Building Algorithm Profiles for Prior Model Selection in Knowledge Discovery Systems

Melanie Hilario & Alexandros Kalousis
CSD, University of Geneva
CH-1211 Geneva 4, Switzerland
hilario|kalousis@cui.unige.ch

ABSTRACT

We propose the use of learning algorithm profiles to address the model selection problem in knowledge discovery systems. These profiles consist of metalevel feature-value vectors which describe learning algorithms from the point of view of their representation and functionality, efficiency, robustness, and practicality. Values for these features are assigned on the basis of author specifications, expert consensus or previous empirical studies. We review past evaluations of the better known learning algorithms and suggest an experimental strategy for building algorithm profiles on more quantitative grounds. Preliminary experiments have disconfirmed expert judgments on certain algorithm features, thus showing the need to build and refine such profiles via controlled experiments.

1 WHY BUILD LEARNING ALGORITHM PROFILES?

The emergence of multi-paradigm data mining tool suites has paradoxically become a source of perplexity to the end-user who often finds herself at a loss as to which learning model or algorithm to use. Thus the model selection problem, broadly defined as the problem of selecting the most appropriate learning model or algorithm for a given application task, has come to the forefront of research in machine learning and data mining.

The model selection problem has typically been addressed by systematic experimentation using methods like cross-validation. However, exhaustive experimentation is becoming more and more impractical with the growing number of learning techniques available. It is thus important, at the outset, to be able to restrict the hypothesis space by focusing on a subset of the available learning algorithms. We shall call this pre-selection of the most promising learning models/algorithms *prior model selection*. These pre-selected models are then instantiated and applied to the dataset, giving rise to a number of learned models. We shall use the term *posterior model selection* to designate the choice of the most appropriate learned model (or hypothesis) from this final candidate set.

Posterior model selection is easier than prior model selection in the sense that it can rely on actual, quantifiable data, namely the evaluation measures taken on the learning process as well as the learned model. Examples of these are the learned model's off-sample error rate, time and space taken to train the final model, and time and space taken by this learned model to execute its task (e.g., classification). On the contrary, prior model selection has no such precise measures to rely on. Evaluation metrics gathered for a particular learning algorithm cannot be used as such on new problems. Rather they should be decontextualized and abstracted across the most diverse training episodes to yield more general properties that we shall call learning model characteristics. In other words, model characteristics—as opposed to evaluation metrics—are meta-attributes associated with a learning model or algorithm insofar as they are considered generic to the class of all the learned models or hypotheses generated using this model/algorithm. These characteristics are grouped together to form a model/algorithm profile. The claim of this paper is that prior model selection can rely on these algorithm profiles in much the same way as posterior model selection relies on evaluation metrics. While the scope of this paper is limited to learning algorithms for classification tasks, it should be fairly clear that these characteristics can be applied to learning models for other tasks such as regression or association.

This paper proposes a standard vocabulary for building profiles of learning algorithms. Section 2 explores the sources from which we draw knowledge of the characteristics of a given learning algorithm. The rest of the paper discusses three groups of model characteristics depending on whether they concern the model's representation and functionality (Section 3), efficiency (Section 4) or robustness (Section 5). For each characteristic, we point out its utility in prior model selection and explore issues concerning its representation and evaluation. We also review how familiar algorithms have been assessed or situated with respect to each characteristic in the current state of the art. We conclude with preliminary experimental findings which disconfirm widely held judgments, thus highlighting the need for more rigorous empirical grounds on which to build algorithm profiles.

2 KNOWLEDGE SOURCES OF MODEL CHARACTERISTICS

Our knowledge of model characteristics comes from three different types of sources. First, certain characteristics are given explicitly in algorithm specifications; they concern the basic requirements, capabilities or limitations of an algorithm. It is possible, but hardly useful, to determine them empirically. Examples of such author-specified characteristics are the number and definition of user-tuned runtime parameters, the ability of algorithms to handle misclassification costs, or the types of attributes supported.

A second source is expert knowledge, more precisely consensus knowledge of experts in the domain of machine learning and data mining. However, when experts disagree or are simply in doubt, one can turn to the third source of model characteristics, that is, evaluation metrics from past learning episodes. Evaluation metrics typically pertain to a specific execution of a learning model or algorithm; they are measures taken on either the learned model or the training process that produced it. The most commonly used evaluation metrics are:

- predictive accuracy of the learned model, i.e., its hit (or error) rate on an independent test set

- training time: length of the training process, or time taken to produce the learned model
- training space: memory demand of the training process, or memory used to produce the learned model
- execution time: time taken by the learned model to classify test instances
- execution space: memory resources used by the learned model to classify test instances

These metrics describe specific training processes or models generated by a learning algorithm, but they can be averaged or otherwise generalized to characterize that algorithm or its model class. For instance, training time and space are examples of model characteristics that can be derived by analysing the means and variances of the corresponding evaluation metrics. Other model characteristics can be inferred indirectly from evaluation metrics, i.e., by studying the evolution of one or several performance metrics as a function of some independent variable or factor. For instance, the scalability of a learning algorithm—its behavior with respect to the size of the training set—can be determined empirically by plotting, say, its error rate or training time against some measure of dataset size.

In short, for each evaluation metric, there is a corresponding model characteristic—with one significant exception: predictive accuracy. It is now generally admitted, on the basis of theoretical findings such as Schaffer's conservation law of generalization performance [19], that one algorithm cannot systematically beat all others on generalization accuracy. Thus it makes no sense to describe an algorithm's accuracy as being uniformly *high* or *low,* since positive performance in some learning situations is always offset by negative performance in others. The basic question then is not whether, but undere what conditions, a given algorithm can attain high or low accuracy.

The past decade has seen a considerable amount of experimental work aimed at evaluating and comparing learning algorithms on the basis of the above metrics. It is difficult to determine which findings obtained from past experimentation can be used as a basis for building algorithm profiles. Most experiments, with the notable exception of those undertaken in the Statlog project [12] are too limited in scale. While it may be possible to synthesize findings from a number of small-scale experiments, it is difficult to factor out differences in implementation, measures, and experimental design, not counting possible observational bias (as very often the experimenter is the author of one of the algorithms being examined). On the whole, there remains a need for large-scale, systematic experimentation aimed at profiling learning algorithms on the basis of a common experimental design. The purpose of this paper is to lay the groundwork for such experiments by identifying and reviewing current understanding of algorithms and their characteristics, grouped according to four dimensions: representation and functionality, efficiency, robustness and practicality. For reasons of space, only the first three dimensions are discussed in this paper.

3 CHARACTERIZING REPRESENTATION AND FUNCTIONALITY

The first group of characteristics concerns the representational power and functionality of a learning algorithm. Examples are the types of attribute values that can be processed by the algorithm, the bias-variance profile of the model class it represents, its incrementality, and its ability to deal with costs. We shall focus on the bias/variance profile and incrementality.

3.1 Bias/variance profile

The generalization error of a model can be decomposed into two parts, bias and variance. Bias, or approximation error, is the systematic part of the error that is due to the choice of the model. Variance, or estimation error, is the part of the error that comes from random variations in the data. There is a trade-off between bias and variance in that low bias often comes with high variance and high bias with low variance [7]. Bias and variance can be measured only with regard to the expectation of a learned model's actual output, averaged over the ensemble of possible training sets D for a fixed sample size N. However, a prototypical bias-variance profile can be associated with a learning model/algorithm. The bias-variance profile is a rough, qualitative indication of the direction in which the algorithm tends to resolve the trade-off between bias and variance.

Learning algorithms with a high-bias profile usually generate simple, highly constrained models which are quite insensitive to data fluctuations, so that variance is low. Linear discriminants and perceptrons are situated at the high-bias endpoint of the spectrum as they have a highly limited set of models to fit the data. The assumption is that the class can be expressed as a linear combination of attribute values, so that the contribution of bias to predictive error will be quite high if the data are not linearly separable. Logistic discriminants have slightly lower—though still relatively high—bias. Naive Bayes is also considered to have high bias because it assumes that the dataset can be summarized by a single probabilistic distribution and that this is sufficient to discriminate between classes.

On the contrary, algorithms with a high-variance profile can generate arbitrarily complex models which fit data variations more readily. Examples of high-variance algorithms are decision trees, neural networks, and the family of memory-based learners. The obvious pitfall of high-variance model classes is overfitting, so learning models of this category usually involve the use of techniques for adjusting complexity. For instance, decision trees typically use pruning techniques whereas neural network complexity can be controlled via regularization, weight decay or early stopping.

3.2 Incrementality

The notion of incrementality in machine learning involves the number of past examples a learning algorithm should reprocess during each learning step. An algorithm is nonincremental if it reprocesses all earlier training instances in order to learn from each new instance. It is incremental if it can build and refine a concept gradually as new training instances come in, without reexamining all instances seen in the past. Each mode of learning has its advantages: nonincremental or batch learners usually avail of overall information about the training sample to take more enlightened decisions; on the other hand, there is an increasing demand for incremental learners as massive volumes of data become available for data mining applications. A typical drawback of incremental learners is their sensitivity to the order of presentation of training instances [18].

Instance-based or lazy learners are by nature incremental learners; since they simply store new instances, there is no need to reexamine past training instances. However, if we consider the classification phase, then standard k-NN is nonincremental in the sense that all past training instances need to be examined in order to find the *k* nearest neighbors. More sophisticated versions called edited nearest neighbor methods store only selected examples and can be considered incremental both as learners and classifiers. Neural network learning is intrinsically nonincremental. Other models were originally designed for batch learning, but have been modified in view of incrementality. For instance, decision trees are typically built by testing attributes of all training instances at each node, but incremental versions have been proposed both for univariate [21] and multivariate decisions trees [3].

4 CHARACTERIZING EFFICIENCY

Characteristics pertaining to a learning algorithm's efficiency are training and execution time as well as training and execution space required. (Recall that the most commonly used performance measure, predictive accuracy, cannot be generalized in order to characterize a learning algorithm or the ensemble of models it generates.)

4.1 Training and execution time

As a performance metric on a single learning episode, training time is the time taken to produce a specific classifier. Comparing time spent by different algorithms remains a delicate process, even if care is taken to standardize measurement units. As was observed in the Statlog project, time measurements do not always measure the same thing. Using an algorithm's default setting may save parameter tuning time and reduce total training time significantly, at the price of a loss in accuracy. Rather than raw time measurements, [12] suggest expressing time in terms of some function of the dataset size.

Although training time varies with the nature of the application task and dataset, specialists generally agree on a partial ordering of the major classes of learning algorithms. For instance, all k-NN based or lazy learning methods require zero training time because the training instance is simply stored. Standard Naive Bayes methods also train very quickly since they require only a single pass on the data either to count frequencies (for discrete variables) or to compute the normal probability density function (for continuous variables under normality assumptions). For nonstandard Bayesian approaches which relax the normality assumption, discretization of continuous variables increases overall training time; more sophisticated methods like kernel density estimation take even longer. Univariate decision trees are also reputed to be quite fast—at any rate, several orders of magnitude faster than oblique decision trees in the presence of partitioning datasets. Neural networks are notorious for their computational cost; this is a recurrent finding in many comparative studies [22, 20, 12]. An exception among neural network models is the perceptron, which achieves training times comparable to those of ID3 or C4.5 in some cases [13]. Among statistical algorithms, nonparametric models such as density estimation usually take longer than the standard techniques. However, systematic experimentation needs to be undertaken in order to convert these partial orderings into a total ordinal scale.

As for execution time, one object of consensus is that k-NN and most of its variants lie on the high endpoint of the spectrum, in particular for large datasets. This fact, empirically observed in large-scale experiments like those of Statlog, is predictable from the approach: k-NN stores all training instances and postpones the burden of computation to the execution stage, where classification time grows linearly with the number of stored training instances. However, this can be reduced to logarithmic time by replacing sequential storage with more sophisticated indexing and memory organization techniques. In general, decision trees and other logic-based methods are fastest at execution time, since they only need to test a small subset of features.

4.2 Training and execution space

Memory demands of learning algorithms have been relatively neglected in past studies. Statlog results indicate the maximum storage (in number of pages) used during run time but no attempt at analysing these storage demands was attempted. The authors themselves warn against misconceptions that might be bred by the measurements reported on time and space demands of algorithms. As with time measurements, they surmise that it might be more useful to quantify storage used in terms of dataset size; if similar results could be stated for all algorithms, comparisons between them would be more meaningful and predictions for new datasets possible. However, it is also necessary to consider whether the algorithm requires that all data be in central memory or can process it in pieces. Currently almost all learning algorithms for classification assume memory-resident datasets; new ways of data partitioning will certainly be needed and proposed in the near future to scale up these algorithms to massive databases.

In the absence of standardized measures, we shall content ourselves with simple analyses which give a rough idea of some algorithms' memory demands. For instance, standard Naive Bayes requires little storage space during both the training and classification stages: the strict minimum is the memory needed to store the prior and conditional probabilities: $2pq$ for continuous attributes (as only their mean and variance per class is needed) and around $pv_{ave}q$ for discrete attributes (where p is the number of independent discrete attributes, v_{ave} is the average number of values per attribute, and q the number of classes) . Beyond this minimum, $O(np)$ storage space is needed if the dataset is loaded, which may not always be the case in a data mining context.

The basic k-NN algorithm uses $O(np)$ storage space for the training phase, and execution space is at least as big as training space. Other instance-based learners have tried to reduce this memory demand by using different techniques [1]. For all non lazy learners, execution space is usually much smaller than training space, since the resulting classifier is usually highly condensed summary of the data. The actual storage demand depends on the representation structures used, but it will certainly be much less than $O(np)$.

5 CHARACTERIZING ROBUSTNESS

Robustness-related characteristics concern the reliability of training results across variations in training conditions or application characteristics. Many of them reflect the resistance of an algorithm to data characteristics that are liable to affect performance adversely. In this section, we shall treat robustness-related characteristics as binary-valued variables to reflect current practice in empirical comparative studies. It is hoped that controlled experiments to be undertaken later will help uncover a finer-grained measure scale for each of these characteristics.

5.1 Scalability

A learning algorithm is scalable if it performs as well on large datasets as on small and medium-sized datasets. Scalability is essential in data mining applications, which typically involve large masses of data. It is thus important to be able to measure and compare the behavior of algorithms as the size of the dataset increases. Scalability can be measured along two dimensions corresponding to the two dimensions of a dataset: scalability in the number of instances N and scalability in the number of features p. For classification tasks, a third component to be considered is the number of classes q, since purely numerical algorithms require that the (qualitative) target variable be reecnoded into q dummy variables.

We are not aware of any comprehensive evaluation of the scalability of known learning algorithms. A number of observations have been reported in the context of several comparative studies. Multivariate decision trees and k-NN based methods are known to scale badly with the number of independent variables, while Naïve Bayes and univariate decision trees seem to have no problem with dataset size, whether in the number of instances or in the number of attributes. However, counterexamples to these findings have been observed. For instance, CART failed in many of the Statlog experiments involving large datasets [12]. Lim et al. [11] did a small scalability study by observing the CPU time of 10 algorithms as the number of instances of four datasets was increased from 1000 to 8000 (by perturbing bootstrap samples from the datasets). As a whole, the logarithm of CPU time grew roughly linearly with the logarithm of N, and the only noteworthy finding was that the line for C4.5 rules rose a bit more steeply than the others—a sign that it does not scale as well as the others.

There is certainly a need for a more systematic comparison of learning algorithms with respect to their scalability. For the independent variable, this should integrate the three components of dataset size mentioned above, either by using a composite *size* variable defined as npq or $n(p+q)$, or by varying one component at a time while keeping the other two constant. The same alternative is available for the dependent variable: one can study predictive accuracy, training and execution time, training and execution space, etc., separately, or one can examine algorithmic behavior by integrating these factors into a single multi-criteria metric as was done in [15]. Such a study is on our research agenda for the immediate future.

5.2 Resistance to missing values

Missing values have been neglected in the pattern recognition literature, presumably because they are rare in applications where data are collected automatically [18]. However, missing values are common in many applications such as medical diagnosis and survey processing. The important issue is the pattern of missingness; some values are missing randomly, while others may be missing on the basis of domain-specific knowledge (e.g., pointlessness of pregnancy tests for men) or constraints (e.g., lab test needed to fill a value is too expensive or difficult). In surveys and other question-answering contexts, missing values may indicate a refusal to reply and therefore carry specific information. In such cases, missing values can be replaced only with the help of domain knowledge or by encoding missingness as a value in itself. A classical example of the latter is the congressional voting dataset from the UCI repository [14]: unknown values cannot be replaced by any of the two other values (yes/no on an issue) but must be given a third interpretation, i.e., abstention due to absence or refusal to take a stand. Determining the pattern of missingness is part of the preprocessing stage; here we consider only the case of randomly missing values and assume that whatever values are left missing for a learning algorithm to handle have been judged random and noninformative.

Some learning algorithms like C4.5/C5 and CART are considered tolerant of missing values because they have built-in mechanisms for handling them during both the training and test phases. Naive Bayes is naturally robust to missing values since these are simply ignored in computing probabilities and hence have no impact on the final decision. Standard k-NN, linear discriminants and neural networks require complete records to do their work: all missing values must have been eliminated, either by deleting the records or features with missing values, or by imputing missing by simplistic methods such as averaging or regression, or by more sophisticated ones like the EM algorithm [4].

The above binary-valued characterization is based simply on whether a given algorithm has built-in mechanisms for handling missing values or not. It does not say much about the effectiveness of such mechanisms, which can only be assessed emprically. We have undertaken such an experimental study, the results of which will be described in a forthcoming paper.

5.3 Resistance to noise

Noise is defined as the presence of non systematic errors in the values of the predictive or the target (class) attributes. Irrelevant attributes, which can also be viewed as a form of noise, will be examined separately in Section 5.4. It is important to be able to assess a learning algorithm's resistance to noise because many real-world datasets are noisy. Noise can come from faulty measurement devices, counting errors, or subjectivity in assigning qualitative values.

When empirical findings are lacking, one can roughly assess an algorithm's resistance to noise by examining the way it generalizes from data. Standard k-NN is generally considered intolerant of noise; its similarity measures can be easily distorted by errors in attribute values, thus leading it to misclassify a new instance on the basis of the wrong nearest neighbors. However, many descendants of k-NN, such as IB3, have been designed to resist noise [1]. Separate-and-conquer rule can also be considered sensitive to noise. They try to build rules which cover all the positive instances and none of the negative instances of a given class C. Noisy attribute or class values can cause them to keep adding rule conditions in order to cover erroneously positive instances of C or to exclude erroneously negative instances of C. In other words, they tend to overfit the noise. Here again, individual algorithms of the model class have added their own specialized mechanisms. CN2 copes with noise by testing for significant differences between the distribution of positive and negative examples covered by a rule and the overall distribution of positive and negative examples. A rule is rejected if it fails the X^2 test with one degree of freedom at the desired significance level. Contrary to k-NN and set-covering rule learners, most decision trees are considered resistant to noise because their tree pruning strategies avoid overfitting the data in general and noisy data in particular. The picture is not that clearcut, though. Since decision trees usually test one attribute at a time, a noisy attribute that is tested early on during learning will quickly throw the classifier off the track, leading to higher error rates [6].

While analysing a learning algorithm allows us to predict whether it will support noisy data, it provides no basis for quantifying its degree of noise tolerance or comparing it with that of other algorithms. Extensive comparative experiments seem to be the only feasible approach to this issue, and there have been very few to date. Some significant studies on noise resistance have been limited to a single learning algorithm (e.g., [16] for ID3) or a small group of closely related algorithms (e.g., [1] for IB1, IB2 and IB3). Cross-paradigm comparisons are relatively rare; Overall, there remains a need for systematic experimentation to quantify and compare noise tolerance of learning algorithms on a common basis.

5.4 Resistance to irrelevant attributes

There is ample experimental evidence that standard k-NN is highly sensitive to irrelevant attributes [1]. It has been shown both by formal analyses and experiments on artificial datatasets that the number of training examples needed by k-NN to achieve 90% accuracy (henceforth called the 90% sample complexity) grows exponentially with the number of irrelevant features, whether for conjunctive concepts or for more sophisticated concept forms such as disjunctive or m-of-n concepts [10].

Naive Bayes appears to be more robust than k-NN on this count: its 90% sample complexity grows only linearly with the number of irrelevant features. Related experiments show that for most of the concept forms studied, C4.5's 90% sample complexity also grows linearly with the number of irrelevant variables. This seems reasonable considering that C4.5 constructs a decision tree by successively testing for attributes that best describe the target concept. However, the picture is not so clearcut: for parity concepts, C4.5 exhibits the same exponential growth as k-NN. Similarly, other experiments have shown that for a given sample size, C4.5's accuracy decreases substantially with the number of irrelevant attributes. Blum and Langley [2] see a possible explanation for this in the greedy attribute selection strategy adopted in C4.5. This approach works fine when there is little interaction between attributes; but when attributes interact (as in parity concepts), examining a relevant attribute in isolation will not distinguish it from an irrelevant one.

We are not aware of similar experiments on neural networks. As a first cut at the problem, one can conjecture that asymptotic accuracy may not be significantly affected insofar as connection weights from irrelevant inputs will gradually tend towards zero as the training process converges. However, training time will be significantly increased, since a lot of time will be wasted on irrelevant portions of the input space; moreover, the greater the dimensionality of the input space, the more training instances are needed to ensure coverage. Thus, in real-world applications, the presence of irrelevant variables can make neural network training very inefficient et even impractible. Extensive experiments will have to be done to quantify the exact impact of irrelevant attributes on the behavior of different neural network models.

5.5 Resistance to redundant attributes

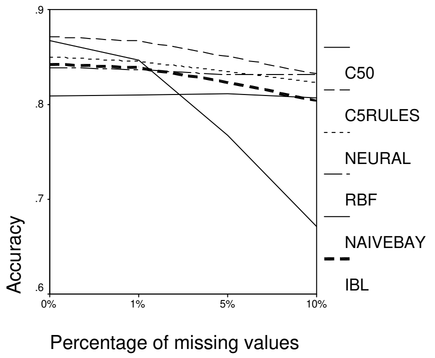
Attributes are redundant when they are interdependent, i.e., when it is possible to predict the value of one on the basis of that of another or several others. When the dependency can be expressed linearly, then the attributes are said to be correlated; correlation measures detect only one form of interdependence among problem features. Heuristic means are usually adopted to detect redundancy among attributes; for instance, attributes are removed one at a time, and performance measures taken before and after removal. Attributes whose removal does not significantly affect the learner's performance are considered redundant. Wettschereck et al. [22] show that k-NN is just as sensitive to redundant features as it is to noise and irrelevant features. This basic shortcoming of k-NN has given rise to a large number of feature weighting schemes which distinguish the many lazy learners that are now available. However, there remains a need for extensive tests in order to determine how effectively feature weighting can immunize lazy learners to feature redundancy.

According to the theory, Naive Bayes is inherently intolerant of interdependent or redundant attributes, since it is based on the assumption that the attributes are conditionally independent given the class. However, Naives Bayes has been shown to perform better than expected when this assumption is violated. In experiments involving 25 UCI datasets and four other learning algorithms (C4.5, CN2, PEBLS and Gaussian Bayes), Domingos and Pazzani [5] observe that Naive Bayes outperforms several of the others even in domains where there is substantial attribute dependence; more importantly, they show that Naive Bayes does not need attribute independence to be optimal under zero-one loss. In short, contrary to popular belief, Naive Bayes is resistant to interdependent attributes.

6 PRELIMINARY EXPERIMENTAL RESULTS

In the preceding sections we proposed a standard vocabulary for characterizing learning algorithms for classification as well as their underlying models. We distinguished three groups of characteristics depending on whether they concern a learning algorithm's representation and functionality, efficiency, or robustness. We defined each characteristic and its potential utility in prior model selection, then evaluated or ranked a number of learning algorithms with respect to this characteristic. For certain characteristics related to functionality (attribute types, cost handling), we relied on algorithm specifications as given by their authors or by expert users. Characteristics related to efficiency (learning and classification time and space) and robustness (scalability, resistance to missing values, noise, irrelevant and redundant attributes) can only be extrapolated from multiple executions of these algorithms over a wide variety of datasets. We tried to quantify these characteristics of prominent learning algorithms on the basis of previous analyses or empirical studies. Of necessity, such characterizations are tentative approximations. We have undertaken a systematic experimental study aimed at refining these qualitative assessments of certain algorithm features.

We focused on tolerance of missing values, which has received little attention in the machine learning literature. We ran five learning algorithms (C5-trees, C5-rules, Naive Bayes, instance-based learning, multilayer perceptrons, and radial basis function networks) on 47 datasets with no missing values from the UCI repository. We then randomly deleted 1%, 5%, and 10% of attribute values, and studied how accuracy and training time evolved with the degree of missing values. We used 10-fold cross-validation for datasets with less than 3000 instances and 2/3-1/3 holdout tests for larger datasets. Each experiment was repeated five times and performance measures averaged over them. Our preliminary findings disconfirm certain well-known expert intuitions on the resistance of learning algorithms to missing values (see 5.2 above). For instance, due to its built-in mechanism for coping with missing values [17], C4.5 has been widely assumed to be resistant to missing values. In fact, certain researchers have used C4.5 as a tool for missing value imputation [9]. Contrary to this conventional wisdom, C5 (an improved version of C4.5) turned out to degrade more rapidly than the other learning methods as the number of missing values increases. Neural networks typically are not tolerant of missing values, but we used the Clementine implementations of MLPs and RBF networks which adopt a fairly simple way of handling them.



The above chart shows that contrary to preconceived ideas about C4.5's tolerance of missing values, its complete-data accuracy plunges by 20% with 10% missing values whereas that of the four other methods, including C5 trees, degrades by at most 5%. Such striking counter-examples to common qualitative estimations of algorithm characteristics prove that there is a need for controlled experiments in view of building more reliable algorithm profiles.

References

[1] D. W. Aha, D. Kibler, M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37-66, 1991.

[2] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245-272, 1997.

[3] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45-77, 1995.

[4] A. P. Dempster and N. M. Laird and D. B. Rubin Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, series B, Vol. 39, pp. 1-38, 1977.

[5] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2/3):103-130, 1997.

[6] D. H. Fisher and K. B. McKusick. An empirical comparison of ID3 and backpropagation. *IJCAI-89*, pages 788-793.

[7] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1-58, 1992.

[8] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In Proc. *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338-345. Morgan Kaufmann, 1995.

[9] K. Lakshminarayan, S. Harp, R. Goldman & T. Samad. Imputation of missing data using machine learing techniques. *KDD-96*, pages 140-145, 1996.

[10] P. Langley and S. Sage. Scaling to domains with irrelevant features. In *Computational Learning Theory and Natural Learning Systems*. MIT Press, 1997.

[11] T. Lim, W. Loh, and Y. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. To appear in *Machine Learning*.

[12] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification*. Prentice-Hall, 1994.

[13] R. Mooney, J. Shavlik, G. Towell, and W. Gove. An experimental comparison of symbolic and connectionist learning algorithms. *IJCAI-89*, pages 775-780.

[14] P. M. Murphy and D.W. Aha. UCI machine learning repository. Irvine, CA: University of California.

[15] G. Nakhaeizadeh and A. Schnabl. Development of multi-criteria metrics for evaluation of data mining algorithms. In *Proc. Third International Conference on Knowledge Discovery and Data Mining*, pages 37-42, Newport Beach, CA, 1997. AAAI Press.

[16] J. R. Quinlan. *The effect of noise in concept learning*, chapter 6, pages 149-166. Morgan Kaufmann, 1986.

[17] *C4.5 : Programs for Machine Learning*. M. Kaufmann, 1993.

[18] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge U. Press, 1996.

[19] C. Schaffer. A conservation law for generalization performance. In W. W. Cohen and H. Hirsh, editors, *Proc. of the 11th International Conference on Machine Learning*, pages 259-265, Rutgers, NJ, 1994. Morgan Kaufmann.

[20] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: an experimental comparison. *Machine Learning*, 6:111-143, 1991.

[21] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161-186, 1989.

[22] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *IJCAI-89*, pages 781-786.

[23] D. Wettschereck, D. W. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11:273-314, 1997.