer to build valid DM workflows. Given an algorithm B, the goal is to find the set of algorithms A_i whose outputs can be used as inputs to B. This is done by estimating the degree of match between the expected output of each algorithm A_i and the required input B. Semantic similarity is computed based on the length of the ontological paths between two concepts along the isA and partOf relations. However (dis)similarity is only one component of a score or cost function that takes account of other factors such as estimated performance or the relaxation of constraints on the input of B. This score induces a finer ranking on the candidate workflows and allows for the early disposal of those whose cost exceeds a given threshold.

KD Ontology [82] and a planner are tightly integrated in an automated workflow composition system that has been developed in conformance with proven standards from the semantic web, namely the Web Ontology Language for ontology modelling and the Planning Domain Definition Language (PDDL) for planning. It has a blend of interesting features not found in other related work. Contrary to IDEA and GMA which generate workflows in the form of linear sequences, it creates workflows as directed acyclic graphs whose nodes represent implemented algorithms; however, these are abstract workflows in the sense that the algorithm parameters need to be instantiated in order to produce executable workflows. In addition, KD Ontology incorporates knowledge about a broad range of data mining algorithms, from standard propositional learners to more advanced algorithms that can handle structured and relational data, thus expanding the power and diversity of workflows that the planner is able to generate. KD Ontology has been tested on two use cases, one in genomics and the other in product engineering.

DMWF and its associated planning environment (eProPlan) [44] have been developed to provide user support in the e-LICO virtual data mining laboratory. A hierarchical task network (HTN) based planner performs a series of task decompositions starting from the initial user task, and generates alternative plans when several methods are available for a given (sub)task. Given the number of operators available to the planner (more than 600 from RapidMiner and Weka alone), the potentially infinite number of valid workflows precludes the approach of enumerating them all and leaving the final choice to the user. Hence the choice of cooperative-interactive workflow planning, in which the planner incrementally expands the current plan and periodically proposes a small number of intermediate extensions or refinements from which the user can choose. The ontology provides the basis for cooperative-interactive workflow planning through the concept of workflow templates, i.e. abstract workflows that can mix executable operators and tasks to be refined later into sub-workflows. These templates serve as the common workspace where user and system can cooperatively design workflows. Automated experimentation can help make intermediate decisions, though this is a viable alternative only when time and computational resources are abundant.

Like the other workflow building systems described above, eProPlan generates a set of correct workflows but has no way of selecting that which is most likely to produce the best results. DMWF models operator preconditions and effects but has no knowledge of the algorithms they implement or the models they are capable of generating. The solution adopted in the e-LICO virtual DM lab is to couple the workflow generator with a meta-miner whose role is to rank the workflows or select the most promising ones based on lessons learned from past data mining experience. The meta-miner relies extensively on deep knowledge of data mining algorithms' biases and capabilities modelled in DMOP (Section 3).

As mentioned above, the vast majority of existing DM ontologies are aimed at supporting workflow construction. One exception is OntoDM [53], whose declared goal is to provide a unified framework for data mining [24]. It contains definitions of the basic concepts used in data mining (e.g., DM task, algorithm, dataset, datatype), which can be combined to define more complex entities such as constraints or data mining experiments. The distinguishing feature of OntoDM is its compliance with ontological best practices defined mainly in the field of biological investigations. It uses a number of upper level ontologies such as Basic Formal Ontology (BFO), the OBO Relation Ontology (RO), and the Information Artefact Ontology (IAO). Its structure has been aligned with the Ontology of Biological Investigations (OBI), and its development follows strict rules like avoiding multiple inheritance and limiting the number of relations. OntoDM is called a general-purpose ontology by its authors and remains to be applied to a concrete data mining use case. More recently, a similar ontology called Exposé [76] has been developed to provide the conceptual basis for a database of data mining experiments [11]. Exposé borrows OntoDM's upper level structure and DMOP's conceptualization of data mining algorithms, and completes these with a description of experiments that serves as the basis of an Experiment Markup Language. OntoDM's and Exposé's alignment with upper ontologies suggests that their primary use is to provide a controlled vocabulary for DM investigations. Among the DM ontologies that do not focus on workflow construction, DMOP is unique in its focus on the problem of optimizing the knowledge discovery process through an in-depth characterization of data and especially of DM algorithm biases and internal mechanisms.

3 An Ontology for Data Mining Optimization

3.1 Objectives and overview

The overall goal of DMOP is to provide support for all decision-making steps that have an impact on the outcome of the knowledge discovery process. It focuses specifically on DM tasks (e.g., learning, feature extraction) whose accomplishment requires non-trivial search in the space of alternative methods. For each such task, the decision process involves two steps that can be guided by prior knowledge from the ontology: *algorithm selection* and *model selection*. While data mining practitioners can profitably consult DMOP to perform "manual" algorithm and model selection, the ontology has been designed to automate these two operations. Thus a third use of DMOP is *meta-learning*, i.e., the analysis of meta-data describing learning episodes in view of extracting patterns and rules to improve algorithm and model selection. Finally, generalizing meta-learning to the complete DM process, DMOP's most innovative objective is to support *meta-mining* or the meta-analysis of complete data mining experiments in order to extract workflow patterns that are predictive of good or bad performance. In short, DMOP charts the higher-order feature space in which meta-learning and meta-mining can take place.

The DMOP ontology's overall structure and foundational role in meta-mining are illustrated in Figure 3. DMOP provides a conceptual framework that defines the

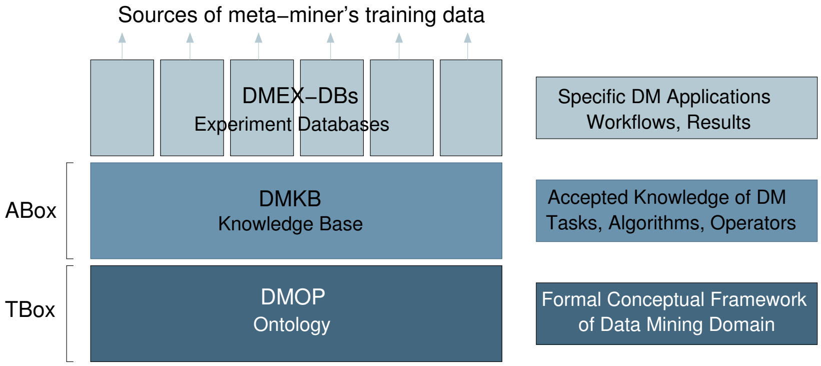


Fig. 3. The DMOP architecture

relationships among the core DM entities such as tasks, algorithms, models, workflows, experiments (Section 3.2). The hierarchy of concepts (classes), together with axioms expressing their properties, relations and restrictions, constitute the terminological box (TBox), or what we can call the ontology proper. Based on this conceptual groundwork, individuals are created as instances of one or several concepts from the TBox; these individuals, and all statements concerning their properties or their relations with other individuals, form the assertional box (ABox), also called the knowledge base.

The **DM knowledge base (DMKB)** captures the DM community's collective expertise; ideally, it would be a compendium of the state of the art in data mining. DMKB builds on DMOP's taxonomy of major data mining tasks and paradigms (broad algorithm families) to classify and characterize individual algorithms that have been developed to date, together with their better known implementations. For instance, DMKB contains formal descriptions of algorithms most often used to solve classification tasks: generative approaches such as Naïve Bayes, discriminative approaches such as Logistic Regression, and discriminant function approaches such as SVMs. To distinguish individual variants of a given algorithm family (e.g. NaiveBayesNormal, NaiveBayesKernel, NaiveBayesDiscretized, MultinomialNaiveBayes, ComplementNaiveBayes), each is described giving specific values to properties defined in the DM ontology. Similarly, operators from DM packages are analysed to identify the algorithms they implement, so that all attempts to explain an operator's performance go beyond low-level programming considerations to reason on the basis of algorithm assumptions and basic components.

DM Experiment data bases (DMEX-DBs) are built using concept and property definitions from DMOP as well as concrete algorithm and operator definitions from DMKB. In contrast to DMKB, which is a compilation of commonly accepted data mining knowledge, a DMEX database is any collection of experimental data concerning a given data mining application task. It is usually domain-specific and contains ground facts about clearly defined use cases, their associated data sets, actual data mining experiments conducted to build predictive or descriptive models that address the task, and the estimated performance of these models.

Thus any number of DM experimental data bases can be built with schemas based on DMOP and DMKB.

3.2 The Core Concepts

To develop the DM concept hierarchy, we start with the two endpoints of the DM process. At one end, the process receives input data relative to a given discovery task; at the other, it outputs knowledge in the form of a descriptive or predictive model, typically accompanied by some kind of report containing the learned model's estimated performance and other meta-data. These three concepts—Data, DM-Model, DM-Report—play a central role in DMOP and have been grouped, for convenience, in a derived class called IO-Object. The major concept hierarchies of the ontology—DM-Task, DM-Algorithm, DM-Operator and DM-Workflow—are structured directly or indirectly by these three types of input/output objects.

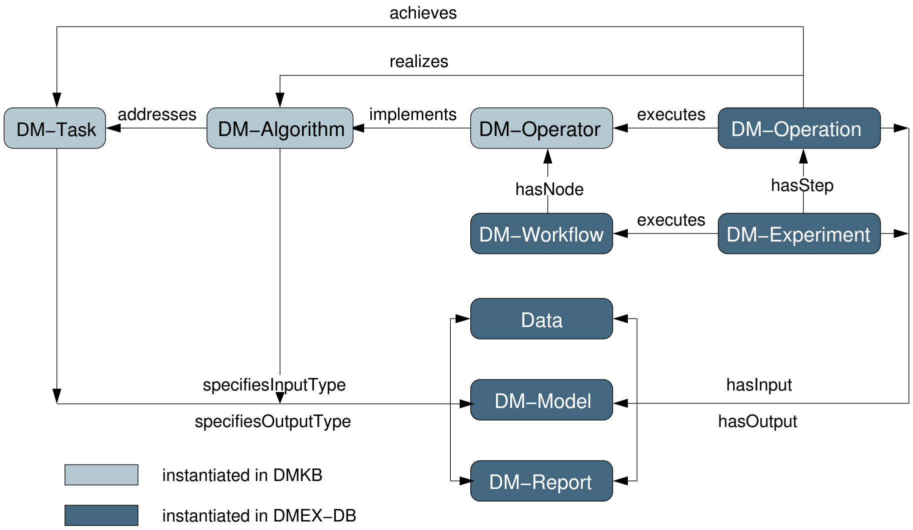


Fig. 4. The core concepts of DMOP

Tasks and algorithms as defined in DMOP are not processes that directly manipulate data or models, rather they are specifications of such processes. A DM-Task is a specification of any piece of work that is part of the DM process, essentially in terms of the input it requires and the output it is expected to produce. A DM-Algorithm is the specification of a procedure that addresses a given Task, while a DM-Operator is a program that implements a given DM-Algorithm (see Figure 4). Instances of DM-Task and DM-Algorithm do no more than specify their input/output types; only processes called DM-Operations have actual inputs and outputs. A process that executes a DM-Operator also realizes the DM-Algorithm implemented by the operator and by the same token achieves the DM-Task addressed by the algorithm. Finally, just as a DM-Workflow is a complex structure composed of DM operators, a DM-Experiment is a complex process composed of operations (or operator executions). A workflow can be represented as a directed acyclic graph in which nodes correspond to operators and edges to IO-Objects, i.e. to the data, models and meta-level reports consumed and produced by DM operations. An experiment is described by all the objects that participate in the process: a workflow, data sets used and produced by the different data processing phases, the resulting models and meta-data quantifying their performance. Instances of DM-Algorithm and DM-Operator are described in the DMKB because they represent consensus data mining knowledge, while instances of DM-Workflow and DM-Experiment are stored in application-specific DM experiment data bases.

Data As the critical resource that feeds the knowledge discovery process, data are a natural starting point for the development of a data mining ontology. Over the past decades many researchers have actively investigated data characteristics that might explain generalization success or failure. An initial set of such statistical and information-theoretic measures was gathered in the StatLog project [50] and extended in the Metal project with other statistical [46], landmarking-based [57,7]and model-based [55,56] characteristics. Data descriptors in DMOP are based on the Metal heritage, which we further extended with geometrical measures of data complexity [6].

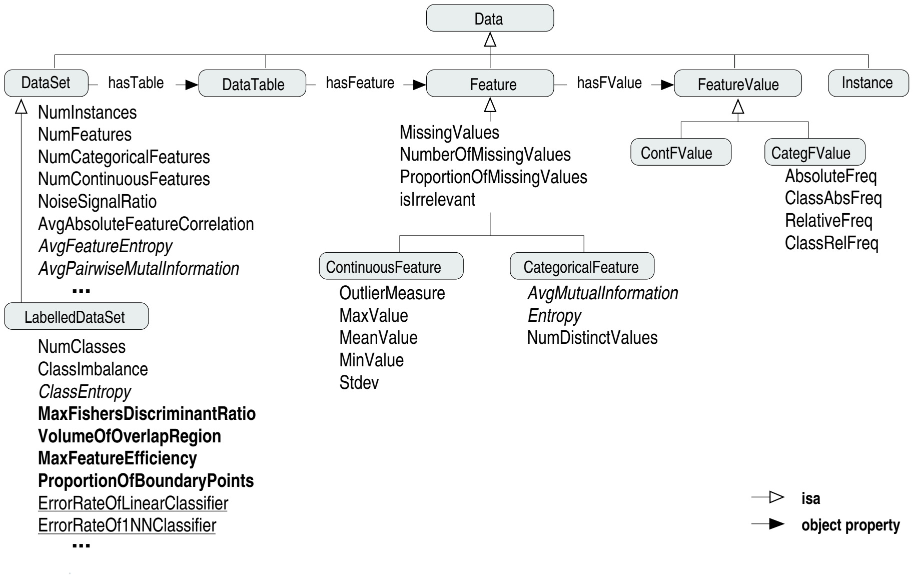


Fig. 5. Data characteristics modelled in DMOP

Figure 5 shows the descriptors associated with the different Data subclasses. Most of these are statistical measures, such as the number of instances or the number of features of a data set, or the absolute or relative frequency of a categorical feature value. Others are information-theoretic measures (italicized in the figure) ; examples are the entropy of a categorical feature or the class entropy of a labelled dataset. Characteristics in bold font, like the max Fisher's discriminant ratio, which measures the highest discriminatory power of any single feature in the data set, or the fraction of data points estimated to be on the class boundary, are geometric indicators of data set complexity; detailed definitions of these characteristics can be found in [38]. Finally, error rates such as those of a linear or a 1-NN classifier (underlined) are data characteristics based on landmarking, which was briefly described in Section 2.1.

DM Tasks As mentioned above, DMOP places special emphasis on so-called core DM tasks—search-intensive or optimization-dependent tasks such as feature construction or learning, as opposed to utility tasks such as reading/writing a data set or sorting a vector of scalars.

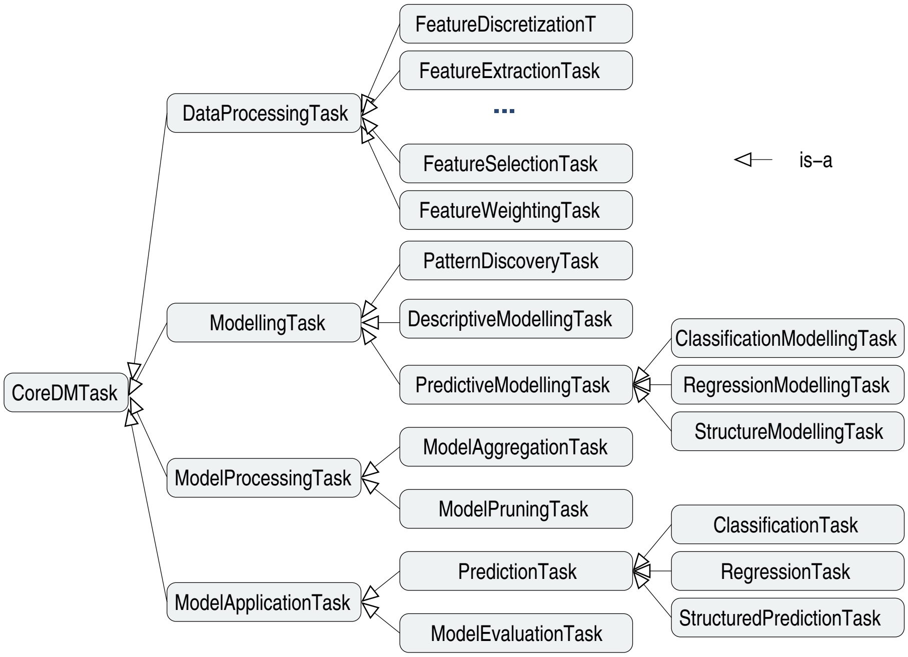


Fig. 6. The CoreDMTask Hierarchy

The CoreDMTask hierarchy (Fig. 6) comprises four major task classes defined by their inputs and outputs: data processing, modelling, model processing, and model application:

$$\texttt{DataProcessingTask} \sqsubseteq \forall\texttt{specifiesInputType.}\underline{\text{Data}} \sqcap \forall\texttt{specifiesOutputType.}\underline{\text{Data}}$$

$$\texttt{ModellingTask} \sqsubseteq \forall\texttt{specifiesInputType.}\underline{\text{Data}} \sqcap \forall\texttt{specifiesOutputType.}\underline{\text{Model}}$$

$$\texttt{ModelProcessingTask} \sqsubseteq \forall\texttt{specifiesInputType.}\underline{\text{Model}} \sqcap \forall\texttt{specifiesOutputType.}\underline{\text{Model}}$$

$$\texttt{ModelApplicationTask} \sqsubseteq \forall\texttt{specifiesInputType.}\underline{\text{Model}} \sqcap \forall\texttt{specifiesOutputType.}\underline{\text{Report}}$$

Specializations of each task are defined by specializing its input and output types. As we move down the tree in Figure 6, the descendant classes of ModellingTask specify input and output types that are successively more specific subclasses of Data and Model respectively:

$$\texttt{PredictiveModellingTask} \sqsubseteq \forall\texttt{specifiesInputType.}\underline{\text{LabelledDataSet}}$$
$$\sqcap \forall\texttt{specifiesOutputType.}\underline{\text{PredictiveModel}}$$

$$\text{ClassificationModellingTask} \sqsubseteq \forall\texttt{specifiesInputType.}\underline{\text{CategoricalLabelledDataSet}}$$
$$\sqcap \forall\texttt{specifiesOutputType.}\underline{\text{ClassificationModel}}$$

Note the distinction between PredictiveModellingTask — the construction of a predictive model — and PredictionTask, which is the simple application of the model built through predictive modelling. The same distinction holds between their respective subclasses, e.g. classification is the application of a classifier built through classification modelling, and similarly for regression and structured prediction. This is in contrast to current usage in the literature, where the term classification, for instance, designates ambiguously the process of building or applying a classifier.

DM Algorithms The top levels of the Algorithm hierarchy reflect those of the Task hierarchy, since each algorithm class is defined by the task it addresses, e.g. $\texttt{DataProcessingAlgorithm} \equiv \texttt{Algorithm} \sqcap \exists\,\texttt{addresses.DataProcessingTask}$. However, the Algorithm hierarchy plunges more deeply than the Task hierarchy: for each leaf class of the task hierarchy, there is an often dense subhierarchy of algorithms that specify diverse ways of addressing each task. For instance, the leaf concept ClassificationModellingTask in Figure 6 is mapped directly onto the hierarchy rooted in the concept of ClassificationModellingAlgorithm in Figure 7.

As shown in the figure, classification modelling algorithms are divided into three broad categories [10]. *Generative methods* compute the class-conditional densities $p(\mathbf{x}|C_k)$ and the priors $p(C_k)$ for each class C_k, then use Bayes' theorem to find posterior class probabilities $p(C_k|\mathbf{x})$. They can also model the joint distribution $p(\mathbf{x}, C_k)$ directly and then normalize to obtain the posteriors. In both cases, they use statistical decision theory to determine the class for each new input. Examples of generative methods are normal (linear or quadratic) discriminant analysis and Naive Bayes. *Discriminative methods* such as logistic regression

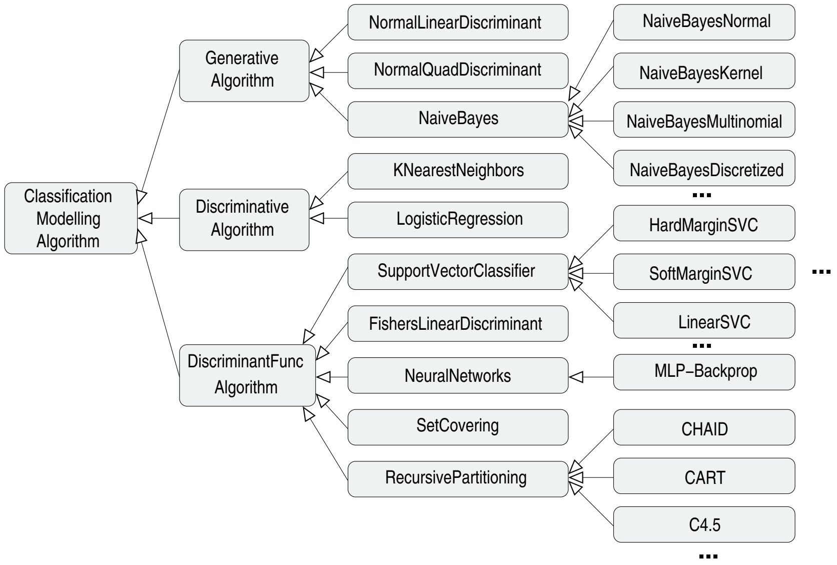


Fig. 7. The ClassificationModellingAlgorithm hierarchy. Only the primitive class hierarchy is shown.

compute posteriors $p(C_k|\mathbf{x})$ directly to determine class membership. *Discriminant functions* build a direct mapping $f(\mathbf{x})$ from input $\mathbf{x}$ onto a class label; neural networks and support vector classifiers (SVCs) are examples of discriminant function methods. These three Algorithm families spawn multiple levels of descendant classes that are distinguished by the type and structure of the models they generate; model structures will be discussed in Section 3.3.

In addition to these primitive classes that form a strict hierarchy (as shown in Figure 7), equivalent class definitions superpose a finer structure on the Algorithm subclasses. For instance, we can distinguish between eager and lazy learners based on whether they compress training data into ready-to-use models or simply store the training data, postponing all processing until a request for prediction is received [1]. Similarly, a classification algorithm can be classified as high-bias or high-variance based on how it tends to control the bias-variance trade-off in its learned models [28,21]. High-bias algorithms can only generate simple models that lack the flexibility to adapt to complex data distributions but for that reason remain stable across different training samples. High-variance algorithms span a broader range of complexity; they can generate highly complex but often unstable models: a slight change in the training sample can yield large changes in the learned models and their predictive behavior. Many other equivalent classes can be defined for modelling algorithms; as a result, algorithm instances can have multiple inheritance links (not shown in the figure) that make this concept hierarchy more of a directed acyclic graph than a simple tree structure.

3.3 Inside the Black Box: A Compositional View of DM Algorithms

As explained in Section 2, a key objective of the proposed meta-mining approach is to pry open the black box of DM algorithms in order to correlate observed behavior of learned models with both algorithm and data characteristics. This is a long-term, labor-intensive task that requires an in-depth analysis of the many data mining algorithms available. In this section, we illustrate our approach on two major data mining tasks, classification modelling and feature selection.

Classification Modelling Algorithms Opening the black box of a modelling or learning algorithm is equivalent to explaining, or describing the sources of, its inductive bias (Section 2.1). DMOP provides a unified framework for conceptualizing a learning algorithm's inductive bias by explicitly representing: 1) its underlying assumptions; 2) its hypothesis language or so-called representational bias through a detailed conceptualization of the class of models it generates; and 3) its preference or search bias through a definition of its underlying optimization problem and the optimization strategy adopted to solve it.

Representational bias and models As its name suggests, the keystone of a modelling algorithm is the **Model** that it was designed to produce (ModellingAlgorithm $\sqsubseteq$ $\exists$specifiesOutput.Model). A detailed characterization of a modelling algorithm's target model is the closest one can get to an actionable statement of its representational bias or hypothesis language. DMOP's characterization of ClassificationModel is summarized in Figure 8. To clarify how the model-related and other relevant concepts are used in describing a classification algorithm, we will use the linear soft-margin SVM classification modelling algorithm (henceforth LinSVC-A for the algorithm and LinSVC-M for the generated model) represented in Figure 9 as our running example.

A model is defined by two essential components: a model structure and a set of model parameters. The **ModelStructure** determines the three main classes of classification models (and hence of classification modelling algorithms). From the definitions given in Section 3.2, it follows that the model structure of a GenerativeModel is a JointProbabilityDistribution, while that of a DiscriminativeModel is a PosteriorProbabilityDistribution. By contrast, DiscriminantFunctionModels compute direct mappings of their inputs to a class label by summarizing the training data in a LogicalStructure (e.g., decision tree, rule set) or a MathematicalExpression (e.g., superposition of functions in neural networks, linear combination of kernels in SVMs). In LinSVC-M, where the kernel itself is linear, the linear combination of kernels is equivalent to a linear combination of features (Fig. 9).

The concept of **ModelParameter** is indissociable from that of ModelStructure. Within each model family, more specific subclasses and individual models are produced by instantiating the model structure with a set of parameters. Probabilistic — generative and discriminative — model structures are unambiguously

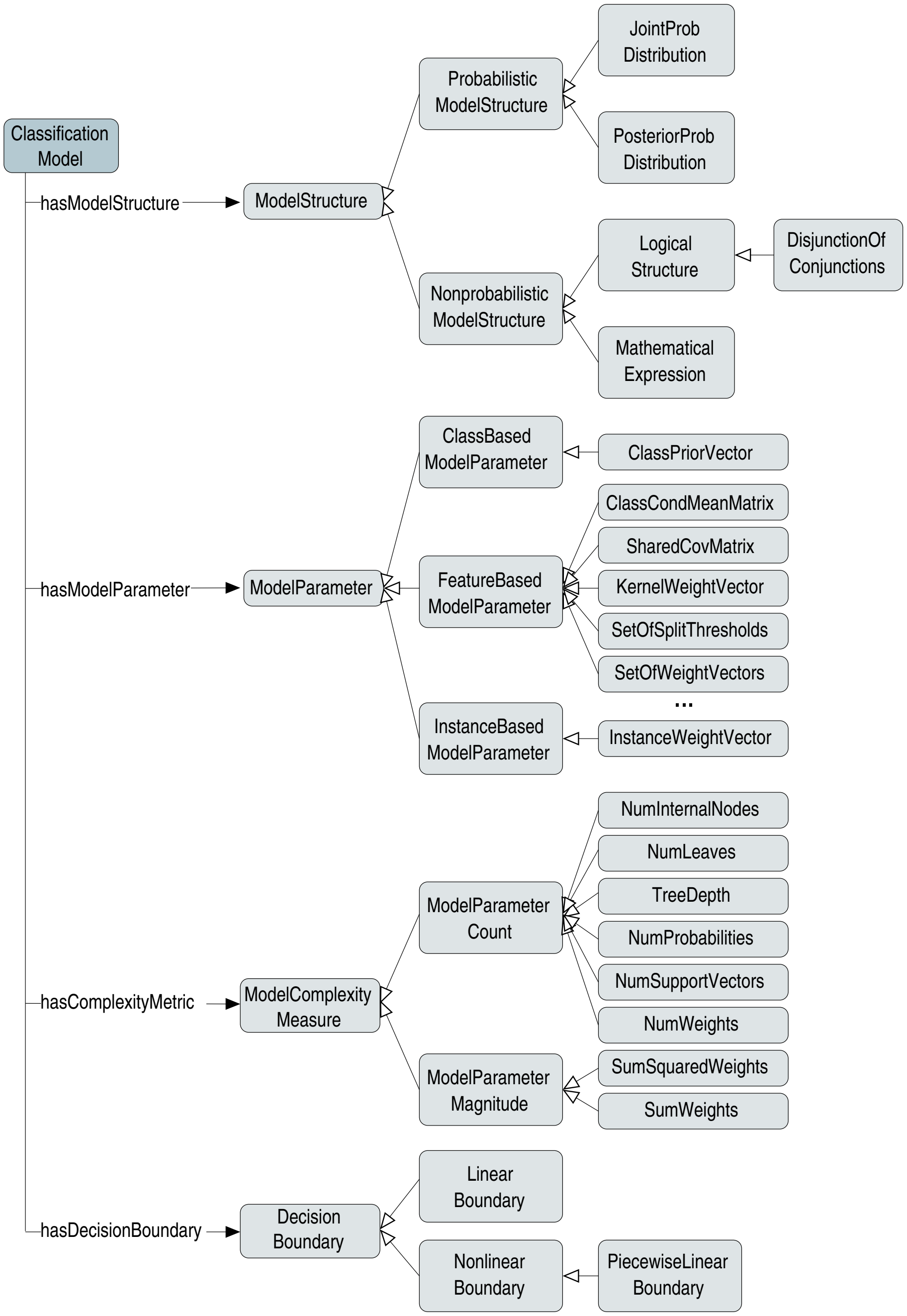


Fig. 8. DMOP's conceptualization of ClassificationModel

specified by the probability distribution that generated the training data. Since this distribution can never be identified with certainty from a finite random sample, the task is often simplified by assuming a family of distributions (e.g., Gaussian in NaiveBayesNormal) or a specific functional form (e.g., the logistic function in LogisticRegression); estimating the probability distribution is then reduced to estimating the values of the distribution parameters (e.g., mean and variance of a Gaussian) or the function parameters (e.g., weights of the linear combination in the logistic function's exponent). In DMOP, the concept of ProbabilisticModelStructure has a specific property, hasDensityEstimation, that identifies the parametric or non-parametric density estimation method used to estimate the model parameters. In non-probabilistic (discriminant function) models, the nature of the model parameters varies based on the type of model structure. In logical structures, which are more or less complex expressions based on the values of individual features, the model parameters are thresholds on feature values that partition the instance space into hyperrectangular decision regions. The natural model parameters of mathematical model structures are the values of the underlying function parameters, e.g. the weights of the hidden units in a neural network or the kernel coefficients in SVMs. In LinSVC-M (Fig. 9), the model parameters are the instance weights and the kernel weights which, as we have seen above, are those of the feature themselves.

In all cases, the number of model parameters confers on the selected model structure the degrees of freedom required to capture the characteristics of the target population. A model that has an inadequate set of parameters will underfit the data and incur systematic errors due to bias; on the other hand, a model with too many model parameters will adapt to chance variations in the sample, in short will overfit the training data and perform poorly on new data due to high variance. Selecting the right number of parameters is no other than selecting the right bias-variance tradeoff or selecting the appropriate capacity or level of complexity for a given model structure. The complexity of each learned model can be quantified using the concept of **ModelComplexityMeasure**, the most important subclass of which is ModelParameterCount. Its sibling, ModelParameterMagnitude, takes into account the view that a model's complexity is also determined by the magnitude of model parameter values [19,5]. The two complexity measures of LinSVC-M (Fig. 9) are NumberOfSupportVectors and SumOfSquaredWeights, subclasses of ModelParameterCount and ModelParameterMagnitude respectively. A final model descriptor is the type of **DecisionBoundary** that is drawn by a given model (family). DMOP's formalization of this concept distinguishes between linear and nonlinear decision boundaries, but more work is needed to develop a more elaborate geometry of decision regions.

Preference bias and optimization strategies Once a model structure and its set of parameters have been selected, the learning process is nothing more or less than the automated adjustment of these parameters to produce a fully specified, operational model. This is the task of the learning algorithm. The goal is to determine the set of parameter values that will maximize classification perfor-

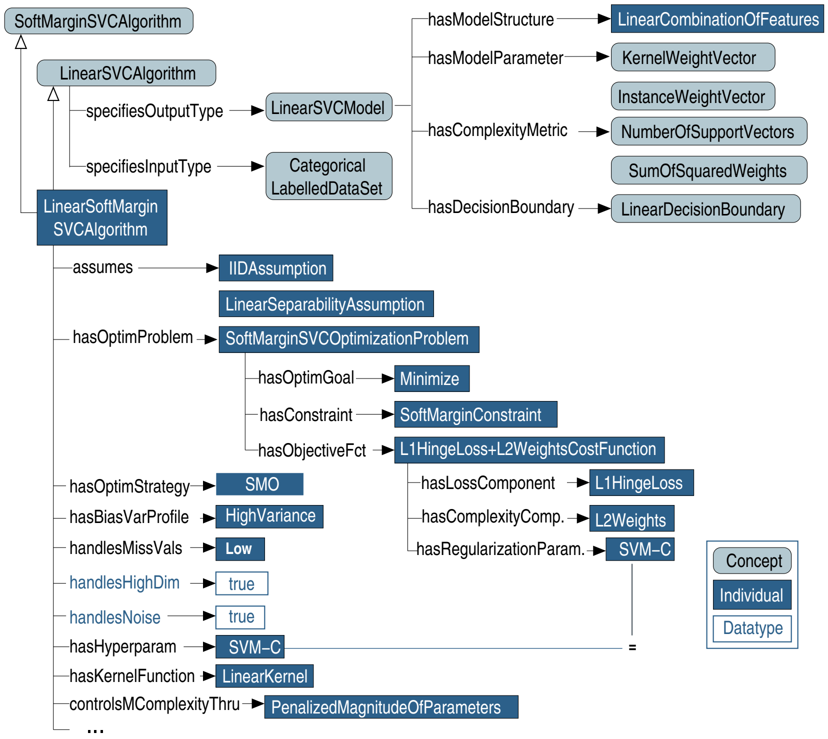


Fig. 9. Linear Soft Margin SVC Algorithm (referred to in the text as LinSVC-A) and the model it specifies (LinSVC-M in the text).

mance as gauged by some criterion. The search for the right parameter setting can be cast as an **OptimizationProblem** that consists in minimizing a cost (or objective) function, with or without a corresponding set of constraints. The cost function quantifies how close the current parameter values are to the optimum. Learning stops when the cost function is minimized. In its simplest version, the cost function is no more than a measure of error or loss (e.g. misclassification rate or sum of squared errors). However, minimizing training set error can lead to overfitting and generalization failure. For this reason many algorithms use a regularized cost function that trades off loss against model complexity. In DMOP, the generic form of the modelling CostFunction is $F = \epsilon + \lambda c$, where ϵ is a measure of loss, c is a measure of model complexity, and λ is a regularization parameter which controls the trade-off between loss and complexity. The optimization problem addressed by the LinSVC-A consists in minimizing the regularized cost function

$$\min_{\xi,\mathbf{w},b} \langle \mathbf{w}.\mathbf{w} \rangle + C \sum_{i=1}^{n} \xi_i^2$$

subject to the soft margin constraint $y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b) \geq 1 - \xi_i$, with $\xi_i \geq 0$, and $i = 1, ..., n$. The ontological representation of this optimization problem is shown (labelled SoftMarginSVCOptimizationProblem) in Figure 9.

DMOP incorporates a detailed hierarchy of strategies adapted to the optimization problems encountered in modelling and in other DM tasks (Fig. 10). These **OptimizationStrategies** fall into two broad categories—continuous and discrete—depending on the type of variables that define the problem. In certain cases, optimization is straightforward. This is the case of several generative algorithms like normal linear/quadratic discriminant analysis and Naive Bayes-Normal, where the cost function is the log likelihood, and the maximum likelihood estimates of the model parameters have a closed form solution. Logistic regression, on the other hand, estimates the maximum likelihood parameters using methods such as Newton-Raphson. In the case of LinSVC-A, the variables involved in the optimization problem defined above call for a continuous optimization strategy. LinSVC-A uses Sequential Minimal Optimization (SMO), a quadratic programming method rendered necessary by the quadratic complexity component of the cost function (L_2 norm of Weights in Fig. 9).

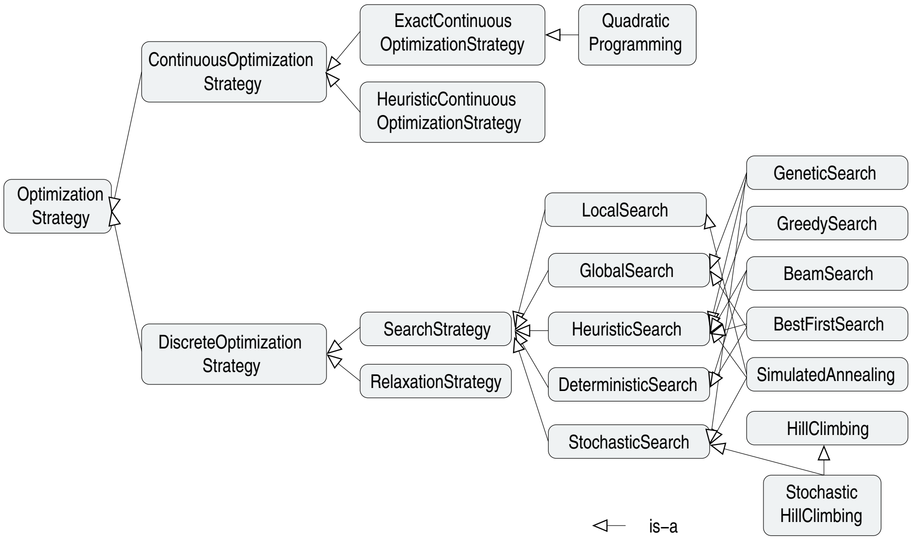


Fig. 10. The OptimizationStrategy hierarchy

The optimization strategy hierarchy plays an important role in DMOP because many core DM tasks other than modelling also have underlying optimization problems. In particular, discrete optimization strategies will come to the fore in feature selection methods.

Feature Selection Algorithms Feature selection is a particular case of dimensionality reduction, which can be defined as follows: given a set of n vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^p$, find a set of lower-dimensional vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{p'}$, $p' < p$, that maximally preserves the information in the data according to some criterion. In a classification task, for instance, the criterion could be some measure of how well the p or $p\prime$ features discriminate between the different classes. Feature selection refers to the specific case where the p' features are a subset of the original p features; dimensionality is reduced by eliminating irrelevant or redundant features. Alternatively, new features can constructed from the original ones via techniques like principal components analysis, and feature selection applied to the set of derived features; this process—feature construction followed by feature selection—is called feature extraction in [33] and in DMOP.

In feature selection, every subset of the original p-dimensional feature vector is represented by a vector $\sigma \in \{0,1\}^p$ of indicator variables, where $\sigma_i = 1$ denotes the presence and $\sigma_i = 0$ the absence of feature i. The task is to find a vector $\sigma^* \in \{0,1\}^p | \forall \sigma', \, f(\sigma^*) \leq f(\sigma')$, where f is some measure of feature set quality. A feature selection algorithm can be described by four properties: its mode of interaction with the learning algorithm, an optimization strategy to guide search in the space of feature subsets, a feature scoring or weighting mechanism to assess the candidate subsets, and a decision strategy to make the final selection.

Interaction with the learner Feature selection methods are commonly classified based on how they are coupled with the learning algorithm. Filter methods perform feature selection as a preprocessing step, independently of the learning method; they must then use learner-independent relevance criteria to evaluate the candidate features, either individually or as subsets of the initial feature set. Wrapper methods wrap feature selection around the learning process and use the estimated performance of the learned model as the selection criterion; the effectiveness of the selected features depends strongly on the specific learning method used. In embedded methods, feature selection is encoded as an integral part of the learning algorithm.

Optimization strategies Feature selection implies extensive search in the discrete space of feature subsets; there are 2^p ways of assigning values to the p-dimensional vector σ, in other words 2^p possible subsets of the initial feature set. Feature selection methods can adopt one of two optimization strategies to solve this kind of problem: SearchStrategy and RelaxationStrategy. Search strategies are based on the combinatorial approach that is a more natural approach to problems in discrete domains, while relaxation strategies relax, as it were, the discreteness constraint and reformulate the problem in a continuous space. The result is then reconverted via a decision rule into a final selection in discrete feature space. Figure 10 shows the two main types of DiscreteOptimizationStrategy. Search strategies, in particular heuristic search strategies that trade off optimality for efficiency or simple feasibility, are by far the most widely used. The subclasses of SearchStrategy are determined by the different properties of search as shown in Figure 11) : its coverage (global or local), its search direction (e.g., forward, backward), its choice policy or what Pearl calls "recovery of pursuit" [54] (irrevocable or tentative), the amount of state knowledge that guides search (blind, informed), and its level of uncertainty (deterministic, stochastic). These properties are For instance, Consistency-based feature selection [49] uses the so-called Las Vegas strategy which is an instance of StochasticHillClimbing, which combines local, greedy, stochastic search. Correlation-based feature selection [35] adopts a forward-selection variant of (non-greedy) BestFirstSearch. Representing the irrevocable choice policy of GreedySearch, C4.5's embedded feature selection algorithm and SVM-RFE [34] use GreedyForwardSelection and GreedyBackwardElimination respectively. The concept RelaxationStrategy has no descendants in the graph because after transposing the discrete problem into a continuous space, on can use any instance of ContinuousOptimizationStrategy. However, most of the feature selection algorithms that use relaxation further simplify the problem by assuming feature independence, reducing the combinatorial problem to that of weighting the p individual features and (implicitly) selecting a subset composed of the top p' features. This is the case of all so-called univariate methods, such as InfoGain, χ^2 and SymmetricalUncertainty (see Figure 12), as well as a few multivariate methods like ReliefF [45,64] and SVMOne. ReliefF solves the continuous problem similarly to univariate methods because it also weights individual features, though in a multivariate context. On the contrary, SVMOne and SVM-RFE use the continuous optimization strategy of the learner in which they are embedded — SMO, which, as we saw above is an instance of QuadraticProgramming. Finally, note the special case of SVM-RFE which actually combines the two discrete optimization strategies: it generates candidate subsets through greedy backward elimination in discrete space, then uses the SVM learner to weight the individual features in continuous space, and finally returns to discrete space by generating a new subset purged of the n features with the lowest weights. This cycle continues until there are no more features to eliminate.

Feature/subset weighting schemes Another characteristic of a feature selection algorithm is its feature weighting scheme. Feature weighting algorithms are divided into two groups based on what is being weighted (hasEvaluationTarget property in Fig. 11): individual features or feature subsets. Single-feature weighting algorithms themselves can be classified as univariate or multivariate depending on the feature context that is brought to bear in weighting the individual feature: univariate algorithms (e.g., those that use information gain or χ^2) weight individual features in isolation from the others, while multivariate algorithms weight individual features in the context of all the others. For instance, ReliefF and SVMOne yield individual feature weights that are determined by taking all the other features into account — when computing nearest neighbors in the case of ReliefF, and in building the linear combination of features or kernels in the case of SVMOne. Finally, feature weighting algorithms are completely specified by adding the evaluation function they use – either individual feature or feature subset weighting algorithms.

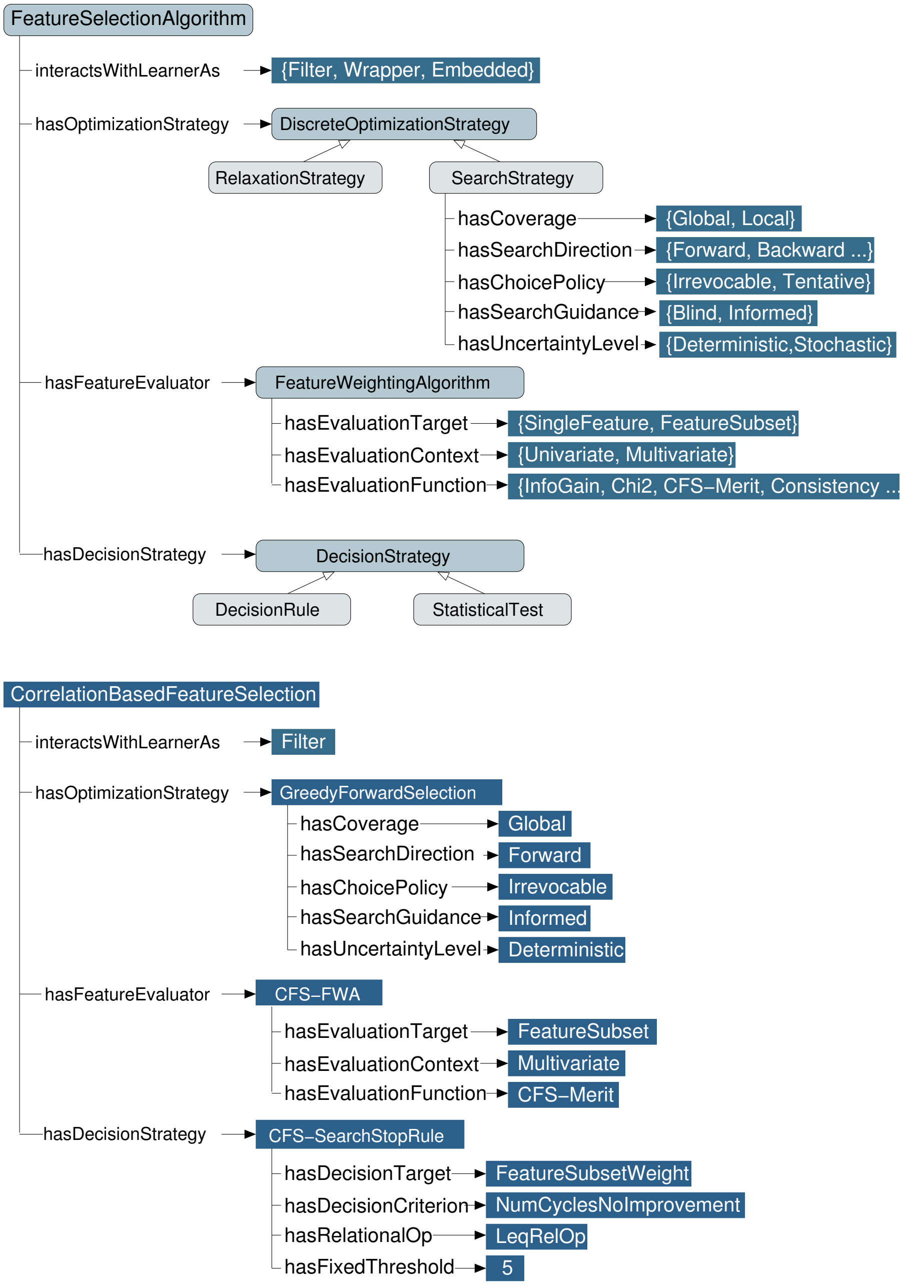
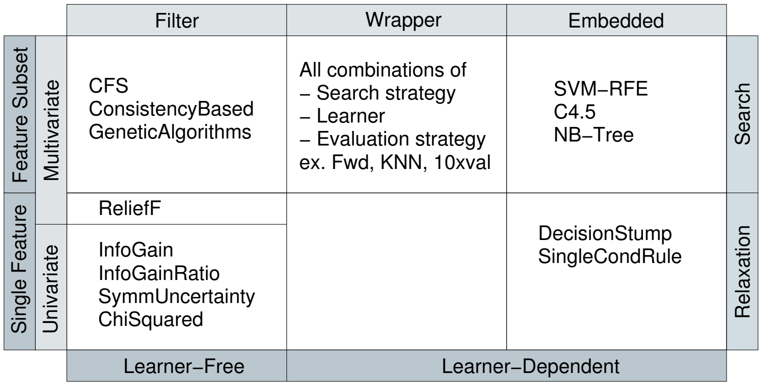


Fig. 11. Algorithms and strategies for feature selection. The upper part of the figure shows the links between the major entities involved: an optimization strategy, a feature weighting algorithm and a decision strategy. The lower part illustrates the use of these concepts in describing Correlation-based Feature Selection.

Decision strategy Once the candidate features or feature subsets have been generated and scored, a feature selection algorithm uses a decision strategy to select the fnal feature subset. This can be a statistical test that uses the resulting p-value as a basis for selection, or any kind of decision rule that sets a threshold on any quantity that describes the evaluated entities, e.g., the weights of the features or subsets, or their ranks.

Figure 12 situates a number of feature selection algorithms according to their characteristics and those of their feature weighting components.



		Filter	Wrapper	Embedded	
Feature Subset	Multivariate	CFS ConsistencyBased GeneticAlgorithms	All combinations of – Search strategy – Learner – Evaluation strategy ex. Fwd, KNN, 10xval	SVM−RFE C4.5 NB−Tree	Search
Single Feature		ReliefF		DecisionStump SingleCondRule	Relaxation
	Univariate	InfoGain InfoGainRatio SymmUncertainty ChiSquared			
		Learner−Free	Learner−Dependent		

Fig. 12. Synoptic view of feature selection methods based on their interaction with the learning algorithm (learner-free=filter, learner-dependent=wrapper, embedded), the optimization strategy used (search, relaxation), and their feature weighting component's evaluation target (single feature, feature subset) and evaluation context (univariate, multivariate).

4 DMOP-based pattern discovery from DM workflows

Data mining workflows are replete with structures that are often reused. A simple example is the workflow segment where the operator Weight by Information Gain is invariably followed by Select by Weights to perform feature selection. This regularity involves individual operators, but it would be even more useful if we could detect the same basic structure had the first operator been replaced by any other that does univariate feature weighting. Similarly, bagging subworkflows should be recognizable despite the diversity of classification modelling operators used. In order to detect patterns with strong support, a frequent pattern search procedure should be capable of generalizing from specific operators to broad algorithm classes. This is one of the roles of the DMOP ontology, where we can follow the executes link from grounded operations to operators, then the implements link from operators to algorithms (Figure 4) in order to analyse the taxonomic (as in Figure 7) and non-taxonomic commonalities between algorithms. In short, prior knowledge modelled in DMOP will support the search for generalized workflow patterns, similar to the generalized sequence patterns extracted via frequent sequence mining in [70].

4.1 Workflow representation for generalized pattern mining

Workflows as hierarchical graphs Data mining workflows are directed acyclic graphs (DAGs), in which nodes correspond to operators and edges between nodes to input/output (I/O) objects, much like the "schemes" described in [40,31]. More precisely, they are hierarchical DAGs, since nodes can represent composite operators (e.g. cross-validation) that are themselves workflows. An example hierarchical DAG representing a RapidMiner workflow is given in Figure 13. The workflow cross-validates feature selection followed by classification model building. X-Validation is a typical example of a composite operator which itself is a workflow. It has two basic blocks, a *training block* which can be any arbitrary workflow that receives as input a dataset and outputs a model, and a *testing block* which receives as input a model and a dataset, and outputs a performance measure. In this specific CV operator, the training block has three steps: computation of feature weights by the Weight by Information Gain operator, selection of a subset of features by the Select by Weights operator, and final model building by the Naive Bayes operator. The testing block consists simply of the Apply Model operator followed by the Compute Performance computation.

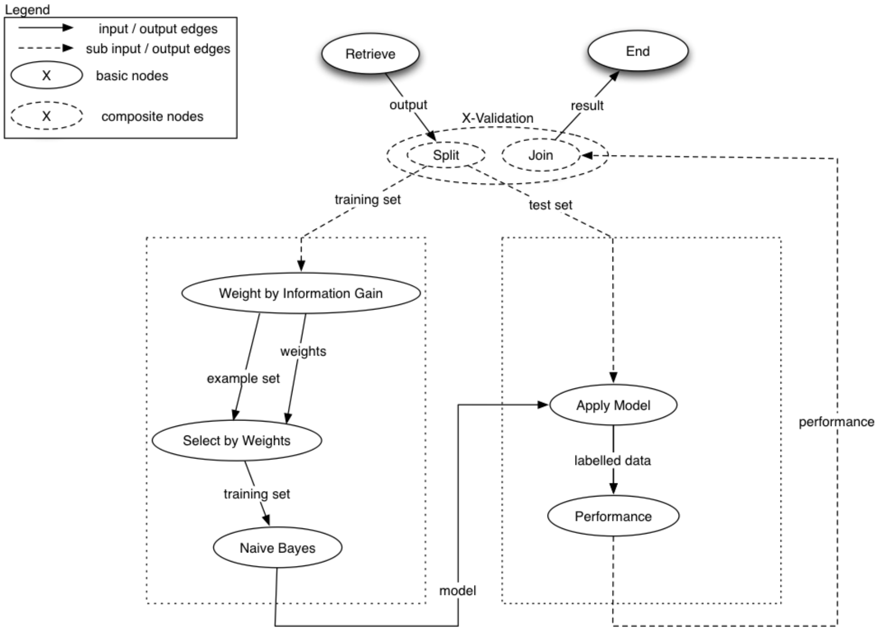


Fig. 13. A DM workflow as a hierarchical DAG

We now give a more formal definition of the hierarchical DAGs that we will use to describe data mining workflows. Let:

- O be the set of all available operators that can appear in a data mining workflow, e.g. classification operators, such as C4.5, SVMs, model combination operators, such as boosting, etc.
- E be the set of all available data types that can appear in a data mining workflow, e.g. the data types of the various I/O objects of some DM workflow, models, datasets, attributes, etc.
- an operator $o \in O$ be defined by its name and the data types of its inputs and outputs. 1

A hierarchical directed acyclic graph, G, that represents a data mining workflow is an ordered pair (O', E') where:

- $O' \subseteq O$ is the set of vertices or nodes that correspond to the operators used in the workflow
- $E' \subseteq E$ is the set of ordered pairs of nodes, (o_i, o_j), called directed edges, that correspond to the data types of the I/O objects, that are passed from operator o_i to operator o_j in the workflow.

E' defines the *data-flow* of the workflow and O' the *control flow.*

Workflows as parse trees A DAG has one or more topological sorts. A *topological sort* is a permutation p of the vertices of a DAG such that an edge (o_i, o_j) indicates that o_i appears before o_j in p [65]. Thus, it is a complete ordering of the nodes of a DAG. If a topological sort has the property that all pairs of consecutive vertices in the sorted order are connected by an edge, then these edges form a directed Hamiltonian path of the DAG. In this case, the topological order is unique. If not, then it is always possible to get the unique topological order by adding a second order such as the lexicographic order of the vertex labels. The topological sort of a DAG can be represented by a parse tree, which is a reduction of the original DAG where the edges have been fully ordered.

The parse tree of Figure 14 gives the topological sort of the DM workflow represented as a hierarchical DAG in Figure 13. As seen clearly, the parse tree is a simplification of the original graph; it represents the order of execution of the different operators and their hierarchical relation but the data-flow is lost (the edges are not labelled).

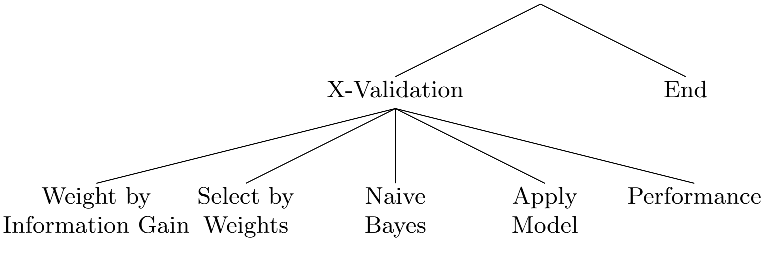


Fig. 14. The parse tree (topological sort) of the DM-workflow given in Figure 13

Augmenting the parse trees Given the parse tree representation of a workflow, the next step is to augment it in view of deriving frequent patterns over generalizations of the workflow components. Generalizations will be based on concepts, relations and subsumptions modelled in DMOP. Starting from the *Operator* level, an operator $o \in O$ *implements* some algorithm $a \in A$ (Figure 4). In addition the DMOP defines a refined algorithm taxonomy, an extract of which is given in Figure 7. Note that contrary to the asserted taxonomy which is a pure tree, the inferred taxonomy can be a DAG (a concept can have multiple ancestors) [60]; consequently the subsumption order is not unique. For this reason we define a distance measure between two concepts C and D, which is related to the terminological axiom of *inclusion*, $\mathcal{C} \sqsubseteq \mathcal{D}$, as the length of the shortest path between the two concepts. This measure will be used to order the subsumptions. For the sake of clarity, we will assume a single-inheritance hierarchy in the example of the (RapidMiner) NaiveBayes operator. Given the taxonomic relations NaiveBayesNormal $\sqsubseteq$ NaiveBayesAlgorithm $\sqsubseteq$ BayesianAlgorithm $\sqsubseteq$ GenerativeAlgorithm, the reasoner infers that NaiveBayes implements someInstance of these superclasses, ordered using the distance measure described. Based on these inferences, an *augmented parse tree* is derived from an original parse tree T by inserting the ordered concept subsumptions between each node $v \in T$ and its parent node. Figure 15 shows the augmented version of the parse tree in Figure 14.

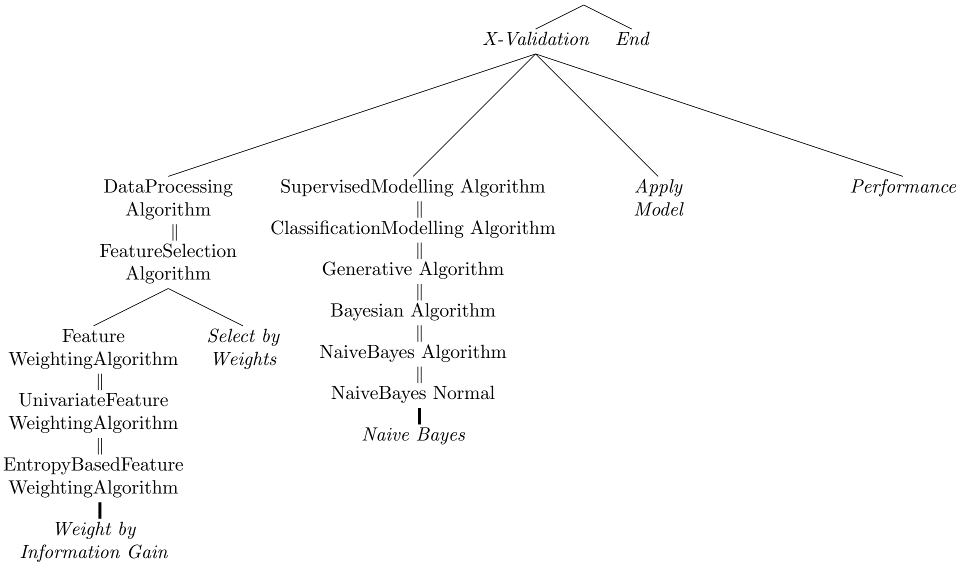

X-Validation
End
DataProcessing Algorithm
FeatureSelection Algorithm
Feature WeightingAlgorithm
UnivariateFeature WeightingAlgorithm
EntropyBasedFeature WeightingAlgorithm
Weight by Information Gain
Select by Weights
SupervisedModelling Algorithm
ClassificationModelling Algorithm
Generative Algorithm
Bayesian Algorithm
NaiveBayes Algorithm
NaiveBayes Normal
Naive Bayes
Apply Model
Performance

Fig. 15. An augmented parse tree. Thin edges depict task decomposition into operators (italics); a thick single line indicates that an operator implements an instance of its parent algorithm; double lines depict subsumption.

4.2 Mining generalized workflow patterns

We are now ready to mine generalized patterns from DM workflows represented as (augmented) parse trees, which we now define more formally. A parse tree is a rooted k-tree [15]. A *rooted* k-tree is a set of k nodes $O' \subseteq O$ where each $o \in O'$, except one called root, has a parent denoted by $\pi(o) \in O'$. The function $l(o)$ returns the label of a node, and the operator $\prec$ denotes the order from left to right among the children of a node.

Induced subtrees We used Zaki et al.'s TreeMiner [81] to search for frequent induced subtrees over the augmented tree representation of workflows. A tree $t\acute{} = (O_{t\acute{}}, E_{t\acute{}})$ is called an induced subtree of $t = (O_t, E_t)$, noted $t' \preceq_i t$, if and only if $O_{t'}$ preserves the direct parent-child relation of O_t. Figure 16 shows a tree T1 and two of its potential induced subtrees, T2 and T3. In the less constraining case where only an indirect ancestor-descendant relation is preserved, the subtree t is called embedded. We had no need for embedded trees: given the way augmented parse trees were built using the DMOP algorithm taxonomy, extending parent-child relationships to ancestor-descendants would only result in semantically redundant patterns with no higher support.

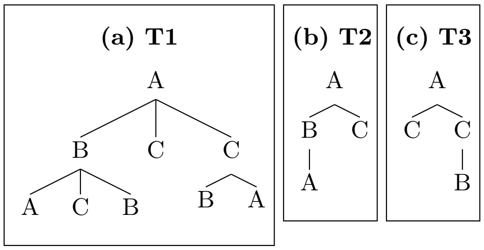
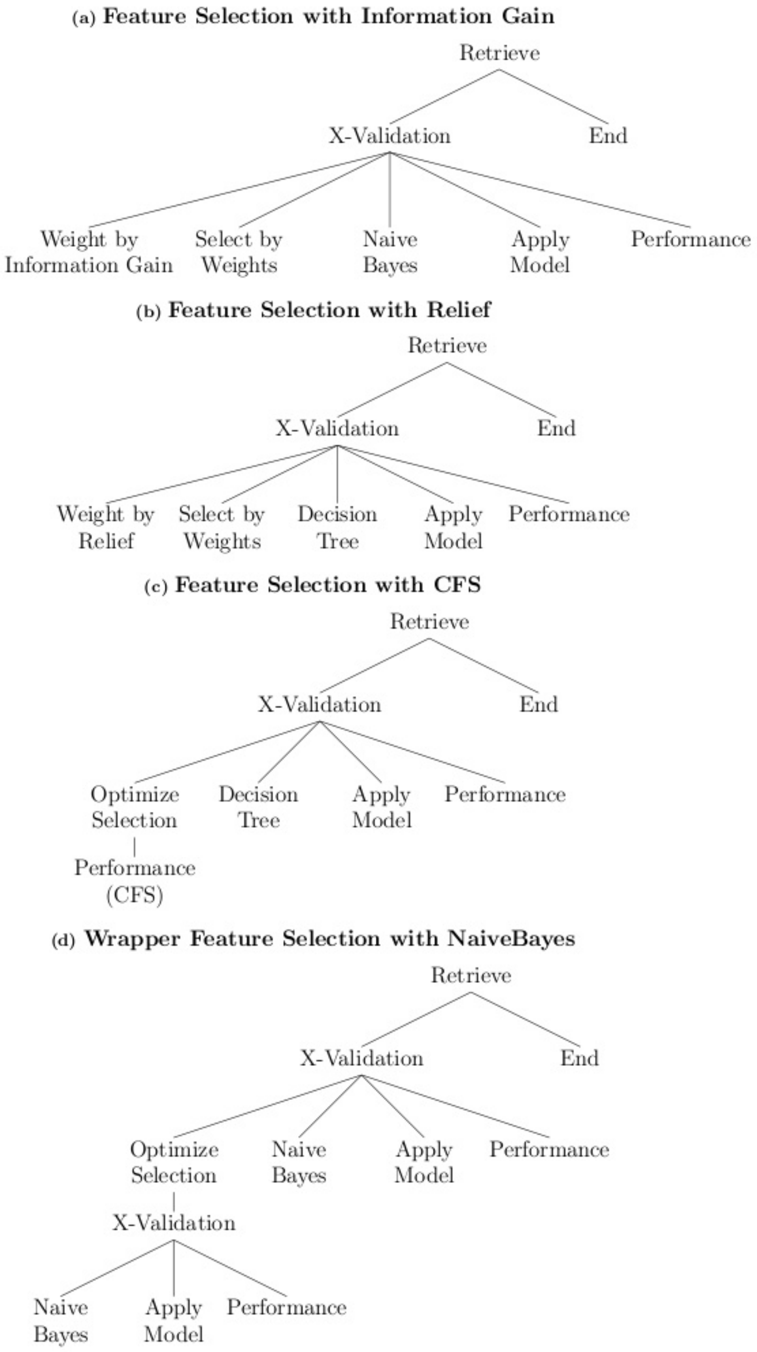


Fig. 16. A tree T1 and two of its induced subtrees ,T2 and T3

Given a database (forest) D of trees, the tree miner algorithm will produce a set $\mathcal{P}$ of induced subtrees (patterns). For a given tree $T_i \in D$ and a pattern $S \in \mathcal{P}$, if $S \preceq_i T_i$, we say that T_i *contains* S or S *occurs* in T_i. Now let $\delta_{T_i}(S)$ denote the number of occurences of the subtree $S \in \mathcal{P}$ in a tree $T_i \in D$, and let d_{T_i} be an indicator variable with $d_{T_i}(S) = 1$ if $\delta_{T_i}(S) > 0$ and $d_{T_i}(S) = 0$ if $\delta_{T_i}(S) = 0$. The *support* of the subtree S in the database D is defined as $sup(S) = \Sigma_{T_i \in D} d_{T_i}(S)$. We call the *support set* of S the set of trees $T_i \in D$ with $d_{T_i}(S) > 0$.

An example We demonstrate frequent pattern extraction from the following workflows that do cross-validated feature selection and classification:



(a) **Feature Selection with Information Gain**

(b) **Feature Selection with Relief**

(c) **Feature Selection with CFS**

(d) **Wrapper Feature Selection with NaiveBayes**

Fig. 17. Parse trees of feature selection/classification workflows

a) feature selection based on Information Gain, classification with Naive Bayes
b) feature selection with Relief, classification with C4.5
c) feature selection with CFS, classification with C4.5
d) wrapper feature selection with Naive Bayes, classification with Naive Bayes.

Their parse trees are given in Figure 17. Workflow a) performs univariate feature selection based on a univariate weighting algorithm. The three remaining workflows are all doing multivariate feature selection, where in b) this is done using a multivariate feature weighting algorithm, and in c) and d) using heuristic search, implemented by the *OptimizeSelection* operator, in the space of feature sets where the cost function used to guide the search is CFS and the Naive Bayes accuracy respectively.

We applied TreeMiner [81] to the augmented version of these parse trees, setting the minimum support to 2 in order to extract frequent induced subtrees. Some of the mined patterns and their support sets are shown in Figure 18.

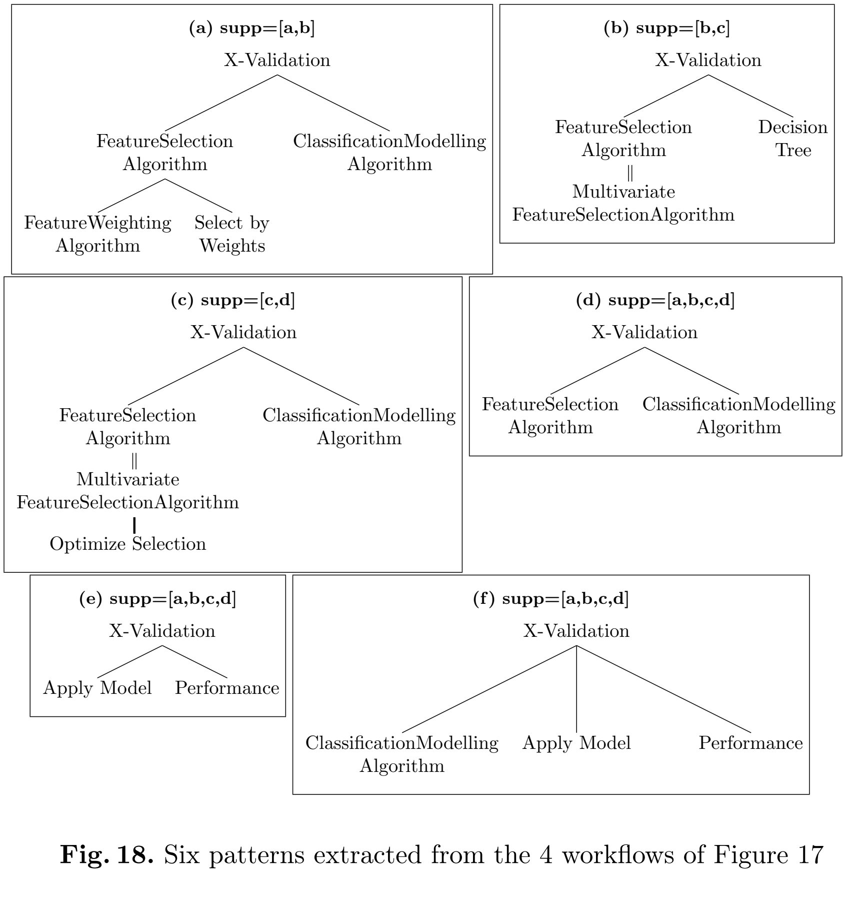

(a) supp=[a,b]
X-Validation
FeatureSelection Algorithm
ClassificationModelling Algorithm
FeatureWeighting Algorithm
Select by Weights
(b) supp=[b,c]
X-Validation
FeatureSelection Algorithm
Decision Tree
Multivariate FeatureSelectionAlgorithm
(c) supp=[c,d]
X-Validation
FeatureSelection Algorithm
ClassificationModelling Algorithm
Multivariate FeatureSelectionAlgorithm
Optimize Selection
(d) supp=[a,b,c,d]
X-Validation
FeatureSelection Algorithm
ClassificationModelling Algorithm
(e) supp=[a,b,c,d]
X-Validation
Apply Model
Performance
(f) supp=[a,b,c,d]
X-Validation
ClassificationModelling Algorithm
Apply Model
Performance

Fig. 18. Six patterns extracted from the 4 workflows of Figure 17

Pattern (a) shows that in two of the four workflows, a) and b), a feature weighting algorithm is followed by the Select by Weights operator, and that this pair forms a feature selection algorithm followed by a classification modelling algorithm, nested all together in a cross-validation operator. Pattern (b) captures the fact that two workflows, b) and c), contain a multivariate feature selection followed by a decision tree algorithm, again nested inside a cross-validation.

Pattern (c) corresponds to MultivariateFeatureSelection performed by OptimizeSelection and followed by some classification algorithm. As mentioned above, OptimizeSelection represents a heuristic search over feature sets using some search strategy and some cost function which are not specified for the moment.

Pattern (d) is a generalization of patterns (a), (b) and (c), and covers all four workflows. It simply says that a feature selection algorithm is followed by a classification modelling algorithm.

Finally, patterns (e) and (f) also cover all four workflows. Pattern (e) corresponds to the validation step where a learned model is applied to a test set using cross-validation, and some performance measure is produced. Pattern (f) is a super pattern of pattern (e) and shows that a model should first be produced by a classification modelling algorithm before it can be applied and evaluated.

5 Experiments in workflow meta-mining

This section describes workflow mining experiments in which the goal is to predict the relative performance of a new workflow, whether handcrafted by the user or designed automatically, e.g. by eProPlan (page 7).

5.1 Experimental design

As discussed in Section 2.1, standard meta-learning has been dominated by the Rice model which considers only data set characteristics to predict the performance of algorithms or algorithm families. We proposed an alternative model which takes into account both data and algorithm characteristics. In this section, we apply the revised Rice model to workflow mining: in the meta-mining view of workflows as compositions of (implemented) algorithms, workflow selection or ranking is grounded on both data and workflow characteristics.

Gathering the meta-mining data To define meta-learning problems and gather the necessary meta-data, we need to collect results from a sizeable number domain-level data mining experiments. We gathered 65 datasets concerning microarray experiments on different types of cancer. Table 1 gives the name (prefixed by the cancer type) and the number of examples, features, and classes for each dataset. As is typical of gene profiling data, all are high-dimensional small samples, i.e. the number of features is several orders of magnitude higher than the number of examples.

The choice of a clearly circumscribed application domain for our meta-learning experiments has a clear rationale. Previous work on meta-learning typically relied on base-level experiments using UCI datasets [26] from the most diverse domains. As a result, the meta-learner groped for regularities in the intractable immensity and sparsity of the space of all possible datasets—from classical toy problems (e.g. Lenses, Tic-tac-toe) to more recent biomedical datasets (e.g. Dorothea, Arcene), where the number of dimensions is one or several orders of magnitude greater than the number of instances. Initially motivated by user rather than

Dataset	N	D	C	Dataset	N	D	C
adrenal_dahia	76	22283	2	leukemia_haslinger	100	12600	2
bladder_blaveri	40	5331	2	leukemia_wei	29	21481	2
bladder_dyrskjot	40	4409	3	leukemia_yagi	53	7241	2
bladder_sanchez-carbayo	157	22283	3	liver_chen	156	2621	2
breast_desmedt	198	22283	2	liver_iizuka	60	7129	2
breast_farmer	49	22215	3	liver_ye	87	8121	3
breast_gruvberger	58	3389	2	lung_barret	54	22283	2
breast_kreike	59	17291	2	lung_beer	86	7129	3
breast_ma_2	60	22575	2	lung_bhattacharjee_2	197	12600	4
breast_minn	68	22283	2	lung_bild	111	54675	2
breast_perou	65	7261	4	lung_wigle	39	1971	2
breast_sharma	60	1368	2	lymphoma_alizadeh	99	8580	2
breast_sotiriou	167	22283	3	lymphoma_booman	36	14362	2
breast_veer	97	24481	2	lymphoma_rosenwald	240	7388	3
breast_wang	286	22283	2	lymphoma_shipp	77	7129	2
breast_west	49	7129	2	medulloblastoma_macdonald	23	1098	2
cervical_wong	33	10692	2	melanoma_talantov	70	22283	3
cns_pomeroy_2	60	7129	2	mixed_chowdary	104	22281	2
colon_alon	62	2000	2	mixed_ramaswamy	76	15539	2
colon_laiho	37	22283	2	myeloma_tian	173	12625	2
colon_lin_1	55	16041	2	oral_odonnell	27	22283	2
colon_watanabe	84	54675	2	ovarian_gilks	23	36534	2
gastric_hippo	30	7127	2	ovarian_jazaeri_3	61	6445	2
glioma_freije	85	22645	2	ovarian_li_and_campbell	54	1536	2
glioma_nutt	50	12625	2	ovarian_schwartz	113	7069	5
glioma_phillips	100	22645	2	pancreas_ishikawa	49	22645	2
glioma_rickman	40	7069	2	prostate_singh	102	12600	2
head_neck_chung	47	9894	2	prostate_tomlins	83	10434	4
headneck_pyeon_2	42	54675	2	prostate_true_2	31	12783	2
leukemia_armstrong	72	12582	3	renal_williams	27	17776	2
leukemia_bullinger_2	116	7983	2	sarcoma_detwiller	54	22283	2
leukemia_golub	72	7129	2	srbct_khan	88	2308	4
leukemia_gutierrez	56	22283	4				

Table 1. The 65 microarray datasets used in the meta-mining experiments. N: number of examples, D: number of features, C: number of classes

meta-learner considerations, so-called third-generation data mining systems [58] promoted the idea of vertical systems which focus on a specific application domain and problem, thus ensuring a more dense and coherent search space as well as the possibility of bringing domain-specific knowledge to bear in the knowledge discovery process. In this spirit, we selected gene expression-based cancer diagnosis as our problem domain, with the explicit proviso that all conclusions drawn from these experiments will apply only to datasets that stem from the same application area or at least share their essential characteristics.

We applied different data mining workflows to these datasets and estimated their performance using ten fold cross-validation. All the workflows were combinations of feature selection and classification algorithms. We used the following feature selection algorithms: Information Gain (IG), Chi-square (CHI), ReliefF (RF), and recursive feature elimination with SVM (SVMRFE); we fixed the number of selected features to ten. For classification we used the following algorithms: one-nearest-neighbor (1NN), decision tree algorithms J48 and CART, Naive Bayes (NB), logistic regression algorithm (LR), and SVMs with linear (SVM-L) and Gaussian (SVM-RBF) kernels. For J48 the C (pruning confidence) and M (minimum number of instances per leaf) parameters were set to 0.25 and 2 respectively; for CART the M and N (number of folds for the minimal cost-complexity pruning) parameters were set to 2 and 5 respectively. The C parameter was set to 1 for both SVM-L and SVM-R, and SVM-R's γ parameter was set to 0.1. We used the implementations of these algorithms in the RapidMiner data mining suite. All the possible combinations of the four feature selection algoriths with the seven classication algorithms gave 28 different learning workflows, each applied to the 65 datasets, for a total of 1820 data mining experiments.

Predicting the performance of a candidate workflow was cast as a classification problem: given a dataset d_j, determine whether workflow wf_i will be among the top performing workflows (class *best*) or not (class *rest*). We assigned these class labels as follows. For each dataset we did a pairwise comparison of the estimated performance of the 28 workflows applied to it using a McNemar's test of statistical significance. For each workflow pair, a score of 1 was assigned to the workflow—if any—which performed significantly better than the other, which scored 0; otherwise both were assigned 0.5. The final performance rank of a workflow for a given dataset was determined by the sum of points it scored on these pairwise comparisons for that dataset. Clearly in the case of 28 workflows the maximum possible score is 27 when a workflow is significantly better than all the other workflows. If there are no significant differences then each workflow gets a score of 13.5. The class label of a workflow for a given dataset was determined based on its score; workflows whose scores were within 1.5 standard deviations of the best performance measure for that dataset were labelled *best* and the remaining workflows *rest*. Under this choice 45% of the experiments, i.e. (d_j, wf_i) pairs, were assigned the label *best* and the remaining 55% the *rest* label.

Representing the meta-data As explained earlier in this section, we used a combination of dataset and workflow characteristics to describe the meta-learning examples.

Data descriptors We took 6 dataset characteristics from the StatLog and METAL projects: class entropy, average feature entropy, average mutual information, noise-signal ratio, outlier measure of continuous features, and proportion of continuous features with outliers. Because our base-level datasets contained only continuous predictive features, average feature entropy and average mutual information were computed via a binary split on the range of continuous feature values, as is done in C4.5 [59]. Detailed descriptions of these data characteristics are given in [50,41].

In addition, we used 12 geometric data complexity measures from [38]. These can be grouped into three categories: (1) measures of overlaps in feature values from different classes (maximum Fisher's discriminant ratio, volume of overlap region, maximum feature efficiency); (2) measures of class separability (fraction of instances on class boundary, ratio of average intra/inter-class distance, and landmarker-type measures like error rates of 1-NN and a linear classifier on the dataset; (3) measures of geometry, topology, and density of manifolds (nonlinearity of linear classifier, nonlinearity of 1NN classifier, fraction of points with retained adherence subsets, and average number of points per dimension). The definitions of these measures, their rationale and formulas, are given in [38].

Workflow descriptors Workflow descriptors were constructed in several steps following the pattern discovery method described in Section 4:

1. We built parse trees (Section 4.1) from the 28 workflows and augmented them using concept subsumptions from the DMOP ontology (Section 4.1); we thus obtained 456 augmented parse trees such as that shown in Figure 15.
2. We applied the TreeMiner algorithm with a minimum support of 3% to the augmented parse trees, thereby extracting 3843 frequent patterns defined as induced subtrees (Section 4.2).
3. We ordered the extracted patterns in order of decreasing generality, then pruned this initial pattern set to retain only closed patterns, i.e. patterns that are maximal with respect to the subsumption ordering of an equivalence class of patterns having the same support set [4]. The final set contained 1051 closed workflow patterns similar to those in Figure 18. In a nutshell, a workflow pattern is simply a fragment of an augmented (workflow) parse tree that has a support above a predefined threshold.
4. Finally, we converted each workflow pattern into a binary feature whose value equals 1 if the given workflow contains the pattern and 0 otherwise; it is these boolean features that we call workflow descriptors. Thus each workflow was represented as a vector of 1051 boolean workflow descriptors. Essentially, what we did was propositionalize the graph structure of the DM workflows.

5.2 Experimental results

We defined two meta-mining scenarios. Scenario A relies mainly on the dataset characteristics to predict performance, while scenario B considers both dataset and workflow characteristics.

Meta-mining scenario A In this scenario we create one meta-mining problem per workflow, producing 28 different problems. We denote by WF this set of meta-mining problems and by WF_i the meta-mining problem associated with workflow wf_i. For each meta-mining problem WF_i, the goal is to build a model that will predict the performance of workflow wf_i on some dataset. Under this formulation each meta-mining problem consists of $|D| = 65$ learning instances, one for each of the datasets in Table 1; the features of these instances are the dataset descriptors. The class label for each dataset d_j is either *best* or *rest*, based on the score of workflow wf_i on dataset d_j as described on page 30.

An issue that arises is how to measure the error for a specific dataset, which is associated with 28 different predictions. One option is to count an error whenever the *set* of workflows that are predicted as *best* is not a subset of the truly best workflow set. This error definition is more appropriate for the task at hand, where the goal is to recommend workflows that are expected to perform best; it is less important to miss some of them (false negatives) than to recommend workflows that will actually underperform (false positives). Here we adopted the simple approach of counting an error whenever the prediction does not match the class label, regardless of the direction of the error. With this method the overall error averaged over the 65 datasets is equal to the average error over the 28 different meta-mining problems WF_i. We denote this error estimate by

$$A_{d_{algo}} = \frac{1}{|WF|} \sum_{i=1}^{|WF|} (f(x) \neq y) = \frac{1}{|D|} \sum_{i=1}^{|D|} (f(x) \neq y),$$

where $f(x)$ denotes the predicted class, y the actual class, and *algo* the learning algorithm that was used to construct the meta-mining models. We use McNemar's test to estimate the statistical difference between the error of the meta-learner and that of the default rule, which simply predicts the majority class for each meta-mining problem WF_i. The average error of the default classifier is denoted by

$$A_{d_{def}} = \frac{1}{|WF|} \sum_{i=1}^{|WF|} (c_{maj} \neq y),$$

where c_{maj} is the majority class for problem $WF_i \in WF$ and y is the actual class or class label.

To generate the meta-mining models we used J48 with the following parameter settings: C=0.25 and M=2. Table 2 shows the error rates of the default rule

($A_{d_{def}}$) and J48 ($A_{d_{J48}}$), averaged over the 28 different meta-mining problems, which is equivalent to the error averaged over the different datasets. The average error rate of the meta-models using dataset characteristics was lower than that of the default rule by around 5%, an improvement that was shown to be statististically significant by McNemar's test.

$A_{d_{def}}$	$A_{d_{J48}}$
45.38	40.44 (+)

Table 2. Average estimated errors for the 28 WF_i meta-mining problems in meta-mining scenario A. A + sign indicates that $A_{d_{J48}}$ was significantly better than $A_{d_{def}}$, an = that there was no significant difference and a - that it was significantly worse.

Meta-mining scenario B The main limitation of meta-mining scenario A is that it is not possible to generalize over the learning workflows. There is no way we can predict the performance of a DM workflow wf_i unless we have meta-mined a model based on training meta-data gathered through systematic experimentation with wf_i itself. To address this limitation we introduce the second meta-mining scenario which exploits *both dataset and workflow descriptions*, and provides the means to generalize not only over datasets but also over workflows.

In scenario B, we have a single meta-mining problem in which each instance corresponds to a base-level data mining experiment where some workflow wf_i is applied to a dataset d_j; the class label is either *best* or *rest,* determined with the same rule as described above. We thus have $65 \times 28 = 1820$ meta-mining instances. The description of an instance combines both dataset and workflow meta-features. This representation makes it possible to predict the performance of workflows which have not been encountered in previous DM experiments, provided they are represented with the set of workflow descriptors used in the predictive meta-model. The quality of performance predictions for such workflows clearly depends on how similar they are to the workflows based on which the meta-model was trained.

The instances of this meta-mining dataset are not independent, since they overlap both with respect to the dataset descriptions and the workflow descriptions. Despite this violation of the learning instance independence assumption, we also applied standard classification algorithms as a first approach. However, we handled performance evaluation with precaution. We first evaluated predictive performance using leave-one-dataset-out, i.e., we removed all meta-instances associated with a given dataset d_i and placed them in the test set. We built a predictive model from the remaining instances and applied it to the test instances. In this way we avoided the risk of information leakage incurred in standard leave-out-out or cross-validation, where both training and test sets are likely to contain instances (experiments) concerning the same dataset. We will denote the predictive error estimated in this manner by $B_{d_{algo}}$, where *algo* is the classification algorithm that was used to construct the meta-mining models. The total number of models built was equal to the number of datasets. For each dataset d_j, the meta-level training set contained $64 \times 28 = 1792$ instances and the test set 28, corresponding to the application of the 28 workflows to d_j.

In addition, we investigated the possibility of predicting the performance of workflows that have not been included in the training set of the meta-mining model. The evaluation was done as follows: in addition to leave-one-dataset-out, we also performed leave-one-workflow-out, removing from the training set all instances of a given workflow. In other words, for each training-test set pair, we removed all meta-mining instances associated with a specific dataset d_j as well as all instances associated with a specific workflow wf_i. We thus did $65 \times 28 = 1820$ iterations of the train-test separation, where the training set consisted of $64 \times 27 = 1728$ instances and the test set of the single meta-mining instance$(d_i, wf_j, label)$. We denote the error thus estimated by $B_{d,wf_{algo}}$.

$A_{d_{def}}$	$B_{d_{J48}}$	$B_{d,wf_{J48}}$
45.38	38.24 (+)	42.25 (=)

Table 3. $B_{d_{J48}}$ and $B_{d,wf_{J48}}$ estimated errors, meta-mining scenario B. A + sign indicates that $B_{d|d,wf_{algo}}$ was significantly better than $A_{d_{def}}$, an = that there was no significant difference and a - that it was significantly worse. Column 2 shows that the meta-miner obtains significantly better results than the default by using both dataset and workflow descriptors. Column 3 gives the results in a more stringent scenario involving workflows never encountered in previous domain-level experiments.

Table 3 gives the estimated errors for meta-mining scenario B, in which the meta-models were also built using J48 with the same parameters as in scenario A, but this time using both dataset and workflow characteristics. McNemar's test was also used to compare their performance against the default rule. Column 2 shows the error rate using leave-one-dataset-out error estimation ($B_{d_{J48}}$), which is significantly lower than that of the default rule, but more importantly, also lower than $A_{d_{j48}}$ (Table 2), the error rate of the meta-model built using dataset characteristics alone. This provides evidence of the discriminatory power of the frequent patterns discovered in the DM workflows and used to build the meta-models.

Column 3 shows the error rate of the meta-model built using the combined leave-one-out-dataset/leave-one-workflow-out error estimation procedure ($B_{d,wf_{J48}}$), which was meant to test the ability of the meta-model to predict the performance of completely new workflows. The estimated error rate is again lower than the baseline error, though not significantly. However, it demonstrates the point that for a new dataset, our approach can predict the performance of workflows never yet encountered in previous data mining experiments. As a matter of fact, these workflows can even contain operators that implement algorithms never seen in previous experiments, provided these algorithms have been described in DMOP.

5.3 Discussion

To gain a bit of insight into the kind of meta-model built by J48 using dataset and workflow meta-features, we reran mining scenario B on the full dataset to derive the decision tree to be deployed on new (dataset, workflow) examples. The result was a complex decision tree with 56 internal nodes (meta-feature tests), but a close analysis of a few top levels proved instructive.

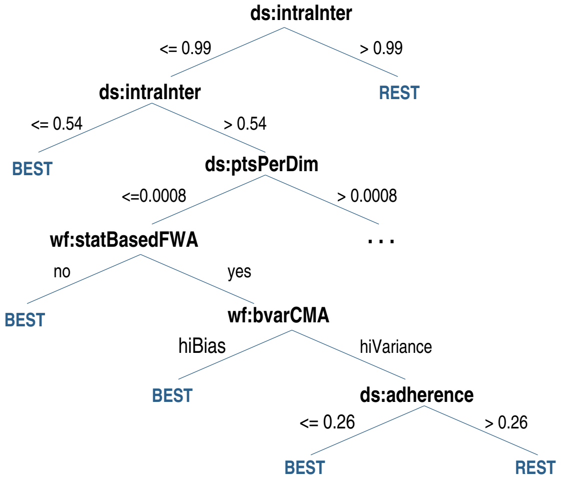


Fig. 19. Top 6 levels of the meta-mined J48 workflow performance predictor based on dataset and workflow characteristics

The top two nodes test the dataset characteristic intraInter, or intra-interclass nearest neighbor (NN) distance [38]. This is a measure of class separability, computed as the ratio of intra (average distance between each data point and its nearest-hit, i.e. its NN from the same class) to inter (average distance between each data point and its nearest-miss, i.e. its NN from a different class):

$$intraInter = \left(\frac{1}{N} \sum_{i=1}^{N} d(x_i, nearestHit(x_i) \right) / \left(\frac{1}{N} \sum_{i=1}^{N} d(x_i, nearestMiss(x_i) \right) .$$

As shown in Fig. 19, the first two tests actually split the intraInter range of values into 3 intervals. At one extreme, $intraInter > 0.99$ indicates a difficult problem where data points from different classes are almost as close as points from the same class; in this case, the meta-model predicts the class REST. At the other extreme, the distance of a data point to members of a different class is almost twice its distance to members of its class ($intraInter \leq 0.54$,); in such cases where classes are neatly separated, the prediction is BEST. Between these two extremes, other tests are needed; the next meta-feature tested is $ptsPerDim = N/D$, where N is the number of data points and D the dimensionality or number of features. The threshold of 0.0008 discriminates between extremely high-dimensional datasets which contain less than 0.0008 instances per feature, or equivalently, more than 1300 features for 1 instance (right branch) and datasets with lower D:N ratio (left branch). We omit the right branch, which grows to a depth of more than 20 levels; in the left branch, by contrast, tests on 2 workflow descriptors and 1 dataset descriptor suffice to classify the remaining instances. The workflow descriptor statBasedFeatureWeightingAlgorithm designates a class of feature weighting algorithms that weight individual features by computing statistics such as χ^2, F-Ratio, or Pearson's correlation coefficient. Workflows that do not use such weighting algorithms (e.g., multivariate algorithms, or univariate methods that use entropy-based weights) are classified as BEST. Among workflows that rely on such statistics to weight features, only those that also use high-bias classification modelling algorithms (e.g. linear discriminants, Naive Bayes) will be predicted to have BEST performance. High-variance algorithms will be classified as BEST only if they are applied to datasets with $adherence < 0.26$. This feature denotes the fraction of data points with maximal adherence subset retained [38]. Intuitively, an adherence subset can be imagined as a ball that is fit around each data point and allowed to grow, covering other data points and merging with other balls, until it hits a data point from a different class. With complex boundaries or highly interleaved classes, the fraction of points with retained (i.e. not merged) adherence subsets will be large. In the learned meta-decision tree, adherence should not be greater than 0.26 for high-variance classification learners to perform BEST.

To summarize, the meta-decision tree described above naturally blends data and workflow characteristics to predict the performance of a candidate workflow on a given dataset. In the vicinity of the root, J48's built-in feature selection mechanism picked up 3 descriptors of data complexity (class separability, dimensionality, and boundary complexity) and 2 workflow descriptors (use of univariate statistics-based feature scores, bias-variance profile of learning algorithm/operator used). Although data descriptors outnumber workflow descriptors in the subtree illustrated in Figure 19, the distribution is remarkably balanced over the whole tree, where 28 of the 56 internal nodes test workflow features. However, most of the workflow features used correspond to simple patterns that express a constraint on a single data mining operator. Only two nodes test a sequence comprising a feature weighting/selection operator and a classification operator. We expect more complex patterns to appear when we feed the meta-learner with workflows from data mining experiments with multi-step data processing. Finally, as mentioned above, the right subtree below ptsPerDim (replaced by "..." in the figure), which corresponds to datasets with more than 1300 features per data point, is considerably more complex; worth noting, however, is the recurrence of the workflow pattern that contains "high-dimensionality tolerant classification modelling algorithm" in branches that lead to a BEST leaf.

6 Conclusion

In this chapter, we proposed a semantic meta-mining approach that contrasts with standard meta-learning in several respects. First, the traditional meta-learning focus on a single task or operator has been replaced by a broader perspective on the full knowledge discovery process. Next, we introduced a revised Rice model that grounds algorithm selection on both data and algorithm characteristics. We operationalized this revised model while mining workflows viewed as compositions of (implemented) algorithms, and performed workflow performance prediction based on both dataset and workflow characteristics. In two distinct meta-mining scenarios, models built using data and workflow characteristics outperformed those based on data characteristics alone, and meta-mined workflow patterns proved discriminatory even for algorithms and workflows not encountered in previous experiments. These experimental results show that the data mining semantics and expertise derived from the DMOP ontology imparts new generalization power to workflow meta-mining.

Though promising, these results can definitely be improved. Performance prediction for DM workflows is still in its infancy, and we have done no more than provide a proof of concept. We certainly need more base-level experiments and more workflows in order to improve the accuracy of learned meta-models. We also need to investigate more thoroughly the different dataset characteristics that have been used in previous meta-learning efforts. Above all, we need more refined strategies for exploring the the joint space of dataset characteristics and workflow characteristics. A simple approach could be to build a model in two stages: first zoom in on the datasets and explore clusters or neighborhoods of datasets with similar characteristics; then within each neighborhood, identify the workflow characteristics that entail good predictive performance. Essentially, what we are trying to solve is a matching problem: the goal is to find the appropriate association of workflow and dataset characteristics, where appropriateness is defined in terms of predictive performance. One way to address this problem is to use collaborative filtering approaches that are also able to account for the properties of the matched objects.

Acknowledgements

This work is partially supported by the European Commission through FP7-ICT-2007-4.4 (Grant No. 231519) project e-LICO: An e-Laboratory for Interdisciplinary Collaborative research in Data Mining and Data-Intensive Sciences. We thank all our partners and colleagues who have contributed to the development of the DMOP ontology: Simon Fischer, Dragan Gamberger, Simon Jupp, Agnieszka Lawrynowicz, Babak Mougouie, Raul Palma, Robert Stevens, Jun Wang.

References

1. D. W. Aha. Lazy learning (editorial). *Artificial Intelligence Review*, 11:7–10, 1997.
2. S. Ali and K. Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1-3):173–186, 2006.
3. M. L. Anderson and T. Oates. A review of recent research in metareasoning and metalearning. *AI Magazine*, 28(1):7–16, 2007.
4. H. Arimura. Efficient algorithms for mining frequent and closed patterns from sem-structured data. In *Proc. Pacific-Asia Conference on Knowledge Discovery in Databases (PAKDD 2008)*, pages 2–13. Springer, 2008.
5. P. Bartlett. For valid generalization, the size of the weights is more important than the size of the network. In *Advances in Nueral Information Processing Systems (NIPS-1997)*, 1997.
6. Mitra Basu and Tim Kam Ho, editors. *Data Complexity in Pattern Recognition.* Springer, 2006.
7. H. Bensusan and C. Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 325–330, 2000.
8. H. Bensusan, C. Giraud-Carrier, and C. Kennedy. A higher-order approach to meta-learning. In *Proceedings of the ECML'2000 workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 109–117, June 2000.
9. A. Bernstein, F. Provost, and S. Hill. Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518, 2005.
10. C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.
11. Hendrik Blockeel and Joaquin Vanschoren. Experiment databases: Towards an improved experimental methodology in machine learning. In *Knowledge Discovery in Databases: PKDD 2007. 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 6–17, Warsaw, Poland, 2007.
12. P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Machine Learning: ECML-94. European Conference on Machine Learning*, pages 83–102, Catania, Italy, 1994. Springer-Verlag.
13. P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, editors. *Metalearning: Applications to Data Mining.* Springer, 2009.
14. P. Brezany, I. Janciak, and A. Min Tjoa. Ontology-based construction of grid data mining workflows. In H. O. Nigro, S. E. Gonzalez Cisaro, and D. H. Xodo, editors, *Data Mining with Ontologies: Implementations, Findings and Frameworks.* IGI Global, 2008.
15. B. Bringmann. Matching in frequent tree discovery. In *Proc.4th IEEE International Conference on Data Mining (ICDM'04)*, pages 335–338, 2004.
16. S. Cacoveanu, C. Vidrighin, and R. Potolea. Evolutional meta-learning framework for automatic classifier selection. In *Proceedings of the IEEE 5th International Conference on Intelligent Computer Communication and Processing (ICCP2009)*, pages 27–30, 2009.
17. M. Cannataro and C. Comito. A data mining ontology for grid programming. In *Proc. 1st Int. Workshop on Semantics in Peer-to-Peer and Grid Computing, in conjunction with WWW2003*, pages 113–134, 2003.

18. P. Chapman, J. Clinton, T. Khabaza, T. Reinartz, and R. Wirth. The CRISP-DM process model. Technical report, CRISP-DM consortium, 1999. http://www.crisp-dm.org.
19. V. Cherkassky. Model complexity control and statistical learning theory. *Natural Computing*, 1:109–133, 2002.
20. C. Diamantini, D. Potena, and E. Storti. Supporting users in KDD process design: A semantic similarity matching approach. In *Proc. 3rd Planning to Learn Workshop (held in conjunction with ECAI-2010)*, pages 27–34, Lisbon, 2010.
21. Pedro Domingos. A unified bias-variance decomposition for zero-one and squared loss. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 564–569, 2000.
22. W. Duch and K. Grudzinski. Meta-learning: Searching in the model space. In *Proc. of the Int. Conf. on Neural Information Processing (ICONIP), Shanghai 2001*, pages 235–240, 2001.
23. W. Duch and K. Grudziński. Meta-learning via search combined with parameter optimization. In *Advances in Soft Computing*, pages 13–22. Springer, 2002.
24. Saso Dzeroski. Towards a general framework for data mining. In *Knowledge Discovery in Inductive Databases*, pages 259–300, 2007.
25. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. MIT Press, 1996.
26. A. Frank and A. Asuncion. UCI machine learning repository, 2010.
27. J. Fürnkranz and J. Petrak. An evaluation of landmarking variants. In *Proceedings of the ECML Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-learning*, pages 57–68, 2001.
28. S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.
29. C. Giraud-Carrier, R. Vilalta, and P. Brazdil. Introduction to the special issue on meta-learning. *Machine Learning*, 54:187–193, 2004.
30. D. Gordon and M. DesJardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20:5–22, 1995.
31. K. Grąbczewski and N. Jankowski. Versatile and efficient meta-learning architecture: knowledge representation and management in computational intelligence. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 51–58, 2007.
32. Data Mining Group. Predictive Model Markup Language (PMML). http://www.dmg.org/.
33. I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, editors. *Feature Extraction: Foundations and Applications*. Springer, 2006.
34. I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.
35. M. Hall. *Correlation-based Feature Selection in Machine Learning*. PhD thesis, University of Waikato, 1999.
36. M. Hilario and A. Kalousis. Fusion of meta-knowledge and meta-data for case-based model selection. In *Principles of Data Mining and Knowledge Discovery. Proceedings of the 5th European Conference*, pages 180–191, Freiburg, Germany, 2001. Springer-Verlag.
37. M. Hilario, A. Kalousis, P. Nguyen, and A. Woznica. A data mining ontology for algorithm selection and meta-mining. In *Workshop on Third-Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD-09)*, 2009.

38. Tin Kam Ho and Mitra Basu. Measures of geometrical complexity in classification problems. In *Data Complexity in Pattern Recognition*, chapter 1, pages 3–23. Springer, 2006.
39. A. Hotho, A. Maedche, S. Staab, and R. Studer. Seal-II - the soft spot between richly structured and unstructured knowledge. *Journal of Universal Computer Science*, 7(7):566–590, 2001.
40. N. Jankowski and K. Grąbczewski. Building meta-learning algorithms basing on search controlled by machine complexity. In *IEEE World Congress on Computational Intelligence*, pages 3600–3607, 2008.
41. A. Kalousis. *Algorithm Selection via Meta-Learning.* PhD thesis, University of Geneva, 2002.
42. A. Kalousis, J. Gama, and M. Hilario. On data and algorithms: understanding inductive performance. *Machine Learning*, 54:275–312, 2004.
43. A. Kalousis and M. Hilario. Representational issues in meta-learning. In *Proc. of the 20th International Conference on Machine Learning*, Washington, DC, 2003. Morgan Kaufmann.
44. J.-U. Kietz, F. Serban, A. Bernstein, and S. Fischer. Data mining workflow templates for intelligent discovery assistance and auto-experimentation. In *Proc. 3rd Workshop on Third-Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD-10)*, pages 1–12, 2010.
45. K. Kira and L. Rendell. The feature selection problem: traditional methods and a new algorithm. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI-92)*, pages 129–134, 1992.
46. C. Köpf and J. Keller. Meta-analysis: from data characterization for meta-learning to meta-regression. In *PKDD-2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP*, 2000.
47. R. Leite and P. Brazdil. Predicting a relative performance of classifiers from samples. In *Proc. International Conference on Machine Learning*, 2005.
48. D. Ler, I. Koprinska, and S. Chawla. Utilising regression-based landmarkers within a meta-learning framework for algorithm selection. In *Proc. ICML-05 Workshop on Meta-Learning*, pages 44–51, 2005.
49. H. Liu and R. Setiono. A probabilistic approach to feature selection—a filter solution. In *Proc. 13th International Conference on Machine Learning (ICML'96)*, pages 319–327, Bari, Italy, 1996.
50. D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification.* Ellis-Horwood, 1994.
51. T. M. Mitchell. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, NJ, 1980.
52. Katharina Morik and Martin Scholz. *Intelligent Technologies for Information Analysis*, chapter The MiningMart Approach to Knowledge Discovery in Databases. Springer, 2004.
53. P. Panov, S. Dzeroski, and L. Soldatova. Ontodm: An ontology of data mining. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, pages 752–760, 2008.
54. J. Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley, 1984.
55. Y. Peng, P. Flach, P. Brazdil, and C. Soares. Decision tree-based data characterization for meta-learning. In *2nd International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, 2002.
56. Y. Peng, P. Flach, C. Soares, and P. Brazdil. Improved dataset characterisation for meta-learning. In *Discovery Science*, pages 141–152, 2002.

57. B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proc. Seventeenth International Conference on Machine Learning, ICML'2000*, pages 743–750, San Francisco, California, June 2000. Morgan Kaufmann.
58. G. Piatetskey-Shapiro. Data mining and knowledge discovery: The third generation. In *Foundations of Intelligent Systems: 10th International Symposium, ISMIS'97*, 1997.
59. J. R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.
60. A. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proc. International Conference on Knowledge Capture (K-CAP'03)*, 2003.
61. L. Rendell, R. Seshu, and D. Tcheng. Layered concept-learning and dynamically variable bias management. In *Proc. of the 10th International Joint Conference on Artificial Intelligence*, pages 308–314, 1987.
62. J. Rice. The algorithm selection problem. *Advances in Computing*, 15:65–118, 1976.
63. C. Schaffer. A conservation law for generalization performance. In *Proc. of the 11th International Conference on Machine Learning*, pages 259–265, 1994.
64. M. R. Sikonja and I. Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53:23–69, 2003.
65. Steven Skiena. *Implementing discrete mathematics: combinatorics and graph theory with Mathematica.* Addison-Wesly Longman Publishing Co., Inc., Boston, MA, USA, 1991.
66. K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1), 2008.
67. C. Soares and P. Brazdil. Zoomed ranking: selection of classification algorithms based on relevant performance information. In *Principles of Data Mining and Knowledge Discovery. Proceedings of the 4th European Conference (PKDD-00*, pages 126–135. Springer, 2000.
68. C. Soares, P. Brazdil, and P. Kuba. A meta-learning method to select the kernel width in support vector regression. *Machine Learning*, 54(3):195–209, 2004.
69. M. Souto, R. Prudêncio, R. Soares, D. Araújo, I. Costa, T. Ludermir, and A. Schliep. Ranking and selecting clustering algorithms using a meta-learning approach. In *International Joint Conference on Neural Networks*, 2008.
70. R. Srikant and R. Agrawal. Mining sequential patterns: generalizations and performance improvements. In *Proc.5th International Conference on Extending Database Technology*, pages 3–17. Springer, 1996.
71. A. Suyama and T. Yamaguchi. Specifying and learning inductive learning systems using ontologies. In *In Working Notes from the 1998 AAAI Workshop on the Methodology of Applying Machine Learning: Problem Definition, Task Decomposition and Technique Selection*, 1998.
72. L. Todorovski and S. Dzeroski. Experiments in meta-level learning with ILP. In *Proc. 3rd European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-99)*, pages 98–106. Springer, 1999.
73. A. Tsymbal, S. Puuronen, and V.Y. Terziyan. Arbiter meta-learning with dynamic selection of classifiers and its experimental investigation. In *Advances in Databases and Information Systems*, pages 205–217, 1999.
74. P. E. Utgoff. *Machine learning of inductive bias.* Kluwer, 1986.

75. P. E. Utgoff. Shift of bias for inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning. An Artificial Intelligence Approach, Vol. 2*, chapter 5, pages 107–148. Morgan Kaufmann, 1986.
76. Joaquin Vanschoren and Larisa Soldatova. Exposé: An ontology for data mining experiments. In *International Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery (SoKD-2010)*, September 2010.
77. R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18:77–95, 2002.
78. R. Vilalta, C. Giraud-Carrier, P. Brazdil, and C. Soares. Using meta-learning to support data mining. *International Journal of Computer Science and Applications*, 1(1):31–45, 2004.
79. D. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1381–1390, 1996.
80. Q. Yang and X. Wu. Ten challenging problems in data mining research. *International Journal of Inform*, 5:594–604, 2006.
81. M. Zaki. Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17, 2005.
82. M. Zakova, P. Kremen, F. Zelezny, and N. Lavrac. Automating knowledge discovery workflow composition through ontology-based planning. *IEEE Transactions on Automation Science and Engineering*, 2010.