Organizing the World's Machine Learning Information

Joaquin Vanschoren[1], Hendrik Blockeel[1], Bernhard Pfahringer[2], and Geoff Holmes[2]

[1] Computer Science Dept., K.U.Leuven, Leuven, Belgium
[2] Computer Science Dept., University of Waikato, Hamilton, New Zealand

Abstract. All around the globe, thousands of learning experiments are being executed on a daily basis, only to be discarded after interpretation. Yet, the information contained in these experiments might have uses beyond their original intent and, if properly stored, could be of great use to future research. In this paper, we hope to stimulate the development of such learning experiment repositories by providing a bird's-eye view of how they can be created and used in practice, bringing together existing approaches and new ideas. We draw parallels between how experiments are being curated in other sciences, and consecutively discuss how both the empirical and theoretical details of learning experiments can be expressed, organized and made universally accessible. Finally, we discuss a range of possible services such a resource can offer, either used directly or integrated into data mining tools.

1 Introduction

Research in machine learning and exploratory data analysis are, to a large extent, guided by the collection and interpretation of performance evaluations of machine learning algorithms. As such, studies in this area comprise extensive experimental evaluations, analyzing the performance of many algorithms on many datasets, or many preprocessed versions of the same dataset. Unfortunately, the results of these experiments are usually interpreted with a single focus of interest, and their details are usually lost after publication or simply not publicly accessible.

Sharing this information with the world would greatly benefit research in these fields, fostering the reuse of previously obtained results for additional and possibly much broader investigation. To realize this in practice, we examine how to collect learning experiments in public repositories and, more importantly, how to organize all this information so that it is both easily accessible and useful. The former implies that the repository should be searchable, allowing easy retrieval of specific results. To achieve the latter, results should be kept in context, relating them to known theoretical properties of the included methods and datasets.

One could envisage many creative uses of such a resource. In machine learning research, pooling the results of many studies would significantly increase the amount of available experimental data, enabling much larger studies aimed

at finding fundamental insights into the dynamics of learning processes and offering well-founded answers to open questions. Also, when developing new learning approaches or enhancements, algorithms are often evaluated against the same benchmark datasets. This means many experiments are needlessly repeated, while the cost of setting up and running them often limits the range of datasets and parameter settings that can be explored. When reusing prior experimental results, algorithms can be compared in more depth by running them instead under a wider range of conditions, thus yielding more generalizable results. Furthermore, meta-level information about algorithms and datasets available in the repository puts those results in context, and can, for instance, be used to investigate how different data properties affect algorithm performance. Finally, it offers a forum for negative results and an easy way to check which approaches have been tried before and what they achieved.

Conversely, in exploratory data analysis, practitioners faced with a specific problem will try different preprocessing and modeling techniques to gain a deeper understanding of the data at hand. In this case, even though each dataset is unique, a searchable repository of previous experiments could be used to build on previous experience and check which methods might be particularly useful. For instance, one might check whether logistic regression is feasible on data with many attributes, and thus whether a feature selection step might be useful. Furthermore, large collections of experimental data could be used to search for similar datasets and the methods that were particularly successful on it, to find meta-rules describing the usability of a method, or to provide training data for data mining assistance tools.

The concept of experiment repositories has been introduced earlier [3, 4]. In this paper, we aim to provide a bird's-eye view of their possibilities and the challenges that need to be addressed before their full potential can be exploited. For those challenges that have been discussed before, we offer pointers to the available literature and suggest improvements. For those that are new, we propose solutions, but also point out directions for further research.

In Section 2 of this paper, we first look at existing approaches towards building experiment repositories in various scientific disciplines. Next, in Section 3, we propose a common experiment description language to allow for learning experiments to be shared freely, and in Section 4, we discuss how all this information can be automatically organized in a searchable database. Finally, Section 5 uses the resulting experiment repository to show how it can assist the development and application of learning algorithms, either by using it directly as an online service, or by integrating it into data mining tools. Section 6 concludes.

2 Previous Work

Many scientific disciplines store experimental data as a means of collaboration between different research groups or to make sure experiments are not needlessly repeated, most notably in fields like high-energy physics where experiments are expensive. Still, most fields lack common standards for experiment description.

2.1 Bioinformatics

Bioinformatics has led the way in describing and collecting experimental data [5]. Probably the best known application can be found in the emergence of microarray databases[3] [14]. The need for reproducibility, as well as recognition of the potential value of microarray results beyond the summarized descriptions found in most papers, have led to the creation of public repositories of microarray data [6]. Submitting experimental data in these repositories has become a condition for publication in several journals [2].

In establishing common standards for describing microarray data, significant progress has been made to ensure that such data can be properly managed and shared. In particular, a set of guidelines was drawn up regarding the required Minimal Information About a Microarray Experiment (MIAME [5]), a MicroArray Gene Expression Markup Language (MAGE-ML [14]) was conceived so that data could be uniformly described and shared between projects, and an ontology (MO [14]) was designed to provide common descriptors required by MIAME for capturing core information about microarray experiments.

Other, more specific projects go even further. The Robot Scientist [13], a fully automated scientific discovery system, expresses all physical aspects of experiment execution and even describes the hypotheses that are under investigation and what has been learned from past experiments.

2.2 Machine Learning

Creating experiment repositories for machine learning inevitably calls for the development of similar standards to describe and share learning experiments. First, similar to the MIAME guidelines, learning experiment descriptions should at least contain the information needed to reproduce the experiment. Such a set of guidelines is described in [4], covering what should minimally be known about the algorithms, datasets and experimental procedures. It also proposes a database schema to store classification experiments, which we shall develop further in Sect. 4.1 to, in addition, capture preprocessing steps and to allow for a more flexible description of different learning tasks besides classification.

3 A Language for Sharing Machine Learning Information

To enable a free exchange of experimental results, it would be useful to have a common description language. As such, learning experiments could be described in a unified way, without having to know how they are physically stored, while allowing them to be automatically verified, uploaded to, retrieved from, and transfered between any existing experiment repository, even if these repositories are implemented differently or distributed geographically. However, as new learning approaches are being developed at a constant rate and new learning

[3] A microarray records the expression levels of thousands of genes.

tasks often put new twists on classical problems, such a format should easily extend to capture new types of learning experiments.

To further the development of such standards, we propose an XML-based markup language, dubbed ExpML, that can be used to describe most classification and regression experiments[4]. An important benefit of XML is that it is hierarchical and extensible. It adapts to experiments of various complexities by extending the description of any aspect of a learning experiment as much as needed. In this section, we first provide a formal definition of this language, after which we will illustrate it with an example.

3.1 ExpML Definition

The XML Schema Definition (XSD) below creates an XML vocabulary for describing machine learning experiments and governs which elements should appear, in what order, and the information they should contain. It ensures that experiments are uniquely defined, and that each of its elements (algorithms, kernels, (preprocessed) datasets, evaluation metrics, etc.) are described in sufficient detail. To capture both (theoretical) meta-information about the experiments and the (empirical) settings and results, it distinguishes between element *definitions* and element *instantiations*. Definitions make sure the element is uniquely defined and can hold known properties and descriptions, while instantiations declare a specific configuration, e.g. including specific parameter settings.

Main structure We highlight the most important parts of the language definition[5], full definitions are available online at `http://expdb.cs.kuleuven.be/`. As the following excerpt shows, each description starts with an arbitrary number of definitions.

```
<xs:element name="expml">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="definition" nobounds>
        <xs:complexType>
          <xs:choice>
            <xs:element name="algorithm" type="algorithmFull"/>
            <xs:element name="kernel" type="algorithmFull"/>
            <xs:element name="dataset" type="datasetFull"/>
            <xs:element name="preprocessor" type="algorithmFull"/>
            <xs:element name="evalmethod" type="algorithmFull"/>
            <xs:element name="metric" type="metricFull"/>
            <xs:element name="environment" type="environmentFull"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
```

Next, an arbitrary number of experiments may be defined, starting with the exact setup, i.e. which instantiations of algorithms, datasets and evaluation procedures are used. Next, we state the results of the evaluation, the predictions generated by the model for each target variable and the used computational environment[6]. Finally, experiments can be labeled to provide additional information.

[4] This language was used in practice to upload all experiments mentioned in Sect. 4.2.
[5] We use 'nobounds' as a shorthand for 'minOccurs="0" maxOccurs="unbounded"'.
[6] Although model description formats exist [8], they are not yet included here.

```
      <xs:element name="experiment" nobounds>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="setting">
              <xs:complexType>
                <xs:all>
                  <xs:element name="algorithm" type="algorithmInst"/>
                  <xs:element name="dataset" type="dataInst"/>
                  <xs:element name="evalmethod" type="evalMethodInst"/>
                </xs:all>
              </xs:complexType>
            </xs:element>
            <xs:element name="evaluation" type="evaluationType"/>
            <xs:element name="prediction" type="predictionType" maxOccurs="unbounded"/>
            <xs:element name="environment" type="xs:string"/>
            <xs:element name="label" type="nameValue" nobounds/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Definitions The definitions for new elements should allow them to be properly used and easily retrieved. This includes descriptions of an algorithm's parameters, whether and how a dataset was preprocessed, how an evaluation metric is calculated and details about the environment used. Most elements can also be annotated further using *properties*, such as dataset and algorithm characterizations or computational benchmarks. Moreover, the required attributes of the elements state the minimal information needed to ensure reproducibility, and whether the definition updates a previous one.

```
<xs:complexType name="algorithmFull">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="parameter" type="parameterFull" nobounds/>
      <xs:element name="property" type="propertyType" nobounds/>
    </xs:sequence>
    <xs:attributeGroup ref="algoInfoFull"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="parameterFull">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="default" type="xs:string"/>
      <xs:element name="property" type="propertyType" nobounds/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="datasetFull">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="preprocessor" type="preprocInst" nobounds/>
      <xs:element name="classindex" type="xs:integer" minOccurs="0"/>
      <xs:element name="property" type="propertyType" nobounds/>
    </xs:sequence>
    <xs:attributeGroup ref="dataInfo"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="metricFull">
  <xs:complexContent>
    <xs:all>
```

```
      <xs:element name="name" type="xs:string"/>
      <xs:element name="formula" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
    </xs:all>
  <xs:attribute name="isUpdate" type="xs:boolean"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="environmentFull">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="cpu" type="xs:string"/>
      <xs:element name="memory" type="xs:string"/>
      <xs:element name="property" type="propertyType" nobounds/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="isUpdate" type="xs:boolean"/>
  </xs:complexContent>
</xs:complexType>
<xs:attributeGroup name="algoInfo">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="libname" type="xs:string"/>
</xs:attributeGroup>
<xs:attributeGroup name="algoInfoFull">
  <xs:attributeGroup ref="algoInfo"/>
  <xs:attribute name="version" type="xs:string" use="required"/>
  <xs:attribute name="libversion" type="xs:string"/>
  <xs:attribute name="url" type="xs:anyURI" use="required"/>
  <xs:attribute name="isUpdate" type="xs:boolean"/>
</xs:attributeGroup>
<xs:attributeGroup name="dataInfo">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="url" type="xs:anyURI" use="required"/>
  <xs:attribute name="isUpdate" type="xs:boolean"/>
</xs:attributeGroup>
```

Instantiations Inside an experiment, instantiations describe a specific application of the defined elements. While their attributes point to the general definition, they additionally define the element's individual configuration. For algorithms, this includes setting parameters or meta-parameters, encapsulating other algorithms (base-learners) or kernels. Datasets, on the other hand, can be instantiated by a number of nested preprocessing steps, which in turn may have parameter settings as well, as does the evaluation method.

```
<xs:complexType name="algorithmInst">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="parameter" type="metaParInst" nobounds/>
    </xs:sequence>
    <xs:attributeGroup ref="algoInfo"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="dataInst">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="preprocessor" type="preprocInst" nobounds/>
      <xs:element name="classindex" type="xs:integer" minOccurs="0"/>
    </xs:sequence>
    <xs:attributeGroup ref="dataInfo"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="preprocInst">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="parameter" type="metaParInst" nobounds/>
    </xs:sequence>
```

```
    <xs:attributeGroup ref="algoInfo"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="evalMethodInst">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="parameter" type="nameValue" nobounds/>
    </xs:sequence>
    <xs:attributeGroup ref="algoInfo"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="nameValue">
  <xs:complexContent>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="metaParInst">
  <xs:complexContent>
    <xs:extension base="nameValue">
      <xs:sequence minOccurs="0">
        <xs:element name="algorithm" type="algorithmInst" minOccurs="0"/>
        <xs:element name="kernel" type="algorithmInst" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Results The result of an experiment encompasses the outcomes of an arbitrary selection of evaluations metrics (depending on the task), and predictions for each data instance. In the case of classification tasks, the latter may also hold probabilities for each class.

```
<xs:complexType name="evaluationType">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="metric" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required"/>
          <xs:attribute name="value" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="predictions">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="instance" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="prob" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="prediction" type="xs:string" use="required"/>
                <xs:attribute name="value" type="xs:string" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        <xs:attribute name="nr" type="xs:integer" use="required"/>
        <xs:attribute name="prediction" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="target" type="xs:string"/>
  <xs:complexContent>
</xs:complexType>
```

3.2 An Example Description

As an illustration, we could use this language to define a new algorithm, run it on a preprocessed classification problem, and store the generated results:

```
<algorithm name="Bagging" version="1.31.2.2" libname="weka"
     libversion="3.4.8" url="http://www.cs.waikato.ac.nz/ml/weka/"
     classpath="weka.classifiers.meta.Bagging">
  <parameter name="P">
    <description>Size of each bag as percentage of data set size</description>
    <default>100</default>
    <property name="suggested_min" value="20"/>
    ...
  </parameter>
  ...
  <property name="class" value="ensemble">
  <property name="handles_classification" value="true">
  ...
</algorithm>
...
<experiment>
  <setting>
    <algorithm name="Bagging" version="1.31.2.2" libname="weka">
      <parameter name="P" value="90"/>
      <parameter name="O" value="false"/>
      <parameter name="I" value="40"/>
      <parameter name="W" value="algorithm">
        <algorithm name="NaiveBayes" version="1.16" libname="weka"/>
      </parameter>
    </algorithm>
    <dataset name="pendigits-90%">
      <preprocessor name="RemovePercentage" version="1.3" libname="weka">
        <parameter name="P" value="10"/>
        <dataset name="pendigits" url="http://archive.ics.uci.edu/ml/">
          <classIndex>-1</classIndex>
        </dataset>
      </preprocessor>
      <classIndex>-1</classIndex>
    </dataset>
    <evalmethod name="CrossValidation" version="1.53" libname="weka"
         libversion="3.4.8">
      <parameter name="nbfolds" value="10"/>
      <parameter name="randomseed" value="1"/>
    </evalmethod>
  </setting>
  <evaluation>
    <metric name="build_cputime" value="5.67"/>
    <metric name="build_memory" value="17929416"/>
    <metric name="mean_absolute_error" value="0.030570337062541805"/>
    <metric name="root_mean_squared_error" value="0.15960607792291556"/>
    <metric name="predictive_accuracy" value="0.8570778748180494"/>
    <metric name="kappa" value="0.8411692914743762"/>
    <metric name="confusion_matrix" value="
        [[0,1,2,3,4,5,6,7,8,9],[1021,0,0,0,2,0,3,0,51,4],[1,883,...],...]]"/>
    ...
  </evaluation>
  <predictions target="0">
    <instance nr="00000" prediction="8">
      <prob prediction="0" value="1.8761967426234115E-5"/>
      ...
      <prob prediction="8" value="0.9991914442703987"/>
      <prob prediction="9" value="3.2190267582597184E-31"/>
    </instance>
    ...
  </predictions>
  <environment>machine14</environment>
  <label spec="type" value="classification"/>
</experiment>
```

In this case, we first added the 'Bagging' algorithm with all the necessary information, descriptions of its parameters and some basic properties. Next, we added an experiment using an instantiation of this algorithm with specific parameter values. Since this is an ensemble algorithm, one of these parameters encapsulates another (parameterless) algorithm, viz. 'NaiveBayes'. The dataset we investigate is 'pendigits', preprocessed by feeding it into the 'RemovePercentage' preprocessor with the stated parameters. The generated model is evaluated using the 10-fold cross-validation technique. The results of this evaluation are stated next, using several evaluation metrics, and are followed by predictions for all the instances, including the probabilities for each class.

3.3 Future Work

While this language is designed to capture a large variety of contemporary classification and regression experiments, it will still need to be developed further. First of all, learning algorithms definitions are still quite limited. It would be instrumental to develop an ontology that models various learning techniques and their relationships. Also, learning tasks in machine learning, such as clustering, link discovery and mining relational data are very different, and might require different 'flavours' of this basic language to suit their needs. Alternatively, an ontology of machine learning techniques could be envisaged to work towards a unified format. Interesting approaches towards building such an ontology can be found in [10] and [1]. Finally, although the language allows the description of sequences of preprocessing steps, more work is needed to capture more complex data mining workflows.

4 Organizing Experimental Data

For all this information to be accessible and useful, it still needs to be stored in an organized fashion. Inserting it into a database seems a good solution [3], allowing powerful query possibilities (SQL) and easy integration into software tools. Ideally, such databases would evolve with the description language to be able to capture future extensions. There may be several interconnected databases, using the common language to transfer stored experiments, or conversely, some databases may be set up locally for sensitive, or preliminary data.

In this section, we focus on designing a database conforming to the structure of the language described previously, thus capturing the basic organization of machine learning experiments. This leads to the schema shown in Fig. 1.

4.1 Anatomy of an Experiment Database

Basically, an `Experiment` consists of a `Learner` run on a `Dataset` using a certain `Machine`, and the resulting `Model` is evaluated with a certain `Evaluation Method`. These are depicted in Fig. 1 by the dashed lines. Each of these components can be defined and instantiated over several other tables. An `Experiment` is stored as a specific combination of component instances.

More specifically, a *learner instance* points to a learning algorithm (`Learner`), which can be characterized by any number of features, and its specific parameter settings. For `Ensemble` learners, parameters can point to other learner instances, and additional records are kept to facilitate querying. A dataset instance is defined by the original dataset and a number of preprocessing steps, which can in turn be described further, including all the involved parameters. The evaluation method (e.g. cross-validation) can also be instantiated. Finally, the evaluation results of each experiment are stored for each employed evaluation metric, and for predictive models, the (non-zero probability) predictions returned for each data instance are recorded as well.

This database is publicly accessible at `http://expdb.cs.kuleuven.be/`. The website also hosts ExpML definitions, the available tools for uploading experiments, a gallery of SQL queries (including the ones used in the next section), and a query interface including visualization tools for displaying returned results.

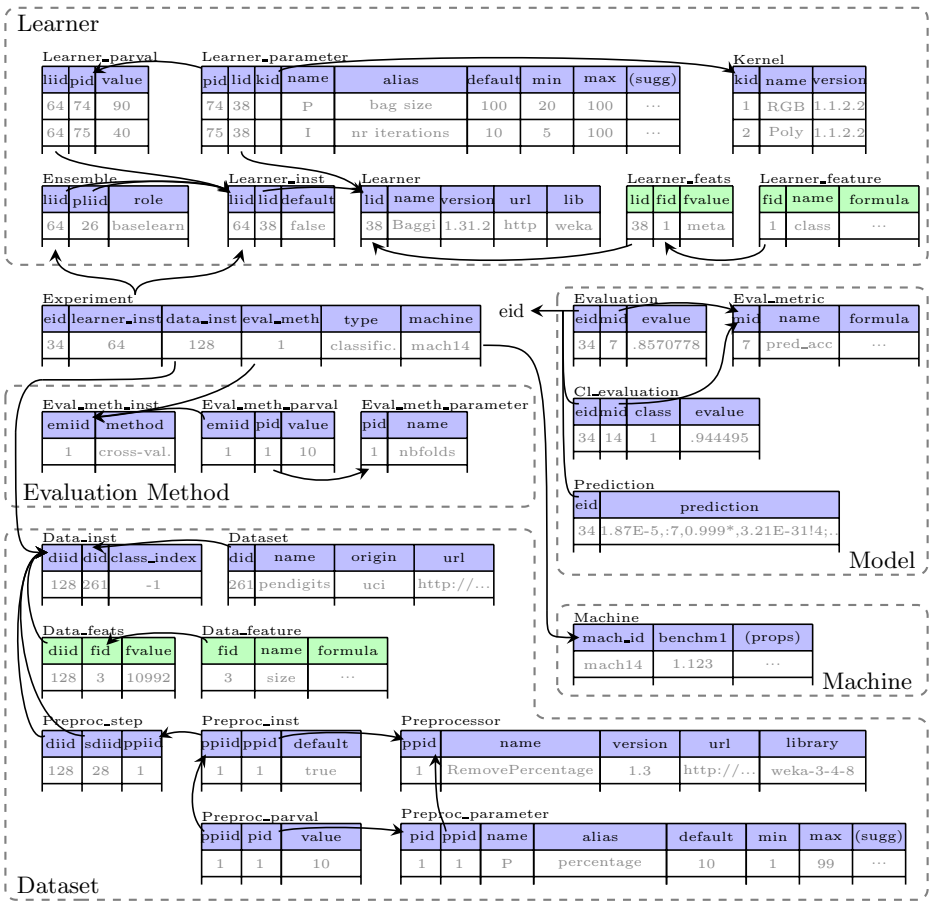

Fig. 1. A (simplified) schema for an experiment database.

4.2 Populating the Database

To fill this database with experiments, we focused on supervised classification. We extended the WEKA platform[16] to output experiments in the format described above, and developed an interface to the database to automatically interpret and store the experiments.

The repository currently holds about 500,000 experiments, using 54 well-known classification algorithms (from WEKA), 86 commonly used classification datasets taken from the UCI repository, and 2 preprocessing methods (also from WEKA). We ran all algorithms, with default parameter settings, on all datasets. Furthermore, the algorithms SMO (an SVM trainer), MultilayerPerceptron, J48 (C4.5), OneR, Random Forests, Bagging and Boosting were varied over their most important parameter settings[7]. For all randomized algorithms, each experiment was repeated 20 times with different random seeds. All experiments were evaluated with 10-fold cross-validation, using the same folds on each dataset, and a large subset was additionally evaluated with a bias-variance analysis.

5 Services of Experiment Repositories

The principled way of annotating algorithms, data, and entire experiments provides a much needed formal grounds for the development of data mining "web services" which allow on-demand retrieval of theoretical en empirical data about learning techniques, and which could then be automatically orchestrated into real data mining processes. In this section, we illustrate some of the possible services offered by an experiment database, either by directly querying the database, or by integrating it in larger data mining tools.

5.1 Public Database Access

All information stored in an experiment database can be accessed directly by writing the right database query (e.g. in SQL), providing a very versatile means to investigate a large number of experimental results, both under very specific and very general conditions. To further facilitate access to this information, a graphical query interface could hide the complexity of SQL queries. We focus here on the different services allowed by querying, a wider range of interesting queries is discussed in [15].

Experiment Reuse As mentioned earlier, when evaluating a new algorithm, one could use the repository to retrieve previously stored results. For instance, to query for the results of all previous algorithms on a specific dataset, we simply ask for all experiments on that dataset and select the algorithm used and the performance recorded. Fig. 2 shows the result on dataset "letter". It is immediately clear how previous algorithms performed, how much variation is caused by parameter tuning, and what the effect is of various ensemble techniques.

[7] For the ensemble methods, all non-ensemble learners were used as possible base-learners, each with default parameter settings.

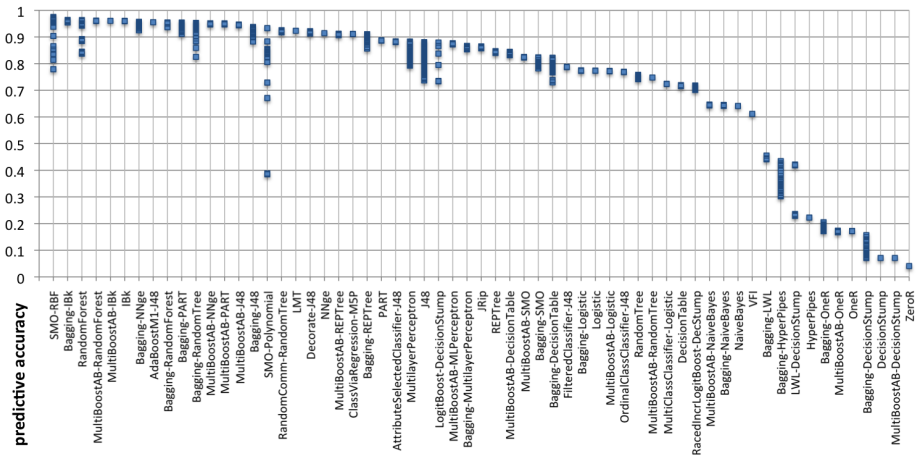

Fig. 2. Performance of all algorithms on dataset 'letter', with base-learners and kernels.

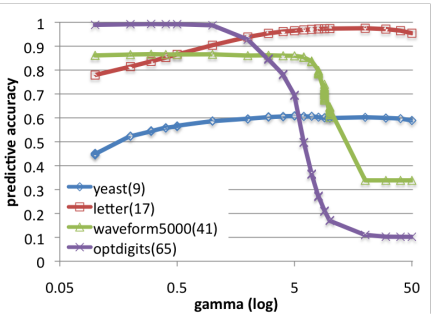

Fig. 3. The effect of parameter gamma of the RBF-kernel in SVMs.

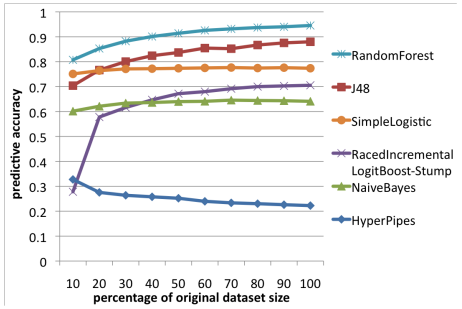

Fig. 4. Learning curves on dataset 'letter'.

Hypothesis Testing Writing queries also provides a fast way to explore the stored data and check hypotheses about any aspect of learning performance. For instance, to see if the variation in the SMO-RBF data from the previous query is caused by a specific parameter, we can 'zoom in' the SMO-RBF data (adding a constraint) and include the value of the 'gamma' parameter (selecting an extra field), yielding Fig. 3. When expanding the query towards several datasets, one can see that the optimal value of that parameter depends heavily on the dataset used, and more specifically on its number of attributes (indicated in brackets)[8].

When analyzing a dataset, one might be interested in how the dataset size affects the performance of an algorithm. Asking for the performance of algorithms on various downsampled versions of a dataset yields the learning curves shown in Fig. 4. Such queries may be useful to decide which algorithm to use based on the amount of available data or, conversely, how much data to collect.

[8] As discussed in [15], this reveals that on data with many attributes, it is better to use small gamma values, suggesting ways to improve the algorithm implementation.

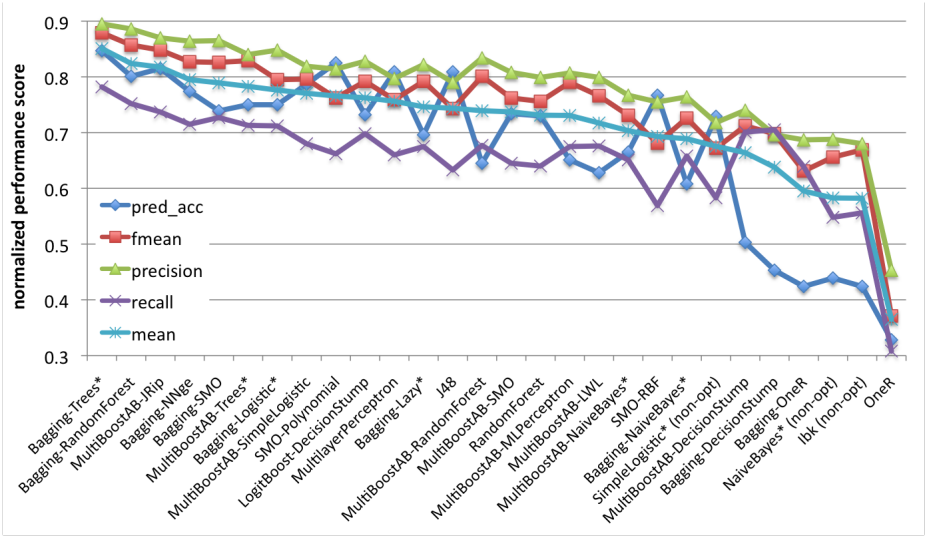

Fig. 5. Ranking of algorithms over all datasets and over different performance metrics. Similar algorithms are compacted in groups, indicated with an asterix (*).

Finding Fundamental Insights Pooling data from different sources enables us to perform studies that would be impossible, or very expensive to setup from scratch, but that may bring very general insights into learning performance. For instance, one could perform a general comparison of all algorithms (selecting optimal parameter settings) on the UCI datasets. Following a technique used by [7], we compare over a range of different evaluation metrics, using SQL aggregation functions to normalize each performance value between baseline and optimal performance, yielding Fig. 5 as the result of a single query. Note that this query can simply be rerun as new algorithms are introduced over time. A complete discussion of the results can be found in [15].

Ranking It is also possible to rank algorithms by writing a query. For instance, to investigate whether some algorithms consistently rank high over various problems, we can query for their average rank (based on optimal parameter settings) over all datasets. Using the Friedman ranking and the Nemenyi test to find the critical difference (the minimal rank gap for algorithms to perform significantly different)[9], we yield Fig. 6 for learning approaches in general, and Fig. 7 for algorithms with specific base-learners and kernels. This shows that indeed, some algorithms rank significantly higher on average than others on the UCI datasets.

Other Uses There are many more uses that can be thought of. For instance, much more could be studied when looking at the stored predictions: we could investigate which instances of a dataset are significantly hard to predict for most algorithms (and why), whether specific combinations of classifiers perform very well, or whether a vote over every single algorithm would result in good performance. Given the large amount of experiments, it could also be very interesting to mine the repository and look for patterns in algorithm performance.

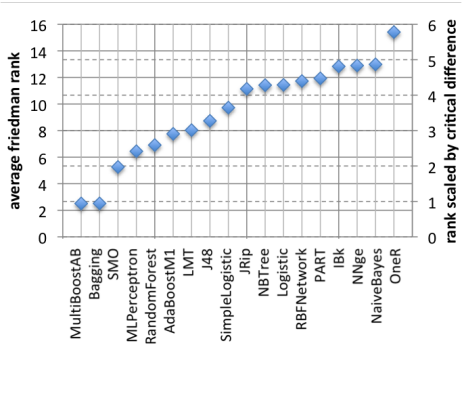

Fig. 6. Average rank, general algorithms.

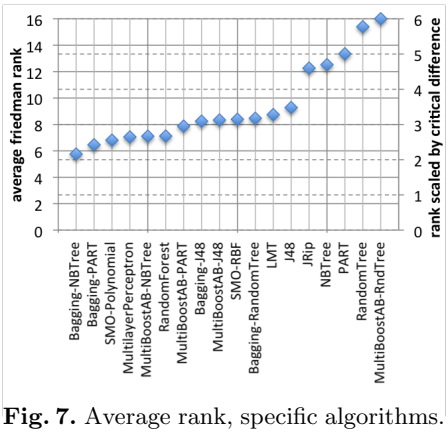

Fig. 7. Average rank, specific algorithms.

5.2 Integration in Data Mining Tools

Besides being queried directly, an experiment database could also be a valuable resource when integrated in a variety of data mining tools. One application would be to avoid unnecessary computation. When performing a large set of experiments, the tool could automatically consult the experiment database to see if some experiments can be reused. Furthermore, one could parallelize the execution of experiments by uploading unfinished experiments to the database and have several computers checking for unfinished experiments to run.

The tool could also automatically export experiments into a local or global experiment repository. Local repositories could typically be used for studies where datasets are not (yet) publicly available or where algorithms are still under development, offering a means to automatically organize all experiments for easier analysis. The experiments (or a selection thereof) could still be shared at a later point in time by transferring them to global repositories.

Finally, one could integrate experiment databases and inductive databases [11], creating repositories with much more powerful querying capabilities [3].

6 Conclusions

Sharing machine learning experiments and organizing them into experiment repositories opens up many opportunities for machine learning research and exploratory data analysis. While such repositories have been used in other sciences, most notably in bio-informatics, they have only recently been introduced into machine learning. To stimulate the future development of these repositories, we have discussed how they can be created and used in practice, and what challenges remain to be addressed to realize their full potential. To allow the free exchange of learning experiments, we have proposed an XML-based description language capturing a wide range of experiments. Next, we have used the same inherent structure to implement a database to capture and automatically organize learning experiments, improving upon earlier suggestions. Finally, we gave an

overview of possible services that such a resource could offer, either by querying it to retrieve relevant information, or by integrating it into data mining tools. It is likely that many more creative uses remain to be discovered.

Acknowledgements

Hendrik Blockeel is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.-Vlaanderen), and this research is further supported by GOA 2003/08 "Inductive Knowledge Bases".

References

1. Allison L.: Models for machine learning and data mining in functional programming. Journal of Functional Programming **15(1)** (2005) 15–32
2. Ball, C.A. and Brazma A. and Causton H. and Chervitz S. and Edgar R. et al.: Submission of Microarray Data to Public Repositories. PLoS Biol **2(9)** (2004) e317
3. Blockeel, H.: Experiment databases: A novel methodology for experimental research. Lecture Notes in Computer Science **3933** (2007) 72-85
4. Blockeel, H. and Vanschoren, J.: Experiment databases: Towards an improved experimental methodology in machine learning. PKDD '07: Proceedings. Lecture Notes in Computer Science **4702** (2007) 6-17
5. Brazma,A., Hingamp,P., Quackenbush,J., Sherlock,G. et al.: Minimum information about a microarray experiment (MIAME): toward standards for microarray data. Nature Genetics **29** (2001) 365371.
6. Brazma A., Parkinson H., Sarkans U., Shojatalab M. et al.: ArrayExpress–a public repository for microarray gene expression data at the EBI. Nucleic Acids Research **31(1)** (2003) 68-71.
7. Caruana R. and Niculescu-Mizil A.: An empirical comparison of supervised learning algorithms. ICML '06: Proc. of the 23rd Intl. Conf. on Mach. learning (2006) 161–168
8. The Data Mining Group: The Predictive Model Markup Language (PMML), version 3.2. `http://www.dmg.org/pmml-v3-2.html`
9. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research **7** (2006) 1–30
10. Džeroski S.: Towards a General Framework for Data Mining. Lecture Notes in Computer Science **4747** (2007) 259–300
11. Imielinski T. and Mannila H.: A database perspective on knowledge discovery. Communications of the ACM **39(11)** (1996) 58–64
12. Perlich, C. and Provost, F. and Simino, J.: Tree induction vs. logistic regression: A learning curve analysis. Journal of Machine Learning Research **4** (2003) 211–255
13. Soldatova L. N., Clare A., Sparkes A., King R. D.: An ontology for a Robot Scientist, Bioinformatics **22(14)** (2006) 464–471
14. Stoeckert,C., Causton,H. and Ball,C.: Microarray databases: standards and ontologies. Nature Genetics **32** (2002) 469–473.
15. Vanschoren J. and Pfahringer B. and Holmes G.: Learning From The Past with Experiment Databases. Working Paper Series 08/2008, Computer Science Department, University of Waikato (2008)
16. Witten, I.H. and Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques (2nd edition). Morgan Kaufmann (2005)