

**Experiment Databases:
Towards an Improved Experimental
Methodology in Machine Learning**

Hendrik Blockeel Joaquin Vanschoren

Report CW483, March 2007



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Experiment Databases: Towards an Improved Experimental Methodology in Machine Learning

Hendrik Blockeel Joaquin Vanschoren

Report CW483, March 2007

Department of Computer Science, K.U.Leuven

Abstract

Machine learning research often has a large experimental component. While the experimental methodology employed in machine learning has improved much over the years, repeatability of experiments and generalizability of results remain a concern. In this paper we propose a methodology based on the use of experiment databases. Experiment databases stimulate large-scale experimentation, guarantee repeatability of experiments, improve reusability of experiments, help explicitating the conditions under which certain results are valid, and support quick hypothesis testing as well as hypothesis generation. We show that they have the potential to significantly increase the ease with which new results in machine learning are obtained and correctly interpreted.

Keywords : Data Mining, Experimental Methodology.

AMS(MOS) Classification : Primary : I.2.6, Secondary : I.5.0, I.5.2.

Experiment Databases: Towards an Improved Experimental Methodology in Machine Learning

Hendrik Blockeel and Joaquin Vanschoren

DTAI research group, Computer Science department, K.U.Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium

Abstract

Machine learning research often has a large experimental component. While the experimental methodology employed in machine learning has improved much over the years, repeatability of experiments and generalizability of results remain a concern. In this paper we propose a methodology based on the use of *experiment databases*. Experiment databases stimulate large-scale experimentation, guarantee repeatability of experiments, improve reusability of experiments, help explicitating the conditions under which certain results are valid, and support quick hypothesis testing as well as hypothesis generation. We show that they have the potential to significantly increase the ease with which new results in machine learning are obtained and correctly interpreted.

1 Introduction

Machine learning is to a large extent an experimental science. Many learning algorithms are heuristic in nature: there is reason to believe that some method will work well, but is often difficult to prove this. Instead, one implements the algorithm and determines its performance empirically.

This experimental component in machine learning research seems unavoidable: we know it is impossible to theoretically prove that one algorithm is superior to another (Wolpert & Macready, 1995), except perhaps under specific conditions. Even then, it may be difficult to specify these conditions precisely, or to find out how relevant they are for real-world problems.

Given that experimental assessment of the performance of machine learning approaches is so important, care should be taken to ensure a correct interpretation of (published) experimental results. To this aim, first, a complete account of the experimental procedure should be available. This goes beyond ensuring repeatability. Stating that algorithm A with parameter settings P produces result R on dataset D should make it possible for other researchers to repeat the experiment and get exactly the same outcome. Yet, for a correct interpretation of a series of experiments, information on how the algorithms, parameter settings and datasets were selected by the researcher is equally important. For instance, tuning the algorithm's parameters towards the test set would give a biased result. Unfortunately, space restrictions imposed on publications often render it practically infeasible to describe experiments in full detail.

Second, even with a full description of the experimental procedure, correct interpretation is not straightforward. Results are easily assumed to generalize beyond the experimental settings described in a paper, sometimes wrongly so. Perlich et al. (2003), for instance, describe how the relative performance of logistic regression and decision trees depends strongly on the dataset size. Similarly, Hoste and Daelemans (2005) show that in text mining, differences in performance between lazy learning and rule induction are dominated by the effect of parameter optimization, data sampling, feature selection, and their interaction. In the light of all this, it is surprising how often general conclusions are drawn from few experiments covering a limited range of parameter settings.

In this paper we propose the use of *experiment databases* as a practical means to alleviate the problems described above. Experiment databases store very large numbers of experiments that are more representative for the whole population of possible experiments, not biased towards optimized results, and described in full detail. As such, they give a full and fair account of conducted research. In addition, the stored experiments are to a large extent reusable: new hypotheses can be tested by just querying such a database, instead of setting up and running new experiments. This may greatly simplify and speed up consecutive research. Overall, experiment databases foster a better experimental methodology by making it much easier for researchers to obtain trustworthy results and to interpret them correctly.

The remainder of this paper is structured as follows. In Section 2 we give more details on the concept and merits of experiment databases. In Section 3 we discuss the structure of such a database, and in Section 4 methods for populating it with experiments. Section 5 presents a case study: we implemented an experimental database and ran a number of queries in order to evaluate how easily it allows verification or refinement of existing knowledge, and/or discovery of new insights. We conclude in Section 6.

2 Experiment Databases

According to Blockeel (2006), an experiment database is a complete account of a (large) number of experiments that have been performed. It contains detailed information on the datasets, algorithms, and parameter settings used in the experiments, as well as the obtained results. It can be seen as a log of performed experiments, but also as a repository of experimental results that can be used to obtain new insights.

Experiment databases can be used in combination with the current experimental methodology in machine learning, but they also allow for a new, rather different methodology. We can summarize these two methodologies as follows:

- Methodology 1 (current): the researcher formulates a hypothesis, designs experiments with the specific aim of testing the hypothesis, runs the experiments and interprets the results.
- Methodology 2 (new): a large number of experiments, covering a wide range of conditions, are run in advance, and described in an experiment database in full detail. When a new hypothesis is to be tested, the necessary information is extracted from the database with a simple query. The hypothesis test is then performed and the result interpreted.

While in Methodology 1 experiment databases are not necessary, they do bring some advantages. Papers often contain a relatively vague account of the exact experimental procedure that was used: algorithms are described on a high level, often without giving the exact implementation (although many results, e.g. on runtime, strongly depend on implementation details), and even parameter values or the way in which they were chosen are not always mentioned clearly. By logging the specifics of all the experiments into an experiment database, which could be put online, a complete account of the experiments remains available, and repeatability of each individual experiment is ensured. Other researchers can use this experiment database to validate or perhaps refine earlier interpretations.

With or without the use of experiment databases, Methodology 1 has the disadvantage that each time a new hypothesis is considered, one needs to design new experiments to test it. The results of previous experiments usually cannot be exploited because they cover different conditions or only the outcomes relevant for testing previous hypotheses were recorded. However, using Methodology 2, each new hypothesis only requires a new query to the database. Each hypothesis test can potentially exploit the information in all the stored experiments, and conversely, each single experiment can potentially be reused for each new hypothesis. If the experiment database is made publicly available, other researchers can also reuse the experiments (possibly in combination with new ones) to test new hypotheses and obtain new insights.

For Methodology 2, it is crucial that the experiment database contain a wide range of experiments, ideally covering the space of possible experiments as completely as possible. While this makes the method computationally expensive, it also enforces generalizability of the results: any results obtained from the full database are necessarily valid under a wide range of conditions. Such results are more general than results obtained from more limited, goal-oriented, experimentation. Focusing on specific experimental conditions is still possible by simply including these conditions in the query. The generality of results obtained this way is of course again limited, but the limitations are crystal clear: the conditions under which the results hold are explicitly listed in the query.

A final advantage of Methodology 2 is that it is also possible to automatically discover (possibly unthought-of) patterns in an algorithm’s behavior, by using a data mining algorithm on (part of) the experiment database, and interpreting the returned model.

Thus, Methodology 2, which is directly connected to the use of experiment databases, supports reusability of experiments, explicitates the generality of results, and stimulates a more thorough analysis of the obtained results by offering easy repeated hypothesis testing and data mining capabilities. All these advantages come on top of better documentation and repeatability of experiments, an advantage that experiment databases also bring to Methodology 1. The main disadvantages of Methodology 2 are that it is computationally expensive and that it makes it more complicated for the user to set up experiments. The first disadvantage is relative: if the experiments are reused often enough, Methodology 2 becomes more efficient than Methodology 1 in terms of the number of experiments run. The second disadvantage can be alleviated by offering good tools for storing experiments into an experiment database, or even entire environments to also facilitate selecting and running them. This work provides a first step in that direction.

3 Database Structure

An experiment database should be designed to store experiments in such detail that they are perfectly repeatable and maximally reusable. This means that they should be described so precisely that the outcome of the experiment is fully determined, and in this section, we consecutively discuss how the learning algorithms, the datasets, and the experimental procedures should be described to achieve this goal. This discussion does not lead to a single best way to design an experiment database: in many cases several options remain, and depending on the purpose of the experiment database different options may be chosen.

3.1 Algorithm

In most cases, storing a complete symbolic description of the implementation of an algorithm is practically impossible. It is more realistic to store name and version of a system, together with a pointer to source code or an executable, so the experiment can be rerun under the same conditions. Some identification of the environment (e.g. the required operating system) completes this description.

As most algorithms have parameters that change their behavior, the values of these parameters must be stored as well. We call an algorithm together with specific values for its parameters an algorithm instantiation. For randomized algorithms, we consider the seed for the random generator that the algorithm uses also as a parameter. In this way, an algorithm instantiation is always a deterministic function.

Optionally, a characterization of the algorithm could be added, consisting of generally known or calculated properties (Van Someren, 2001; Kalousis & Hilario, 2000). Such a characterization could indicate, for instance, the class of approaches the algorithm belongs to (naive bayes, neural net, decision tree learner, . . .), whether it is considered a (semi-)supervised learner, whether it is generally considered to have high or low bias and variance, etc. Although this characterization is not necessary to ensure repeatability of the experiment, it may be useful when interpreting the

results. It is not problematic if parts of this characterization are rather subjective: the degree of subjectiveness can easily be evaluated by analysing the experiment database later on.

Thus, the structure of the experiment database should be such that each experiment refers to an algorithm instantiation, which itself points to an algorithm (with characterization) and a set of parameter settings.

3.2 Dataset

To describe datasets, it is again easiest to just store some attributes for documentation (name and version of dataset, perhaps the URL where it was obtained) and a pointer to a representation of the actual dataset. This dataset could be in any format: the most obvious one is a text file that the algorithm implementations can read, but it could also be a dataset generator together with its parameters (including the generator’s random seed) or a data transformation function (sampling instances, selecting features, defining new features, etc.) together with its parameters and a pointer to its input dataset.

As with algorithms, an optional characterization of the dataset can be added: number of examples, number of attributes, etc. Since this characterization depends only on the dataset, not on the experiment, new features can be added and computed for each dataset (and subsequently used in future analysis) without rerunning any experiments. The same holds for the algorithm characterisation. This is one aspect of the reusability aspect of experiment databases.

3.3 Experimental Procedure

An experiment often involves more than just running a learning algorithm on a dataset and recording the results. For instance, it may involve a cross-validation procedure to estimate the predictive performance of the algorithm on unseen data. To make the experiment fully repeatable, we therefore need to describe this experimental procedure in full detail. In the case of cross-validation, this implies storing (a seed to generate) the exact folds. To describe the outcome variables (e.g. on predictive performance), we need to describe exactly what is measured (i.e., the exact function used to compute them) and of course the values of these outcome variables for each experiment.

What outcome variables should be stored? First of all, the model description itself is an outcome variable. However, while the user may want to visualize a learned model every now and then, it is likely that only specific properties of these models, but not the models themselves, will be involved in the analysis later on. For that reason, one may choose to store only those properties in the database, and not the (perhaps storage-intensive) full representation of the model.

Examples of such properties are: runtime of the entire experiment, time to learn the model (excluding e.g. cross-validation time), size of the learned model, likelihood of the model, estimated predictive accuracy of the model, etc. In the case of classifiers, there are arguments against storing just the error or accuracy; it is better to store the full contingency table (i.e., for each couple of classes (i, j) , the number of cases where class i was predicted as class j).¹

In the limit, for predictive models, individual predictions for each element in a data set could be stored. Although this may not be very storage-efficient, it is nevertheless advisable to store as detailed information as possible. When the user wants to add new evaluation criteria, this can be done most efficiently if these criteria can be derived from the stored metrics. If not, the experiments need to be rerun, which is computationally expensive.

¹Demsar (2006) comments that it is astounding how many papers still evaluate classifiers based on accuracy alone, despite the fact that this has been advised against for many years now. The use of more or less standardized experiment databases that automatically store contingency tables may help eradicate this practice.

4 Populating the Database

In the previous section we discussed the structure of the database. Besides having a good structure, one also needs good contents. As we want to use this database to gain insight in the behavior of machine learning algorithms under various conditions, we need to have experiments that are as diverse as possible. At the same time however, we also want to be able to thoroughly investigate very specific conditions. This means we must cover an area within the space of all possible experiments that is as large as possible, and yet populate this area in a reasonably dense way. This goal is non-trivial to achieve, due to the high dimensionality of the experiment space.

The total experiment space is the product of the spaces describing instantiated algorithms, datasets, and experimental procedure. Even after restricting the number of measured and stored properties (e.g., for datasets, we could only look at the size, number of attributes and number of classes), it remains high-dimensional. Covering a large area of this space in a “reasonably dense” way implies running many experiments.

A simple, yet effective way of doing this is selecting random, but sensible (see below), values for all parameters in our experiments. With the term parameter we mean any stored property of the experiment: the used algorithm, its parameters, its algorithm-independent characterization, the dataset properties, etc.

Assume that each parameter has at most v values (numerical parameters are discretized into v bins). Then running $100v$ experiments with random values for all parameters implies that for each value the average outcomes of about 100 experimental runs will be stored. This seems sufficient to be able to detect most correlations between outcomes and the value of this parameter².

To detect interaction effects between two parameters, $100v^2$ experiments would be needed, and for higher order interactions $100v^n$ with n the order. With $v = 20$ and $n = 3$, this yields 800K experiments: a large number, but (especially for fast algorithms) not infeasible with today’s computation power.

Note how this contrasts to the number of experimental runs typically reported on machine learning papers. There are several justifications for this contrast. First, experiments are usually set up to test one specific hypothesis. If we know in advance that we want to test the effect of one parameter on the outcomes, we can keep the other parameters constant such that no noise is introduced, and systematically vary the one parameter, so that only v experimental runs are needed. But such experiments are useful only for this one goal, they cannot be reused for anything else. Moreover, by keeping the other parameters constant, there is no guarantee that the obtained results generalize towards other parameter settings. The factor 100 is the price we pay for ensuring reusability and generalizability. Especially in the long run, these benefits easily compensate for the extra computational expense.

Second, the above computation assumes that the user may be interested in interaction effects up to the third order. Most existing work does not investigate such effects. If one does want to investigate n ’th order interaction effects between parameters, the v^n factor is unavoidable.

Finally, experiments could in fact be designed in a better way than just randomly generating parameter values; one could look at techniques from active learning or Optimal Experiment Design (OED) (Cohn, 1994) to focus on the most interesting experiments. In itself, however, we do not expect this to reduce the number of experiments needed with orders of magnitude.

Having discussed how many experiments are needed, we now turn to the question of how to generate them in practice.

First, we need to select a specific algorithm from a large set of available algorithms. To choose its parameter settings, one can specify a probability distribution for each different parameter according to which values should be generated (in the simplest case, one can make a list of all possible or reasonable values for each parameter and let the system choose one uniformly at random).

²It may not be sufficient if the randomization of the other parameters causes so much noise that even after averaging over 100 outcomes this noise dominates the effect of the parameter studied; but this effectively implies that the effect of the studied parameter, while it may exist, is much smaller than the effect of other parameters.

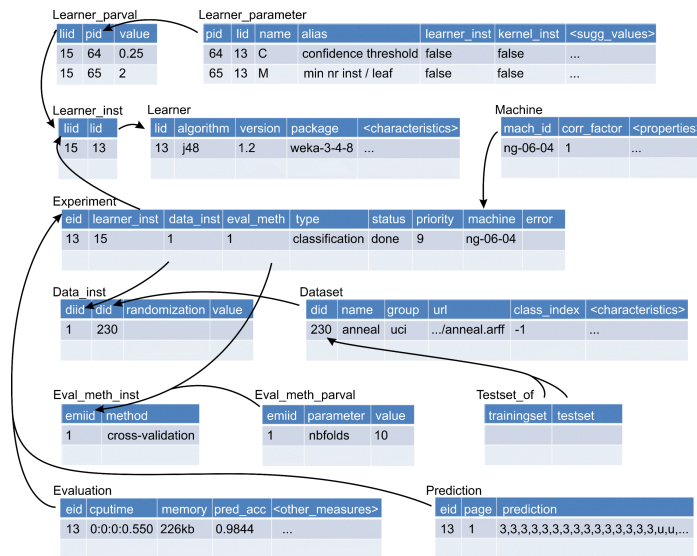


Figure 1: A possible implementation of an experiment database.

Covering the dataset space is harder. One can select a dataset from a large number of real-world datasets, including for instance the UCI repository. This gives a relatively small number of possibilities. One can however implement a number of data transformation methods (e.g., artificially limiting the number of examples of the dataset; performing feature selection; etc.) and derive variants of real-world datasets in this way. Finally, one could use synthetic datasets, produced by dataset generators. This seems a very promising direction, but the construction of dataset generators that cover a reasonably interesting area in the space of all datasets is non-trivial.

While the matter of generating experiments that cover the whole space of real-world datasets in a satisfactory way remains unsolved, it is clear that even the trivial approach of just including publicly available datasets would already ensure a coverage that is equal to or greater than that of many published papers on general-purpose machine learning techniques.

5 A Case Study

In this section we discuss one specific implementation of an experiment database. First, we describe the schema of our database and which experiments we have inserted into it. Then, we illustrate its use with a few example queries. Our experiment database is public and can be consulted online (the URL will be included in the non-anonymous version of this paper).

5.1 A Relational Experiment Database

We implemented an experiment database for classifiers in a standard RDBMS (MySQL). A simplified schema for this database is shown in figure 1.

Central in the figure is a table of experiments listing the used *instances* of learning algorithms, datasets and evaluation methods, the experimental procedure, and the machine it was run on³. An instance of a learning algorithm points to an entry in the table of learning algorithms, which are described by name, version, download url and a short list of simple characteristics. Each parameter

³Some extra status information is stored to facilitate automatic execution.

value assignment is stored as a new row (here only two are shown) in a separate table⁴, referring to the parameter and the learning instance in which they are used. Parameters can be further described by the learner they belong to, suggested values, and some special properties which are used for interpreting their values. For instance, when a parameter points to a learner instance (as is the case in meta-algorithms, e.g. bagging) its value will simply consist of the corresponding learner instance id.

Dataset instances point to a dataset and a (possibly randomized) order on the attributes and examples; the latter might be useful, for instance, when experimenting with incremental learners. Datasets are described by name, group (e.g. UCI), download url, class index (if needed), and 56 characterization metrics (including landmarks, entropy, skewness, kurtosis, mstatistics, . . .) most of which are mentioned in Peng et al. (2002). Dataset generators or transformers are currently not supported.

Next, the experimental procedure and its parameters are stored (e.g., 10-fold cross-validation).

Finally, the experiment outcome is described by a wide range of evaluation metrics for classification, including the contingency tables. To compare cpu times, a factor describing the relative speed of the used machine is stored as part of the machine description. The last table in this figure stores the predictions returned by each experiment, which, for classification, can be compressed by only storing the misclassified examples.

5.2 Populating the Database

To populate the database, we have first selected a set of datasets and algorithms, after which two series of experiments were performed. Although these experiments focus on classification tasks, it is easy to see that experiments on other tasks could be performed just as easily.

First of all, we selected and inserted 54 classifier implementations from the Weka platform (Witten & Frank, 2005). Their parameters were also added, all of which were given a default value (the one used in Weka), and some of which were in addition given a set of suggested values. This was the case for the parameters in the Weka implementations of SMO (for training a support vector machine), MultilayerPerceptron, C4.5, 1R and Random Forests. Finally, if an algorithm had a random seed, a set of 20 different randomly selected seeds was given. As for the datasets, we took 86 commonly used classification datasets from the UCI repository.

This information was then used by an experimentation program for generating the experiments, scheduling them on a computer cluster, running them with WEKA and inserting the results in the database. All experiments used 10-fold cross-validation. In a first series of experiments, we ran all algorithms on all datasets, first using the default parameter settings, then varying one of the above described parameters over their suggested range while keeping all other parameters at default. In a second series of experiments, we generated random parameter settings for a small number of algorithms (including J48 and OneR, Weka’s implementation of C4.5 and 1R), until we had about 1,000 examples on each dataset. Over the course of three weeks, about 250,000 experiments were performed.

5.3 Querying

We are now ready to start querying the database. In a first query, we compare the performance of all algorithms on a specific dataset, which we can do by launching the following query:

```
SELECT l.algorithm, v.pred_acc
FROM experiment e, learner_inst li, learner l, data_inst di, dataset d, evaluation v
WHERE e.learner_inst = li.liid and li.lid = l.lid and e.data_inst = di.diid and
di.did = d.did and d.name='waveform-5000' and v.eid = e.eid
```

In this query, we gather all experiments where the dataset is `waveform-5000`, and select the algorithm used and the predictive accuracy registered. We visualize the returned data in Figure 2,

⁴This is necessary as different algorithms have different parameters. An alternative option is to use one parameter table per algorithm, although this would make it harder to add new algorithms.

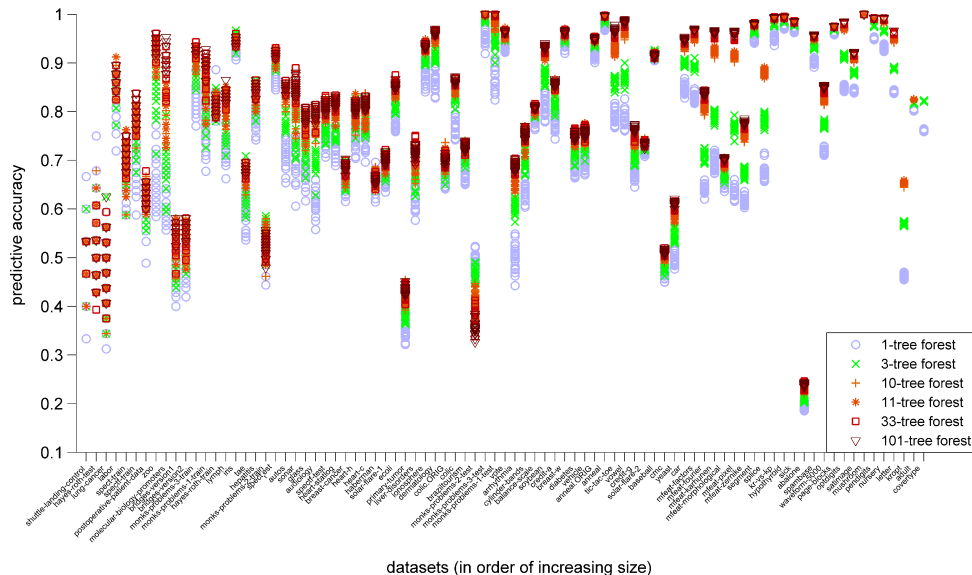


Figure 4: The effect of dataset size and the number of trees for random forests.

We can also investigate combined effects of dataset characteristics and parameter settings. For instance, we can test whether the performance ‘jumps’ of `RandomForest` are linked to the number of trees in a forest and the dataset size:

```
SELECT d.name, d.nr_examples, lv.value, v.pred_acc
FROM experiment e, learner_inst li, learner l, learner_parval lv, learner_parameter p,
data_inst di, dataset d, evaluation v
WHERE e.learner_inst = li.liid and li.lid = l.lid and l.algorithm='RandomForest' and
lv.liid = li.liid and lv.pid = p.pid and p.alias='nb of trees in forest' and v.eid = e.eid
ORDER BY d.nr_examples
```

In this query we select the dataset name and number of examples, the parameter value of the parameter named `nb of trees in forest` of algorithm `RandomForest` and the corresponding predictive accuracy. The results are returned in order of dataset size. When plotted in Figure 4, this clearly shows that predictive accuracy increases with the number of trees, usually leveling off between 33 and 101 trees, but with one exception: on the `monks-problems-2-test` dataset the base learner performs so badly (less than 50% accuracy, though there are only two classes) that the ensemble just performs worse when more trees are included. We also see that as the dataset size grows, the accuracies for a given forest size vary much less, which is indeed what we would expect as the trees become more stable on large datasets.

5.4 Mining

5.4.1 A Closer Look

From Figure 4, we learned that the behavior of `RandomForest` is rather atypical on dataset `monks-problems-2-test`. We can now take a closer look at the behavior of other learners on this dataset, which represents a binary classification problem with 6 attributes (each having 2 up to 4 nominal values). The task is to separate those instances for which exactly two of the attributes are assigned their first values. It is somewhat similar to parity problems, in the sense

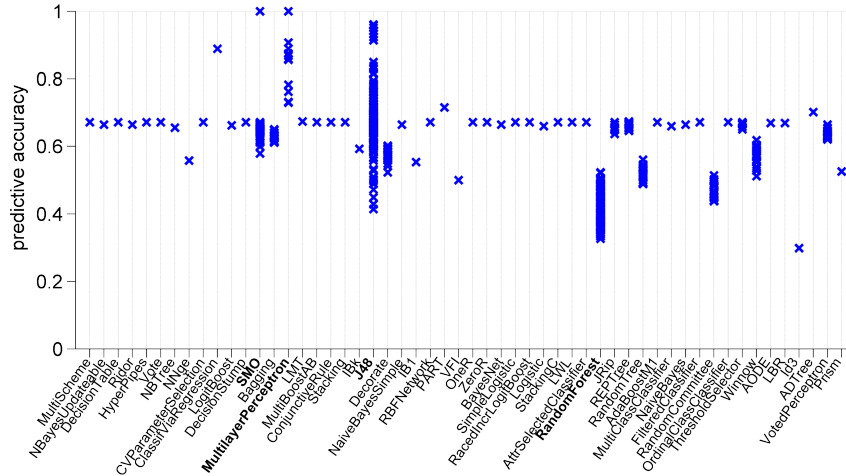


Figure 5: Performance comparison of all algorithms on the `monks-problems-2_test` dataset.

that higher-order interactions between the attributes (by which we mean that the value of an attribute depends on the values of several other attributes) make it hard to learn for most learning algorithms.

A query similar to the first one yields Figure 5. We see that indeed, most algorithms do not surpass the default accuracy of 67% for this dataset (at least not with default parameters). There are however some notable exceptions. Some algorithms score very badly, including `RandomForest`, `RandomCommittee`, `RandomTree`, `Winnow` and `Decorate`. Two algorithms reach perfect accuracy under certain parameter settings: `MultilayerPerceptron` when having 3 or more hidden nodes (all experiments used only one hidden layer) and `SMO` when using a polynomial kernel of order 2 or higher (default accuracy is never surpassed when using first order polynomial or RBF kernels). These requirements are not surprising for dealing with a near-parity problem.

5.4.2 Modelling Parameter Effects

The performance of J48 is less easily linked to certain parameter settings. As can be seen from Figure 5 a lot of results for J48 score below the default accuracy, some lie above this default accuracy and a few models built by J48 were able to reach accuracies higher than 85% (a clear performance jump can be seen in Figure 5). In order to get insight in the relation between the parameter settings of J48 and its accuracy on the `monks-problems-2_test` dataset, we constructed a meta-dataset by querying for all parameter settings of J48 on this dataset (we skip the query, which returned 773 results). Then, we used J48 to generate a (meta-)decision tree that predicts in which interval (below default accuracy, above default accuracy or higher than 85%) the accuracy lies of a tree built with a certain combination of parameters. The resulting tree, which obtained 97.3% accuracy by 10-fold cross-validation, is shown in Figure 6.

The parameter settings used in the meta-tree are `binary_splits` (whether binary splits are used in the tree), `min_inst` (the minimal number of instances in a leaf of the tree), `use_rep` (whether reduced error pruning is used), `pruning` (whether pruning is used) and `conf_thresh` (the confidence threshold for regular pruning). As can be derived from the tree in Figure 6, J48 is able to build the best models if binary splits are used, if the minimal number of instances per leaf is smaller than 3 and if reduced error pruning is not used as the pruning technique. Knowing the concept of `monks-problems-2_test`, it is indeed clear that binary splits are a better option than splits that use all attribute values, as only the first value of each of the attributes is related to the class value. Moreover, more examples (and thus more splits) remain in each branch,

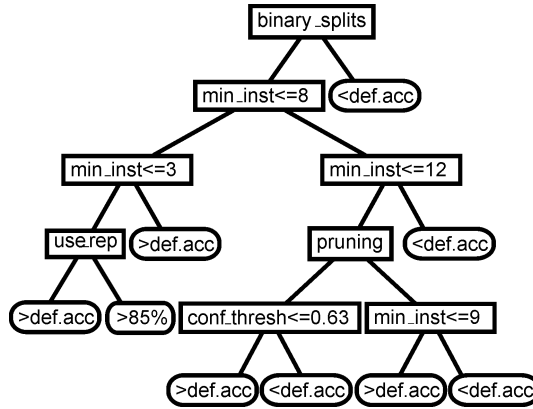


Figure 6: A meta-tree learned on a meta-dataset concerning predictive accuracies of trees learned on the `monks-problems-2_test` dataset.

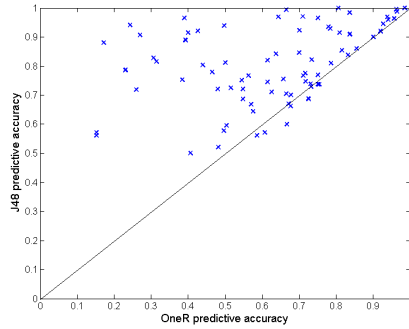


Figure 7: Relative performance of J48 and OneR.

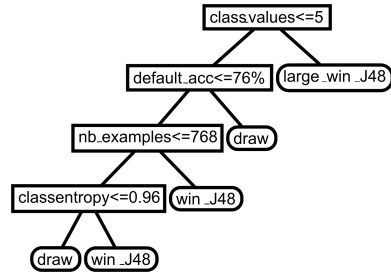


Figure 8: A meta tree predicting the accuracy improvement of J48 over OneR on 77 UCI datasets.

allowing better generalization. Concerning the minimal number of instances in the leaves it is indeed necessary for a (near-)parity problem like this, to grow the tree (nearly) up to the level of individual instances. The fact that reduced error pruning seems to reduce the performance over normal pruning could be explained by the validation set that is used to do the pruning, leaving less examples for training.

5.5 Verifying and Refining Existing Knowledge

As said before, an experiment database can also be useful to verify or refine existing knowledge. To illustrate this, we verify the result of Holte (1993) that very simple classification rules (like 1R) perform almost as good as complex ones (like C4, a predecessor of C4.5) on most datasets. We compare the average predictive performance (over experiments using default parameters) of J48 with that of OneR for each dataset. We skip the query as it is quite complex. Plotting the average performance of the two algorithms against each other yields Figure 7.

We see that J48 almost consistently outperforms OneR, in many cases performing a little bit better, and in some cases much better. This is not essentially different from Holte’s results, though the average improvement does seem larger here (17.95% versus 5.7% for Holte’s results).

We also see that J48 never goes below 50% accuracy, while OneR regularly does, which makes it less suitable as a weak learner in ensemble learning.

We can automatically learn under which conditions J48 clearly outperforms OneR and under which conditions their predictive performance is similar. To do this, we queried for the difference in predictive accuracy between J48 and OneR for each dataset, together with all dataset characteristics (number of examples, number of attributes, default accuracy, . . .). Discretizing the predictive accuracy yields a classification problem with 3 class values: “draw” (OneR wins or J48 wins slightly: accuracy gain less than 4%), “win_J48” (4% to 20% gain), and “large_win_J48” (20% to 70% gain). The tree returned by J48 on this meta-dataset is shown in Figure 8.

As can be seen from the tree, the number of values for the class attribute yields the most informative condition. If the class contains at least 5 values, then J48 has a large win. These results indeed seem to explain a lot. OneR works by choosing the most informative single attribute and builds a rule on this attribute alone. In order to be able to predict all class values, this attribute must also partition the dataset into at least as many parts. From the tree we also observe that if the default accuracy is high, then the predictive performance of OneR and J48 is similar. We also learn that J48 outperforms OneR on quite large datasets and datasets with high class entropy⁵.

The fact that the average improvement of the decision tree learner seems larger than was reported by Holte may indicate an improvement in decision tree learners and/or a shift towards more complex datasets in the last decade. Interestingly, when comparing the dataset characteristics of our datasets with those of the 16 datasets used by Holte, we learn that in his study only one dataset contained more than 5 class values, which might explain why smaller accuracy differences were reported.

5.6 Final remarks

These queries only scratched the surface of all possible hypotheses that can be tested using the experiments generated for this case study. One could easily launch new queries to gain further insights into the behavior of an algorithm, the impact of a parameter or the way algorithms behave on a specific dataset (or any combined effects). Also, one can reuse this data (possibly augmented with further experiments) when researching the covered learning techniques, or when one wishes to compare it with new experimental results. Finally, one can use our experiment database implementation to set up other experiment databases, e.g. for regression or clustering problems.

6 Conclusions

We advocate the use of experiment databases in machine learning research. Combined with the current methodology, experiment databases ensure repeatability. Combined with a new methodology that consists of running many more experiments in a semi-automated fashion, storing them all in an experiment database, and then querying that database, experiment databases in addition foster reusability, generalizability, and easy and thorough analysis of experimental results. Furthermore, as these databases can be put online, they provide a detailed log of performed experiments, and a repository of experimental results that can be used to obtain new insights. As such, they have the potential to speed up future research and at the same time make it more reliable, especially when supported by the development of good experimental tools. We have discussed the construction of experiment databases, and demonstrated the feasibility and merits of this approach by presenting an publicly available experiment database containing 250,000 experiments and illustrating its use.

⁵The class entropy was scaled to remove the effect of the number of class values

References

- Blockeel, H. (2006). Experiment databases: A novel methodology for experimental research. *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Revised, Selected and Invited Papers* (pp. 72–85).
- Cohn, D. (1994). Neural network exploration using optimal experiment design. *Advances in Neural Information Processing Systems* (pp. 679–686).
- Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 1–30.
- 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'. *Proceedings of the Workshop on Meta-Learning (ICML-2005)* (pp. 20–27).
- Kalousis, A., & Hilario, M. (2000). Building algorithm profiles for prior model selection in knowledge discovery systems. *Engineering Intelligent Systems*, 8(2).
- Peng, Y., Flach, P., Soares, C., & Brazdil, P. (2002). Improved dataset characterisation for meta-learning. *Lecture Notes in Computer Science*, 2534, 141–152.
- Perlich, C., Provost, F., & Siminoff, J. (2003). Tree induction vs. logistic regression: A learning curve analysis. *Journal of Machine Learning Research* 4:211-255.
- 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.
- Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann Series in Data Management Sys. Morgan Kaufmann. Second edition.
- Wolpert, D., & Macready, W. (1995). *No free lunch theorems for search*. (Technical Report SFI-TR-95-02-010). Santa Fe Institute.