Experiment Databases: Towards an Improved Experimental Methodology in Machine Learning

Hendrik Blockeel and Joaquin Vanschoren

Computer Science Dept., 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 facilitate 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 can be obtained and correctly interpreted.

1 Introduction

Experimental assessment is a key aspect of machine learning research. Indeed, many learning algorithms are heuristic in nature, each making assumptions about the structure of the given data, and although there may be good reason to believe a method will work well in general, this is difficult to prove. In fact, it is impossible to theoretically prove that one algorithm is superior to another [15], except 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. Therefore, one usually verifies a learning algorithm's performance empirically, by implementing it and running it on (real-world) datasets.

Since empirical assessment is so important, it has repeatedly been argued that care should be taken to ensure that (published) experimental results can be interpreted correctly [8]. First of all, it should be clear how the experiments can be reproduced. This involves providing a complete description of both the experimental setup (which algorithms to run with which parameters on which datasets, including how these settings were chosen) and the experimental procedure (how the algorithms are run and evaluated). Since space is limited in paper publications, an online log seems the most viable option.

Secondly, it should be clear how generalizable the reported results are, which implies that the experiments should be general enough to test this. In time series analysis research, for instance, it has been shown that many studies were biased towards the datasets being used, leading to ill-founded or contradictory results [8]. In machine learning, Perlich et al. [10] describe how the relative performance of logistic regression and decision trees depends strongly on the size of dataset samples. Similarly, Hoste and Daelemans [6] show that in text mining, the relative performance of lazy learning and rule induction is dominated by the effect of parameter optimization, data sampling, feature selection, and their interaction. As such, there are good reasons for strongly varying the conditions under which experiments are run, and projects like Statlog [12] and METAL [11] made the first inroads into this direction.

In light of the above, it would be useful to have an environment for machine learning research that facilitates storage of the exact conditions under which experiments have been performed as well as large-scale experimentation under widely varying conditions. To achieve this goal, Blockeel [1] proposed the use of *experiment databases*. Such databases are designed to store detailed information on large numbers of learning experiments, selected to be highly representative for a wide range of possible experiments, improving reproducibility, generalizability and interpretability of experimental results. In addition, they can be made available online, forming "experiment repositories" which allow other researchers to query for and reuse the experiments to test new hypotheses (in a way similar to how dataset repositories are used to test the performance of new algorithms).

Blockeel introduced the ideas behind experiment databases and discussed their potential advantages, but did not present details on how to construct such a database, nor considered whether it is even realistic to assume this is possible. In this paper, we answer those questions. We propose concrete design guidelines for experiment databases, present a specific implementation consistent with these guidelines, and illustrate the use of this database. By querying it for specific experiments, we can directly test a wide range of hypotheses on the covered algorithms and verify or refine existing results. Finally, the database itself is a contribution to the machine learning community: this database, containing the results of 250,000 runs of well-known classification systems under varying conditions, is publicly accessible on the web to be queried by other researchers.

The remainder of this paper is structured as follows. In Sect. 2 we summarize the merits of experiment databases. In Sect. 3 we discuss the structure of such a database, and in Sect. 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 of existing knowledge and discovery of new insights. We conclude in Sect. 6.

2 Experiment Databases

An experiment database is a database designed to store a (large) number of experiments, containing detailed information on the datasets, algorithms, and parameter settings used, as well as the evaluation procedure and the obtained results. It can be used as a log of performed experiments, but also as a repository of experimental results that can be reused for further research.

The currently most popular experimental methodology in machine learning is to first come up with an hypothesis about the algorithms under study, then perform experiments explicitly designed to test this hypothesis, and finally interpret the results. In this context, experiment databases make it easier to keep an unambiguous log of all the performed experiments, including all information necessary to repeat the experiments.

However, experiment databases also support a new methodology: instead of designing experiments to test a specific hypothesis, one can design them to cover, as well as possible, the space of all experiments that are of interest in the given context. A specific hypothesis can then be tested by querying the database for those experiments most relevant for the hypothesis, and interpreting the returned results. With this methodology, many more experiments are needed to evaluate the learning algorithms under a variety of conditions (parameter settings, datasets,...), but the same experiments can be reused for many different hypotheses. For instance, by adjusting the query, we can test how much the observed performance changes if we add or remove restrictions on the datasets or parameter settings. Furthermore, as the query explictly mentions all restrictions, it is easy to see under which conditions the returned results are valid.

As an example, say Ann wants to test the effect of dataset size on the complexity of trees learned by C4.5. To do this, she selects a number of datasets of varying sizes, runs C4.5 (with default parameters) on those datasets, and interprets the results. Bob, a proponent of the new methodology proposed here, would instead build a large database of C4.5 runs (with various parameter settings) on a large number of datasets, possibly reusing a number of experiments from existing experiment databases. Bob then queries the database for C4.5 runs, selecting the dataset size and tree size for all runs with default parameter settings (explicitly mentioning this condition in his query), and plotting them against each other. If Ann wants to test whether her results on default settings for C4.5 are representative for C4.5 in general, she needs to set up new experiments. Bob, on the other hand, only has to ask a second query, this time not including the condition. This way, he can easily investigate under which conditions a certain effect will occur, and be more confident about the generality of his results.

The second methodology requires a larger initial investment with respect to experimentation, but may pay off in the long run, especially if many different hypotheses are to be tested, and if many researchers make use of experiments stored in such databases. For instance, say another researcher is more interested in the runtime (or another performance metric) of C4.5 on these experiments. Since this is recorded in the experiment database as well, the experiments will not have to be repeated. A final advantage is that, given the amount of experiments, Bob can train a learning algorithm on the available meta-data, gaining models which may provide further insights in C4.5's behavior.

Note that the use of experiment databases is not strongly tied to the choice of methodology. Although experiment databases are necessary for the second methodology, they can also be used with the first methodology, allowing experiments to be more easily reproduced and reused.

3 Database Structure

An experiment database should be designed to store experiments in such detail that they are perfectly repeatable and maximally reusable. 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 store the seed for the random generator they use also as a parameter. As such, an algorithm instantiation is always a deterministic function.

Optionally, a characterization of the algorithm could be added, consisting of generally known or calculated properties [13, 7]. Such a characterization could indicate, for instance, the class of approaches the algorithm belongs to (naive bayes, neural net, decision tree learner,. . .), whether it generally has 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 or when investigating specific types of algorithms.

3.2 Dataset

To describe datasets, one can store name, version and a pointer to a representation of the actual dataset. The latter could be an online text file (possibly in multiple formats) 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 the input dataset. If storage space is not an issue, one could also store the dataset itself in the database.

As with algorithms, an optional characterization of the dataset can be added: number of examples, number of attributes, class entropy, etc. These are useful to investigate how the performance of an algorithm is linked to properties of the training data. 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 underlines the reusability aspect of experiment databases.

3.3 Experimental Procedure

To correctly interpret (and repeat) the outcome of the experiment, we need to describe exactly how the algorithm is run (e.g. on which machine) and evaluated. For instance, in case we use a cross-validation procedure to estimate the predictive performance of the algorithm on unseen data, this implies storing (a seed to generate) the exact folds[1]. Also the exact functions used to compute these estimates (error, accuracy,...) should be described. To make the experiments more reusable, it is advisable to compute a variety of much used metrics, or to store the information from which they can be derived. In the case of classifiers, this includes storing 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).[2]

Another important outcome of the experiment is the model generated by the algorithm. As such, we should at least store specific properties of these models, such as the time to learn the model, its size, and model-specific properties (e.g. tree depth) for further analysis. If storage space allows this, also a full representation of the model could be stored for later visualisation[3]. For predictive models, it might also be useful to store the individual (probabilities of) predictions for each example in the dataset. This allows to add and compute more evaluation criteria without rerunning the experiment.

4 Populating the Database

Next to storing experiment in a structured way, one also needs to select the right experiments. 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. To achieve this in practice, we first need to select the algorithm(s) of interest 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, this could be a uniformly sampled list of reasonable values).

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. Yet, one can also implement a number of data transformation methods (e.g., sampling the dataset, performing feature selection,...) and derive variants of real-world datasets in this way. Finally, one could use synthetic datasets, produced by

[1] Note that although algorithms should be compared using the same folds, these folds (seeds) should also be varied to allow true random sampling.

[2] Demšar [3] 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. Experiment databases may help eradicate this practice.

[3] Some recent work focuses on efficiently storing models in databases [4].

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. This is a challenge, not a limitation, as even the trivial approach of only 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.

At the same time however, we also want to be able to thoroughly investigate very specific conditions (e.g. very large datasets). This means we must not only cover a large area within the space of all interesting experiments[4], but also populate this area in a reasonably dense way. Given that the number of possible algorithm instantiations and datasets (and experimental procedures) is possibly quite large, the space of interesting experiments might be very high-dimensional, and covering a large area of such a high-dimensional space in a "reasonably dense" way implies running many experiments.

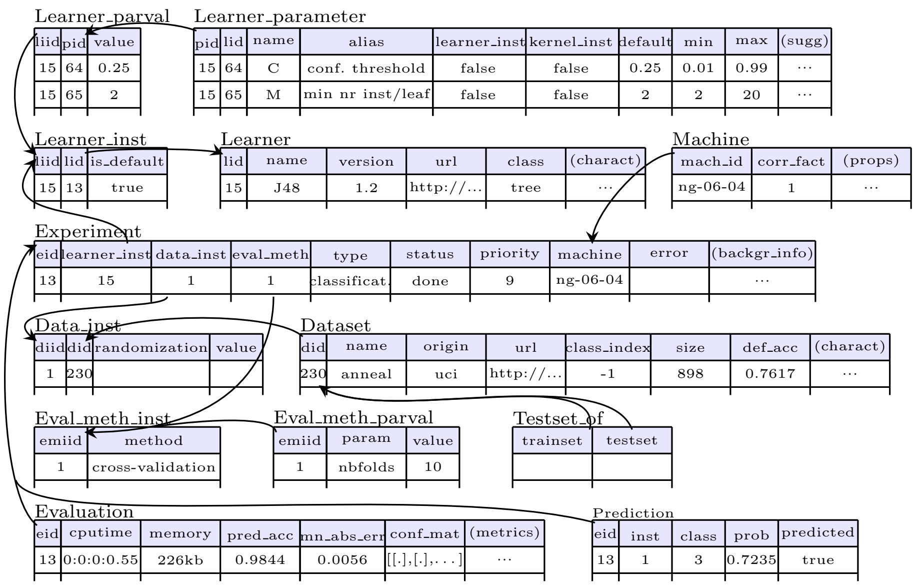
A simple, yet effective way of doing this is selecting random, but sensible, values for all parameters in our experiments. With the term parameter we mean any stored property of the experiment: the used algorithm, its parameters, its algorithm-independent characterization, the dataset properties, etc.

To imagine how many experiments would be needed in this case, assume that each of these parameter has on average v values (numerical parameters are discretized into v bins). Running $100v$ experiments with random values for all parameters implies that for each value of any single parameter, 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 parameter. To detect n-th order interaction effects between parameters, $100v^n$ experiments would be needed. Taking, for example, $v = 20$ and $n = 2$ or $n = 3$, this yields a large number of experiments, 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. Yet, when keeping many parameters constant to test a specific hypothesis, there is no guarantee that the obtained results generalize towards other parameter settings, and they cannot easily be reused for testing other hypotheses. 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. The v^n factor is unavoidable if one wants to investigate n'th order interaction effects between parameters. Most existing work does not study effects higher than the second order.

Finally, experiments could in fact be designed in a better way than just randomly generating parameter values. For instance, one could look at techniques from active learning or Optimal Experiment Design (OED) [2] to focus on the most interesting experiments given the outcome of previous experiments.

[4] These are the experiments that seem most interesting in the studied context, given the available resources.



Learner_parval

liid	pid	value
15	64	0.25
15	65	2

Learner_parameter

pid	lid	name	alias	learner_inst	kernel_inst	default	min	max	(sugg)
15	64	C	conf. threshold	false	false	0.25	0.01	0.99	⋯
15	65	M	min nr inst/leaf	false	false	2	2	20	⋯

Learner_inst

liid	lid	is_default
15	13	true

Learner

lid	name	version	url	class	(charact)
15	J48	1.2	http://...	tree	⋯

Machine

mach_id	corr_fact	(props)
ng-06-04	1	⋯

Experiment

eid	learner_inst	data_inst	eval_meth	type	status	priority	machine	error	(backgr_info)
13	15	1	1	classificat.	done	9	ng-06-04		⋯

Data_inst

diid	did	randomization	value
1	230		

Dataset

did	name	origin	url	class_index	size	def_acc	(charact)
230	anneal	uci	http://...	-1	898	0.7617	⋯

Eval_meth_inst

emiid	method
1	cross-validation

Eval_meth_parval

emiid	param	value
1	nbfolds	10

Testset_of

trainset	testset

Evaluation

eid	cputime	memory	pred_acc	mn_abs_err	conf_mat	(metrics)
13	0:0:0:0.55	226kb	0.9844	0.0056	[[.],[.],...]	⋯

Prediction

eid	inst	class	prob	predicted
13	1	3	0.7235	true

Fig. 1. A possible implementation of an experiment database.

5 A Case Study

In this section we discuss one specific implementation of an experiment database. We describe the structure of this database and the experiments that populate it. Then, we illustrate its use with a few example queries. The experiment database is publicly available on `http://www.cs.kuleuven.be/~dtai/expdb`.

5.1 A Relational Experiment Database

We implemented an experiment database for classifiers in a standard RDBMS (MySQL), designed to allow queries about all aspects of the involved learning algorithms, datasets, experimental procedures and results. This leads to the database schema shown in Fig. 1. Central in the figure is a table of experiments listing the used *instantiations* of learning algorithms, datasets and evaluation methods, the experimental procedure, and the machine it was run on.

First, a *learner instantiation* points to a learning algorithm (`Learner`), which is described by the algorithm name, version number, a url where it can be downloaded and a list of characteristics. Furthermore, if an algorithm is parameterized, the parameter settings used in each learner instantiation (one of which is flagged as default) are stored in table `Learner_parval`. Because algorithms have different numbers and kinds of parameters, we store each parameter value assignment in a different row (in Fig. 1 only two are shown). The parameters are further described in table `Learner_parameter` with the learner it belongs to, its name and a specification of sensible values. If a parameter's value points to a learner instantiation (as occurs in ensemble algorithms) this is indicated.

Secondly, the used dataset, which can be instantiated with a randomization of the order of its attributes or examples (e.g. for incremental learners), is described in table `Dataset` by its name, download url(s), the index of the class attribute and 56 characterization metrics, most of which are mentioned in [9]. Information on the origin of the dataset can also be stored (e.g. whether it was taken from a repository or how it was preprocessed or generated).

Finally, we must store an evaluation of the experiments. The evaluation method (e.g. cross-validation) is stored together with its (list of) parameters (e.g. the number of folds). If a dataset is divided into a training set and a test set, this is defined in table `Testset_of`. The results of the evaluation of each experiment is described in table `Evaluation` by a wide range of evaluation metrics for classification, including the contingency tables[5]. The last table in Fig. 1 stores the (non-zero probability) predictions returned by each experiment.

5.2 Populating the Database

To populate the database, we first selected 54 classification algorithms from the WEKA platform[14] and inserted them together with all their parameters. Also, 86 commonly used classification datasets were taken from the UCI repository and inserted together with their calculated characteristics[6].

To generate a sample of classification experiments that covers a wide range of conditions, while also allowing to test the performance of some algorithms under very specific conditions, a number of algorithms were explored more thoroughly than others. In a first series of experiments, we ran all experiments with their default parameter settings on all datasets. In a second series, we defined at most 20 suggested values for the most important parameters of the algorithms SMO, MultilayerPerceptron, J48 (C4.5), 1R and Random Forests. We then varied each of these parameters one by one, while keeping all other parameters at default. In a final series, we defined sensible ranges for all parameters of the algorithms J48 and 1R, and selected random parameter settings (thus fully exploring their parameter spaces) until we had about 1000 experiments of each algorithm on each dataset. For all randomized algorithms, each experiment was repeated 20 times with different random seeds. All experiments (about 250,000 in total) where evaluated with 10-fold cross-validation, using the same folds on each dataset.

5.3 Querying and Mining

We will now illustrate how easy it is to use this experiment database to test a wide range of hypotheses on the behavior of these learning algorithms by simply writing the right queries and interpreting the results, or by applying data mining algorithms to model more complex interactions. In a first query, we compare the performance of all algorithms on a specific dataset:

[5] To help compare cpu times, a diagnostic test might be run on each machine and its relative speed stored as part of the machine description.

[6] As the database stores a ‘standard’ description of the experiments, other algorithm (implementations) or datasets can be used just as easily.

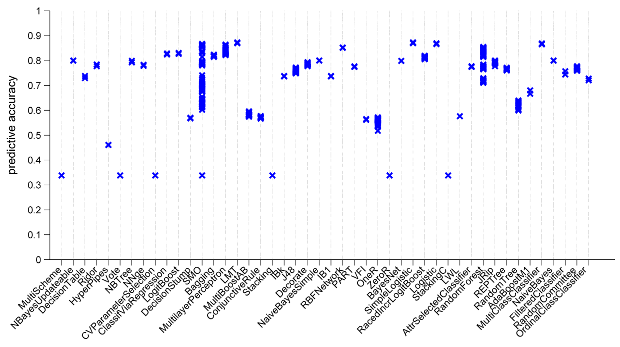


Fig. 2. Performance comparison of all algorithms on the `waveform-5000` dataset.

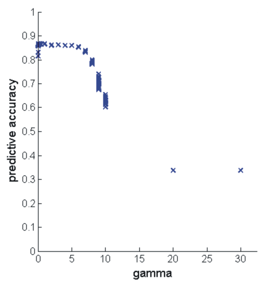


Fig. 3. Impact of the γ-parameter on SMO.

```
SELECT l.name, 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 select the algorithm used and the predictive accuracy registered in all experiments on dataset `waveform-5000`. We visualize the returned data in Fig. 2, which shows that most algorithms reach over 75% accuracy, although a few do much worse. Some do not surpass the default accuracy of 34%: besides SMO and ZeroR, these are ensemble methods that use ZeroR by default.

It is also immediately clear how much the performance of these algorithms varies as we change their parameter settings, which illustrates the generality of the returned results. SMO varies a lot (from default accuracy up to 87%), while J48 and (to a lesser extent) MultiLayerPerceptron are much more stable in this respect. The performance of RandomForest (and to a lesser extent that of SMO) seems to jump at certain points, which is likely bound to a different parameter value. These are all hypotheses we can now test by querying further.

For instance, we could examine which bad parameter setting causes SMO to drop to default accuracy. After some querying, a clear explanation is found by selecting the predictive accuracy and the gamma-value (kernel width) of the RBF kernel from all experiments with algorithm `SMO` and dataset `waveform-5000` and plotting them (Fig. 3). We see that accuracy drops sharply when the gamma value is set too high, and while the other modified parameters cause some variation, it is not enough to jeopardize the generality of the trend.

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. Therefore, we select the dataset name and number of examples, the parameter value of the parameter named `nb of trees in forest` of algorithm

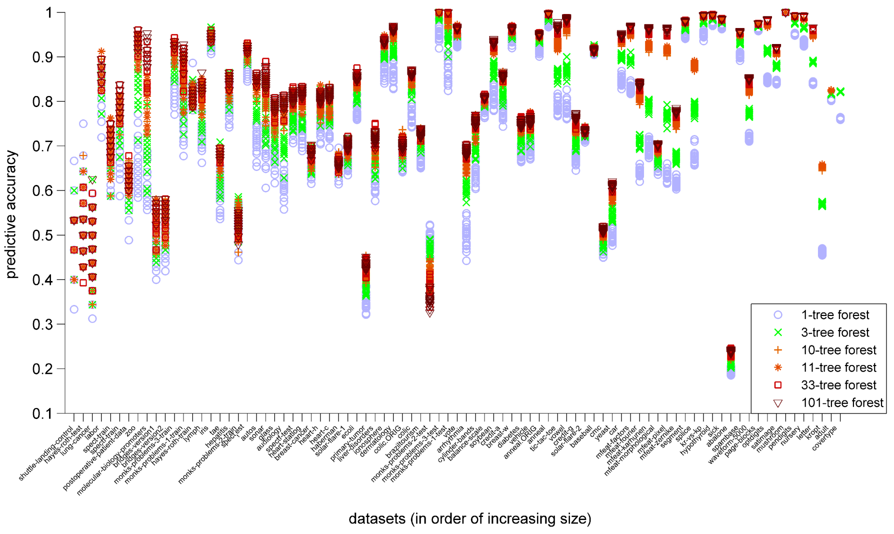


Fig. 4. The effect of dataset size and the number of trees for random forests.

`RandomForest` and the corresponding predictive accuracy. The results are returned in order of 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.name='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
```

When plotted in Fig. 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 less, which is indeed what we would expect as trees become more stable on large datasets.

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 [5] 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 Fig. 5.

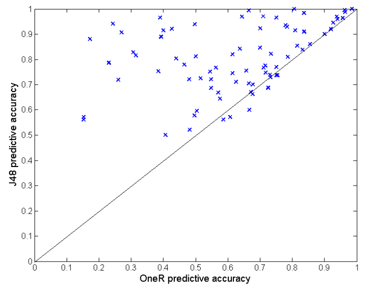


Fig. 5. Relative performance of J48 and OneR.

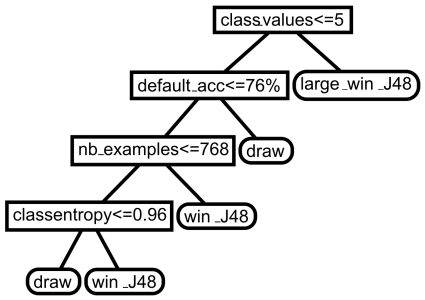


Fig. 6. A meta-decision tree on dataset characteristics.

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 a bit larger here (which may indicate an improvement in decision tree learners and/or a shift towards more complex datasets).

We can also automatically learn under which conditions J48 clearly outperforms OneR. To do this, we queried for the difference in predictive accuracy between J48 and OneR for each dataset, together with all dataset characteristics. Discretizing the predictive accuracy yields a classification problem with 3 class values: "draw", "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 Fig. 6, showing that a high number of class values often leads to a large win of J48 over 1R. Interestingly, Holte's study contained only one dataset with more than 5 class values, which might explain why smaller accuracy differences were reported.

Yet 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 request the results of certain experiments, and gain further insights into the behavior of the algorithms. Also, one can reuse this data (possibly augmented with further experiments) when researching the covered learning techniques. Finally, one can also use our 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 foster 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 experimentational 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.

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. Blockeel, H.: Experiment databases: A novel methodology for experimental research. Lecture Notes in Computer Science **3933**, Springer (2006) 72–85
2. Cohn, D.A.: Neural Network Exploration Using Optimal Experiment Design. Advances in Neural Information Processing Systems **6** (1994) 679–686
3. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research **7** (2006) 1–30
4. Fromont E. and Blockeel H. and Struyf J.: Integrating Decision Tree Learning into Inductive Databases. In Revised selected papers of the workshop KDID'06, Lecture Notes in Computer Science **(to appear)**, Springer (2007)
5. Holte, R.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **11** (1993) 63–91
6. Hoste, V. and Daelemans, W.: Comparing Learning Approaches to Coreference Resolution. There is More to it Than 'Bias'. Proceedings of the Workshop on Meta-Learning (ICML-2005) (2005) 20–27
7. Kalousis, A. and Hilario, M.: Building Algorithm Profiles for prior Model Selection in Knowledge Discovery Systems. Engineering Intelligent Systems **8(2)** (2000)
8. Keogh, E. and Kasetty, S.: On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2002) 102–111
9. Peng, Y. et al.: Improved Dataset Characterisation for Meta-Learning. Lecture Notes in Computer Science **2534** (2002) 141–152
10. Perlich, C. and Provost, F. and Siminoff, J.: Tree induction vs. logistic regression: A learning curve analysis. Journal of Machine Learning Research **4** (2003) 211–255
11. METAL-consortium: METAL Data Mining Advisor. `http://www.metal-kdd.org`
12. Michie, D. and Spiegelhalter D. J. and Taylor C. C.: Machine Learning, Neural and Statistical Classification. Ellis Horwood, New York (1994)
13. Van Someren, M.: Model Class Selection and Construction: Beyond the Procrustean Approach to Machine Learning Applications. Lecture Notes in Computer Science **2049** (2001) 196–217
14. Witten, I.H. and Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques (2nd edition). Morgan Kaufmann (2005)
15. Wolpert, D. and Macready, W.: No free lunch theorems for search. **SFI-TR-95-02-010** Santa Fe Institute (1995)