Appears in Tools with AI '94

$\mathcal{MLC}$++: 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

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

> *Newton said he saw farther because he stood on the shoulders of giants. Computer programmers stand on each other's toes — James Coggins [5]*

In supervised machine learning [18, 2], 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. Many researchers have implemented similar algorithms in isolation, inevitably duplicating much work. Many algorithms currently exist, yet few are publicly available, and a researcher who wishes to compare a new algorithm with existing algorithms, or run a few algorithms on a real dataset, finds the task daunting. Furthermore, as algorithms become more complex, and as hybrid algorithms combining several approaches are suggested, the task of implementing such algorithms from scratch becomes hopeless.

The $\mathcal{MLC}$++ project at Stanford aims to help those 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 of existing ones.

$\mathcal{MLC}$++ 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 such as perceptron trees, multivariate trees 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 full version of this paper can be retrieved using URL
`http://robotics.stanford.edu:/users/ronnyk/mlc.html`
or using anonymous ftp from
`ftp://starry.stanford.edu:pub/ronnyk/intromlc.ps` .

1.1 Problems with current methodology

> *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 ...*
> — *Robert Holte [7]*

Although many experimental results have appeared in the literature, 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 experiments and comparisons. Extensive experiments, and broad comparisons are only possible when algorithms are available in an integrated uniform environment. We describe six problems with the current methodology.

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.

Comparisons on different datasets
Results are given on different datasets, and cannot be easily compared.

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.

Experiments are hard to replicate
Replication plays an important role in science, since it ensures that results are robust and reliable before they are widely accepted.

Lesion studies and hybrid algorithms
Machine learning systems contain many 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. Implementing such algorithms from scratch, and with sufficient generality to support varying of many components (factor experiments) is an arduous task.

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 [17] and CART [3] give only a rudimentary display of an induced tree.

A library providing a common framework and basic tools for implementing learning algorithms 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 framework, researchers will publish more of their code.

2 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 similar concerns.

An extensive collection of over 100 datasets has been collected by Murphy and Aha at the University of California at Irvine [14]. 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 [13].

Mooney has a collection of a few machine learning algorithms implemented in Lisp at UCI [14], but they are not an integrated environment, and are not very efficient.

StatLog [19] is an ESPRIT project studying the behavior of over twenty algorithms (mostly in the ML-Toolbox), on over twenty datasets. *StatLog* is an instance of a good experimental study, but does not provide the tools to aid researchers in performing similar studies.

Wray Buntine has recently suggested a unified approach to some machine learning paradigms using graphical models. We believe $\mathcal{MLC}$++ could be a base on which such an automatic compiler could be built.

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

In this section we describe the main decomposition of $\mathcal{MLC}$++ giving an overview of several of the important classes in $\mathcal{MLC}$++. The library is divided into five 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 [15], and dot from AT&T [6].

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 data, converting attribute types between formats (*e.g.*, local encoding, binary encoding), and normalizing values.
- **Wrappers** Algorithms that "wrap around" an induction algorithm to generate accuracies from test sets, estimate accuracy using cross-validation or bootstrap, 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.

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, perceptron categorizer, nearest-neighbor categorizer, decision tree categorizer, and decision graph 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 [16] always uses attribute categorizers for nominal attributes and threshold categorizers for real attributes. To generate multivariate trees with perceptrons at nodes, the induction algorithm can put perceptron categorizers at the nodes.

Induction algorithms
Induction algorithms induce categorizers. The library currently provides a majority inducer, a nearest-neighbor inducer [1], an ID3-like decision tree inducer [16], and an inducer for oblivious read-once decision graphs [9].

Visualization tools
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.

Visualization of structures
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 [6].

Visualization of discrete data
For viewing datasets and induced concepts, we have implemented General Logic Diagrams [20], a method for diagrammatic visualization of discrete data. After running an induction algorithm, users may gain insight about the induced concept by inspecting the GLD.

Currently, $\mathcal{MLC}$++ consists of over 25,000 lines of code (ignoring public domain code used), and over 7,000 lines of test code to verify correctness. 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.

4 Summary

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?
— Frederick Brooks [4]

$\mathcal{MLC}$++ is an attempt at providing such an off-the-shelf package to researchers and users of machine learning algorithms. 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 [8, 9, 10, 11, 12]. 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

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

[2] 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.

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

[4] 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.

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

[6] 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.

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

[8] George John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann, July 1994. Available by anonymous ftp from: `starry.Stanford.EDU:pub/ronnyk/ml94.ps`.

[9] Ron Kohavi. Bottom-up induction of oblivious, read-once decision graphs : strengths and limitations. In *Twelfth National Conference on Artificial Intelligence*, pages 613–618, July 1994. Available by anonymous ftp from `Starry.Stanford.EDU:pub/ronnyk/aaai94.ps`.

[10] Ron Kohavi. Feature subset selection as search with probabilistic estimates. In *AAAI Fall Symposium on Relevance*, pages 122–126, November 1994. Available by anonymous ftp from: `starry.Stanford.EDU: pub/ronnyk/aaaiSymposium94.ps`.

[11] Ron Kohavi. A third dimension to rough sets. In *Third International Workshop on Rough Sets and Soft Computing*, pages 244–251, November 1994. Available by anonymous ftp from: `starry.Stanford.EDU:pub/ronnyk/roughOODG.ps`.

[12] Ron Kohavi. Useful feature subsets and rough set reducts. In *Third International Workshop on Rough Sets and Soft Computing*, pages 310–317, November 1994. Available by anonymous ftp from: `starry.Stanford.EDU:pub/ronnyk/rough.ps`.

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

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

[15] 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`.

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

[17] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, Los Altos, California, 1993.

[18] 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.

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

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