Experiment databases: a novel methodology for experimental research

Hendrik Blockeel

Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, 3001 Leuven, Belgium
`Hendrik.Blockeel@cs.kuleuven.be`

Abstract. Data mining and machine learning are experimental sciences: a lot of insight in the behaviour of algorithms is obtained by implementing them and studying how they behave when run on datasets. However, such experiments are often not as extensive and systematic as they ideally would be, and therefore the experimental results must be interpreted with caution. In this paper we present a new experimental methodology that is based on the concept of "experiment databases". An experiment database can be seen as a special kind of inductive database, and the experimental methodology consists of filling and then querying this database. We show that the novel methodology has numerous advantages over the existing one. As such, this paper presents a novel and interesting application of inductive databases that may have a significant impact on experimental research in machine learning and data mining.

1 Introduction

Data mining and machine learning are experimental sciences: much insight in the behaviour of algorithms is obtained by implementing them and studying their behaviour on specific datasets. E.g., one might run different learners using different algorithms to see which approach works best on a particular dataset. Or one may try to obtain insight into which kind of learners work best on which kind of datasets, or for which parameter settings a parametrized learner performs best on a certain task.

Such experimental research is difficult to interpret. When one learner performs better than another on a few datasets, how generalizable is this result? The reason for a difference in performance may be in the parameter settings used, it may be due to certain properties of the datasets, etc. Similarly, when we vary one parameter in the hope of understanding its effect on the learner's performance, any effect we notice might in fact be specific for this dataset or application; it might be due to interaction with other, uncontrolled, effects; and even if we eliminate this interaction by keeping the values for other parameters fixed, perhaps the effect would have been different with other values for those parameters.

As a consequence of this complex situation, overly general conclusions are sometimes drawn. For instance, it has recently been shown [5] that the relative performance of different learners depends on the size of the dataset they are used on. Consequently, any comparative results obtained in the literature that do not take dataset size explicitly into account (which is probably the large majority) are to be interpreted with caution.

Clearly, the methodology usually followed for experimental research in machine learning has its drawbacks. In this paper we present a new methodology that avoids these drawbacks and allows a much cleaner interpretation of results. It is based on the concept of *experiment databases.* An experiment database can be seen as a kind of inductive database, and the methodology we propose essentially consists of querying this database for patterns. As such, this paper presents a novel and potentially interesting application of inductive databases that may have a significant impact on how experimental research in machine learning and data mining is conducted in the future.

We present the basic ideas of the approach, but many details are left open and there remain several interesting questions for further research.

We will first give an overview of the shortcomings and caveats of the classical experimental methodology (Section 2), then we introduce informally the concept of experiment databases and show how they can be used (Section 3). We summarize our conclusions in Section 4.

2 The classical experimental methodology

Let us look at a typical case of an experimental comparison of algorithms. A realistic setup of the experiments is:

– A number of datasets is chosen; these may be existing benchmarks, or synthetic datasets with specific built-in properties (for instance, one may want to control the skewness of the class distribution in the dataset)
– On all of these datasets, a number of algorithms are run. These algorithms may have different parameter settings; typically they are run with "suitable" (not necessarily optimal) parameters.
– Certain performance criteria are measured for all these algorithms.

The above methodology, while very often followed, has two important disadvantages: the generalizability of the findings is often unclear, and the experiments are not reusable.

2.1 Unclear Generalizability

The conclusions drawn from experiments may not hold as generally as one might expect, because the experiments typically cover a limited range of **datasets** as well as **parameter settings**.

Comparisons typically happen on a relatively small number of datasets, in the range of 1-30. Imagine describing all datasets using a number of properties such as the number of examples in the dataset, the number of attributes, the skewedness of the class distribution, the noise level, level of missing values, etc. Many such properties can be thought of, leading to a description of these datasets in a high-dimensional space, let us call it D-space (D for datasets). Clearly, in such a high-dimensional space, a sample of 1-30 points (datasets) is extremely sparse. As a consequence, any experimental results obtained with such a small number of datasets, no matter how thoroughly the experiments have been performed, are necessarily limited with respect to their generalizability towards other datasets.

This is not a purely theoretical issue; as we already mentioned, recent work [5] has shown how the relative performance of different learning algorithms in terms of predictive accuracy may depend strongly on the size of the dataset. This sheds a new light on hundreds of scientific papers in machine learning and data mining. Indeed, many authors implicitly assume the predictive accuracy of algorithms, relative to each other, to be independent of data set size (or any other data set parameters, for that matter).

A second possible cause for limited generalizability is that many algorithms are highly parametrized. Let us call an algorithm with completely instantiated parameters a ground algorithm. Then typically, a limited number of ground algorithms is used in the experiments. If, similar to D-space, we define for each parameterized algorithm or class of algorithms its parameter space (P-space), then again the experiments involve a very sparse sample from this space, and the results may not be representative for the average or optimal behaviour of the algorithm.

An additional problem here is that authors presenting new algorithms often understand their own algorithm better and may be better at choosing optimal parameter values for their own approach, putting the existing algorithm at a small disadvantage.

The above discussion was from viewpoint of comparing algorithms, but the generalizability problem also occurs when, for instance, the effect of a single parameter of the algorithm or dataset on the performance of the system is investigated. For instance, to study the robustness of an algorithm to noise, one would typically run it on a variety of data sets with increasing noise levels. Using synthetic datasets in which the noise level can be controlled, it makes sense to increase the noise level while keeping all other parameters of the data set and the algorithm constant, and plot performance as a function of the noise level.

Here, too, generalizability is problematic. If we look at such approaches in $D \times P$-space or P-space, it is clear that by varying one parameter of the dataset or algorithm, one constructs a sample that lies in a one-dimensional subspace of the high-dimensional space. The sample is typically dense within this subspace, but still located in a very limited area of the overall space, so, again, the generalizability of the results may be low due to this. For instance, one might conclude that a certain parameter has a large influence on the efficiency of an algorithm, when in fact this holds only for datasets having certain specific properties.

2.2 No Reusability

In the classical methodology, the experimental setup is typically oriented towards a specific goal. The above example regarding the study of the effect of noise illustrates this: since the researcher knows that she wants to study the effect of noise, she varies the noise level and nothing else. Such an experimental setup is clearly goal-oriented. Each time the researcher has a new experimental hypothesis to be tested or wants to investigate a new effect, this will involve setting up and running new experiments. This obviously takes additional work. Moreover, there may be practical problems involved. For instance, if there is a large time span between the original experiments and the newly planned experiments, certain algorithm implementations may have evolved since the time of the original experiments, making the new results incompatible with the old ones.

2.3 Summary

The above problems can be summarized as follows:

1. Experimental results regarding the relative performance of different methods and the effect that certain parameters of the algorithm or properties of the dataset have, may have limited generalizability.
2. For each additional experimental hypothesis that is to be investigated, new experiments must be set up.

We will next show how the use of experiment databases can solve these problems.

3 Experiment databases

An experiment database (in short, an ExpDB) is a database that contains results of many random experiments. The experiments in themselves are unfocused; the focused, goal-oriented experiments mentioned above will be replaced by specific queries to the database.

In the following, we first describe how the database can be created, and next, how it can be mined to obtain useful knowledge. For simplicity, we start with the case where we are interested in the behaviour of a single algorithm. The extension towards a comparison of multiple algorithms is non-trivial but will briefly be discussed afterwards.

3.1 Creating an experiment database

Assume we have a single algorithm A with a parameter space P. Assume furthermore that some fixed method for describing datasets is given, which gives rise to a D-space D. (Note that there is a difference between the dataset space and the D-space; the D-space is an n-dimensional space containing *descriptions* of datasets, not the datasets themselves). Finally, we denote with M a set of performance metrics; M may include runtime, predictive accuracy, true and false positive rates, precision, recall, etc.

We further assume that a dataset generator G_D is given. G_D generates datasets at random, according to some distribution over D. This distribution need not be uniform (this would be impossible if some parameters are unbounded), but it should *cover* all of D, i.e., for each $d \in D$, the distribution must assign a non-zero probability to an element "sufficiently close to" d. For instance, the "dataset size" parameter is continuous but one could choose to use only the values 10^k with $k = 2, 3, 4, 5, 6, 7, 8, 9$. A dataset generator using these values could be said to "cover" datasets of a hundred up to a billion instances.

Finally, we assume we have a random generator G_P for parameter values of the algorithm; again this generator should generate values according to a distribution that covers, in the same sense as above, all of P.

Now we can create a table of experimental results, as follows:

```
Experiment Database Creation:
Input: $A$, $D$, $G_D$, $P$, $G_P$, $M$
Output: a table $T$
Create a table $T$ with attributes from $D \times P \times M$.
for i = 1 to $k$:
    Generate a random data set DS using $G_D$
    Let $d$ be the D-space description of DS
    Generate random parameter values $p$ according to $G_P$
    Run $A$ with parameter values $p$ on DS
    Let $m$ be the result of this run in terms of the performance metrics $M$
    Insert a new row containing $d, p, m$ in the table
```

The table now contains the results of k random experiments, that is, runs of algorithm A with random parameters on random datasets. We will discuss later how large k would need to be for this table to be useful.

A final note regarding the creation of the database: we have assumed that datasets and algorithm parameters are chosen at random, and we will continue our discussion under this assumption. However, total randomness is not required; we require only that the whole $D \times P$-space is covered. As several researchers have pointed out (personal communication), it may well be possible to do better, for instance, by filling the table according to experiment design principles. The analysis we further make on the required size of the table should therefore be considered a worst-case analysis.

3.2 Mining the database

We have a database of "random" experiments, but we are in fact interested in testing specific hypotheses about the behaviour of the algorithm A, or investigating the influence of certain parameters or dataset properties on A's performance.

If the table is considered an inductive database and we can query for patterns in the table, then such knowledge is immediately obtainable in a very straightforward way.

Suppose A is some frequent itemset discovery algorithm, and we want to see the influence of the total number of items in the dataset on A's runtime. Assuming NItems (number of items in the dataset) is one dimension of D and the attribute Runtime is included in M, the following simple SQL query

```
SELECT NItems, Runtime
FROM EXP
SORT BY NItems
```

gives us the results. (In practice we would of course graphically plot Runtime against NItems; such a plot can be readily derived from the result of the SQL query. In this text we are mainly concerned with how the results to be visualized can be obtained with a query language, rather than the visualization itself.)

The Runtime attribute is of course related to many other parameters, which vary randomly, and as a result the above query may result in a very jagged plot (e.g., Runtime for NItems=100 might be larger than Runtime for NItems=1000 just because lower values for the MinSupport parameter were used for the former). In the classical experimental setting, one would keep all other parameters equal when one is interested in the effect of the number of items only. For instance, knowing that the minimal support parameter of a frequent itemset mining algorithm typically has a large influence on the run time, one might keep this parameter fixed at, say, 0.05. This can easily be simulated with our approach:

```
SELECT NItems, Runtime
FROM EXP
WHERE MinSupport = 0.05
SORT BY NItems
```

Assuming that G_P generates 10 different values for MinSupport, uniformly distributed, the result of this query is based on roughly 10% of the database. Compared to the classical setting where the experimenter chooses in advance to use only MinSupport=0.05, we need to populate the database with 10 times as many experiments. Figure 1 illustrates how the WHERE constraint gives rise to fewer points with a clearer trend.

Now, as said before, the influence of NItems on Runtime might be different if we vary other parameters, such as MinSupport. We can easily vary the MinSupport condition in the above query to check whether this is the case. For instance, using an ad-hoc scripting language that allows to plot query results, and where we use the notation \$x inside an SQL query to refer to the value of a variable x defined outside the query, we could write

```
FOR ms = 0.001, 0.005, 0.01, 0.05, 0.1 DO
  PLOT
    SELECT NItems, Runtime
    FROM EXP
    WHERE MinSupport = $ms
    SORT BY NItems
```

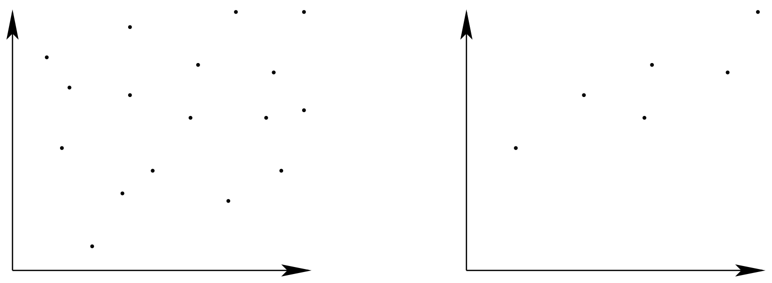

Fig. 1. An impression of what plots would typically look like under different constraints in the SQL query. The left plot shows a cloud obtained from all data points, the right plot shows a cloud obtained using MinSupport=0.05. The latter contains a subset of the former's points, but with a clearer trend.

and get a figure showing different curves indicating the effect of NItems, one for each value of MinSupport.

In the classical experimental setting, if the experimenter realizes that this would be a better approach only after having performed the experiments, she needs to set up new experiments. The total number of experiments will then be equal to the number of experiments in our database.

From the above discussion, two differences between the classical approach and the ExpDB approach become clear.

1. *In the classical approach, experiments are set up from the beginning to test a specific hypothesis; to test new hypotheses, new experiments are needed. With the ExpDB approach, many experiments are run once, in advance, independent of the goal; to test new hypotheses, new queries are run on the database.*
2. *Due to stronger randomization of the experiments, the experiment database approach tends to yield results in which fewer parameter values are kept constant. This may lead to less convincing (though more generalizable) results. To some extent, queries can be written so as to counter this effect.*

Looking at the above SQL queries, another advantage of the ExpDB approach becomes clear. As said before, the second query has the advantage of producing a less jagged curve, in other words, less variance in the results, but this comes at the cost of obtaining less generalizable results. It is immediately clear from the form of the second query that the obtained results are for MinSupport=0.05, whereas in the classical approach this is left implicit. In both queries, nothing is explicitly assumed about the unmentioned parameters, and indeed in the actual experiments these parameters get random values and we may assume that roughly the whole range of values is covered. Thus, these queries explicitly state how general a result is. The first query yields the most general results (and, as a consequence, the least conclusive results, as far as the detection of trends or rejection of hypotheses is concerned). The second query yields results for a more specific case; the results are more conclusive but may be less generalizable.

3. *The ExpDB approach explicitates the conditions under which the results are valid.*

What if, at some point, the researcher realizes that besides MinSupport, also the number of transactions NTrans in the database might influence the effect of NItems on Runtime? Now the researcher following the classical approach has several options. He may run new experiments, fixing MinSupport at 0.05 but now varying NTrans. He might also decide to combine each value of MinSupport with each value of NTrans, which would give a detailed account of the effect of both together on Runtime. Or he might randomize the MinSupport setting while controlling NTrans.

In each case, new experiments are needed. The original experiments are not usable because the NTrans parameter was not varied. Even if it was varied, the necessary statistics to test the new hypothesis have not been recorded in the original experiments because they were not needed for the goal of those experiments.

The situation is entirely different with the ExpDB approach. The experiment database was created to be as generally useful as possible; a large number of statistics, potentially useful for a variety of purposes, have been recorded. When the researcher wants to test a new hypothesis, he just needs to query the experiment database. In other words, a fixed set of experimental results is re-used for many different hypothesis tests or other kinds of investigations. This leads us to a fourth difference:

4. *The ExpDB approach is more efficient than the classical approach if multiple hypotheses will be tested.*

Clearly, the ExpDB approach makes controlled experimentation much easier. As a result, such experimentation can easily be performed in much more depth than with the classical approach. For instance, classically, higher-order effects are usually not investigated. Researchers vary one parameter P1 to investigate its effect, then vary another parameter P2 to investigate its effect. This leads to the discovery of only so-called marginal effects of the parameters. By varying P1 and P2 together, one can discover so-called higher order effects; for instance, one might discover that the effect of P1 on performance is large when P2 is low but not so large when P2 is high.

With the ad-hoc language introduced before, such interaction can be studied easily, for instance using the following SQL query:

```
FOR a=0.01, 0.02, 0.05, 0.1 DO
  FOR b = 1000, 10000, 100000, 1000000 DO
    PLOT
        SELECT NItems, Runtime
        FROM EXP
        WHERE MinSupport = $a AND $b <= NTrans < 10*$b
        SORT BY NItems
```

This shows that

5. *The ExpDB approach makes it easy to perform in-depth analyses of both marginal and higher-order effects of parameters and dataset properties.*

The inductive database approach, where mining is performed by querying for patterns using a special-purpose inductive query language [1], allows us to go further. While the above kind of queries amount to checking manually for the effects of specific parameters or dataset characteristics, or interactions between them, one can easily think of more sophisticated data mining approaches that allow the researcher to ask questions such as "what is the parameter that has the strongest influence on the predictive accuracy of my decision tree system", or "are there any dataset characteristics that interact with the effect of parameter P on predictive accuracy", etc. Clearly, inductive database query languages are needed for this purpose. A possible query would be

```
SELECT ParName, Var(A) / Avg(V) as Effect
FROM AlgorithmParameters,
     SELECT $ParName, Var(Runtime) as V, Avg(Runtime) as A
     FROM EXP
     GROUP BY $ParName
GROUP BY ParName
SORT BY Effect
```

This SQL-like query (it is not standard SQL)[1] requires some explanation. The inner SELECT query takes a parameter $ParName (e.g., $ParName = 'MinSupport') and computes the average A and variance V of the runtimes measured for specific values of $ParName. If $ParName = 'MinSupport', then the result of the inner query is a table with attributes MinSupport, A, V, and for each occurring value of MinSupport the corresponding average runtime is listed as well as the runtimes' variance.

The outer SELECT query takes a table AlgorithmParameters that is supposed to have an attribute ParName. For each value of ParName, the inner query is instantiated and run. We again use the convention that $ParName refers by definition to the *value* of ParName. The result of this construction is a "table" with attributes ParName, $ParName, A, V. (It is not a standard SQL table because the second attribute does not have a fixed name.) The SELECT part of the outer query projects this onto ParName and Effect, sorting the parameters according to their influence on runtime, where this influence is defined as the ratio of the variance of the averages of the different groups to the average variance within these groups.

6. *The ExpDB approach, if accompanied by suitable inductive querying languages, allows for a much more direct kind of questions, along the lines of "which parameter has most influence on runtime", instead of finding this out with repeated specific questions.*

[1] The same query could be expressed in standard SQL if the parameter names listed in ParNames are hardcoded in the query, but this makes the query lengthy and cumbersome, and less reusable. We prefer this more compact and intuitive notation.

A final advantage of the ExpDB approach is their reusability: experiment databases could be published on the web, so that other researchers can investigate the database in ways not thought of by the original researcher. Currently, the tendency is to make available the datasets themselves, possibly also implementations of systems used, but the actual experiments and conclusions are described on paper (and sometimes the experimental settings are not described in sufficient detail for others to reconstruct the experiments exactly). By following the ExpDB approach and publishing the experiment database, a detailed log of the experiments remains available, and it becomes possible for other researchers to, e.g., refine conclusions drawn by previous researchers from these experiments. It is likely that such refinements would happen less frequently than is currently the case, exactly because the ExpDB approach enforces a much more diligent experimental procedure.

7. *The ExpDB approach leads to better reusability of experiments and better reproducibility of results.*

3.3 A summary of the advantages

We can summarize the advantages of the experiment database approach as follows:

- Efficiency. The same set of runs (of algorithms on datasets) is reused for many different goal-oriented experiments.
- Generalizability. The generalizability of experimental results in the original setting is often unclear. With the experiment database approach, due to randomization of all parameters not under investigation, it is always clear to what extent results are generalizable. Results are obtained from a relatively large sample in $P \times D$-space that covers the whole space, instead of from a small sample covering a small part of the space.
- Depth of analysis. It is easy to investigate the combined effect of two or more parameters on some performance metric, or, in general, to check for higher-order interactions (in the statistical sense) between algorithm parameters, dataset properties, and performance criteria.
- True data mining capacity. With a suitable query language, one can also ask questions such as "what algorithm parameters have the most influence on the accuracy of the algorithm?" (and hence are most important to tune).
- Reusability. Publishing an experimental database guarantees that a detailed log of the experiments is available. It makes it easier for other researchers to reproduce the experiments, and it makes it possible for them to investigate other hypotheses than the ones described by the authors of the experiment database.

3.4 Size of the table

How large should the number of tuples in the experiment table, k, be? Assume that we want each point that we plot to be the average of at least e examples and no parameter or dataset characteristic has more than v values (continuous variables are discretized). $e = 30$ and $v = 10$ are reasonable values. Then we need to have $ve = 300$ experiments to measure the effect of any single parameter on any single performance metric. Measuring the effect while keeping the value of a single other parameter constant, may require up to v times more data to obtain equally precise results (e.g., averaged over 30 measurements); the same holds for measuring any second-order effects. In general, to measure mth-order effects, we need ev^m experiments. Thus, a table listing a few thousand experimental results is typically enough to perform more thorough experiments than what is typically done with the classical approach. Note that it is not problematic for the creation of this table to create hours or even days, as this can be done off-line, possibly as background computations, and subsequent experimentation will take require very little time. In this way, assuming the use of reasonably efficient algorithms, creation of a database of 10,000 to 100,000 experiments is quite feasible.

As mentioned before, this analysis holds for a method where the experiment database is filled randomly. Following experiment design principles, it may be possible to improve this number, but we have not looked into this issue yet. The main conclusion here is that even with the randomized approach, the number of experiments needed is not prohibitive.

Note that there is always the possibility of filling the database also in a goal-oriented way, by only generating tuples with specific values for certain attributes. This defeats part of the purpose of the ExpDB, but makes it possible to build an ExpDB with exactly the same number of experiments as would have been needed in the classical setting, while still allowing deeper analysis.

3.5 The multiple-algorithm case

Up till now we have assumed that the experiments recorded in the ExpDB involve one single algorithm. In practice, it will be useful to build ExpDB's with information on multiple algorithms.

One problem that then arises, is that different algorithms typically have different sets of parameters. As a result, there exists no elegant schema for the single table we considered up till now.

A solution is to have multiple tables for describing parameter settings, where each class of algorithms has its own different table and schema. An example is shown in Figure 2. This necessitates a relational data mining approach [2]. The SQL-like “mining” approach that we have discussed before is not limited to querying a single experiment table, and hence simple queries can be asked to compare for instance the best-case, worst-case, average-case behaviour of different algorithms, possibly with constraints on the parameters of the algorithms and datasets. For instance, the query

```
SELECT AVG(s.Accuracy) - AVG(t.Accuracy),
       VAR(s.Accuracy), VAR(t.Accuracy)
FROM (ExpDB JOIN Alg1) s , (ExpDB JOIN Alg2) t
WHERE Alg1.C=0 and Examples < 1000
```

for the database schema shown in Figure 2 compares the average and variance of the accuracy of algorithms Alg1 and Alg2 under the constraint that Alg1's C parameter is 0 (e.g., the default), on datasets of less than one thousand examples.

ExpDB

ExpID	Attr	Examples	Target Complexity	Runtime	Accuracy

Alg1

ExpID	A	B	C	D

Alg2

ExpID	A	B	E	F

Fig. 2. Schema for an experiment database containing data for two algorithms Alg1 and Alg2 with different parameters.

An alternative to this approach could be to define a generic description of algorithms: a so-called A-space, in which any algorithm is described using algorithm-independent properties, similarly to the D-space that describes datasets without giving full information on the dataset. An advantage would be that one would be able to detect dependencies between algorithm-independent characteristics of learners, and the performance on certain kinds of datasets. It is unclear, however, what kind of characteristics those should be.

3.6 A connection to Meta-learning

Meta-learning is a subfield of machine learning that is concerned with learning to understand machine learning algorithms by applying machine learning methods to obtained experimental results. While our ExpDB approach has been presented as a way to improve the experimental methodology typically followed by machine learning and data mining researchers, it is clear that the approach is very suitable for meta-learning:

- Working with synthetic datasets solves the problem of sparse data that is so typical of meta-learning. While the UCI repository, for instance, is a sizeable collection of machine learning problems, from the meta-learning point of view each UCI dataset yields a single example, so the meta-learning dataset derived from the UCI repository contains only a few dozen examples. This makes it difficult to derive any conclusions. A synthetic dataset generator, such as used in the ExpDB approach, appears crucial for the success of meta-learning.
- As explained, our approach allows thorough investigation of the interactions between algorithm parameters, dataset characteristics, and performance metrics, in addition to allowing a comparison between different kinds of algorithms.

Conversely, a lot of existing work in meta-learning is very useful for the concept of experiment databases. For instance, significant efforts have been invested in the description of datasets and algorithms [6, 4] and in methods for generating synthetic datasets (Soares, Džeroski, personal communication). All of these can be used to give the work on experiment databases a headstart.

4 Conclusions

We have presented a novel methodology for experimental research in machine learning and data mining. The methodology makes use of the concept of experiment databases. The idea behind this is that a database of random experiments is first created, then hypotheses can be tested ad libitum by just querying the database instead of repeatedly setting up new experiments. The experiment database approach has many advantages with respect to reusability, reproducibility and generalizability of results, efficiency of obtaining them, ease of performing thorough and sophisticated analysis, and explicitness of assumptions under which the obtained results are valid.

The current paper is obviously very preliminary; it presents the basic ideas and promises of experiment databases. There are many open questions, such as:

– *The format of the D-space*: it is easy to list some characteristics of datasets that might be useful, but difficult to ensure no important ones are missed. Some work has already been done on this in the meta-learning community [4], but we expect further efforts on this may yield more results.
– *The dataset generator*: such a generator generates data according to a certain distribution; how do we specify this distribution? For supervised learners a target concept must be included; how do we generate this target concept? Information on this concept (e.g., its complexity) is part of the D-space.
– *An inductive query language*: In the above we have used an ad hoc language for inductive queries. It is necessary to define a suitable inductive query language for the kind of patterns we are interested in. It is not clear if any of the existing query languages are suitable; for instance, languages for finding frequent itemsets or association rules [3] are not immediately applicable. It seems that a kind of standard SQL that allows the user to mix the meta and object level in a single query, would be useful.

We believe that further research along the proposed direction has the potential to lead to much better experimental research in machine learning and data mining, and to ultimately lead to a greatly improved understanding of the strengths and weaknesses of different approaches.

Acknowledgements

The author is a post-doctoral fellow of the Fund for Scientific Research of Flanders, Belgium (FWO-Vlaanderen). He thanks Sašo Džeroski, Carlos Soares, Ashwin Srinivasan, and Joaquin Vanschoren for interesting comments and suggestions.

References

1. L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, 2002.
2. S. Džeroski and N. Lavrač, editors. *Relational Data Mining.* Springer-Verlag, 2001.
3. R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules in SQL. *Data Mining and Knowledge Discovery*, 2:195 – 224, 1998.
4. Y. Peng, P. Flach, C. Soares, and P. Brazdil. Improved dataset characterisation for meta-learning. In *Proceedings of the 5th International Conference on Discovery Science*, volume 2534 of *Lecture Notes in Computer Science*, pages 141–152. Springer-Verlag, 2002.
5. C. Perlich, F. Provost, and J. Siminoff. Tree induction vs. logicstic regression: A learning curve analysis. *Journal of Machine Learning Research*, 4:211–255, 2003.
6. B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 743–750. Morgan Kaufmann, 2000.