

# The experiment database for machine learning (Demo)

Joaquin Vanschoren<sup>1</sup>

**Abstract.** We demonstrate the use of the experiment database for machine learning, a community-based platform for the sharing, reuse, and in-depth investigation of the thousands of machine learning experiments executed every day. It is aimed at researchers and practitioners of data mining techniques, and is publicly available at <http://expdb.cs.kuleuven.be>. This demo gives a hands-on overview of how to share novel experimental results, how to integrate the database in existing data mining toolboxes, and how to query the database through an intuitive graphical query interface.

## 1 Introduction

Experimentation is the lifeblood of machine learning (ML) research. A considerable amount of effort and resources are invested in assessing the usefulness of new algorithms, finding the optimal approach for new applications or just to gain some insight into, for instance, the effect of a parameter. Yet in spite of all these efforts, experimental results are often discarded or forgotten shortly after they are obtained, or at best averaged out to be published, which again limits their future use. If we could collect all these ML experiments in a central resource and make them publicly available in an organized (searchable) fashion, the combined results would provide a highly detailed picture of the performance of algorithms on a wide range of data configurations, speeding up ML research.

In this paper, we demonstrate a community-based platform designed to do just this: the *experiment database for machine learning*. 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, 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 stored.

### 1.1 Meta-learning

Instead of being purely empirical, these 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 [7] and landmarks [10], while algorithms can be tagged by model properties, the average ratio of bias or variance error, or their sensitivity to noise [3].

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.

### 1.2 Overview of benefits

We can summarize the benefits of this platform as follows:

**Reproducibility** The database stores all details of the experimental setup, resulting in 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.

**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.

**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 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.

The remainder of this paper is organized as follows. Sect. 2 outlines how we constructed our pilot experiment database and the underlying models and languages that enable the free exchange of experiments. In Sect. 3, we demonstrate how it can be used to quickly discover new insights into a wide range of research questions and to verify prior studies. Sect. 4 concludes.

---

<sup>1</sup> LIACS, Leiden University, The Netherlands, email: [joaquin@liacs.nl](mailto:joaquin@liacs.nl)

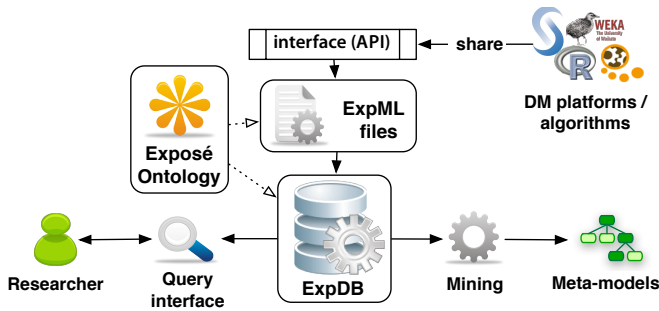


Figure 1. Components of the experiment database framework.

## 2 Framework description

In this section, we outline the design of this collaborative framework, outlined in Fig. 1. 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. Full versions of the ontologies, languages and database models discussed below will be available 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:

**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. 2.3.2).

**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 [1].

### 2.1 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.

#### 2.1.1 Collaborative Ontology Design

Several other useful ontologies are being developed in parallel: OntoDM [8] is a top-level ontology for data mining concepts, EXPO [11] models scientific experiments, DMOP [4] describes learning algorithms (including their internal mechanisms and models) and workflows, and the KD ontology [13] and eProPlan ontology [5]

describe large arrays of DM operators, including information about their use to support automatic workflow planning.

To streamline ontology development, a ‘core’ ontology was defined, and an open ontology development forum was created: the Data Mining Ontology (DMO) Foundry<sup>2</sup>. 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 concepts and relationships from established top-level scientific ontologies: BFO,<sup>3</sup> OBI,<sup>4</sup> IAO,<sup>5</sup> and RO.<sup>6</sup> We often use subproperties, e.g. *implements* for *concretizes*, and *runs* for *realizes*, to reflect common usage in the field. Exposé is designed to integrate or be similar to the above mentioned ontologies, but focusses on aspects related to experimental evaluation.

#### 2.1.2 Top-level View

Fig. 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 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 sub-workflows, 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. 1.1), which are useful to interpret results afterwards. Finally, algorithms, functions or parameters can play certain *roles* in a complex setup: an algorithm can sometimes act as a *base-learner* in an ensemble algorithm, and a dataset can act as a *training set* in one experiment and as a *test set* in the next.

<sup>2</sup> The DMO Foundry: <http://dmo-foundry.org>

<sup>3</sup> The Basic Formal Ontology (BFO): <http://www.ifomis.org/bfo>

<sup>4</sup> The Ontology for Biomedical Investigations (OBI): <http://obi-ontology.org>

<sup>5</sup> The Information Artifact Ontology (IAO): <http://bioportal.bioontology.org/ontologies/40642>

<sup>6</sup> The Relation Ontology (RO): <http://www.obofoundry.org/ro>

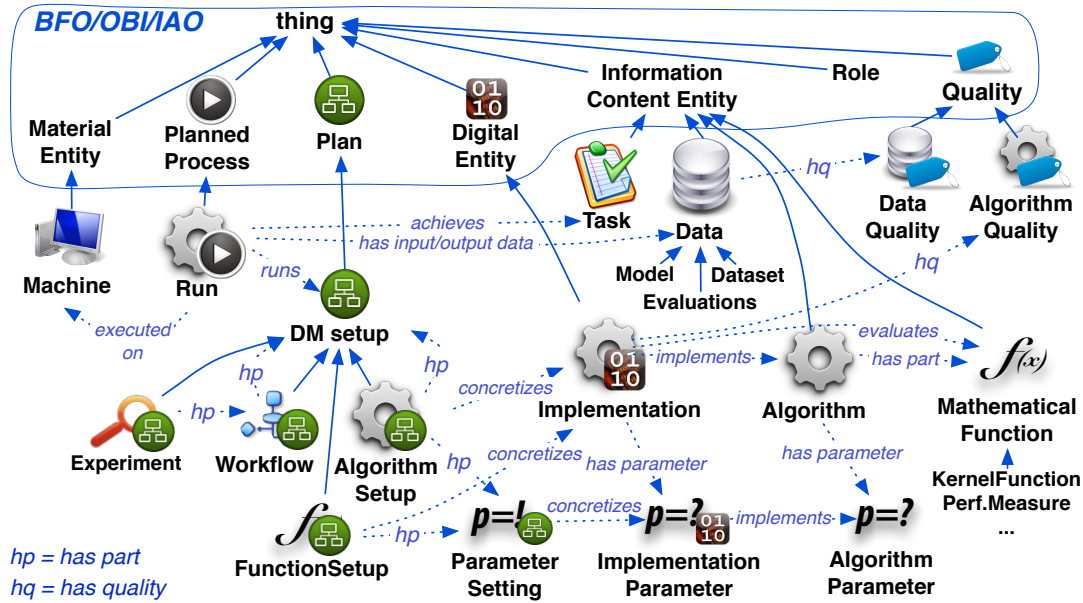


Figure 2. An overview of the top-level concepts in the Exposé ontology.

### 2.1.3 Experiments

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.

One experiment run may generate several workflow runs (with different input values), and a workflow run may consist of smaller algorithm runs. *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, the level of detail is chosen by the experimenter. Especially for complex workflows, it might be interesting to afterwards query the results of certain sub-runs.

## 2.2 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<sup>7</sup> 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. Below, we show a small example of ExpML output to illustrate our modeling of data mining workflows.

### 2.2.1 Workflow Runs

Fig. 3 shows a *workflow run* in ExpML, executed in WEKA [2] and exported through the aforementioned API, and a schematic representation is shown in Fig. 4. 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 Figure 3 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.

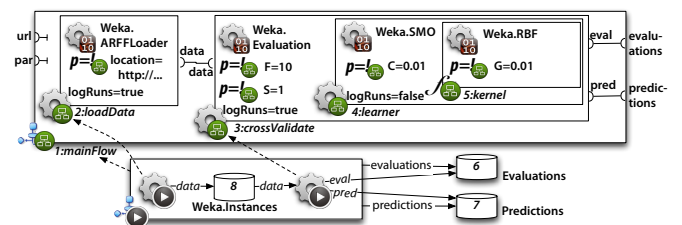


Figure 4. A schematic representation of the run.

<sup>7</sup> Resource Description Framework: <http://www.w3.org/RDF>

```

<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>
  <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>
</Workflow>
<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"/>
    ... </Evaluations>
  </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>

```

Figure 3. A workflow run in ExpML.

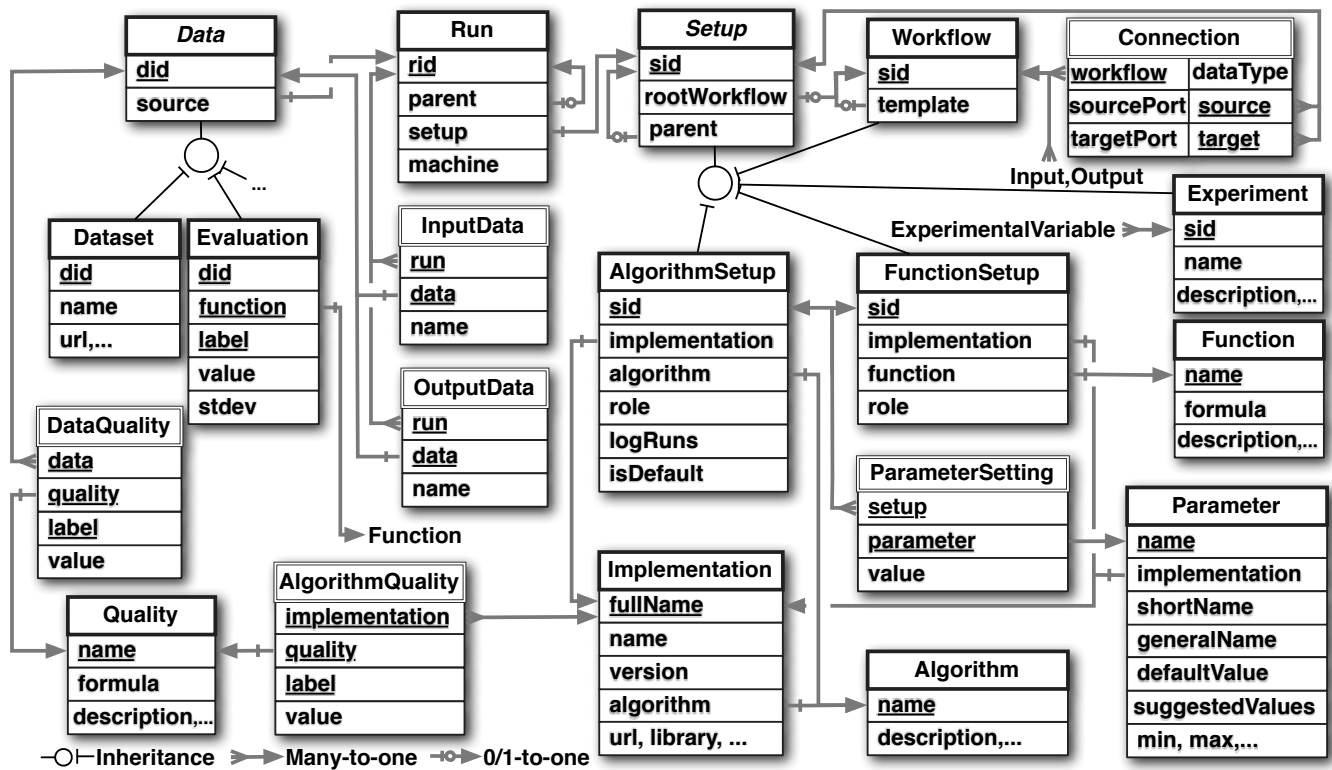


Figure 5. The general structure of the experiment database. Underlined columns indicate primary keys, the arrows denote foreign keys. Tables in italics are abstract: their fields only exist in child tables.

## 2.3 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 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.

### 2.3.1 Anatomy of an Experiment Database

Fig. 5 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. `AlgorithmSetups` and `FunctionSetups` can have `ParameterSettings`, a specific `Implementation` and a general `Algorithm` or `Function`. `Implementations` and `Datasets` can also have `Qualities`, stored in `AlgorithmQuality` and `DataQuality`, respectively. `Data`, runs and setups have unique id's, while algorithms, functions, parameters and qualities have unique names defined in Exposé.

### 2.3.2 Accessing the Experiment Database

The experiment database is available at <http://expdb.cs.kuleuven.be>. A graphical query interface is provided (see the examples below) that hides the complexity of the database, but still supports most types of queries. In addition, it is possible to run standard SQL queries (a library of example queries is available. Several video tutorials help the user to get started quickly. We are currently updating the database, query interface and submission system, and a public submission interface for new experiments (described in ExpML) will be available shortly.

## 3 Example Queries

In this section, we illustrate the use of the experiment database.<sup>8</sup> In doing this, we aim to take advantage of the theoretical information stored with the experiments to gain deeper insights.

### 3.1 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. In the graphical query interface, see Fig. 6, this can be done by starting with the `CrossValidation` node, which will be connected to the input `Dataset`, the outputted `Evaluations` and the underlying `Learner` (algorithm setup). Green nodes represent data, blue nodes are setups and white nodes are qualities (runs are hidden). By clicking a node it can be expanded to include other parts of the workflow setup (see below). For instance, 'Learner' expands into the underlying implementation, parameter settings, base-learners and sub-functions (e.g. kernels). By clicking a node one can also add a selection (in green, e.g. the used learning algorithm) or a constraint (in red, e.g. a preferred evaluation function). The user is always given

<sup>8</sup> See [12] for a much more extensive list of possible queries

a list of all available options, in this case a list of all evaluation functions present in the database. Here, we choose a specific input dataset and a specific evaluation function, and we aim to plot the evaluation value against the used algorithm.

Running the query returns all known experiment results, which are scatterplotted in Fig. 7, 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. Each of them can be explored by adding further constraints.

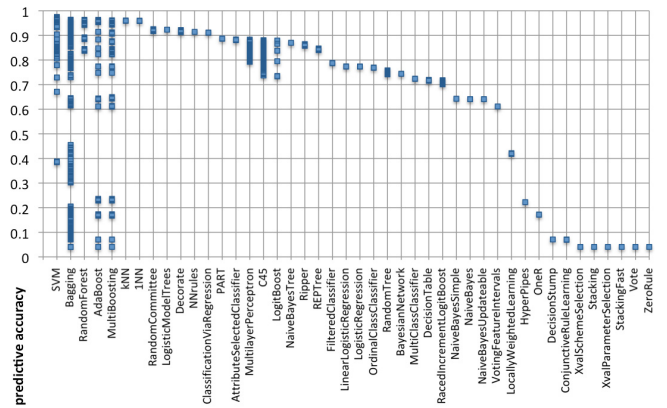


Figure 7. Performance of all algorithms on dataset 'letter'.

### 3.2 Investigating Parameter Effects

For instance, we can examine the effect of the used kernel, or even the parameters of a given kernel. Building on our first query, we *zoom in* on these results by adding two constraints: the algorithm should be an SVM<sup>9</sup> 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. 10. First, note that much of the variation seen for SVMs on the 'letter' dataset (see Fig. 7) 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). Further study shows that some SVM implementations indeed tend to overfit on datasets with many attributes [12].

### 3.3 Preprocessing Effects

The database also stores workflows with preprocessing methods, and thus we can investigate their effect on the performance of learning

<sup>9</sup> Alternatively, we could ask for a specific implementation, i.e., 'implementation=weka.SMO'.



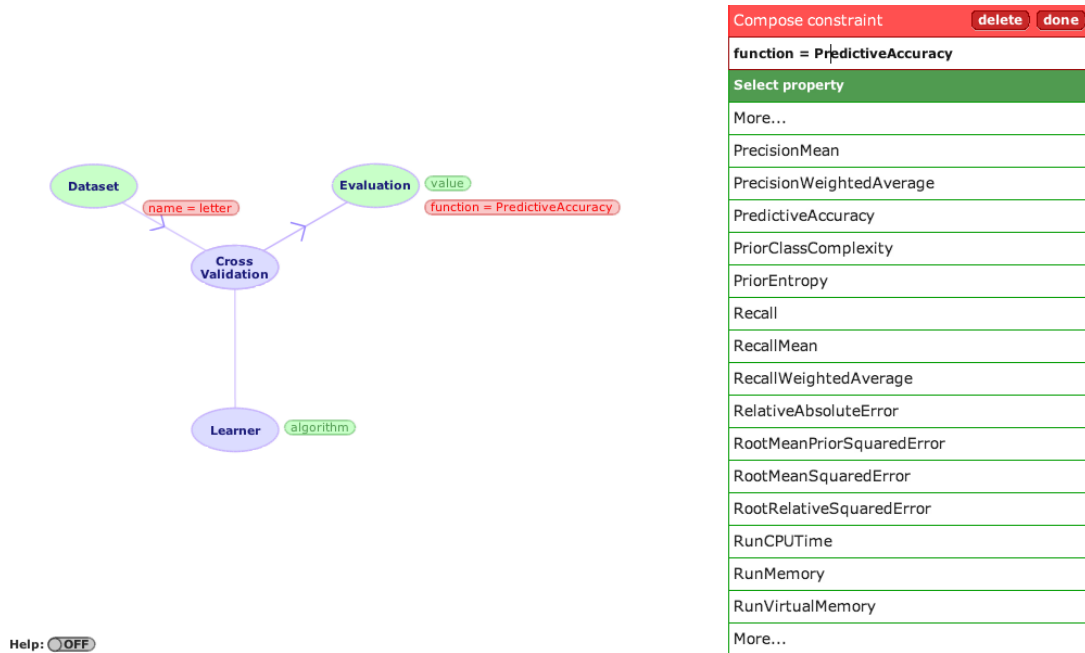


Figure 6. The graphical query interface.

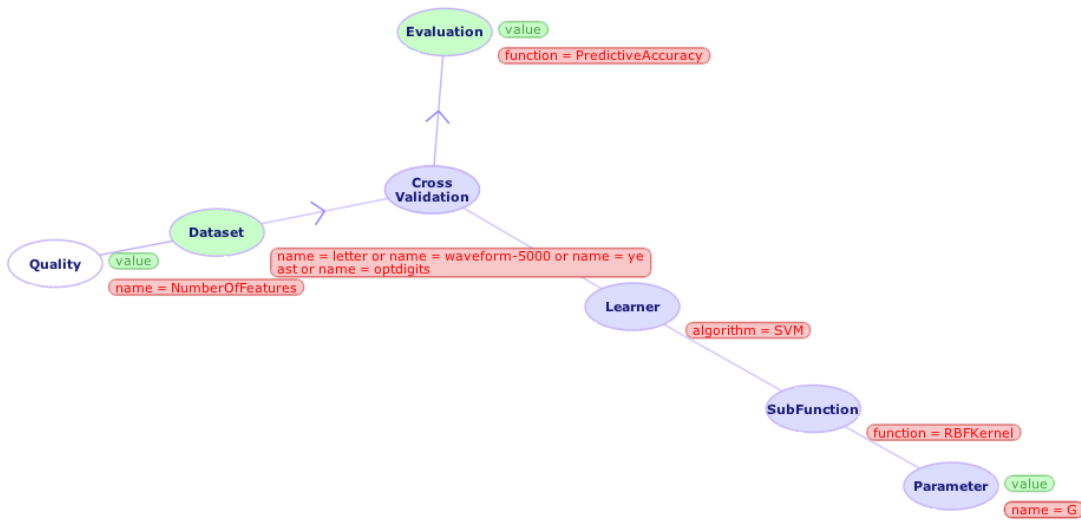


Figure 8. Querying the performance of SVMs with different kernel widths on datasets of different dimensionalities.

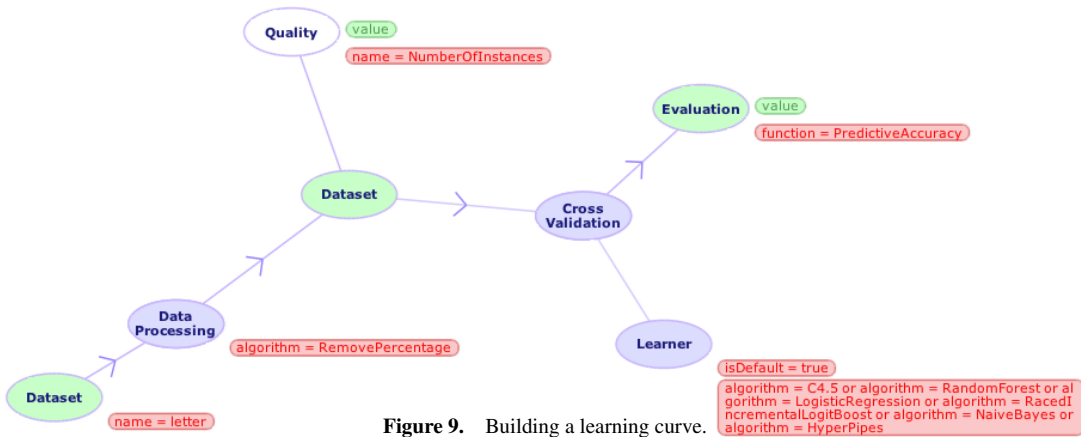
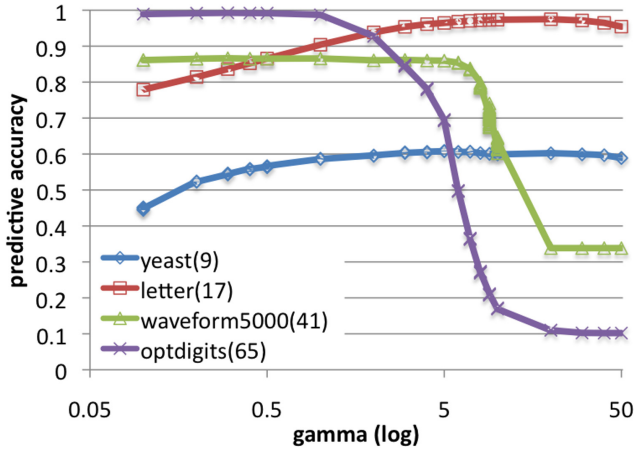
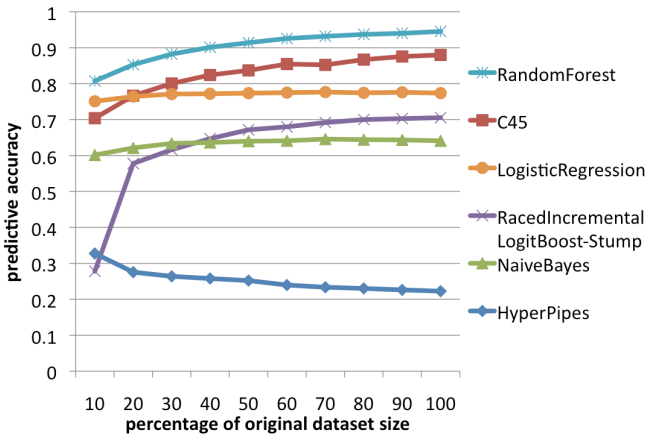


Figure 9. Building a learning curve.



**Figure 10.** The effect of parameter gamma of the RBF kernel in SVMs on a number of different datasets (number of attributes shown in brackets).

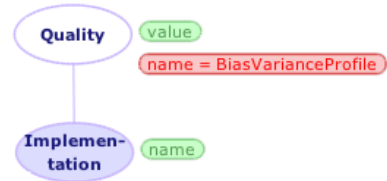


**Figure 11.** Learning curves on the Letter-dataset.

algorithms. For instance, when querying for workflows that include a downsampling method, we can draw learning curves by plotting learning performance against sample size. Fig. 9 shows the query: a preprocessing step is added and we query for the resulting number of instances, and the performance of a range of learning algorithms (with default parameter settings). The result is shown in Fig. 11. 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 [9]. 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.

### 3.4 Bias-Variance Profiles

The database also stores a series of algorithm properties, many of them calculated based on large numbers of experiments. One interesting algorithm property is its bias-variance profile. Because the



**Figure 12.** Query for the bias-variance profile of algorithms.

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. Fig. 13 shows, for each algorithm, the proportion of the total error that can be attributed to bias error, calculated according to [6], using default parameter settings and averaged over all datasets. The simple query is shown in Fig. 12. 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 RandomForest has relatively good bias management, but generates more variance error than NaiveBayes. When looking at the ensemble methods, Fig. 13 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).

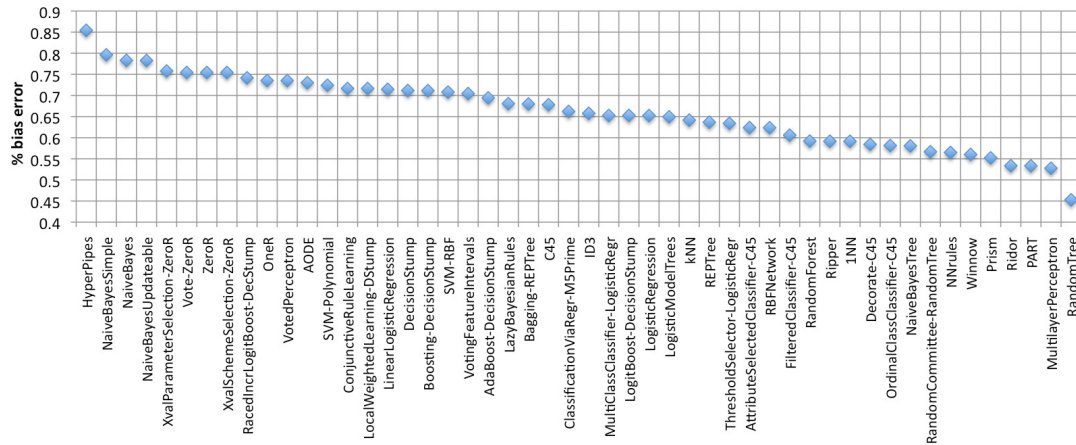
### 3.5 Further queries

These are just a few examples the queries that can be answered using the database. Other queries allow algorithm comparisons using multiple evaluation measures, algorithm rankings, statistical significance tests, analysis of ensemble learners, and especially the inclusion of many more dataset properties and algorithm properties to study how algorithms are affected by certain types of data. Please see [12] and the database website for more examples.

## 4 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.

This demo paper gives an overview of the design of the framework, the underlying ontologies, and the resulting data exchange formats and database structures. It discusses how these can be used to share novel experimental results, to integrate the database in existing data mining toolboxes, and how to query the database through an intuitive graphical query interface. By design, the database also calculates and stores a wide range of known or measurable properties of datasets and algorithms. 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. This results in a great resource for meta-learning and its applications.



**Figure 13.** The average percentage of bias-related error for each algorithm averaged over all datasets.

## Acknowledgements

We acknowledge the support of BigGrid, the Dutch e-Science Grid, supported by the Netherlands Organisation for Scientific Research, NWO. We like to thank Larisa Soldatova and Pance Panov for many fruitful discussions on ontology design.

## REFERENCES

- [1] P Brazdil, C Giraud-Carrier, C Soares, and R Vilalta, 'Metalearning: Applications to data mining', Springer, (2009).
- [2] MA Hall, E Frank, G Holmes, B Pfahringer, P Reutemann, and IH Witten, 'The WEKA data mining software: An update', *SIGKDD Explorations*, **11**(1), 10–18, (2009).
- [3] M Hilario and A Kalousis, 'Building algorithm profiles for prior model selection in knowledge discovery systems', *Engineering Intelligent Systems*, **8**(2), 956–961, (2000).
- [4] M Hilario, A Kalousis, P Nguyen, and A Woznica, 'A data mining ontology for algorithm selection and meta-mining', *Proceedings of the ECML-PKDD'09 Workshop on Service-oriented Knowledge Discovery*, 76–87, (2009).
- [5] J Kietz, F Serban, A Bernstein, and S Fischer, 'Towards co-operative planning of data mining workflows', *Proceedings of the ECML-PKDD'09 Workshop on Service-oriented Knowledge Discovery*, 1–12, (2009).
- [6] R Kohavi and D Wolpert, 'Bias plus variance decomposition for zero-one loss functions', *Proceedings of the International Conference on Machine Learning (ICML)*, 275–283, (1996).
- [7] D Michie, D Spiegelhalter, and C Taylor, 'Machine learning, neural and statistical classification', Ellis Horwood, (1994).
- [8] P Panov, LN Soldatova, and S Džeroski, 'Towards an ontology of data mining investigations', *Lecture Notes in Artificial Intelligence*, **5808**, 257–271, (2009).
- [9] C Perlich, F Provost, and J Simonoff, 'Tree induction vs. logistic regression: A learning-curve analysis', *Journal of Machine Learning Research*, **4**, 211–255, (2003).
- [10] B Pfahringer, H Bensusan, and C Giraud-Carrier, 'Meta-learning by landmarking various learning algorithms', *Proceedings of the International Conference on Machine Learning (ICML)*, 743–750, (2000).
- [11] LN Soldatova and RD King, 'An ontology of scientific experiments', *Journal of the Royal Society Interface*, **3**(11), 795–803, (2006).
- [12] J Vanschoren, H Blockeel, B Pfahringer, and G Holmes, 'Experiment databases: A new way to share, organize and learn from experiments', *Machine Learning*, **87**(2), (2012).
- [13] M Zakova, P Kremen, F Zelezny, and N Lavrac, 'Planning to learn with a knowledge discovery ontology', *Proceedings of the ICML/UAI/COLT'08 Workshop on Planning to Learn*, 29–34, (2008).