Mach Learn (2012) 87:127–158
DOI 10.1007/s10994-011-5277-0

Experiment databases

A new way to share, organize and learn from experiments

Joaquin Vanschoren · Hendrik Blockeel · Bernhard Pfahringer · Geoffrey Holmes

Received: 10 March 2009 / Accepted: 16 December 2011 / Published online: 5 January 2012

Abstract Thousands of machine learning research papers contain extensive experimental comparisons. However, the details of those experiments are often lost after publication, making it impossible to reuse these experiments in further research, or reproduce them to verify the claims made. In this paper, we present a collaboration framework designed to easily share machine learning experiments with the community, and automatically organize them in public databases. This enables immediate reuse of experiments for subsequent, possibly much broader investigation and offers faster and more thorough analysis based on a large set of varied results. We describe how we designed such an experiment database, currently holding over 650,000 classification experiments, and demonstrate its use by answering a wide range of interesting research questions and by verifying a number of recent studies.

Keywords Experimental methodology · Machine learning · Databases · Meta-learning

Editor: Carla Brodley.

J. Vanschoren (✉) · H. Blockeel
LIACS, Universiteit Leiden, Niels Bohrweg 1, 2333CA Leiden, The Netherlands
e-mail: joaquin@liacs.nl

H. Blockeel
e-mail: hendrik.blockeel@cs.kuleuven.be

J. Vanschoren · H. Blockeel
Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium

B. Pfahringer · G. Holmes
Dept. of Computer Science, The University of Waikato, Private Bag 3105, Hamilton 3240, New Zealand

B. Pfahringer
e-mail: bernhard@cs.waikato.ac.nz

G. Holmes
e-mail: geoff@cs.waikato.ac.nz

1 Motivation

"Study the past," Confucius said, "if you would divine the future." Whether we aim to develop better learning algorithms or analyze new sets of data, it is wise to (re)examine earlier machine learning studies and build on their results to gain a deeper understanding of the behavior of learning algorithms, the effect of parameters and the utility of data preprocessing.

Because learning algorithms are typically heuristic in nature, *machine learning experiments* form the main source of objective information. To know how algorithms behave, we need to actually run them on real-world or synthetic datasets and analyze the ensuing results. This leads to thousands of empirical studies appearing in the literature. Unfortunately, experiment descriptions in papers are usually limited: the results are often hard to reproduce and reuse, and it is frequently difficult to interpret how generally valid they are.

1.1 Reproducibility and reuse

Indeed, while much care and effort goes into these studies, they are essentially conducted with a single focus of interest and summarize the empirical results accordingly. The individual experiments are usually not made publicly available, thus making it impossible to reuse them for further or broader investigation. Reproducing them is also very hard, because space restrictions imposed on publications often make it practically infeasible to publish all details of the experiment setup. All this makes us duplicate a lot of effort, makes it more difficult to verify results, and ultimately slows down the whole field of machine learning.

This lack of reproducibility has been warned against repeatedly (Keogh and Kasetty 2003; Sonnenburg et al. 2007; Pedersen 2008), has been highlighted as one of the most important challenges in data mining research (Hirsh 2008), and some major conferences have started to require that all submitted research be fully reproducible (Manolescu et al. 2008). Still, there are few tools that facilitate the publication of reproducible results.

1.2 Generalizability and interpretation

A second issue is that of generalizability: in order to ensure that results are generally valid, the empirical evaluation needs to be equally general: it must cover many different conditions such as various parameter settings and various kinds of datasets, e.g., differing in size, skewness, noisiness or with or without being preprocessed with basic techniques such as feature selection. This typically requires a large set of experiments. Unfortunately, many studies limit themselves to algorithm benchmarking under a rather small set of different conditions. It has long been recognized that such studies are only 'case studies' (Aha 1992) that should be interpreted with caution, and may even create a false sense of progress (Hand 2006).

In fact, a number of studies have illustrated how easy it is to draw overly general conclusions. In time series analysis research, for instance, it has been shown that studies regularly contradict each other because they are biased toward different datasets (Keogh and Kasetty 2003). Furthermore, Perlich et al. (2003) prove that the relative performance of logistic regression and decision trees depends strongly on the *size* of the dataset samples in the training set, which is often not varied or taken into account. The same holds for parameter optimization and feature selection, which can easily dominate any measured performance difference between algorithms (Hoste and Daelemans 2005). These studies underline that there are very good reasons to thoroughly explore different conditions, even when this requires a larger set of experiments. Otherwise, it is very hard, especially for other researchers, to correctly interpret the results.

1.3 Experiment databases

Both issues can be tackled by enabling researchers and practitioners to easily share the full details and results of their experiments in public community databases. We call these *experiment databases*: databases specifically designed to collect and organize all the details of large numbers of past experiments, performed by many different researchers, and make them immediately available to everyone. First, such databases can keep a full and fair account of conducted research so that experiments can be easily logged and reproduced, and second, they form a rich source of reusable, unambiguous information on learning algorithms, tested under various conditions, for wider investigation and application.

They form the perfect complement to journal publications: whereas journals form our collective *long-term memory*, such databases effectively create a collective *short-term working memory* (Nielsen 2008), in which empirical results can be recombined and reused by many researchers to quickly answer specific questions and test new ideas. Given the amount of time and effort invested in empirical assessment, often aimed at replicating earlier results, sharing this detailed information has the potential to dramatically speed up machine learning research and deepen our understanding of learning algorithms.

In this paper, we propose a principled approach to construct and use such experiment databases. First, experiments are automatically transcribed in a common language that captures the exact experiment setup and all details needed to reproduce them. Then, they are uploaded to pre-designed databases where they are stored in an organized fashion: the results of every experiment are linked to the exact underlying components (such as the algorithm, parameter settings and dataset used) and thus also integrated with all prior results. Finally, to answer any question about algorithm behavior, we only have to write a query to the database to sift through millions of experiments and retrieve all results of interest. As we shall demonstrate, the expressiveness of database query languages warrants that many kinds of questions can be answered in one or perhaps a few queries, thus enabling fast and thorough analysis of large numbers of collected results. The results can also be interpreted unambiguously, as all conditions under which they are valid are explicitly stated in the queries.

1.4 Meta-learning

Instead of being purely empirical, experiment databases also store known or measurable properties of datasets and algorithms. For datasets, this can include the number of features, statistical and information-theoretic properties (Michie et al. 1994) and landmarkers (Pfahringer et al. 2000), while algorithms can be tagged by model properties, the average ratio of bias or variance error, or their sensitivity to noise (Hilario and Kalousis 2000).

As such, all *empirical* results, past and present, are immediately linked to all known *theoretical* properties of algorithms and datasets, providing new grounds for deeper analysis. For instance, algorithm designers can include these properties in queries to gain precise insights on how their algorithms are affected by certain kinds of data or how they relate to other algorithms. As we shall illustrate, such insights can lead to algorithm improvements.

1.5 Overview of benefits

In all, sharing machine learning experiments and storing them in public databases constitutes a new, *collaborative* approach to experimentation with many benefits:

Reproducibility The database stores all details of the experimental setup, thus attaining the scientific goal of truly reproducible research.

Reference All experiments, including algorithms and datasets, are automatically organized in one resource, creating an overview of the state-of-the-art, and a useful 'map' of all known approaches, their properties, and their performance. This also includes *negative results*, which usually do not get published in the literature.

Querying When faced with a question on the performance of learning algorithms, e.g., 'What is the effect of the training set size on runtime?', we can answer it in seconds by writing a query, instead of spending days (or weeks) setting up new experiments. Moreover, we can draw upon many more experiments, on many more algorithms and datasets, than we can afford to run ourselves.

Reuse It saves time and energy, as previous experiments can be readily reused. For instance, when benchmarking a new algorithm, there is no need to benchmark the older algorithms over and over again as well: their evaluations are likely stored online, and can simply be downloaded. Moreover, reusing the same benchmarks across studies makes these studies more comparable, and the results are more trustworthy since the original authors typically know best how to setup and tune their algorithms.

Larger studies Studies covering many algorithms, parameter settings and datasets are very expensive to run, but could become much more feasible if a large portion of the necessary experiments are available online. Even if many experiments are missing, one can use the existing experiments to get a first idea, and run additional experiments to fill in the blanks. And even when all the experiments have yet to be run, the automatic storage and organization of experimental results markedly simplifies conducting such large scale experimentation and thorough analysis thereof.

Visibility By using the database, users may learn about (new) algorithms they were not previously aware of.

Standardization The formal description of experiments may catalyze the standardization of experiment design, execution and exchange across labs and data mining tools. Comparable experiments can then be set up more easily and results shared automatically.

The remainder of this article is organized as follows. Section 2 discusses existing experiment repositories in other scientific disciplines, as well as related work in machine learning. Next, Sect. 3 outlines how we constructed our pilot experiment database and the underlying models and languages that enable the free exchange of experiments. In Sect. 4, we query this database to demonstrate how it can be used to quickly discover new insights into a wide range of research questions and to verify prior studies. Finally, Sect. 5 focuses on future work and further extensions. Section 6 concludes.

2 Related work

The idea of sharing empirical results is certainly not new: it is an intrinsic aspect of many other sciences, especially *e-Sciences*: computationally intensive sciences which use the Internet as a global, collaborative workspace. Ultimately, their aim is to create an "open scientific culture where as much information as possible is moved out of people's heads and labs, onto the network and into tools that can help us structure and filter the information" (Nielsen 2008). In each of these sciences, online infrastructures have been built for experiment exchange, using more or less the same three components:

A formal representation language: to enable a free exchange of experimental data, a standard and formal representation language needs to be agreed upon. Such a language should also contain guidelines about the information necessary to ensure reproducibility.

Ontologies: to avoid ambiguity, we need a shared vocabulary in which the interpretation of each concept is clearly defined. This can be represented in an *ontology* (Chandrasekaran and Josephson 1999): a formal, machine-interpretable representation of knowledge as a set of concepts and relationships. It ensures that experiment descriptions can be interpreted unambiguously and allows more powerful querying of the stored information.

A searchable repository: to reuse experimental data, we need to locate it first. Experiment repositories therefore collect and organize all data to make it easily retrievable.

2.1 e-Sciences

Bioinformatics Expression levels of thousands of genes, recorded to pinpoint their functions, are collected in *microarray databases* (Stoeckert et al. 2002), and submission of experiments is now a condition for publication in major journals (Ball et al. 2004). In particular, a set of guidelines was drawn up regarding the required Minimal Information About a Microarray Experiment (MIAME (Brazma et al. 2001)), a MicroArray Gene Expression Markup Language (MAGE-ML) was conceived so that data could be exchanged uniformly, and an ontology (MAGE-MO) was designed (Stoeckert et al. 2002) to provide a controlled core vocabulary, in addition to more specific ontologies, such as the Gene Ontology (Ashburner et al. 2000). Their success has instigated similar approaches in related fields, such as proteomics (Vizcaino et al. 2009) and mass spectrometry data analysis. Likewise, work is underway to better document and organize results from clinical trials.[1] One remaining drawback is that experiment description is still partially performed manually. However, some projects are automating the process further: the Robot Scientist (King et al. 2009) runs and stores all experiments automatically, including all physical aspects of their execution and the hypotheses under study. It has autonomously made several novel scientific discoveries.

Astronomy Large numbers of astronomical observations are collected in so-called Virtual Observatories (Szalay and Gray 2001). They are supported by an extensive list of different protocols, such as XML formats for tabular information (VOTable) (Ochsenbein et al. 2004) and astronomical binary data (FITS), an Astronomical Data Query Language (ADQL) (Yasuda et al. 2004) and informal ontologies (Derriere et al. 2006). The data is stored in databases all over the world and is queried for by a variety of portals (Schaaff 2007), now seen as indispensable to analyze the constant flood of data.

Physics Various subfields of physics also share their experimental results. Low-energy nuclear reaction data can be expressed using the Evaluated Nuclear Data File (ENDF) format and collected into searchable ENDF libraries.[2] In high-energy particle physics, the HEPDATA[3] website scans the literature and downloads the experimental details directly from the machines performing the experiments. Finally, XML-formats and databases have been proposed for high-energy nuclear physics as well (Brown et al. 2007).

2.2 Machine learning

In machine learning, datasets can be shared in the UCI (Asuncion and Newman 2007) and LDC[4] repositories (amongst others), and algorithms can be published on the MLOSS web-

[1] Development of a Systematic Review Data Repository, http://www.effectivehealthcare.ahrq.gov.

[2] http://www.nndc.bnl.gov/exfor/endf00.jsp.

[3] http://durpdg.dur.ac.uk/hepdata/.

[4] http://www.ldc.upenn.edu.

site.[5] The MLcomp website[6] stores both algorithms and datasets, and if the algorithms implement a given interface, users can run the algorithms on the server. Unfortunately, there is no public repository of results: users can only see their own results.

In fact, many data mining platforms can run experiments and save their setups and results. However, they each use their own export format, which only covers experiments run with the platform in question. To the best of our knowledge, the only *standardized* format is the Predictive Model Markup Language (PMML),[7] which describes predictive models, but not detailed experimental setups nor other outputs such as algorithm evaluations.

Some public repositories of experimental results do exist. In meta-learning, the StatLog (Michie et al. 1994) and METAL (Brazdil et al. 2009) projects collected a large number of experimental results to search for patterns in learning behavior. Unfortunately, only experiments with default parameter settings were stored. Furthermore, data mining challenge platforms such as Kaggle (Carpenter 2011) and TunedIT (Wojnarski et al. 2010), and challenge-based conferences such as TREC[8] evaluate algorithms on public datasets and publish the results online. However, the use of these results in future research is limited, as parameter settings are often highly optimized to a particular dataset, and the results are not organized to allow thorough querying.

Our research over the last few years has focused on bringing the benefits of these approaches together. Blockeel (2006) first proposed to store machine learning experiments in databases designed for thorough querying of the stored results, although without presenting details on how to construct such databases, or considering whether it was realistic to do so.

Blockeel and Vanschoren (2007) offered a first implementation of experiment databases for supervised classification supporting a wide range of querying capabilities, including explorations of the effects of parameters and dataset properties. Further work focused on how to query the database (Vanschoren and Blockeel 2008), how to exchange experiments with the database (Vanschoren et al. 2009), and how experiments from previous studies can be combined and reused to learn from past results (Vanschoren et al. 2008).

This article goes far beyond this earlier work in machine learning. We present an open collaborative framework, similar to those used in other sciences, through which machine learning experiments can be freely shared, organized and reused for further study.

Specifically, as in many e-sciences, we start by building an ontology for machine learning experimentation, called Exposé, which serves as a rich formal domain model to uniformly describe different types of experiments. It includes complete data mining workflows and detailed algorithm descriptions and evaluation procedures. Next, we will use this ontology to create much more general and extensible experiment description languages and detailed database models. As such, we bring this work in line with experiment repositories in other sciences, allow researchers to adapt experiment descriptions to their own needs, and facilitate participation of the community in extending the ontology towards other machine learning subfields and new types of experiments.

[5] http://mloss.org.

[6] http://www.mlcomp.org.

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

[8] http://trec.nist.gov/.

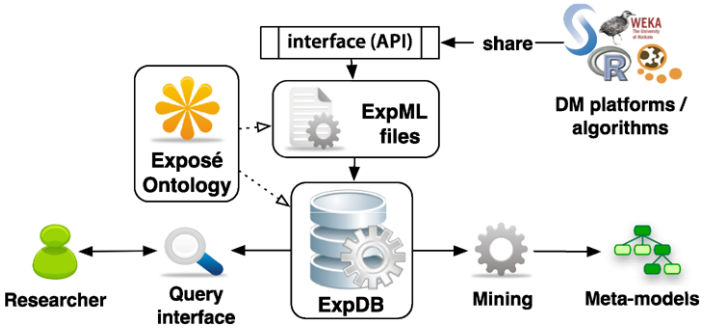


Fig. 1 Components of the experiment database framework

3 A pilot experiment database

In this section, we outline the design of this collaborative framework, outlined in Fig. 1. As in the sciences discussed above, we first establish a controlled vocabulary for data mining experimentation in the form of an open ontology (Exposé), before mapping it to an experiment description language (called ExpML) and an experiment database (ExpDB). These three elements (boxed in Fig. 1) will be discussed in the next three subsections. Although currently focused on supervised classification, Exposé is open to any future extensions, e.g., towards new techniques or models, which can then be mapped to updated description languages and databases. Full versions of the ontologies, languages and database models discussed below can be found on http://expdb.cs.kuleuven.be.

Experiments are shared (see Fig. 1) by entering all experiment setup details and results through the framework's interface (API), which exports them as ExpML files or directly streams them to an ExpDB. Any data mining platform or custom algorithm can thus use this API to add a 'sharing' feature that publishes new experiments. The ExpDB can be set up locally, e.g., for a single person or a single lab, or globally, a central database open to submissions from all over the world. Finally, the bottom of the figure shows different ways to tap into the stored information, which will be amply illustrated in Sect. 4:

Querying. Querying interfaces allow researchers to formulate questions about the stored experiments, and immediately get all results of interest. We currently offer various such interfaces, including graphical ones (see Sect. 3.3.3).

Mining. A second use is to automatically look for patterns in algorithm performance by mining the stored evaluation results and theoretical meta-data. These *meta-models* can then be used, for instance, in algorithm recommendation (Brazdil et al. 2009).

3.1 The Exposé ontology

The Exposé ontology describes the concepts and the structure of data mining experiments. It establishes an unambiguous and machine-interpretable (semantic) vocabulary, through which experiments can be automatically shared, organized and queried. We will also use it to define a common experiment description language and database models, as we shall illustrate below. Ontologies can be easily extended and refined, which is a key concern since data mining and machine learning are ever-expanding fields.

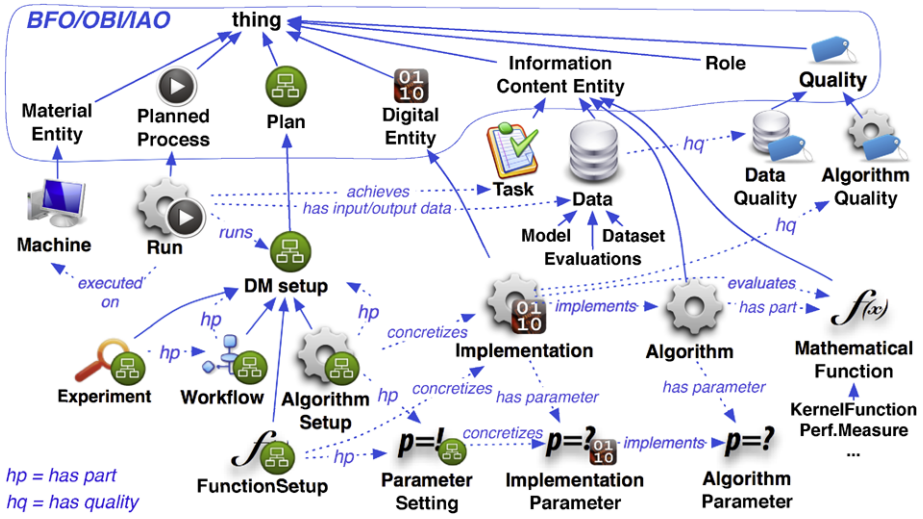


Fig. 2 An overview of the top-level concepts in the Exposé ontology

3.1.1 Collaborative ontology design

Several other useful ontologies are being developed in parallel: OntoDM (Panov et al. 2009) is a top-level ontology for data mining concepts, EXPO (Soldatova and King 2006) models scientific experiments, DMOP (Hilario et al. 2009) describes learning algorithms (including their internal mechanisms and models) and workflows, and the KD ontology (Záková et al. 2008) and eProPlan ontology (Kietz et al. 2009) describe large arrays of DM operators.

To streamline ontology development, a 'core' ontology was defined, and an open ontology development forum was created: the Data Mining Ontology (DMO) Foundry.[9] The goal is to make the ontologies interoperable and orthogonal, each focusing on a particular aspect of the data mining field. Moreover, following best practices in ontology engineering, we reuse commonly accepted concepts and relationships from established top-level scientific ontologies: BFO,[10] OBI,[11] IAO,[12] and RO.[13] We will describe how Exposé interacts with all other ontologies below.

3.1.2 Top-level view

Figure 2 shows Exposé's high-level concepts and relationships. The full arrows symbolize *is-a* relationships, meaning that the first concept is a subclass of the second, and the dashed arrows symbolize other common relationships. The most top-level concepts are reused from the aforementioned top-level scientific ontologies, and help to describe the exact semantics

[9]The DMO Foundry: http://dmo-foundry.org.

[10]The Basic Formal Ontology (BFO): http://www.ifomis.org/bfo.

[11]The Ontology for Biomedical Investigations (OBI): http://obi-ontology.org.

[12]The Information Artifact Ontology (IAO): http://bioportal.bioontology.org/ontologies/40642.

[13]The Relation Ontology (RO): http://www.obofoundry.org/ro. We often use subproperties, e.g. *implements* for *concretizes*, and *runs* for *realizes*, to reflect common usage in the field.

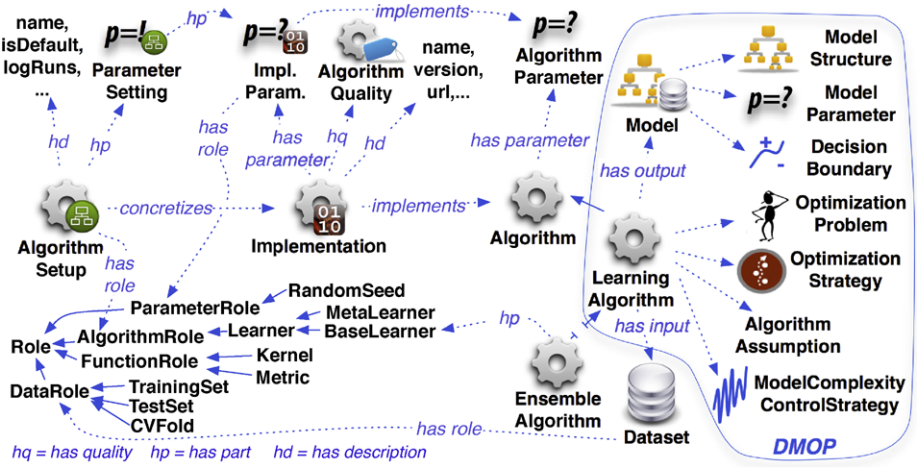


Fig. 3 Learning algorithms in the Exposé ontology

of many data mining concepts. For instance, when speaking of a 'data mining algorithm', we can semantically distinguish an abstract *algorithm* (e.g., C4.5 in pseudo-code), a concrete *algorithm implementation* (e.g., WEKA's `J48` implementation of C4.5), and a specific *algorithm setup*, including *parameter settings* and subcomponent setups. The latter may include other algorithm setups, e.g. for base-learners in ensemble algorithms, as well as mathematical *functions* such as kernels, distance functions and evaluation measures. A *function setup* details the implementation and parameter settings used to evaluate the function.

An algorithm setup thus defines a deterministic function which can be directly linked to a specific result: it can be *run* on a *machine* given specific input data (e.g., a *dataset*), and produce specific output data (e.g., new datasets, *models* or *evaluations*). As such, we can trace any output result back to the inputs and processes that generated it (data provenance). For instance, we can query for evaluation results, and link them to the specific algorithm, implementation or individual parameter settings used, as well as the exact input data.

Algorithm setups can be combined in *workflows*, which additionally describe how data is passed between multiple algorithms. Workflows are hierarchical: they can contain subworkflows, and algorithm setups themselves can contain internal workflows (e.g., a cross-validation setup may define a workflow to train and evaluate learning algorithms). The level of detail is chosen by the author of an experiment: a simple experiment may require a single algorithm setup, while others involve complex scientific workflows.

Tasks cover different data mining (sub)tasks, e.g., supervised classification. *Qualities* are known or measurable properties of algorithms and datasets (see Sect. 1.4), which are useful to interpret results afterwards. Finally, algorithms, functions or parameters can play certain *roles* in a complex setup. As illustrated in Fig. 3, an algorithm can sometimes act as a *base-learner* in an ensemble algorithm, a dataset can act as a *training set* in one experiment and as a *test set* in the next, and some parameters also have special roles, such as setting a random seed or selecting one of several implemented functions (e.g., kernels) in an algorithm.

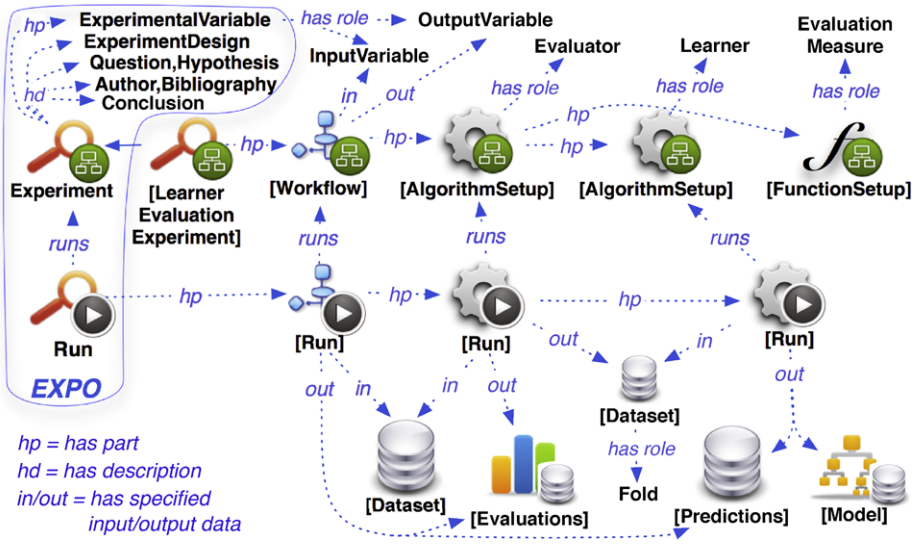


Fig. 4 An example experiment in the Exposé ontology. Elements shown in brackets are instantiations (individuals) of the concepts (classes) in question

3.1.3 Algorithms

Algorithms, especially learning algorithms, are very complex objects, sometimes consisting of many components, and often differing from other algorithms in only one key aspect. Thus, it is important to describe how one algorithm (or implementation) differs from the next.

On the right of Fig. 3, a *learning algorithm* is shown: it will output a model, which can be described using concepts from the DMOP ontology: the *model structure* (e.g., a decision tree), the *model parameters* and the type of *decision boundary* (for classification). It also requires a dataset as input, which can be specified further by its structure (e.g., table, graph or time series) and specific qualities (e.g., purely numeric data). The algorithm itself can be further described by its *optimization problem* (e.g., support vector classification with an L1 loss function), its *optimization strategy* (e.g., sequential minimal optimization (SMO)) and any *model complexity control strategies* (e.g., regularization), also defined by DMOP. This information can be very useful when comparing the results of different algorithms.

Algorithm implementations are further described by their version number, download url (compiled or source code), algorithm qualities and their parameters, including default values and sensible value ranges. Algorithm setups may include a custom name (for reference), whether it is the default setup, and whether its runs should be stored. As previously stated, they may fulfill certain roles, which can be used in queries: for instance, to find the *learner* in a complex DM workflow. Many examples of such querying will be given in Sect. 4.

3.1.4 Experiments

Experiments are modeled as shown in Fig. 4. A workflow has certain *input variables*, e.g., an input dataset, algorithm or parameter setting, and *output variables*. An *experiment* tries to answer a question (in exploratory settings) or test a hypothesis by assigning certain values to these input variables. It has *experimental variables*: *independent variables* with a range

of possible values, *controlled variables* with a single value, or *dependent variables*, i.e., a monitored output. The *experiment design* (e.g., *full factorial*) defines which combinations of input values are used. These concepts are reused from EXPO (Soldatova and King 2006).

One experiment run will typically generate many workflow runs (with different input values), and a workflow run may consist of smaller algorithm runs, as illustrated in Fig. 4. It shows an instantiation of an experiment whose workflow contains a cross-validation (CV) procedure, in turn including a learning algorithm and evaluation measures.

Runs are triples consisting of input data, a setup and output data. Any sub-runs, such as the 10 algorithm runs within a 10-fold CV run, could also be stored with the exact input data (folds) and output data (predictions). Again, this level of detail is defined by the experimenter in the algorithm setup (see Sect. 3.1.3). Especially for complex workflows, it might be interesting to afterwards query the results of certain sub-runs. To maintain reproducibility, all runs involving external services (e.g. web services) should be stored as well.

Finally, it is worth noting that while this model is very general, some types of experiments, e.g., *simulations* (Frawley 1989), will still require further ontology extensions.

3.1.5 Extending the ontology

In all, the Exposé ontology currently defines over 850 concepts (not including the related ontologies), many of them covering dataset and algorithm qualities, experimental designs and evaluation functions. Still, it can be extended much further, especially towards other algorithms and other data mining tasks. Extending the ontology can be done by proposing extensions on the DMO Foundry website (see Sect. 3.1). Additionally, we are in the process of setting up a *semantic wiki*[14] where one can add new concepts simply by starting a new wiki page, and add relationships by using *property links*: whereas the wiki link `[[J48]]` placed on the C4.5 page will create an *unlabeled* link to the J48 page, the property link `[[Has implementation::J48]]` will create a *labeled* link between both pages, which can be interpreted by the wiki engine. It will then automatically add J48 as an implementation of C4.5 in the underlying ontology. As such, extending the ontology is as easy as editing wiki's, and every implementation, algorithm, function and dataset will have its own wiki page. Vice versa, the wiki can query the ontology to generate up-to-date information, e.g., a list of all known C4.5 implementations or algorithm benchmarks. Concepts that require a minimal amount of information to ensure reproducibility, such as implementations and their version numbers, download urls and parameters, will be added through forms on the wiki. A small number of editors can curate and complement the user-generated content.

3.2 ExpML: a common language

Returning to our framework in Fig. 1, we now use this ontology to define a common language to describe experiments. The most straightforward way to do this would be to describe experiments in Exposé, export them in RDF[15] and store everything in RDF databases (triple-stores). However, such databases are still under active development, and many researchers are more familiar with XML and relational databases, which are also widely supported by many current data mining tools. Therefore, we will also map the ontology to a simple XML-based language, ExpML, and a relational database schema. Technical details of this mapping are outside the scope of this paper and can be found on the database website, which

[14] http://semantic-mediawiki.org.

[15] Resource Description Framework: http://www.w3.org/RDF.

also hosts an API to read and write ExpML. Below, we show a small example of ExpML output to illustrate our modeling of data mining workflows.

3.2.1 Workflow runs

Figure 5 shows a *workflow run* in ExpML, executed in WEKA (Hall et al. 2009) and exported through the aforementioned API, and a schematic representation.[16] The workflow has two inputs: a dataset URL and parameter settings. It also contains two algorithm setups: the first loads a dataset from the given URL, and then passes it to a cross-validation setup (10 folds, random seed 1). The latter evaluates a Support Vector Machine (SVM) implementation, using the given parameter settings, and outputs evaluations and predictions. Note that the workflow is completely concretized: all parameter settings and implementations are fixed.

The bottom of Fig. 5 shows the workflow run and its two algorithm sub-runs, each pointing to the setup used. Here, we chose not to output the 10 per-fold SVM runs.

The final output consists of *Evaluations* and *Predictions*. As shown in the ExpML code, these have a predefined structure so that they can be automatically interpreted and organized. Evaluations contain, for each evaluation function (as defined in Exposé), the evaluation value and standard deviation. They can also be labeled, as for the per-class precision results. Predictions can be probabilistic, with a probability for each class, and a final prediction for each instance. For storing models, we can use existing formats such as PMML. Finally, note that id's are generated for each setup and dataset, so that they can be referenced.

The evaluation process needs to be described in such high detail because there are many pitfalls associated with the statistical evaluation of algorithms (Salzberg 1999). For instance, it is often important to perform multiple cross-validation runs (Bradford and Brodley 2001), use multiple evaluation metrics (Demsar 2006), and use the appropriate statistical tests (Dietterich 1998). By describing (and storing) the exact evaluation procedures, we can later query for those results that can be confidently reused.

3.2.2 Experiments

Figure 6 shows a partial experiment description, including the independent variables and their value ranges (defined as sets of labeled tuples), and the experiment design (i.e., full factorial). The workflow is a generalization of the one shown in Fig. 5: some attributes are defined in terms of the input variables, i.e., 'input:<input name>:<label>'. This can be done for any attribute. Assigning specific values to the workflow inputs (according to the experiment design) will generate concrete workflows which can be run. These abstract workflow descriptions can also be stored in the database, or shared on workflow sharing platforms (De Roure et al. 2009; Leake and Kendall-Morwick 2008; Morik and Scholz 2004). Finally, one can add textual conclusions, stating what was learned or, in the case or negative results, why an idea was interesting or made perfect sense, even though it didn't work.

3.3 Organizing machine learning information

The final step in our framework (see Fig. 1) is organizing all this information in searchable databases such that it can be retrieved, rearranged, and reused in further studies. This is done

[16] In the future, we aim to generate workflow schemas automatically. Additionally, each setup element can be given a 'canvasXY' attribute so that canvas-based data mining platforms can export workflow schemas.

```
<Run machine="" timestamp="" author="">
 <Workflow id="1:mainflow" template="10:mainflow">
   <AlgorithmSetup id="2:loadData" impl="Weka.ARFFLoader(1.22)" logRuns="true">
    <ParameterSetting name="location" value="http://.../lymph.arff"/>
   </AlgorithmSetup>
   <AlgorithmSetup id="3:crossValidate" impl="Weka.Evaluator(1.25)" logRuns="true" role="
        CrossValidation">
    <ParameterSetting name="F" value="10"/>
    <ParameterSetting name="S" value="1"/>
    <AlgorithmSetup id="4:learner" impl="Weka.SMO(1.68)" logRuns="false" role="Learner">
     <ParameterSetting name="C" value="0.01"/>
     <FunctionSetup id="5:RBFKernel" impl="Weka.RBF(1.3.1)" role="Kernel">
      <ParameterSetting name="G" value="0.1"/>
     </FunctionSetup>
    </AlgorithmSetup>
   </AlgorithmSetup>
   <Input name="url" dataType="Tuples" value="http://.../lymph.arff"/>
   <Input name="par" dataType="Tuples" value="[name:G,value:0.1]"/>
   <Output name="evaluations" dataType="Evaluations"/>
   <Output name="predictions" dataType="Predictions"/>
   <Connection source="2:loadData" sourcePort="data" target="3:crossValidate" targetPort="
        data"
   dataType="Weka.Instances"/>
   <Connection source="3:crossValidate" sourcePort="evaluations"
   target="1:mainflow" targetPort="evaluations" dataType="Evaluations"/>
   <Connection source="3:crossValidate" sourcePort="predictions" target="1:mainflow"
   targetPort="predictions" dataType="Predictions"/>
   </Workflow>
 <OutputData name="evaluations">
  <Evaluations id="6">
   <Evaluation function="PredictiveAccuracy" value="0.831081" stDev="0.02"/>
   <Evaluation function="Precision" label="class:normal" value="0" stDev="0"/>
   <Evaluation function="Precision" label="class:metastases" value="0.8021" stDev="0.01"/>
  ...</Evaluations>
 </OutputData>
 <OutputData name="predictions">
  <Predictions id="7">
   <Prediction instance="0" value="normal" probability="0"/>
   <Prediction instance="0" value="metastases" probability="0.333333"/>
   <Prediction instance="0" value="malign_lymph" probability="0.5" final="true"/>
   <Prediction instance="0" value="fibrosis" probability="0.166667"/>
  ...</Predictions>
 </OutputData>
 <Run setup="2:loadData">
  <OutputData name="data">
   <Dataset id="8" name="lymph" url="http://.../lymph.arff"
   dataType="Weka.Instances"/>
  </OutputData>
 </Run>
 <Run setup="3:crossValidate">
  <InputData name="data"> <Dataset ref="8"/> </InputData>
  <OutputData name="evaluations"> <Evaluations ref="6"/> </OutputData>
  <OutputData name="predictions"> <Predictions ref="7"/> </OutputData>
 </Run>
</Run>
```

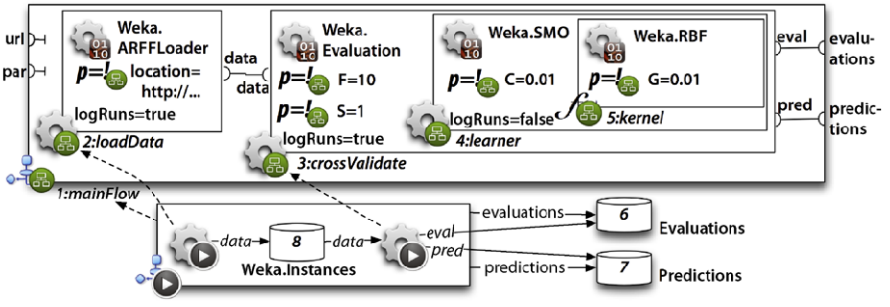


Fig. 5 A *workflow run* in ExpML and a schematic representation

```
<Experiment id="9:exp1" experimentDesign="FullFactorial"author="" question="">
 <ExperimentalVariable name="url" variableType ="independent">{http
  ://.../anneal.arff,http://.../colic.arff}
 </ExperimentalVariable>
 <ExperimentalVariable name="par" variableType ="independent"> {{[name:G,value:0.01]},{[
 name:G,value:0.1]}}</ExperimentalVariable>
 <Workflow id="10:mainflow" name="mainflow">
   ...
    <ParameterSetting name="location" value="input:url"/>
   ...
    <ParameterSetting name="input:par:name" value="input:par:value"/>
   ...
   <Input name="url" dataType="Tuples"/>
   <Input name="par" dataType="Tuples"/>
   ...
 </Workflow>
 <Conclusion>...</Conclusion>
</Experiment>
```

Fig. 6 An example experiment setup in ExpML, used to generate the workflow run in Fig. 5

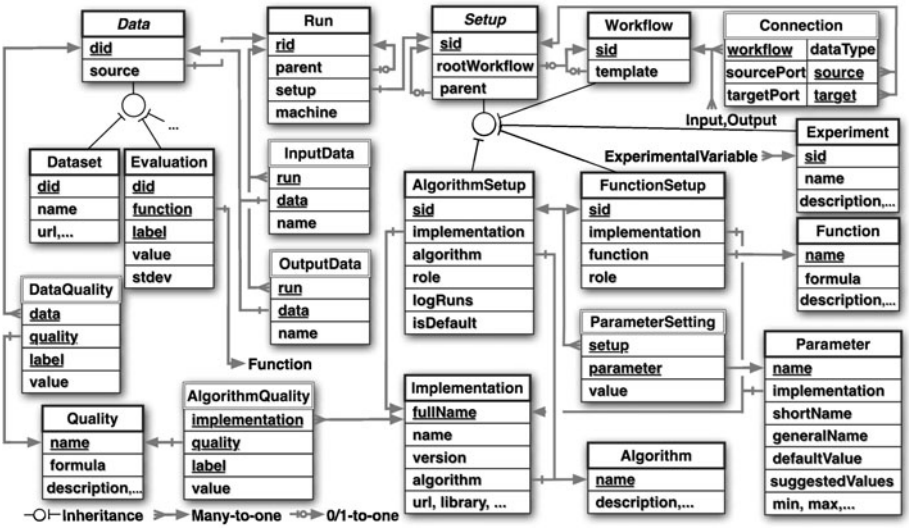


Fig. 7 The general structure of the experiment database. *Underlined columns* indicate primary keys, the *arrows* denote foreign keys. *Tables in italics* are abstract: their fields only exist in child tables

by collecting ExpML descriptions and storing all details in a predefined database. To design such a database, we mapped Exposé to a relational database model. In this section, we offer a brief overview of the model to help interpret the queries in the remainder of this paper.

3.3.1 Anatomy of an experiment database

Figure 7 shows the most important tables, columns and links of the database model. `Runs` are linked to their input- and output data through the join tables `InputData` and `OutputData`, and data always has a *source* run, i.e., the run that generated it. Runs can have *parent* runs, and a specific `Setup`: either a `Workflow` or `AlgorithmSetup`, which can

also be hierarchical. All setups have a (top-level) *root workflow*.[17] `AlgorithmSetups` and `FunctionSetups` can have `ParameterSettings`, a specific `Implementation` and a general `Algorithm` or `Function`. `Implementations` and `Datasets` can also have `Qualities`, stored in `Algorithm Quality` and `DataQuality`, respectively. Data, runs and setups have unique id's, while algorithms, functions, parameters and qualities have unique names defined in Exposé.

3.3.2 Populating the database

Because we want to use this database to gain insight into the behavior of machine learning algorithms under various conditions, we need to populate it with experiments that are as diverse as possible. In this study, we selected 54 well-known classification algorithms from the WEKA platform (Hall et al. 2009) and inserted them together with descriptions of all their parameters. Next, 86 commonly used classification datasets were taken from the UCI repository and inserted together with 56 data characteristics, calculated for each dataset. Finally, two data preprocessors were added: feature selection with Correlation-based Feature Subset Selection (Hall 1998), and a subsampling procedure.

To strike a balance between the breadth and depth of our exploration, a number of algorithms were explored more thoroughly than others. A first series of experiments simply varied the chosen algorithm, running them with default parameter settings on all datasets. To study parameter effects, a second series varied the parameter settings, with at least 20 different values, of a smaller selection of popular algorithms: SMO (an SVM trainer), MultilayerPerceptron, J48 (C4.5), 1R, RandomForest, Bagging and Boosting. In addition to parameter settings, component settings were also varied: two different kernels were used in the SVM, with their own range of parameter settings, and all non-ensemble learners were used as base-learners for ensemble learners. In the case of multiple varied parameters, we used a one-factor-at-a-time design: each parameter is varied in turn while keeping all other parameters at default. Finally, a third series of experiments used a random sampling design to uniformly cover the entire parameter space (with at least 1000 settings) of an even smaller selection of algorithms: J48, Bagging and 1R. All parameter settings were run on all datasets, and for all randomized algorithms, each experiment was repeated 20 times with different random seeds. Detailed information on exactly which parameter settings were used can be found in the stored experiment context descriptions. While we could have explored more algorithms in more depth, this already constitutes a quite computationally expensive setup, with over 650,000 experiments.

All experiments where evaluated with 10-fold cross-validation, using the same folds on each dataset, against 45 evaluation metrics. A large portion was additionally evaluated with a bias-variance analysis (Kohavi and Wolpert 1996). To run all these experiments, we wrote a WEKA plug-in that exported all experiments in ExpML, as shown in Fig. 1.

3.3.3 Accessing the experiment database

The experiment database is available at http://expdb.cs.kuleuven.be/. Two query interfaces are provided: one for standard SQL queries (a library of example queries is available), as well as a graphical interface that hides the complexity of the database, but still supports most types of queries. Querying can be done on the website itself or with a desktop application, and both support several useful visualization techniques to display the results. Finally, several video tutorials help the user to get started quickly.

[17] In `Experiments`, this workflow is the one that will be instantiated and run, as shown in Fig. 6.

4 Learning from the past

In this section, we use the database described in the previous section to evaluate how easily the collected experiments can be exploited to discover new insights into a wide range of research questions, and to verify a number of recent studies.[18] In doing this, we aim to take advantage of the theoretical information stored with the experiments to gain deeper insights.

More specifically, we distinguish three types of studies, increasingly making use of this theoretical information, and offering increasingly generalizable results:[19]

1. Model-level analysis. These studies evaluate the produced models through a range of performance measures, but consider only individual datasets and algorithms. They identify HOW a specific algorithm performs, either on average or under specific conditions.
2. Data-level analysis. These studies investigate how known or measured *data properties*, not individual datasets, affect the performance of algorithms. They identify WHEN (on which kinds of data) an algorithm can be expected to behave in a certain way.
3. Method-level analysis. These studies don't look at individual algorithms, but take general *algorithm properties* (e.g. their bias-variance profile) into account to identify WHY an algorithm behaves in a certain way.

4.1 Model-level analysis

In the first type of study, we are interested in how individual algorithms perform on specific datasets. This type of study is typically used to benchmark, compare or rank algorithms, but also to investigate how specific parameter settings affect performance.

4.1.1 Comparing algorithms

To compare the performance of all algorithms on one specific dataset, we can plot the outcomes of cross-validation (CV) runs against the algorithm names. This can be translated to SQL as shown graphically in Fig. 8: we select (in brackets) the evaluation values obtained from CV runs, and the name of the algorithm used in the CV setup. We also add constraints (below the table names) to select a specific evaluation function, i.e., predictive accuracy, and the name of the dataset inputted into the CV run. Since CV queries are very common, we offer an indexed view, `CVRun`, shown in Fig. 9 (left). This simplifies the query: with three simple joins, we link evaluations to the algorithm setup and input dataset, as shown in Fig. 9 (right). Should we need a specific type of CV, we'd include and constrain the CV setup.

To further simplify querying, we provide a graphical query interface (see Sect. 3.3.3) based on these query graphs. It allows you to click nodes to expand the graph, select the desired values and add constraints, without needing to know the exact table or column names.

Running the query returns all known experiment results, which are scatterplotted in Fig. 10, ordered by performance. This immediately provides a complete overview of how each algorithm performed. Because the results are as general as allowed by the constraints written in the query, the results on sub-optimal parameter settings are shown as well (at least for those algorithms whose parameters were varied), clearly indicating the performance variance they create. As expected, ensemble and kernel methods are dependent on the selection of the correct kernel, base-learner, and other parameter settings.

[18] Some of these illustrations have appeared before (Blockeel and Vanschoren 2007; Vanschoren et al. 2008, 2009) but are also included here to provide a more complete overview of the querying possibilities.

[19] A similar distinction is identified by Van Someren (2001).

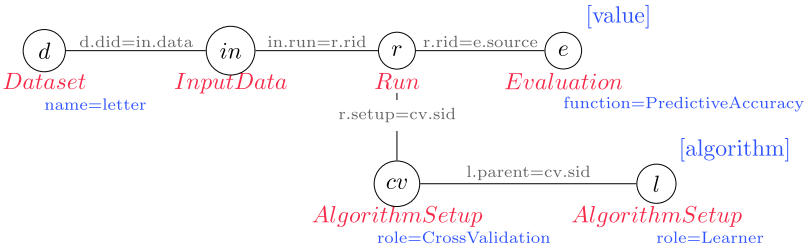


Fig. 8 A graph representation of our first query

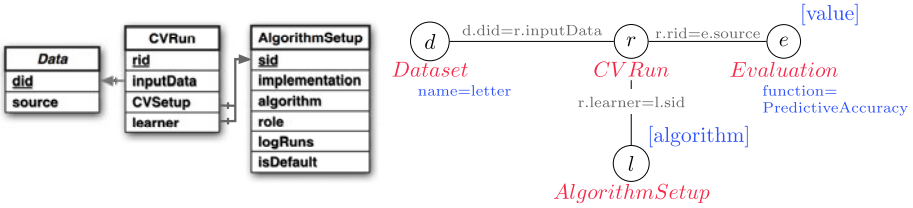


Fig. 9 The first query (*right*), simplified by using the cross-validation run view (*left*)

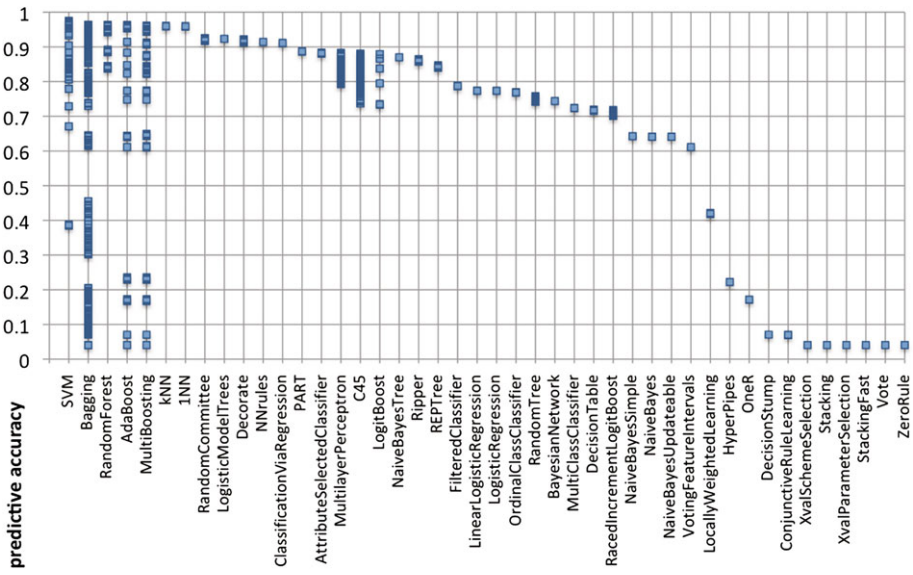


Fig. 10 Performance of all algorithms on dataset 'letter'

We can however extend the query to ask for more details about these algorithms. Figure 11 shows how to append the name of the kernel or base-learner in algorithms that have such components, yielding the results shown in Fig. 12. This provides a detailed overview of learning performance on this dataset. For instance, when looking at SVMs, it is clear that the RBF-kernel yields good performance, which is not unexpected given that RBF kernels are popular in letter recognition problems. However, there is still variation in the performance

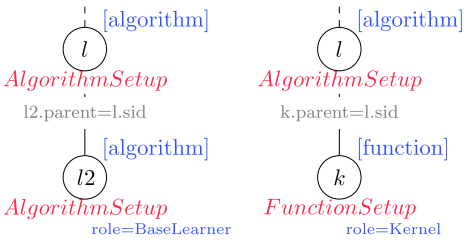


Fig. 11 A partial graph representation of the extended query, showing how to select the base-learners of an ensemble method (*left*) and kernels (*right*). The rest of the query is the same as in Fig. 9

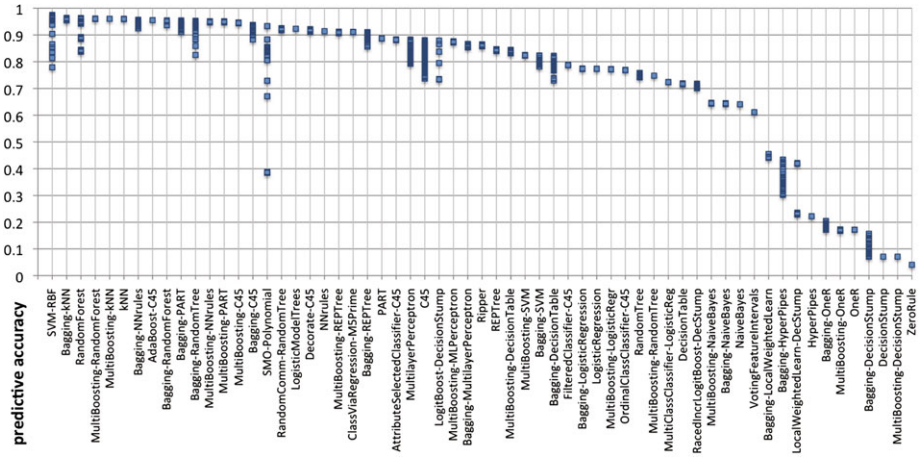


Fig. 12 Performance of all algorithms on dataset 'letter', including base-learners and kernels. Some similar (and similarly performing) algorithms were omitted to allow a clear presentation

of the RBF-based SVMs, so it might be interesting to investigate this in more detail. Note that there are large jumps in the performance of SVMs and RandomForests, which are, in all likelihood, caused by parameters that heavily affect their performance.

Moreover, when looking at the effects of bagging and boosting, it is clear that some base-learners are more useful than others. For instance, it appears that bagging and boosting have almost no effect on logistic regression and naive Bayes. Indeed, bagged logistic regression has been shown to perform poorly (Perlich et al. 2003), and naive Bayes, which generally produces very little variance error (see Sect. 4.3) is unlikely to benefit from bagging, a variance-reduction method. Conversely, bagging random trees seems to be hugely profitable, but this does not hold for boosting. A possible explanation for this is the fact that random trees are prone to variance error, which is primarily improved by bagging but much less by boosting (Bauer and Kohavi 1999). Another explanation is that boosting might have stopped early. Some learners, including the random tree learner, can yield a model with zero training error, causing boosting to stop after one iteration. A new query showed that on half of the datasets, the boosted random trees indeed yield exactly the same performance as a single random tree. This shows that boosting is best not used with base learners that can achieve zero training error, such as random trees, random forests and SVMs with RBF or high-degree polynomial kernels. It also suggests a research direction: is there an optimal, non-zero error rate for boosting, and can we 'regularize' strong learners to exhibit that error during boosting such that it runs to completion? Finally, it seems more rewarding to fine-tune random forests, multi-layer perceptrons and SVMs than to bag or boost their default

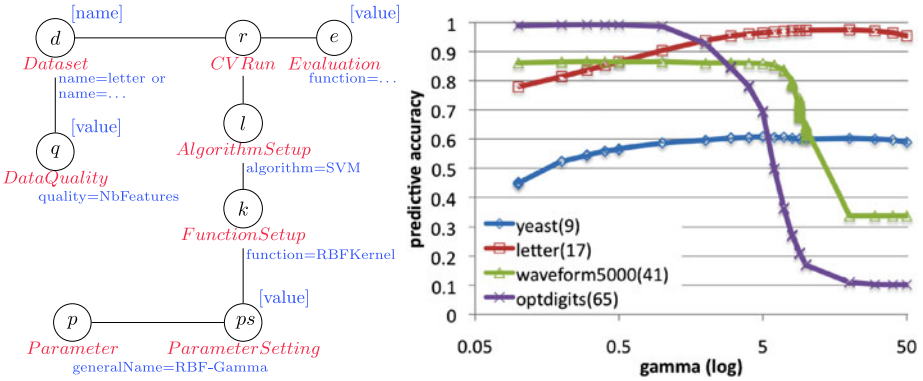


Fig. 13 The effect of parameter gamma of the RBF kernel in SVMs on a number of different datasets, with their number of attributes shown in brackets, and the accompanying query graph

setting, whereas for C4.5, bagging and boosting outperform fine-tuning. Note that these observations are made on one dataset. Section 4.1.3 examines whether they hold over all datasets.

The remainder of this section investigates each of these findings in more detail. Given the variety of stored experiments, queries often lead to unexpected observations, stimulating further study. In that respect, ExpDBs are a great tool for exploring learning behavior.

4.1.2 Investigating parameter effects

First, we examine the effect of the parameters of the RBF kernel, using the query in Fig. 13. Building on our first query (Fig. 9), we *zoom in* on these results by adding two constraints: the algorithm should be an SVM[20] and contain an RBF kernel. Next, we select the value of the 'gamma' parameter (kernel width) of that kernel. We also relax the constraint on the dataset by including three more datasets, and ask for the number of features in each dataset.

The result is shown in Fig. 13 (right). First, note that the variation on the letter dataset (Fig. 12) is indeed explained by the effect of this parameter. We also see that its effect on other datasets is markedly different: on some datasets, performance increases until reaching an optimum and then slowly declines, while on other datasets, performance decreases slowly up to a point, after which it quickly drops to default accuracy, i.e., the SVM is simply predicting the majority class. This behavior seems to correlate with the number of features in each dataset (shown in brackets), which we will investigate further in Sect. 4.2.1.

4.1.3 General comparisons

Previous queries studied the performance of algorithms under rather specific conditions. Yet, by simply dropping the dataset constraints, the query will return results over a large number of different datasets. Furthermore, instead of only considering predictive accuracy, we can select a range of evaluation functions, and use a normalization technique used by Caruana and Niculescu-Mizil (2006): normalize all performance metrics between the baseline performance and the best observed performance over all algorithms on each dataset. Using the

[20] Alternatively, we could ask for a specific implementation, i.e., 'implementation=weka.SMO'.

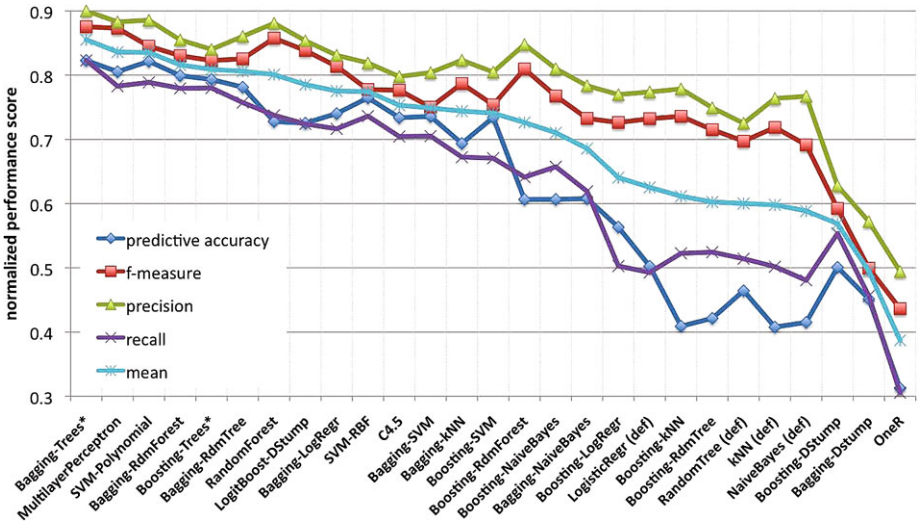


Fig. 14 Ranking of algorithms over all datasets and over different performance metrics

aggregation functions of SQL, we can do this normalization on the fly, as part of the query. We won't print any more queries here, but they can be found in full on the database website.

As such, we can perform general comparisons of supervised learning algorithms. We select all UCI datasets and all algorithms whose parameters were varied (see Sect. 3.3.2) and, though only as a point of comparison, logistic regression, nearest neighbors (kNN), naive Bayes and RandomTree with their default parameter settings. As for the performance metrics, we used predictive accuracy, F-measure, precision and recall, the last three of which were averaged over all classes within a particular dataset. We then queried for the maximal (normalized) performance of each algorithm for each metric on each dataset, averaged each of these scores over all datasets, and finally ranked all classifiers by the average of predictive accuracy, precision and recall.[21] The results of this query are shown in Fig. 14.

Taking care not to overload the figure, we compacted groups of similar and similarly performing algorithms, indicated with an asterix (*). The overall best performing algorithms are mostly bagged and boosted ensembles. Especially bagged and boosted trees (including C4.5, PART, Ripper, Ridor, NaiveBayesTree, REPTree and similar tree-based learners), and SVM and MultilayerPerceptron perform very well, in agreement with the results reported by Caruana and Niculescu-Mizil (2006). Another shared conclusion is that boosting full trees performs dramatically better than boosting stumps (see Boosting-DStump) or boosting random trees.

However, one notable difference is that C4.5 performs slightly better than Random-Forests, though only for predictive accuracy, not for any of the other measures. A possible explanation lies in the fact that performance is averaged over both binary and multi-class datasets: because some algorithms perform much better on binary than on multi-class datasets, we can expect some differences. It is easy to investigate this effect: we add the

[21] Because all algorithms were evaluated over all of the datasets (with 10-fold cross-validation), we could not optimize their parameters on a separate calibration set for this comparison. To limit the effect of overfitting, we only included a limited set of parameter settings, all of which are fairly close to the default setting. Nevertheless, these results should be interpreted with caution as they might be optimistic.

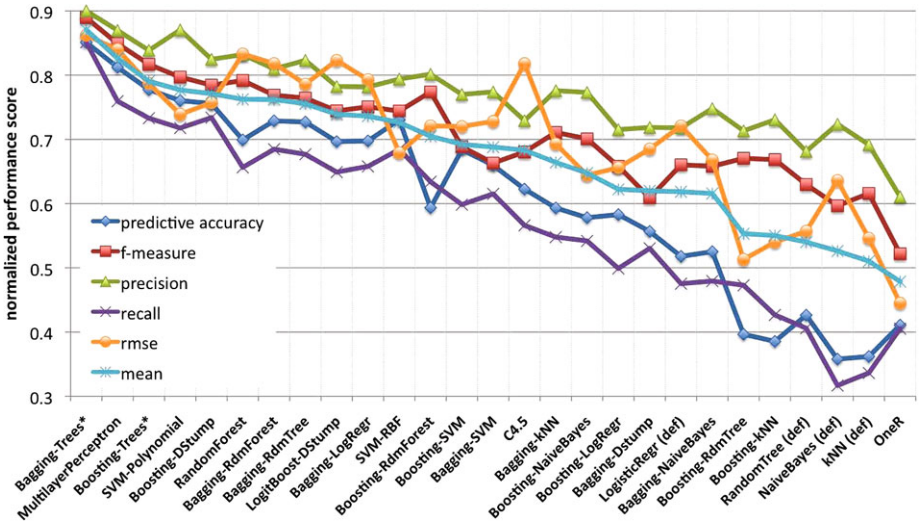


Fig. 15 Ranking of algorithms over all binary datasets and over different performance metrics

constraint that datasets should be binary, which also allows us to include another evaluation function: root mean squared error (RMSE). This yields Fig. 15.

On the 35 binary datasets, RandomForest outperforms C4.5 on all evaluation functions, while bagged and boosted trees are still at the top of the list. Note that boosted stumps score higher on binary datasets than they do on non-binary ones.

Furthermore, since this study contains many more algorithms, we can make a number of additional observations. In Fig. 14, for instance, the bagged versions of most strong learners (SVM, C4.5, RandomForest, etc.) don't seem to improve much compared to the original base-learners with optimized parameters. Apparently, tuning the parameters of these strong learners has a much larger effect on performance than bagging or boosting. On binary datasets, the relative performances of most algorithms seem relatively unaffected by the choice of metric, except perhaps for RMSE.

We can also check whether our observations from Sect. 4.1.1 still hold over multiple datasets and evaluation metrics. First, averaged over all datasets, the polynomial kernel performs better than the RBF kernel in SVMs. Also contrary to what was observed earlier, bagging does generally improve the performance of logistic regression, while boosting still won't. The other observations hold: boosting (high-bias) naive Bayes classifiers is much better than bagging them, bagging (high-variance) random trees is dramatically better than boosting them, and whereas boosting trees is generally beneficial, boosting SVMs and RandomForests is not. This is further evidence that boosting stops early on these algorithms, whereas pruning mechanisms in tree learners avoid overfitting and thus allow boosting to perform many iterations.

Finally, note that although this is a comprehensive comparison of learning algorithms, each such comparison is still only a snapshot in time. However, as new algorithms, datasets and experiments are added to the database, one can at any time rerun the query and immediately see how things have evolved.

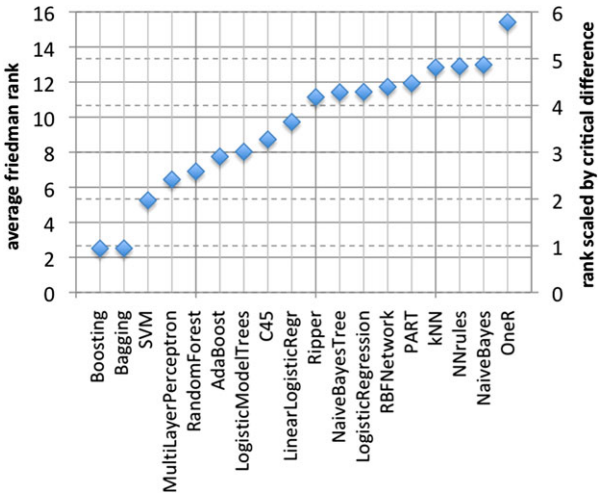


Fig. 16 Average rank, general algorithms

4.1.4 Ranking algorithms

In some applications, one prefers to have a ranking of learning approaches, preferably using a statistical significance test. This can also be written as a query in most databases. For instance, to investigate whether some algorithms consistently rank high over various problems, we can query for their average rank (using each algorithm's optimal observed performance) over a large number of datasets. Figure 16 shows the result of a single query over all UCI datasets, in which we selected 18 algorithms to limit the amount of statistical error generated by using 'only' 87 datasets. To check which algorithms perform significantly different, we used the Friedman test, as discussed in Demsar (2006). The right axis shows the average rank divided by the *critical difference*, meaning that two algorithms perform significantly different if the average ranks of two algorithms differ by at least one unit on that scale. The critical difference was calculated using the Nemenyi test (Demsar 2006) with $p = 0.1$, 18 algorithms and 87 datasets.

This immediately shows that indeed, some algorithms rank much higher on average than others over a large selection of UCI datasets. Boosting and bagging, if used with the correct base-learners, perform significantly better than SVMs, and SVMs in turn perform better than C4.5. We cannot yet say that SVMs are better than MultilayerPerceptrons or RandomForests: more datasets (or fewer algorithms in the comparison) are needed to reduce the critical difference. Note that the average rank of bagging and boosting is close to two, suggesting that a (theoretical) meta-algorithm that reliably chooses between the two approaches and the underlying base-learner would yield a very high rank. Indeed, rerunning the query while joining bagging and boosting yields an average rank of 1.7, down from 2.5.

Of course, to be fair, we should again differentiate between different base-learners and kernels. We can drill down through the previous results by adjusting the query, additionally asking for the base-learners and kernels involved, yielding Fig. 17. Bagged Naive Bayes trees seem to come in first, but the difference is not significant compared to that of SVMs with a polynomial kernel (although it is compared to the RBF kernel). Also note that, just as in Sect. 4.1.1, bagging and boosting PART and NBTrees yields big performance boosts on all these datasets, whereas boosting random trees will, in general, not improve performance.

In any such comparison, it is important to keep the No Free Lunch theorem (Wolpert 2001) in mind: if all possible data distributions are equally likely, "... for any algorithm,

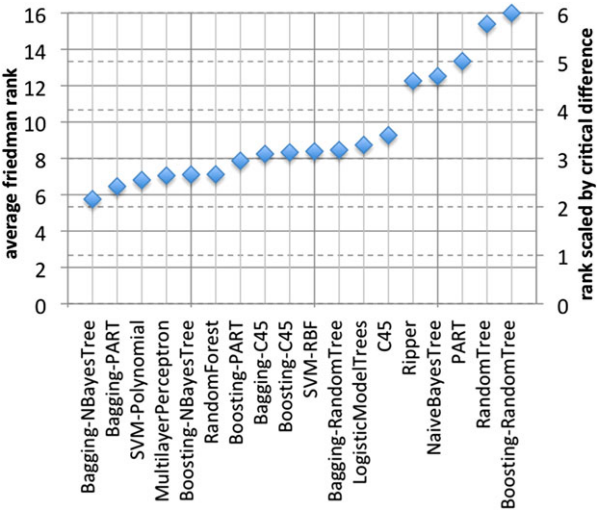


Fig. 17 Average rank, specific algorithm setups

any elevated performance over one class of problems is exactly paid for in performance over another class." Even if method A is better than method B across a variety of datasets, such as the UCI datasets in Fig. 17, this could be attributed to certain properties of those datasets, and results may be very different over a group of somehow different datasets. An interesting avenue of research would be to repeat these queries on various collections of datasets, with different properties or belonging to specific tasks, to investigate such dependencies.

4.2 Data-level analysis

While the queries in the previous section allow a detailed analysis of learning performance, they give no indication of exactly *when* (on which kind of datasets) a certain behavior is to be expected. In order to obtain results that generalize over different datasets, we need to look at the properties of individual datasets, and investigate how they affect learning performance.

4.2.1 Data property effects

In a first such study, we examine whether the 'performance jumps' that we noticed with the Random Forest algorithm in Fig. 12 are linked to the dataset size. Querying for the performance and size (number of trees) of all random forests on all datasets, ordered from small to large, yields Fig. 18.

On most datasets, performance increases as we increase the number of trees (darker labels), usually leveling off between 33 and 101 trees. One dataset, in the middle of Fig. 18, is a notable exception. A single random tree achieves less than 50% accuracy on this binary dataset, such that voting over many such trees actually increases the chance of making the wrong prediction. More importantly, we see that as the dataset size increases, the accuracies for a given forest size vary less because trees become more stable on large datasets, eventually causing clear performance jumps on very large datasets. Contrarily, on small datasets, the benefit of using more trees is overpowered by the randomness in the trees. All this illustrates that even quite simple queries can give a detailed picture of an algorithm's behavior, showing the combined effects of algorithm parameters and data properties.

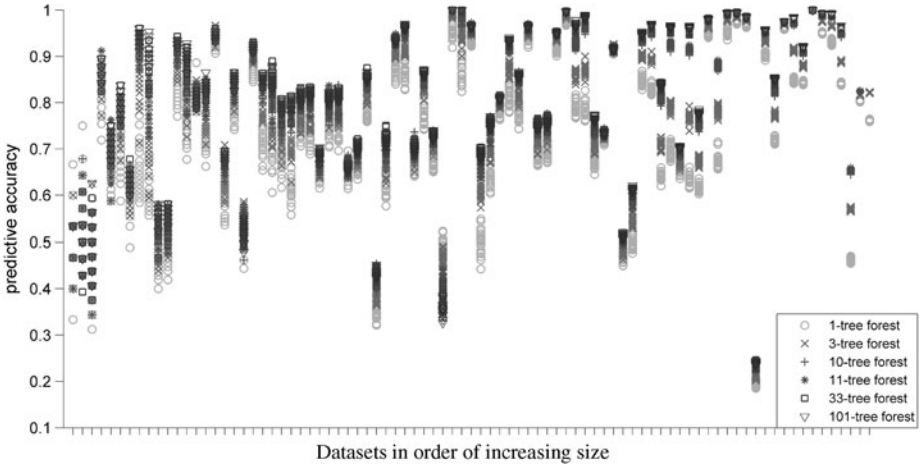
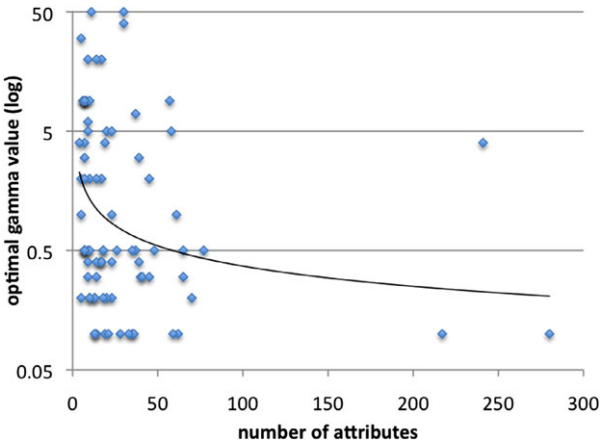


Fig. 18 Performance of random forests for different forests sizes (*darker labels* represent larger forests) on all datasets, ordered from small to large. Dataset names are omitted because there are too many to print legibly

Fig. 19 Number of attributes vs. optimal gamma



A second effect we can investigate is whether the optimal value for the gamma-parameter of the RBF-kernel is indeed linked to the number of attributes in the dataset. After querying for the relationship between the gamma-value corresponding with the optimal performance and the number of attributes in the dataset used, we get Fig. 19.

Although the number of attributes and the optimal gamma-value are not directly correlated, it looks as though high optimal gamma values predominantly occur on datasets with a small number of attributes, also indicated by the fitted curve. A possible explanation for this is that WEKA's SVM implementation normalizes all attributes into the interval [0,1]. Therefore, the maximal squared distance between two examples, $\sum (a_i - b_i)^2$ for every attribute i, is equal to the number of attributes. Because the RBF kernel computes $e^{(-\gamma * \sum (a_i - b_i)^2)}$, the kernel value will go to zero quickly for large gamma-values and a large number of attributes, making the non-zero neighborhood around a support vector small. Consequently, the SVM will overfit these support vectors, resulting in low accuracies. This suggests that the RBF

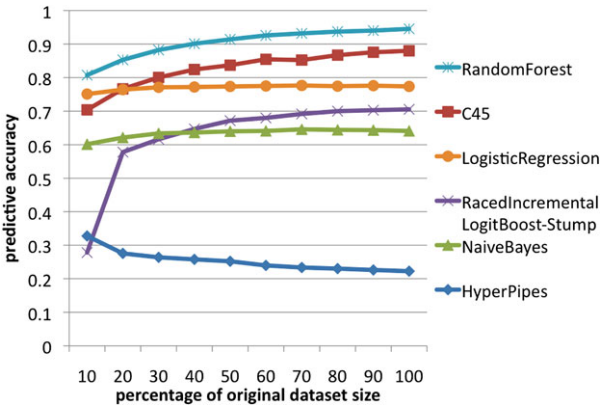


Fig. 20 Learning curves on the Letter-dataset

kernel should take the number of attributes into account to make the default gamma value more suitable across a range of datasets.

This finding illustrates how experiment databases can assist in algorithm development, and that it is important to describe algorithms at the level of individual implementations, in order to link performance results to the exact underlying procedures.

4.2.2 Preprocessing effects

The database also stores workflows with preprocessing methods, and thus we can investigate their effect on the performance of learning algorithms. For instance, when querying for workflows that include a downsampling method, we can draw learning curves by plotting learning performance against sample size, as shown in Fig. 20. From these results, it is clear that the ranking of algorithm performances depends on the size of the sample: the curves cross. While logistic regression is initially stronger than C4.5, the latter keeps improving when given more data, confirming earlier analysis by Perlich et al. (2003). Note that RandomForest performs consistently better for all sample sizes, that RacedIncrementalLogitBoost crosses two other curves, and that HyperPipes actually performs worse when given more data, which suggests that its initially higher score was largely due to chance.

4.2.3 Mining for patterns in learning behavior

As shown in Fig. 1, another way to tap into the stored information is to use data mining techniques to automatically model the effects of many different data properties on an algorithm's performance. For instance, when looking at Fig. 14, we see that OneR performs much worse than the other algorithms. Earlier studies, most notably one by Holte (1993), found little performance difference between OneR and the more complex C4.5. To study this discrepancy in more detail, we can query for the default performance of OneR and J48 (a C4.5 implementation) on all UCI datasets, and plot them against each other, as shown in Fig. 21(a). This shows that on many datasets, the performances are indeed similar (crossing near the diagonal), while on many others, J48 is the clear winner. Note that J48's performance never drops below 50%, making it much more useful in ensemble methods than OneR, which is also clear from Fig. 14. Interestingly, if we add the constraint that only datasets published at the time of these earlier studies can be used, the dominance of J48 is much less pronounced.

To model the circumstances under which J48 performs better than OneR, we first discretize these results into three classes as shown in Fig. 21(a): "draw", "win_J48" (4% to

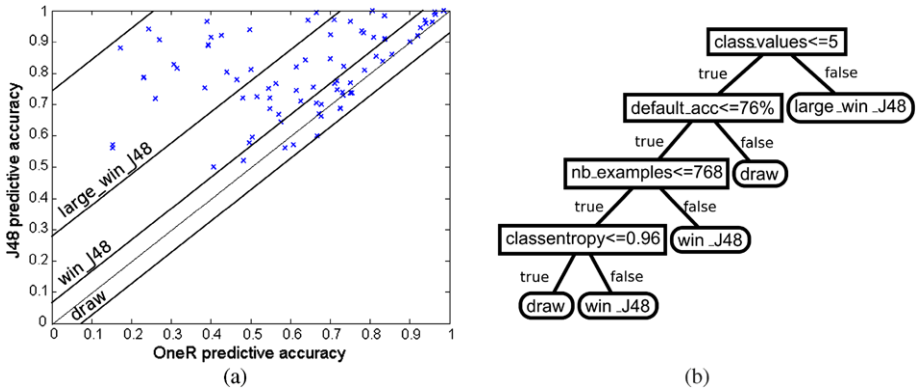


Fig. 21 **(a)** J48's performance against OneR's for all datasets, discretized into 3 classes. **(b)** A meta-decision tree predicting algorithm superiority based on data characteristics

20% gain), and "large_win_J48" (20% to 70% gain). We then extend the query by asking for all stored characteristics of the datasets used, and train a meta-decision tree on the returned data, predicting whether the algorithms will draw, or how large J48's advantage will be (see Fig. 21(b)). From this we learn that J48 has a clear advantage on datasets with many class values. This can be explained by the fact that OneR bases its prediction only on the most predictive attribute. Thus, if there are more classes than there are values for the attribute selected by OneR, performance will necessarily suffer. Another short query tells us that indeed, the datasets published after Holte (1993) generally had more class values.

4.3 Method-level analysis

While the results in the previous section are clearly more generalizable towards the datasets used, they only consider individual algorithms and do not generalize over different techniques. Hence, we need to include the stored algorithm properties in our queries as well.

4.3.1 Bias-variance profiles

One very interesting algorithm property is its bias-variance profile. Because the database contains a large number of bias-variance decomposition experiments, we can give a realistic numerical assessment of how capable each algorithm is in reducing bias and variance error. Figure 22 shows, for each algorithm, the proportion of the total error that can be attributed to bias error, calculated according to Kohavi and Wolpert (1996), using default parameter settings and averaged over all datasets. The algorithms are ordered from large bias (low variance), to low bias (high variance). NaiveBayes is, as expected, one of the algorithms whose error consists primarily of bias error, whereas RandomTree has relatively good bias management, but generates more variance error than NaiveBayes. When looking at the ensemble methods, Fig. 22 shows that bagging is a variance-reduction method, as it causes REPTree to shift significantly to the left. Conversely, boosting reduces bias, shifting DecisionStump to the right in AdaBoost and LogitBoost (additive logistic regression).

4.3.2 Bias-variance effects

As a final study, we investigate the claim by Brain and Webb (2002) that on large datasets, the bias-component of the error becomes the most important factor, and that we should

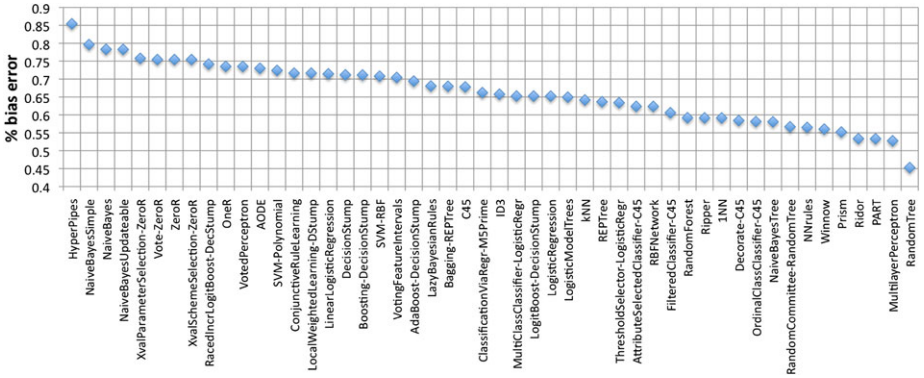


Fig. 22 The average percentage of bias-related error for each algorithm averaged over all datasets

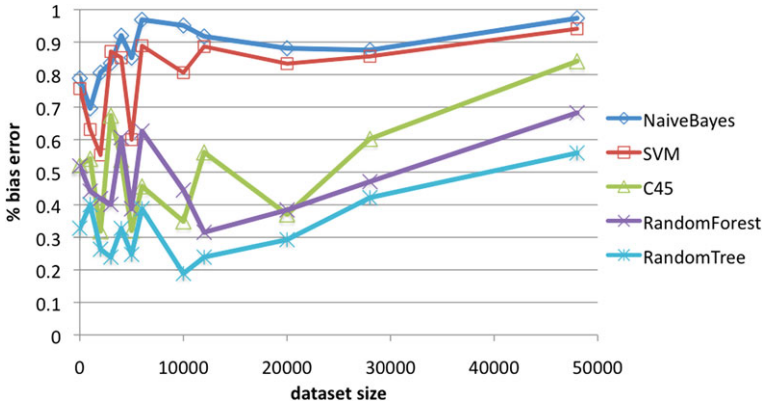


Fig. 23 The average percentage of bias-related error in algorithms as a function of dataset size

use algorithms with good bias management to tackle them. To verify this, we look for a connection between the dataset size and the proportion of bias error in the total error of a number of algorithms, using the previous figure to select algorithms with very different bias-variance profiles. Averaging the bias-variance results over datasets of similar size for each algorithm produces the result shown in Fig. 23. It shows that bias error is of varying significance on small datasets, but steadily increases in importance on larger datasets, for all algorithms. This validates the previous study on a larger set of datasets. In this case (on UCI datasets), bias becomes the most important factor on datasets larger than 50000 examples, no matter which algorithm is used. As such, it is indeed advisable to look to algorithms with good bias management when dealing with large datasets.

Remark Many more types of queries could be written to delve deeper into the available experimental results. Moreover, we have typically selected only a few variables in each query, and more advanced visualizations should be tried to analyze higher-dimensional results.

5 Future work

In many ways, the experiment database system presented here is only a proof of concept. Many avenues of further work exist. First, the ontology should be extended toward other tasks, such as clustering, graph mining and data stream mining. Furthermore, we did not investigate in depth how to handle other types of experiments: simulation experiments and experiments on data streams will probably require quite different database models than the ones used here. Still, the same experimental principles apply. The main differences lie in the evaluation techniques, evaluation functions and the structure of the experiment workflow. It should thus be possible to extend the ontology in these areas, and to translate these changes into appropriate XML definitions and databases allowing the queries most useful in those domains. Regarding reinforcement learning, lessons can be learned from the RL logbook project.[22] Probably the greatest caveat is that we do not yet store models in a way that allows us to write queries about their structure. Here, many lessons may be learned from inductive databases to remedy this (Imielinski and Mannila 1996; Fromont et al. 2007).

5.1 An improved experimental methodology

That said, there are many ways in which these experiment databases could be directly used in contemporary machine learning research. In it simplest form, it serves as a personal log ensuring that researchers can reproduce and reuse their own experiments at a later point in time. Still, the problem of generalizability remains. In the current methodology, experiments are designed to test specific hypotheses, often under many constraints (e.g., default parameter settings). Testing whether these observations are generally valid requires the laborious design of new experiments to measure each possible effect.

However, as we have illustrated in this paper, experiment databases also enable an improved methodology: design experiments such that they measure the performance of algorithms under as many different conditions as possible, store the results, and query the database to test certain hypotheses. While this requires more experiments, we can launch one query after another about every possible aspect of the algorithm. Moreover, these queries will return results that are generally valid (under the constraints listed in the query), such that we can be confident that we interpreted the results correctly. Finally, such general experiments are likely to be useful in further research, and the more researchers do this type of general experimentation, the greater the probability that we can simply download a large portion of the required experiments from a public repository.

5.2 Building a collaboration infrastructure

Thus, sharing experiments boosts the value of experimental results. The network effect of many researchers contributing their own findings will paint an increasingly detailed picture of learning performance, and we will be able to reuse ever more prior results, perhaps even perform some studies without running any new experiments at all. Moreover, mining the database for patterns in algorithm behavior will also become ever more feasible, possibly leading to unexpected new insights. Even if only a few popular data mining tools would enable their users to export their experiments to the current ExpDB, or any other public repository, this would be tremendously useful for machine learning research.

[22] http://logbook.rl-community.org/.

Some problems remain. First, tools and algorithms may be buggy and generate faulty results. In some e-Science platforms, experiments are annotated with a 'trustworthiness' ranking, reflecting whether results were verified or whether they are disputed (e.g., the experiment cannot be repeated). Another issue is that some research uses proprietary data, which can only be shared in local databases, not public ones. However, these experiments can still be useful to the community if sufficient meta-data about the datasets is provided (see Sect. 1.4), or when, similar to the MLcomp service (see Sect. 2.2), it is possible to evaluate algorithms on a server, using a trusted procedure, without making the data itself public.

5.3 Statistical significance tests

Another improvement would be to add more support for statistical significance testing (Dietterich 1998; Demsar 2006). In the current database, we need to include such tests manually in the query (as we did in Sect. 4.1.4), or do them as an extra step after downloading the returned results. Moreover, given the number of returned results, we may need include adjustments, e.g., for multiple comparisons (Jensen and Cohen 2000). To offer better assistance, these tests should be integrated in query interfaces, or in specialized query languages.

5.4 Towards automated experimentation

Finally, running a query on an ExpDB will only return *known* experiments, which may not be sufficient to conclusively answer a given question. Ideally, the ExpDB would run additional experiments when needed. Further research is required to automatically generate experiments based on queries. For instance, active learning may be used to select the most important remaining experiments given the ones that are already stored, or new query languages may be designed that answer common research questions by running experiments.

6 Conclusions

Experiment databases are databases specifically designed to collect all the details on large numbers of experiments, performed and shared by many different researchers, and make them immediately available to everyone. They ensure that experiments are repeatable and automatically organize them such that they can be easily reused in future studies.

In this paper, we have introduced a principled framework for the design of such experiment databases for machine learning, modeled after similar repositories that are actively used in other sciences. First, we establish a controlled vocabulary for experiment description by building an open ontology for data mining experiments, which is then translated to an extensible experiment description language for free experiment exchange, and a detailed database model for storing and querying all shared results. We follow a collaborative approach, facilitating participation of the community in applying and developing them further.

Using a pilot experiment database, focused on supervised classification, we found that such databases indeed enable fast and thorough analysis of learning algorithms based on a myriad of collected results. In a series of increasingly in-depth studies, we performed elaborate comparisons and rankings of supervised classification algorithms and investigated the effects of algorithm parameters and data properties, Moreover, we suggested algorithm improvements, generated meta-models of algorithm performance, and gained novel insight into the bias-variance profiles of learning algorithms.

While several directions for future work still remain open, experiment databases offer the possibility to truly unite the results of machine learning studies all over the world, enhance cooperation, and facilitate empirical studies that used to be extremely laborious or prohibitively expensive to run from scratch, while making it convenient to thoroughly investigate the ensuing results. As such, we are confident that they can contribute greatly to the vigor of machine learning research.

Our experiment database is available online at http://expdb.cs.kuleuven.be/.

Acknowledgements Hendrik Blockeel was a Postdoctoral Fellow of the Fund for Scientific Research—Flanders (Belgium) (F.W.O.-Vlaanderen) at the time of this work, and this research is further supported by GOA 2003/08 "Inductive Knowledge Bases" and F.W.O.-Vlaanderen G.0108.06 "Foundations of Inductive Databases for Data Mining". We further acknowledge the support of the K.U. Leuven High Performance Computing center and BiG Grid, the Dutch e-Science Grid, supported by the Netherlands Organization for Scientific Research, NWO. We like to thank Larisa Soldatova and Pance Panov for many fruitful discussions on ontology design, and the editor and reviewers for their useful comments and suggestions.

References

Aha, D. (1992). Generalizing from case studies: a case study. In *Proceedings of the international conference on machine learning (ICML)* (pp. 1–10).

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, MA, Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., & Sherlock, G. (2000). Gene ontology: tool for the unification of biology. *Nature Genetics*, *25*, 25–29.

Asuncion, A., & Newman, D. J. (2007). *UCI machine learning repository*. University of California, School of Information and Computer Science.

Ball, C. A., Brazma, A., Causton, H. C., & Chervitz, S. (2004). Submission of microarray data to public repositories. *PLoS Biology*, *2*(9), e317.

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning*, *36*(1–2), 105–139.

Blockeel, H. (2006). Experiment databases: A novel methodology for experimental research. *Lecture Notes in Computer Science*, *3933*, 72–85.

Blockeel, H., & Vanschoren, J. (2007). Experiment databases: towards an improved experimental methodology in machine learning. *Lecture Notes in Computer Science*, *4702*, 6–17.

Bradford, J., & Brodley, C. (2001). The effect of instance-space partition on significance. *Machine Learning*, *42*, 269–286.

Brain, D., & Webb, G. (2002). The need for low bias algorithms in classification learning from large data sets. *Lecture Notes in Artificial Intelligence*, *2431*, 62–73.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Metalearning: applications to data mining*. Berlin: Springer.

Brazma, A., Hingamp, P., Quackenbush, J., Sherlock, G., Spellman, P., Stoeckert, C., Aach, J., Ansorge, W., Ball, C., Causton, H. C., Gaasterland, T., Glenisson, P., Holstege, F., Kim, I., Markowitz, V., Matese, J., Parkinson, H., Robinson, A., Sarkans, U., Schulze-Kremer, S., Stewart, J., Taylor, R., & Vingron, J. (2001). Minimum information about a microarray experiment. *Nature Genetics*, *29*, 365–371.

Brown, D., Vogt, R., Beck, B., & Pruet, J. (2007). High energy nuclear database: a testbed for nuclear data information technology. In *Proceedings of the international conference on nuclear data for science and technology*, article 250.

Carpenter, J. (2011). May the best analyst win. *Science*, *331*(6018), 698–699.

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the international conf. on machine learning* (pp. 161–168).

Chandrasekaran, B., & Josephson, J. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, *14*(1), 20–26.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.
Derriere, S., Preite-Martinez, A., & Richard, A. (2006). UCDs and ontologies. *ASP Conference Series*, *351*, 449.
Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1923.
Frawley, W. (1989). The role of simulation in machine learning research. In *Proceedings of the annual symposium on simulation (ANSS)* (pp. 119–127).
Fromont, E., Blockeel, H., & Struyf, J. (2007). Integrating decision tree learning into inductive databases. *Lecture Notes in Computer Science*, *4747*, 81–96.
Hall, M. (1998). *Correlation-based feature selection for machine learning*. PhD Thesis, Waikato University.
Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18.
Hand, D. (2006). Classifier technology and the illusion of progress. *Statistical Science*, *21*(1), 1–14.
Hilario, M., & Kalousis, A. (2000). Building algorithm profiles for prior model selection in knowledge discovery systems. *Engineering Intelligent Systems*, *8*(2), 956–961.
Hilario, M., Kalousis, A., Nguyen, P., & Woznica, A. (2009). A data mining ontology for algorithm selection and meta-mining. In *Proceedings of the ECML-PKDD'09 workshop on service-oriented knowledge discovery* (pp. 76–87).
Hirsh, H. (2008). Data mining research: Current status and future opportunities. *Statistical Analysis and Data Mining*, *1*(2), 104–107.
Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–91.
Hoste, V., & Daelemans, W. (2005). Comparing learning approaches to coreference resolution. There is more to it than bias. In *Proceedings of the ICML'05 workshop on meta-learning* (pp. 20–27).
Imielinski, T., & Mannila, H. (1996). A database perspective on knowledge discovery. *Communications of the ACM*, *39*(11), 58–64.
Jensen, D., & Cohen, P. (2000). Multiple comparisons in induction algorithms. *Machine Learning*, *38*, 309–338.
Keogh, E., & Kasetty, S. (2003). On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge Discovery*, *7*(4), 349–371.
Kietz, J., Serban, F., Bernstein, A., & Fischer, S. (2009). Towards cooperative planning of data mining workflows. In *Proceedings of the ECML-PKDD'09 workshop on service-oriented knowledge discovery* (pp. 1–12).
King, R., Rowland, J., Oliver, S., Young, M., Aubrey, W., Byrne, E., Liakata, M., Markham, M., Pir, P., Soldatova, L. N., Sparkes, A., Whelan, K. E., & Clare, A. (2009). The automation of science. *Science*, *324*(5923), 85–89.
Kohavi, R., & Wolpert, D. (1996). Bias plus variance decomposition for zero-one loss functions. In *Proceedings of the international conference on machine learning* (pp. 275–283).
Leake, D., & Kendall-Morwick, J. (2008). Towards case-based support for e-science workflow generation by mining provenance. *Lecture Notes in Computer Science*, *5239*, 269–283.
Manolescu, I., Afanasiev, L., Arion, A., Dittrich, J., Manegold, S., Polyzotis, N., Schnaitter, K., Senellart, P., & Zoupanos, S. (2008). The repeatability experiment of SIGMOD 2008. *ACM SIGMOD Record*, *37*(1), 39–45.
Michie, D., Spiegelhalter, D., & Taylor, C. (1994). *Machine learning, neural and statistical classification*. Ellis Horwood: Chichester.
Morik, K., & Scholz, M. (2004). The MiningMart approach to knowledge discovery in databases. In N. Zhong & J. Liu (Eds.), *Intelligent technologies for information analysis* (pp. 47–65). Berlin: Springer.
Nielsen, M. (2008). The future of science: building a better collective memory. *APS Physics*, 17(10).
Ochsenbein, F., Williams, R. W., Davenhall, C., Durand, D., Fernique, P., Hanisch, R., Giaretta, D., McGlynn, T., Szalay, A., & Wicenec, A. (2004). VOTable: tabular data for the Virtual Observatory. In Q. Peter & G. Krzysztof (Eds.), *Toward an international virtual observatory* (Vol. 30, pp. 118–123). Berlin: Springer.
Panov, P., Soldatova, L. N., & Džeroski, S. (2009). Towards an ontology of data mining investigations. *Lecture Notes in Artificial Intelligence*, *5808*, 257–271.
Pedersen, T. (2008). Empiricism is not a matter of faith. *Computational Linguistics*, *34*, 465–470.
Perlich, C., Provost, F., & Simonoff, J. (2003). Tree induction vs. logistic regression: a learning-curve analysis. *Journal of Machine Learning Research*, *4*, 211–255.
Pfahringer, B., Bensusan, H., & Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In *Proceedings of the international conference on machine learning (ICML)* (pp. 743–750).
De Roure, D., Goble, C., & Stevens, R. (2009). The design and realisation of the myExperiment virtual research environment for social sharing of workflows. *Future Generations Computer Systems*, *25*, 561–567.

Salzberg, S. (1999). On comparing classifiers: a critique of current research and methods. *Data Mining and Knowledge Discovery*, *1*, 1–12.

Schaaff, A. (2007). Data in astronomy: from the pipeline to the virtual observatory. *Lecture Notes in Computer Science*, *4832*, 52–62.

Soldatova, L., & King, R. (2006). An ontology of scientific experiments. *Journal of the Royal Society Interface*, *3*(11), 795–803.

Sonnenburg, S., Braun, M., Ong, C., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Muller, K., Pereira, F., Rasmussen, C., Ratsch, G., Scholkopf, B., Smola, A., Vincent, P., Weston, J., & Williamson, R. (2007). The need for open source software in machine learning. *Journal of Machine Learning Research*, *8*, 2443–2466.

Stoeckert, C., Causton, H. C., & Ball, C. A. (2002). Microarray databases: standards and ontologies. *Nature Genetics*, *32*, 469–473.

Szalay, A., & Gray, J. (2001). The world-wide telescope. *Science*, *293*, 2037–2040.

van Someren, M. (2001). Model class selection and construction: beyond the procrustean approach to machine learning applications. *Lecture Notes in Computer Science*, *2049*, 196–217.

Vanschoren, J., & Blockeel, H. (2008). Investigating classifier learning behavior with experiment databases. *Studies in Classification, Data Analysis, and Knowledge Organization*, *5*, 421–428.

Vanschoren, J., Pfahringer, B., & Holmes, G. (2008). Learning from the past with experiment databases. *Lecture Notes in Artificial Intelligence*, *5351*, 485–492.

Vanschoren, J., Blockeel, H., Pfahringer, B., & Holmes, G. (2009). Organizing the world's machine learning information. *Communications in Computer and Information Science*, *17*(12), 693–708.

Vizcaino, J., Cote, R., Reisinger, F., Foster, J., Mueller, M., Rameseder, J., Hermjakob, H., & Martens, L. (2009). A guide to the Proteomics Identifications Database proteomics data repository. *Proteomics*, *9*(18), 4276–4283.

Wojnarski, M., Stawicki, S., & Wojnarowski, P. (2010). TunedIT.org: system for automated evaluation of algorithms in repeatable experiments. *Lecture Notes in Computer Science*, *6086*, 20–29.

Wolpert, D. (2001). The supervised learning no-free-lunch theorems. In *Proceedings of the online world conference on soft computing in industrial applications* (pp. 25–42).

Yasuda, N., Mizumoto, Y., Ohishi, M., O'Mullane, W., Budavari, T., Haridas, V., Li, N., Malik, T., Szalay, A., Hill, M., Linde, T., Mann, B., & Page, C. (2004). Astronomical data query language: simple query protocol for the virtual observatory. *ASP Conference Series*, *314*, 293.

Záková, M., Kremen, P., Zelezný, F., & Lavrač, N. (2008). Planning to learn with a knowledge discovery ontology. In *Proceedings of the ICML/UAI/COLT'08 workshop on planning to learn* (pp. 29–34).