$\mathcal{MLC}$++: A Machine Learning Library in C++

Ron Kohavi George John Richard Long
David Manley Karl Pfleger

Computer Science Department
Stanford University
Stanford, CA 94305
mlc@CS.Stanford.EDU

August 23, 1994

Abstract

We present $\mathcal{MLC}$++, a library of C++ classes and tools for supervised Machine Learning. While $\mathcal{MLC}$++ provides general learning algorithms that can be used by end users, the main objective is to provide researchers and experts with a wide variety of tools that can accelerate algorithm development, increase software reliability, provide comparison tools, and display information visually. More than just a collection of existing algorithms, $\mathcal{MLC}$++ is an attempt to extract commonalities of algorithms and decompose them for a unified view that is simple, coherent, and extensible. In this paper we discuss the problems $\mathcal{MLC}$++ aims to solve, the design of $\mathcal{MLC}$++, and the current functionality.

1 Introduction

In supervised machine learning [SL92, Ang92, Val84, AS83], one tries to find a rule (a categorizer) that can be used to accurately predict the class label of novel instances. During the last decade, the machine learning community has developed a plethora of algorithms for this task. Regrettably, there is no clear understanding of the circumstances under which specific algorithms work and how they compare to each other. Thrun *et al.* open the 112-page report [TBB+91] on the Monk's problems with the following:

> After listening more than one week to a wide variety of learning algorithms, they [the students at the summer school] felt rather confused: Which algorithm would be optimal? And which one to avoid? As a consequence of this dilemma, they created a simple task on which all learning algorithms ought to be compared: the *three Monk's problems.*

While it is clear that no learning algorithm is optimal for all tasks, one of the main goals of machine learning is to identify situations in which given algorithms should perform well. Since a unified theory is currently out of reach, to a large extent algorithms can only be evaluated experimentally. Expected accuracy can be estimated using an unseen test set or cross-validation.

The $\mathcal{MLC}$++ project at Stanford aims to help users who need to use machine learning algorithms and researchers who wish to experiment with new algorithms or make modifications to existing algorithms. The goal of $\mathcal{MLC}$++ is to provide a library of C++ classes and functions implementing the most common algorithms, and to enable effortless programming of variants through object-oriented mechanisms such as inheritance and virtual functions.

$\mathcal{MLC}$++ not only provides different implementations of algorithms, but also integrates them in a C++ library with one unified coding methodology and decomposition of components into C++ classes. Not only will coding a new algorithm be substantially easier, but the array of tools provided by $\mathcal{MLC}$++ will give the researcher a broad picture of the performance of the algorithm when compared to other algorithms on different domains.

Our main goal is to provide capability to do the following tasks easily and quickly:

1. Implement and test new ideas and variants of supervised learning algorithms.
2. Generate performance statistics such as accuracy, learning rates, and confusion matrices.
3. Compare algorithms on different datasets (*e.g.*, compare algorithms A, B, C on datasets X, Y, Z).
4. Experiment with hybrid algorithms (*e.g.*, perceptron trees [Utg88], multivariate trees [BU92] and entropy nets [KP88]) and hierarchical structures.

5. Graphically display the learned structure (*e.g.*, display the decision tree).
6. Graphically display the learned concept and deviations from the target concept (when it is known). A useful tool for this purpose is Michalski's General Logic Diagrams [WM94, WSWM90, Mic78].

In the next section, we identify the main problems with the current experimental methodology in the machine learning community. In Section 3, we cover related work. Section 4 is an overview of the $\mathcal{MLC}$++ objectives, design issues, and implementation. In Section 5 we show several examples of the output of the library. In Section 6, we discuss plans for extensions and future work, and conclude with a summary.

2 Problems with the Current Methodology

Despite progress on the theoretical front, mostly related to Valiant's PAC model (cf. [Ang92] for a survey), most supervised learning algorithms are too complex for formal analysis, and empirical studies retain a central role.

Although many experimental results have appeared (and continue to appear), the field seems to be in a state of disarray. There are too many algorithms and variations of algorithms, each claiming to do better on a few datasets. We believe the field has reached a stage of maturity that calls for more systematic experimentation and analysis. Some of the main problems that exist are described below (see [LK91, KL88] for further discussions):

Comparisons with other algorithms Results are usually given in isolation. When a learning algorithm is presented, comparison is usually limited to a trivial straw algorithm, as opposed to state-of-the-art algorithms. Since authors do not use a standard data file format, even those programs which are publicly available often cannot be easily compared on the same datasets. For example, the Irvine repository [MA94] does not provide a mechanism for parsing its data files.

Comparisons on different datasets Results are given on different datasets, and cannot be compared. The Irvine repository [MA94] partially solves this problem, as it supplies a set of "standard" datasets, but most files do not have test sets, and researchers differ on their evaluation of algorithms. Holte [Hol93, Appendix C] collected accuracy results from many papers and experimented on his own, but as he writes, these results should be taken with a grain of salt:

> The results included in this survey were produced under a very wide variety of experimental conditions, and therefore it is impossible to compare them in any detailed manner. ...the number of runs varies considerably, as does the ratio of the sizes of the training and test set. ...it is virtually certain

that some papers reporting results on a dataset have used slightly different versions of the dataset than others, it being common practice to make "small" changes to a dataset for the purposes of a particular experiment.

Programs are written in isolation Authors of machine learning algorithms seldom make their code available, and thus other researchers working in the same area must start from scratch in order to implement the previous algorithm and modify it. Algorithms and input/output routines are coded over and over, without using existing code. To quote James Coggins [Cog90], who designed the C++ COOL library:

> Newton said he saw farther because he stood on the shoulders of giants. Computer programmers stand on each other's toes.

Experiments are hard to replicate Replication plays an important role in science, since it ensures that results are robust and reliable before widely accepted. Currently there are very few results in machine learning that have been thoroughly examined and verified.

Lesion studies and hybrid algorithms Machine learning systems contain many independent components, and most authors report results on one modification. It is not clear how such variations interact; while some combinations of variations might be synergistic, other combinations may nullify the individual gains. Conversely, hybrid approaches, where more than one machine learning paradigm is used, seem to be a promising avenue of research in machine learning. Such hybrid systems are time consuming to program from scratch.

Display of information In order to understand the problem better, and perhaps bias a learning algorithm, it is helpful to view the resulting structures (*e.g.*, decision tree), or the data. Even commercially available programs such as C4.5 [Qui92] and CART [BFOS84] give only a rudimentary display of an induced tree.

A library providing a common basis would alleviate the pain involved in programming from scratch. Since researchers will need to write less code, they can write higher quality code and do it in shorter time. Such code currently exists in the $\mathcal{MLC}$++ library. We hope that with the existence of this common basis, researchers will publish more of their code.

3 Related Work

Although $\mathcal{MLC}$++ is a unique project, there have been previous efforts to create large resources for the machine learning community. Below, we mention previous projects addressing these concerns, and attempt to identify how $\mathcal{MLC}$++ either differs or perhaps solves the same problem in a more general way.

3.1 Data Collections

An extensive collection of over 80 datasets has been collected by Murphy and Aha at the University of California at Irvine [MA94]. We do not intend to duplicate any of this effort; in fact, we use their data formats as much as possible. There is also a large repository of data used by statisticians in the StatLib archive, created and maintained by Meyer [Mey]. Not all datasets are for supervised learning—some are intended for clustering or exploratory data analysis.

3.2 Learning Algorithm Collections

Mooney [Moo91] has re-implemented many learning algorithms in lisp for the purpose of teaching students about machine learning. This is a nice collection of easy-to-read implementations of common machine learning algorithms. $\mathcal{MLC}$++ would like to retain the simplicity and understandability in Mooney's implementations while providing a framework allowing efficient runtime behavior and the extensibility inherent in object-oriented design. Moreover, Mooney's work points out an important benefit of $\mathcal{MLC}$++ as a tool for teaching machine learning courses. Students using $\mathcal{MLC}$++ would be able to spend their time doing interesting experimental studies or modifying existing algorithms.

The *Consultant* system [Sle], headed by Derek Sleeman, was created in Europe to help domain experts choose a machine learning algorithm suitable for their needs. Consultant is also a library of learning algorithms, and the selected algorithm is then run on the user's data. This addresses an important need which we intend to address in $\mathcal{MLC}$++ as well: choosing an algorithm to achieve the highest accuracy on unseen test sets from the plethora of algorithms presented in the literature. Consultant is basically an expert system, programmed with rules which allow it to interview a user and recommend a machine learning algorithm which might be appropriate. As a result there are several important differences between Consultant and $\mathcal{MLC}$++. First, Consultant is a tool for choosing an algorithm, and not a tool for creating variants of algorithms or testing new ones. Second, $\mathcal{MLC}$++ will be able to suggest an appropriate algorithm to a user, but by employing more general and direct methods such as cross-validation [BFOS84, WK91] to estimate the accuracy of the algorithms using only the user-provided training data. While more computationally intensive, one would expect such direct data-driven methods to yield better performance than the high-level rules in Consultant.

3.3 Large-Scale Comparisons: StatLog and MLToolbox

StatLog [TMS94] is an ESPRIT project studying the behavior of over twenty algorithms (mostly in the MLToolbox), on over twenty datasets. The StatLog book gives a nice introduction to a wide variety of statistical and machine learning algorithms and an interesting discussion of the results. $\mathcal{MLC}$++ can be used to generate such experiments as well. MLToolbox is a collection of many publicly available algorithms and re-implementations of others. Only a subset of the programs are available, and some need to be bought (CN2 and NewID, for example, cost £150 for academic users and £450 for non-academic users). Not all algorithms share code and a common interface. Thus, the comparisons in the study were not as controlled as they are in $\mathcal{MLC}$++. Also, the goal of the project was just to provide the experimental data mentioned, not to provide a toolbox for comparing and implementing machine learning algorithms.

4 Design of the $\mathcal{MLC}$++ library

In this section we describe the main decomposition of $\mathcal{MLC}$++ into different types of functionality. We then give an overview of several of the important classes in $\mathcal{MLC}$++, and conclude with our reasons for choosing C++ as the language for implementation.

4.1 Decomposition of $\mathcal{MLC}$++

$\mathcal{MLC}$++ is divided into four main types of functions:

General Support Classes These classes provide support for general operations, not specific to machine learning. These include classes for arrays, linked lists, strings, random numbers, and graphs. We attempt to use as much educational and public domain software as possible for this part of $\mathcal{MLC}$++. For example, the graph manipulations are done using LEDA (Library of Efficient Data Structures) written by Stefan Näher [Nae92], and dot from AT&T [GKNV93, KN].

Core classes These are the basic tools that are shared by many algorithms in supervised machine learning. They further divide into three types of functionality:

- **Input/Output** Classes for reading and writing data files.
- **Generation and Conversions** Classes for generating artificial datasets, converting attribute types between formats (*e.g.*, local encoding[1], binary encoding), and normalizing values.
- **Wrappers** Algorithms that "wrap around" an induction algorithm to generate accuracies from test sets, estimate accuracy using cross-validation, and generate learning curves.

Since the core classes are the most useful to researchers implementing algorithms, most of the work on $\mathcal{MLC}$++ is concentrated on these classes.

[1] For example, the local encoding of a four-valued nominal $\{a, b, c, d\}$ would be $\{0001, 0010, 0100, 1000\}$.

Categorizers Categorizers are functions mapping instances to categories, or classes. These are the basic structures that induction algorithms induce. $\mathcal{MLC}$++ provides the most common categorizers such as a constant categorizer (which returns the same category regardless of the instance), attribute categorizer (which uses only a single attribute to predict the category), threshold categorizer, nearest-neighbor categorizer, decision tree and decision graph categorizers, perceptron categorizer, and neural net categorizer. Categorizers are built recursively; for example, in a decision tree categorizer, the branching nodes are categorizers themselves (mapping the set of instances into the set of children of that node), and the induction algorithm can use any categorizer, including the possibility of recursive decision trees. ID3 [Qui86] always uses attribute categorizers for nominal attributes and threshold categorizers for real attributes. To generate multivariate trees [BU92, HKS93] with perceptrons at nodes, the induction algorithm can put perceptron categorizers at the nodes. In the future we will provide *mappers*, a generalization of categorizers where the output is a real number instead of a discrete category label.

Induction algorithms Induction algorithms induce categorizers. $\mathcal{MLC}$++ currently provides a majority inducer, a nearest-neighbor inducer [Aha92, AKA91], an ID3-like decision tree inducer [Qui86], HOODG for inducing oblivious read-once decision graphs [Koh94b, Koh94a].

Figure 1 illustrates the interactions between inducers, categorizers, testers, and wrappers. The decoupling of the inducer and the categorizer is critically important in supervised learning and plays an important role in the overall decomposition of the $\mathcal{MLC}$++ library.

4.2 Overview of $\mathcal{MLC}$++ classes

This section provides a more detailed description of the $\mathcal{MLC}$++ classes.

Sets of Instances Sets (actually multi-sets, or bags) of instances are used for reading, storing, and manipulating instances (such as from a training set or a test set). $\mathcal{MLC}$++ recognizes files in Quinlan's format [Qui92], which shares the same data file format with the Irvine repository [MA94]. Sets can be converted to other formats as well as other encodings (*e.g.*, binary, local). This class provides functions for inserting and deleting instances as well as iterating through all instances in the set. Sets may be split based on label value or based on attribute values using a categorizer. Other functions provide independent samples of instances, reordering the instances, and removing instances with unknown or conflicting labels.

Categorizers Categorizers have already been introduced in Section 4.1. A generalization of decision trees, rooted decision graph categorizers use a structure with categorizers at interior nodes and constant categorizers at leaf nodes to predict the category for a

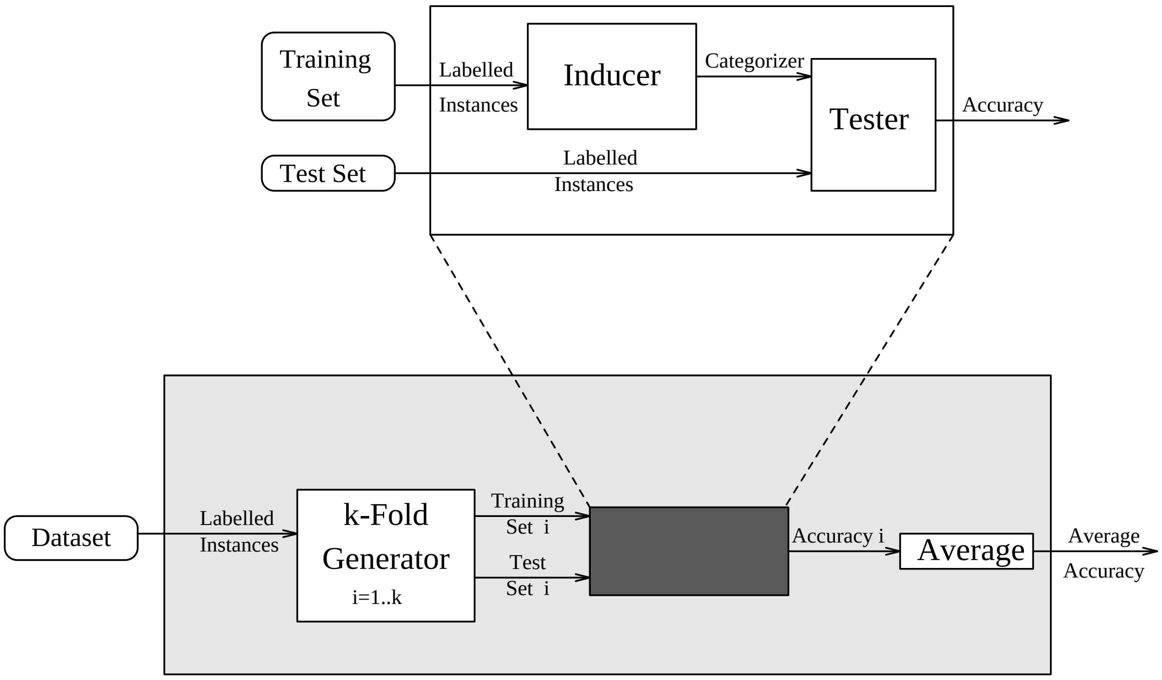

Figure 1: The upper portion illustrates the induction and subsequent testing of a categorizer. The lower portion denotes a cross-validation wrapper estimating the accuracy of an inducer on a dataset.

given instance. Neural network categorizers use networks of neuron mappers. Nearest neighbor categorizers store a set of data points and use a distance metric to predict the category of new instances.

All categorizers provide a display function. Both rooted decision graph and decision tree categorizers provide graphical displays that can also be sent to an X window or to a Postscript file (see Figures 2 and 3). A simple ASCII display format is also available.

Inducers Inducers induce categorizers based on a training set. A majority inducer creates a constant categorizer which returns the majority category of the training set.

The top-down decision tree inducer is a class for inducing a decision tree top-down by building smaller training sets and inducing decision trees for them recursively. Different variations of top-down decision tree induction can be implemented by subclassing and providing a function for determining how to branch (*i.e.*, which attribute to split on) at each node. ID3 inducer is a subclass of the top-down decision-tree inducer that uses the ID3 algorithm to induce the decision tree [Qui86].

The OODG (Oblivious, read-Once Decision Graph) inducer uses a bottom-up method for building levelled rooted decision graphs [Koh94a, Koh94b]. The HOODG inducer is a subclass of the OODG inducer that provides a hill-climbing implementation of the

OODG algorithm.

A nearest neighbor inducer uses the IB algorithms described in [Aha92].

Test result Test result is a class that provides information about the performance of a categorizer on a test set. It provides the number of instances correctly and incorrectly categorized, the accuracy, and the confusion matrix.

Cross validator Cross validator is a class that is used to estimate the accuracy of an inducer on unseen test sets. Cross validator performs k-fold cross-validation on an inducer and a given training set. It divides the training set into k mutually exclusive test partitions of approximately equal size, and then repeatedly trains the inducer on $k-1$ partitions and tests on the remaining partition. Averaging over all instances yields the estimated accuracy. Cross validator uses the test result class to return the average accuracy and the variance of these accuracy measurements. The class provides the option of dumping all of the training sets generated during cross-validation to files for later re-use.

Learning curve Learning curve is a class which takes an inducer and training set, and generates a learning curve to show how the amount of training data affects the accuracy of the induced categorizer. The inducer is trained using a certain number of instances chosen at random from the given set, and trained several times at that level. The average accuracy predicting the entire training set at that level is recorded using the test result class. The data can then be plotted using any plot package, such as gnuplot or Mathematica.

GLD The GLD class creates General Logic Diagrams, which are graphical projections of multi-dimensional discrete spaces onto two dimensions. They are similar to Karnaugh maps, but are generalized to non Boolean inputs and outputs [WM94, WSWM90, Mic78]. A GLD may be generated for a test set, the induced categorizer's predictions for the entire instance space, and an overlay, showing the errors. GLDs can be displayed in X windows, Postscript, and xfig, the X windows drawing editor.

Attributes Attributes may have different types, such as nominal, real, linear, and linear nominal.

Discrete data generators The data generator class allows the generation of labelled instances (possibly the full space) for a given instance space where the label is determined by a user-provided function.

General support classes Support is provided in addition to the above mentioned machine learning specific classes. Template classes for one and two-dimensional arrays and doubly-linked lists are implemented. Other useful classes include a string class and a random number generator.

4.3 The Choice of the C++ Language

C++ [Str91] was chosen as the best language for implementing the library because of its object-oriented capability, good type checking, and efficiency. Inheritance, virtual functions, and overloading make overriding defaults and modifying specific parts of an algorithm effortless.

The library has high coding standards. Code is well documented, and each class has a specific tester that tests it. Development utilizes the ObjectCenter and TestCenter products which ensure that there are no illegal memory accesses and no memory leaks. Profiling is done regularly to improve efficiency.

$\mathcal{MLC}$++, ignoring public domain code used, consists of over 20,000 lines of code, and over 5,000 lines of test code to verify correctness.

5 Examples from the Library

The most important aspects of $\mathcal{MLC}$++ are the ease with which one can modify previously-written algorithms, and the graphical utilities which allow the user to inspect datasets and induced concepts. Aside from the functionality, perhaps the most important aspect of a library is the ease of use and readability of code written using the library. Many relatively complicated machine learning experiments can be performed with just a few lines of $\mathcal{MLC}$++. It is difficult to convey the ease of use and readability without resorting to including actual code fragments.

5.1 Exploiting Modularity

One of the main goals in the design of $\mathcal{MLC}$++ was to benefit from the object-oriented features of C++ so that code could be shared as much as possible. This has the effect of making many of the algorithms in the library easy to implement. By reducing algorithms to their essential components, this modularity leads to a better conceptual understanding of the relationships between different algorithms and produces interesting ideas about combinations of modules that had not been considered before.

Table 1 gives a fragment of the header file for the `ID3Inducer`. *Very* few functions are actually defined by the `ID3Inducer` since it inherits much code from its parent class `TDDTInducer` (Top-down decision tree inducer) and its grandparent class `Inducer`. In fact, given these other classes the only methods needed to specify the ID3 algorithm are the `best_split` splitting criterion and the `create_subinducer` function, which specifies that the ID3 algorithm itself is to be called on the subsets created by the split.

```
#include <TDDTInducer.h>

class ID3Inducer : public TDDTInducer {
public:
 ID3Inducer(const MString& dscr, CGraph* aCgraph = NULL);
 virtual ~ID3Inducer();
 virtual Categorizer* best_split(MStringArray*& catNames) const;
 virtual TDDTInducer* create_subinducer(const MString& dscr,
                                        CGraph& aCgraph) const;
};
```

Table 1: The header file showing the functions needed to write the ID3 algorithm.

5.2 Visualizing Structures, Datasets, Concepts, and Performance

From the outset, one of our top priorities was to provide visualization tools to the user. Graphical displays of datasets and induced concepts can provide key insights. Without such visualization tools, the user must rely totally on simple performance statistics (such as accuracy on a test set) that provides little understanding.

An important tool in $\mathcal{MLC}$++ allows the user to view the actual structures induced by the learning algorithms. While most studies of supervised machine learning discuss accuracy on an unseen test set as the performance component, in many cases it is equally or more important to induce comprehensible structures which give the users new insight regarding their data. Decision trees and decision graphs are excellent examples of interpretable structures, and $\mathcal{MLC}$++ interfaces the excellent graph-drawing programs dot and dotty provided by AT&T [GKNV93, KN]. Examples of induced structures are shown in Figures 2 and 3.

For viewing datasets and induced concepts, we have implemented General Logic Diagrams. After running an induction algorithm, users may gain insight about the induced concept by inspecting the GLD. High dimensional concepts are hard to visualize; Michalski [Mic78] introduced General Logic Diagrams as a visualization technique to help understand concepts. As well as a tool for researchers to gain insights into their algorithms, GLDs are a tool for users to test different algorithms on their own data to see which learning algorithm actually induces sensible-looking concepts. We provide General Logic Diagrams for discrete concepts and allow users to view the induced concept, and, whenever the target concept is known, the symmetric errors. Figures 4 and 5 show GLDs for two common datasets.

Simple performance statistics, such as the accuracy of a learning algorithm when trained on a standard training set and tested on a standard test set, are the typical information given to a user by a learning algorithm. A much more interesting metric is how the performance

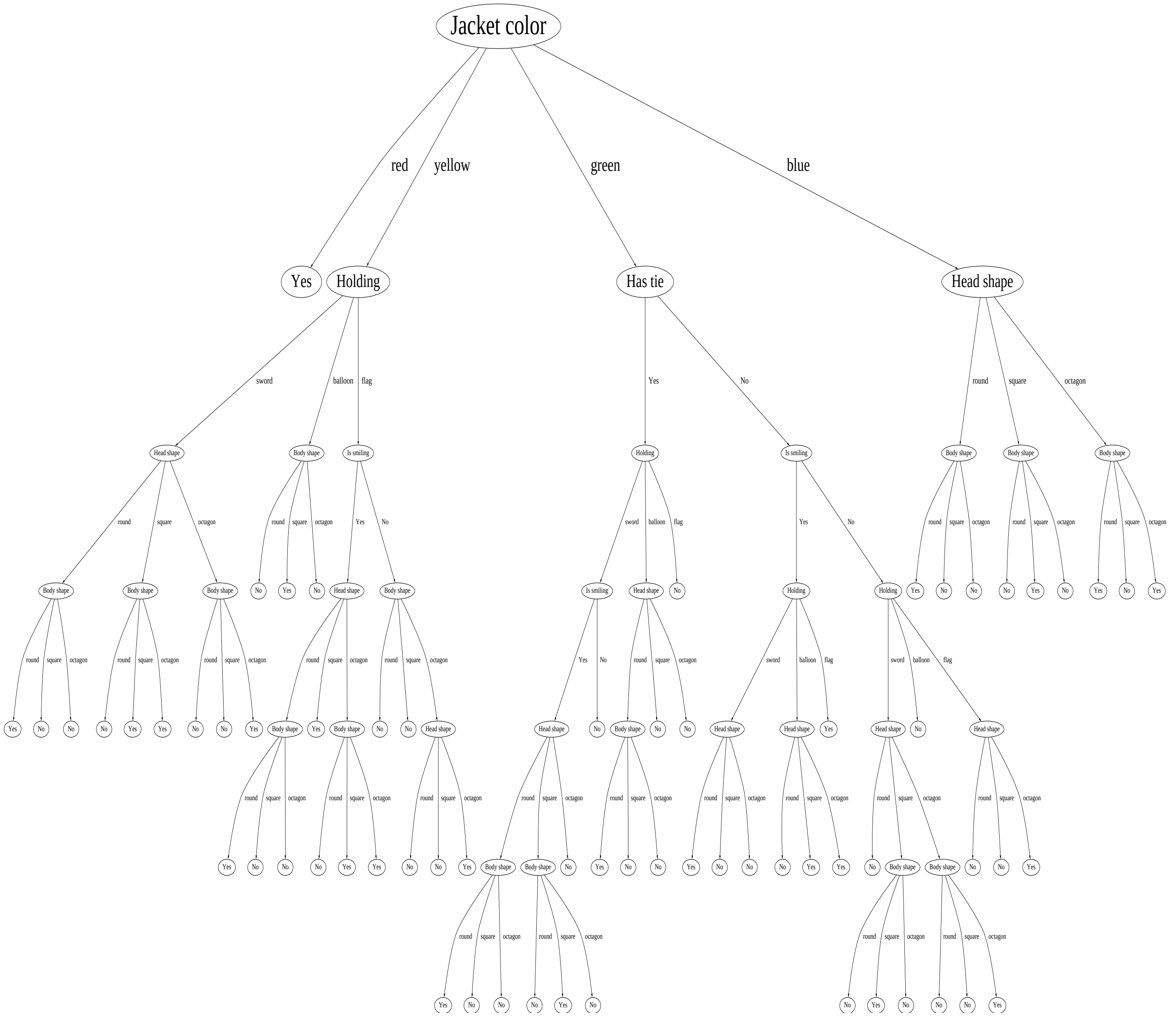

Figure 2: The decision tree induced by ID3 on the Monk1 dataset.

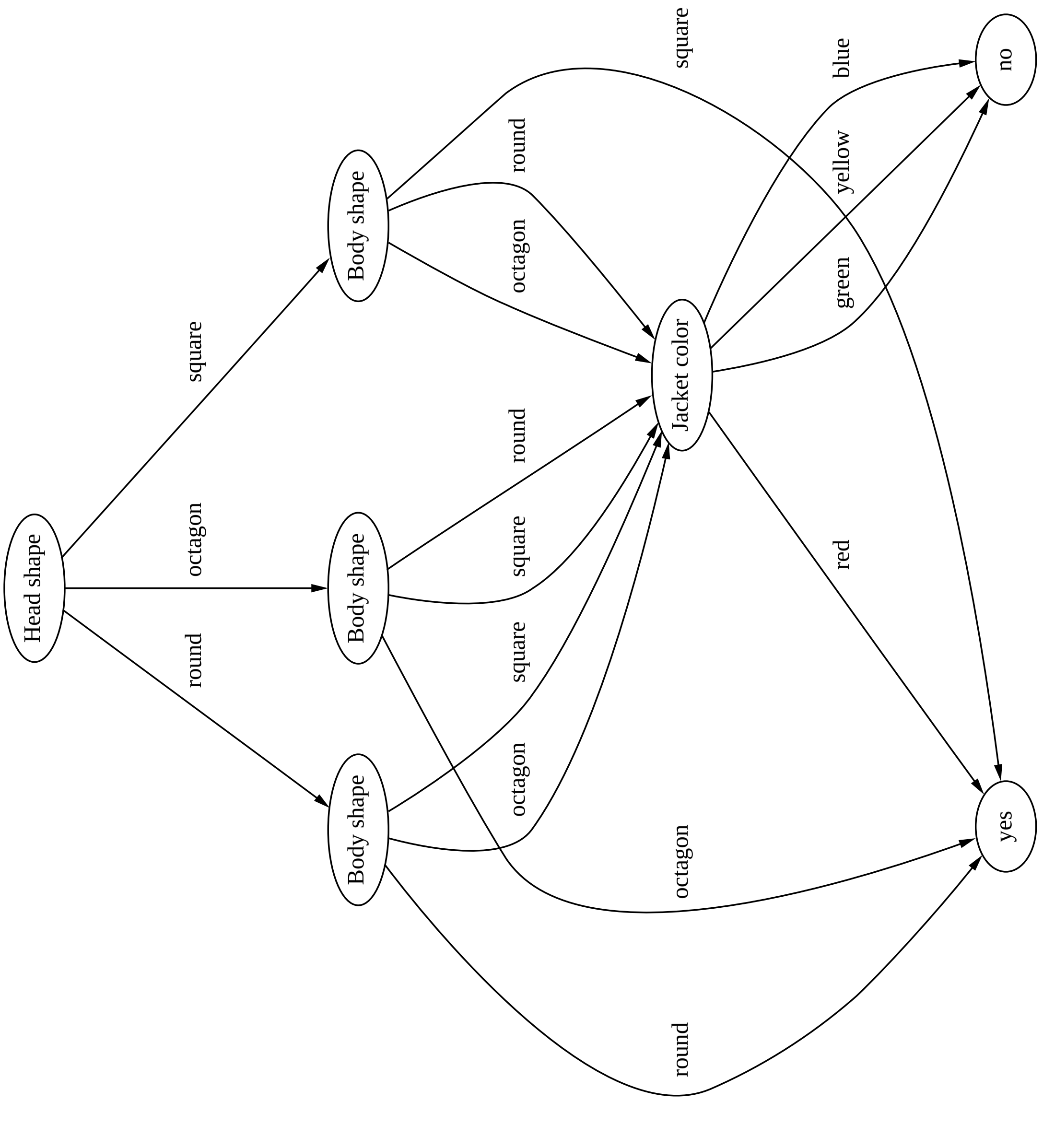

Figure 3: The decision graph induced by OODG on the Monk1 dataset.

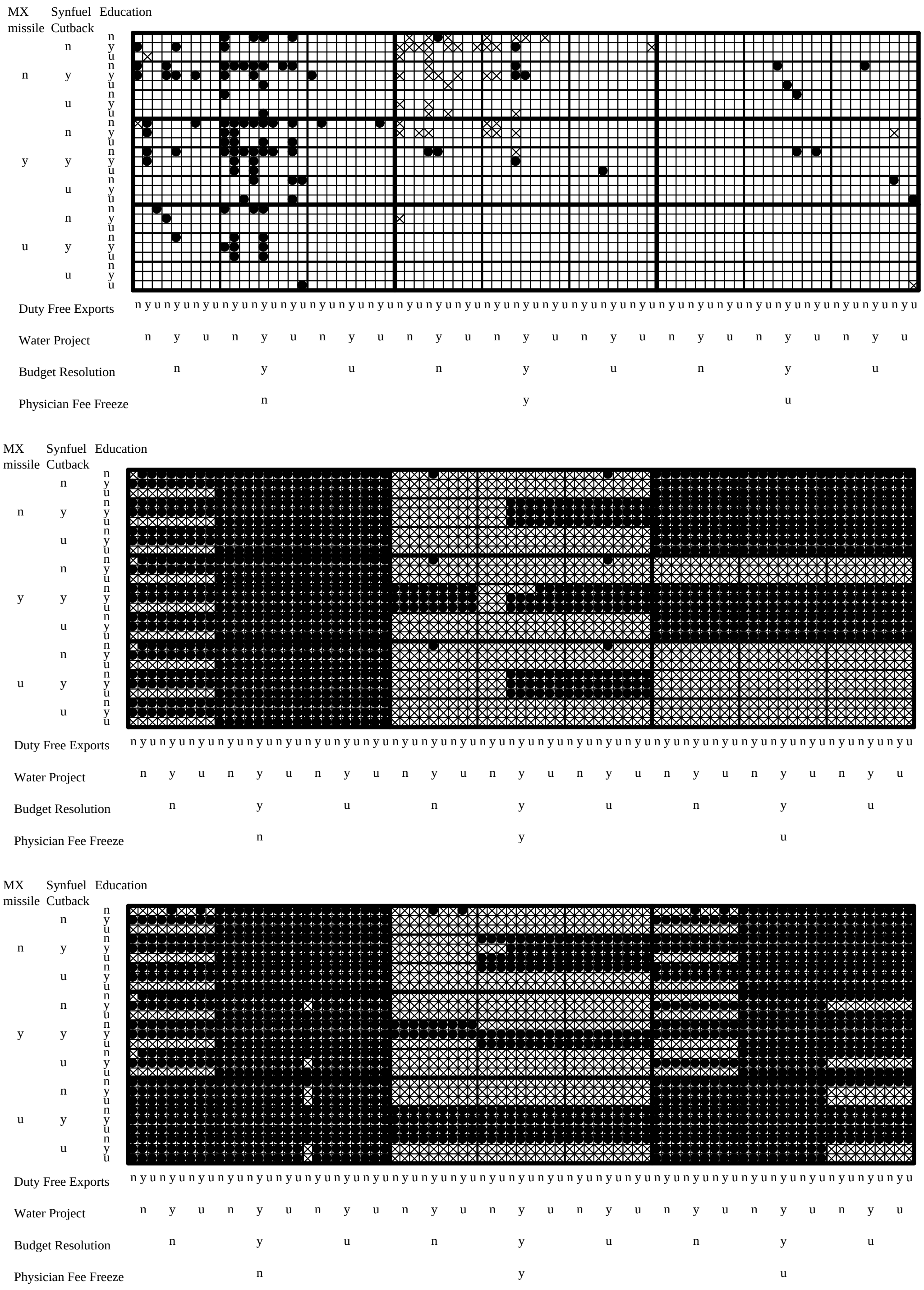

Figure 4: The vote dataset (top), the concept induced by ID3 (center), and the concept induced by HOODG (bottom).

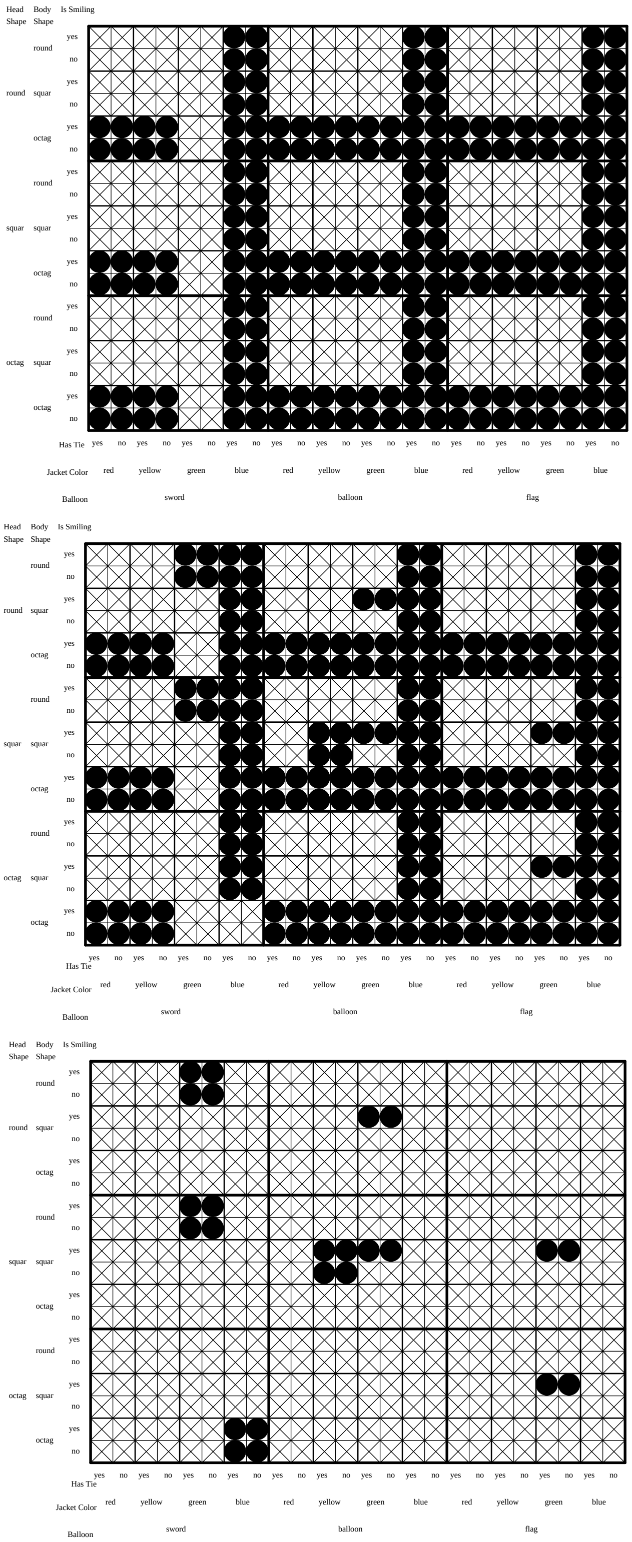

Figure 5: The Monk3 dataset (top), the concept predicted by ID3 (center), and the errors in black (bottom).

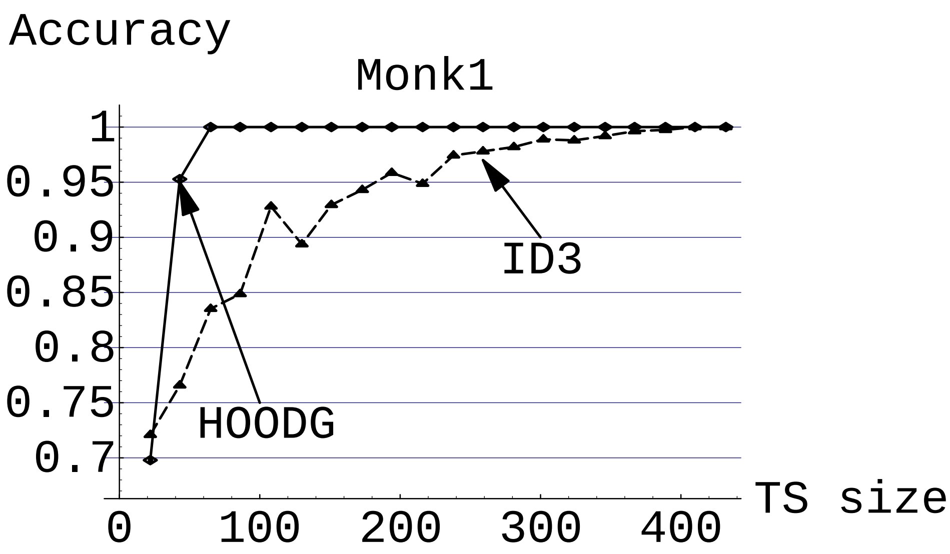

Figure 6: Learning curves generated using $\mathcal{MLC}$++.

of the learning algorithm scales with the number of inputs. Plots of accuracy versus number of instances in the training set are called learning curves, and are an important means of evaluating a learning algorithm. In $\mathcal{MLC}$++, a learning curve generator takes an inducer and a few other parameters, and generates data such as that plotted in Figure 6, which shows the accuracy for ID3 and HOODG.

6 Extensions and Future Work

$\mathcal{MLC}$++ can be extended by adding many different inducers and categorizers. For example, Bayesian classifiers, decision lists, AQ-11 type inducers, and many other examples exist and might be implemented. However, we feel that our effort should be concentrated on the core functions as much as possible. The additions to the core functions that we have in mind include:

Feature manipulation Functions for discovering irrelevant attributes and ranking the relative importance of attributes [KR92, JKP94], functions to help in constructing new features [PH90], and discretizing real values into ranges.

Search algorithms Turing in [Tur50] already recognized that learning can be viewed as a type of search when he wrote "The learning process may be regarded as a search for a form of behavior which will satisfy the teacher." While many researchers noted that learning can be viewed as search through the space of hypotheses [SL92, Bun90, Mit82], most algorithms couple the search and the bias of the algorithm.

ID3 [Qui86] and many variants are searching for a small tree, guided by the heuristic that splits lowering the entropy should yield small trees. One problem is that since there is no backtracking, decisions cannot be changed (although they can be undone by pruning). By *abstracting* away the search problem, we can use search algorithms such as beam search and IDA* [PK87] to systematically find better trees.

Wrappers Given a learning algorithm, we wish to provide more wrappers to evaluate the performance. Variants on cross-validation such as stratification can be useful. Similarly, other methods of estimating accuracy, such as Efron's Bootstrap [ET93, Efr82], are possible. Given different induction algorithms, we would like to add significance tests to test whether an improvement is non-negligible. Comparing on which instances the predictions differ is also important, and the GLDs provide one way of comparing predictions.

Graphical displays The General Logic Diagrams will be much more useful in color.

Data generator The data generator should be extended to allow generating non-discrete data, to incorporate different noise models, and varying distributions.

Hybrid approaches Hybrid algorithms of different kinds can be tested.

One of the more intriguing things we wish to do is meta-level learning [BGH94, BaGH93], *i.e.*, to have a machine learning algorithm induce rules that will define (empirically) when an algorithm is applicable, or which algorithm should be used for a given dataset, and with what parameterization. We believe that only through projects like $\mathcal{MLC}$++ and [TMS94] can we reach such a goal.

7 Summary

Frederick Brooks in his famous paper "No Silver Bullet" [Bro86] wrote

> The most radical possible solution for constructing software is not to construct it at all. ... The key issue, of course, is applicability. Can I use an available off-the-shelf package to perform my task?

$\mathcal{MLC}$++ is an attempt at providing such an off-the-shelf package to researchers and users of machine learning algorithms. We often asked ourselves this question while writing $\mathcal{MLC}$++, and where appropriate we have made use of publicly available software. We have described several problems researchers in machine learning currently face, and we believe that these problems can be solved with the right tool. We described $\mathcal{MLC}$++, our attempt at building such a tool. Much work has already been done on $\mathcal{MLC}$++, and it has already aided some of us in our research [Koh94a, Koh94b, JKP94, Koh94c]. We trust that other researchers will also enjoy productivity gains when using $\mathcal{MLC}$++, and will contribute to this effort.

Acknowledgments The $\mathcal{MLC}$++ project is partly funded by ONR grant N00014-94-1-0448 and NSF grant IRI-9116399. George John is supported by an NSF Graduate Research Fellowship. Nils Nilsson and Yoav Shoham have provided crucial support for the $\mathcal{MLC}$++ project. Wray Buntine, Pat Langley, Ofer Matan, and Scott Roy have contributed important ideas. Finally, we wish to thank everyone working on $\mathcal{MLC}$++, including James Dougherty, Brian Frasca, Svetlozar Nestorov, and Yeo-Girl Yun.

References

[Aha92] David W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36(1):267–287, 1992.

[AKA91] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[Ang92] Dana Angluin. Computational learning theory: Survey and selected bibliography. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 351–369. ACM Press, 1992.

[AS83] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

[BaGH93] Pavel Brazdil, Jo ao Gama, and Bob Henery. Comparisons of ML and statistical approaches using meta level learning. In *Proceedings of the ECML93 workshop on Real-World Applications of Machine Learning*. Springer, 1993.

[BFOS84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[BGH94] Pavel Brazdil, Joao Gama, and Bob Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Proceedings of the European Conference on Machine Learning*, April 1994.

[Bro86] Frederick P. Brooks. No silver bullets. In H. J. Kugler, editor, *Information Processing*. Elsevier Science Publishers, North Holland, 1986. Reprinted in Unix Review November 1987.

[BU92] Carla E. Brodley and Paul Utgoff. Multivariate decision trees. Technical Report MASSCS 92-93, University of Massachusetts, Amherst, 1992.

[Bun90] Wray Buntine. Myths and legends in learning classification rules. In *Eighth National Conference on Artificial Intelligence*, pages 736–742. MIT Press, 1990.

[Cog90] James Coggins. Designing C++ libraries. *The C++ Journal*, 1(1):25–32, 1990.

[Efr82] Bradley Efron. *The jackknife, the bootstrap, and other resampling plans.* Society for Industrial and Applied Mathematics, 1982.

[ET93] Brad Efron and Rob Tibshirani. *An introduction to the bootstrap.* Chapman & Hall, 1993.

[GKNV93] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. In *IEEE Transactions on Software Engineering*, pages 214–230, 1993.

[HKS93] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *13th International Joint Conference on Artificial Intelligence*, pages 1002–1007. Morgan Kaufmann, 1993.

[Hol93] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–90, 1993.

[JKP94] George John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann, 1994.

[KL88] Dennis Kibler and Pat Langley. Machine learning as an experimental science. In *Proceedings of the Third European Working Session on Learning*. Pitman Publishing Ltd., London and Morgan Kaufman Publishing, San Mateo, CA., 1988. Also appears in Readings in Machine Learning by Shavlik and Dietterich.

[KN] Eleftherios Koutsofios and Stephen C. North. Drawing graphs with dot. Available by anonymous ftp from
`research.att.com:dist/drawdag/dotdoc.ps.Z`.

[Koh94a] Ron Kohavi. Bottom-up induction of oblivious, read-once decision graphs. In *Proceedings of the European Conference on Machine Learning*, April 1994. Paper available by anonymous ftp from
`starry.Stanford.EDU:pub/ronnyk/euroML94.ps`.

[Koh94b] Ron Kohavi. Bottom-up induction of oblivious, read-once decision graphs : Strengths and limitations. In *Twelfth National Conference on Artificial Intelligence*, 1994. Paper available by anonymous ftp from `Starry.Stanford.EDU:pub/ronnyk/aaai94.ps`.

[Koh94c] Ron Kohavi. Feature subset selection as search with probabilistic estimates. In *AAAI Fall Symposium on Relevance*, 1994. To appear.

[KP88] C. Koutsougeras and C. A. Papachristou. Training of a neural network for pattern classification based on an entropy measure. In *IEEE International Conference on Neural Networks*, pages 247–254. IEEE, IEEE Press, 1988.

[KR92] Kenji Kira and Larry A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Tenth National Conference on Artificial Intelligence*, pages 129–134. MIT Press, 1992.

[LK91] Pat Langley and Dennis Kibler. The experimental study of machine learning. Unpublished paper, 1991.

[MA94] Patrick M. Murphy and David W. Aha. UCI repository of machine learning databases. For information contact ml-repository@ics.uci.edu, 1994.

[Mey] Mike Meyer. Statlib. Available at `lib.stat.cmu.edu`.

[Mic78] Ryszard S. Michalski. A planar geometric model for representing multidimensional discrete spaces and multiple-valued logic functions. Technical Report UIUCDCS-R-78-897, University of Illinois at Urbaba-Champaign, 1978.

[Mit82] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982. Reprinted in Shavlik and Dietterich (eds.) Readings in Machine Learning.

[Moo91] Raymond J. Mooney. Ml-code machine learning archive. Available by anonymous ftp in `cs.utexas.edu:/pub/mooney`, 1991.

[Nae92] Stefan Naeher. *LEDA: A Library of Efficient Data Types and Algorithms*. Max-Planck-Institut fuer Informatik, IM Stadtwald, D-66123 Saarbruecken, FRG, 3.0 edition, 1992. Available by anonymous ftp in `ftp.cs.uni-sb.de:LEDA`.

[PH90] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[PK87] Judea Pearl and Richard E. Korf. Search techniques. *Annual Review of Computer Science*, 2:451–467, 1987.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. Reprinted in Shavlik and Dietterich (eds.) Readings in Machine Learning.

[Qui92] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, Los Altos, California, 1992.

[SL92] Jeffrey C. Schlimmer and Pat Langley. Learning, Machine. In Stuart Shapiro and David Eckroth, editors, *The Encyclopedia of Artificial Intelligence*, pages 785–805. Wiley-Interscience, 2nd edition, 1992.

[Sle] Derek Sleeman. The role of CONSULTANT in helping domain experts use machine learning. Workshop on Fielded Applications of Machine Learning 1993, University of Massachusetts, Amherst.

[Str91] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley Publishing Company, second edition, 1991.

[TBB+91] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Weldel, W. Wenzel, J. Wnek, and J. Zhang. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[TMS94] C.C. Taylor, D. Michie, and D.J. Spiegalhalter. *Machine Learning, Neural and Statistical Classification.* Paramount Publishing International, 1994.

[Tur50] A.M. Turing. Computing machinery and intelligence. *Mind*, 59, 1950. Reprinted in 'Computers and Thought', Feigenbaum and Feldman [eds.], McGraw-Hill, New York, 1963.

[Utg88] Paul E. Utgoff. Perceptron trees: a case study in hybrid concept representation. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 601–606. Morgan Kaufmann, 1988.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[WK91] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems that Learn.* Morgan Kaufmann, San Mateo, CA, 1991.

[WM94] Janusz Wnek and Ryszard S. Michalski. Hypothesis-driven constructive induction in AQ17-HCI : A method and experiments. *Machine Learning*, 14(2):139–168, 1994.

[WSWM90] Janusz Wnek, Jayshree Sarma, Ashraf A. Wahab, and Ryszard S. Michalski. Comparing learning paradigms via diagrammatic visualization. In *Methodologies for Intelligent Systems, 5. Proceedings of the Fifth International Symposium*, pages 428–437, 1990. Also technical report MLI90-2, University of Illinois at Urbaba-Champaign.