Updated September 25, 1998

D2.1.2 $\mathcal{MLC}$++

Ron Kohavi
ronnyk@engr.sgi.com

Dan Sommerfield
sommda@engr.sgi.com

Data Mining and Visualization
Silicon Graphics, Inc.
2011 N. Shoreline Blvd, M/S 8U-850
Mountain View, CA, 94043

Abstract

$\mathcal{MLC}$++, the Machine Learning library in C++ is a set of libraries and utilities that can aid developers in interfacing machine learning technology, aid users in selecting an appropriate algorithm for a given task, and aid researchers in developing new algorithms, especially hybrid algorithms and multi-strategy algorithms.

D2.1.2.1 Motivation for $\mathcal{MLC}$++

A learning algorithm cannot outperform any other learning algorithm when the performance measure is the expected generalization accuracy, assuming all possible target concepts are equally likely. This theoretical result, sometimes called the No Free Lunch Theorem or Conservation Law (Mitchell 1982, Wolpert 1994, Schaffer 1994), implies that it is impossible for any algorithm to have the highest generalization accuracy in all domains.

The theorem has few interesting consequences in practice because users are interested in performance on a specific domain or a set of domains, and not all concepts are equally probable. For a given domain, comparisons certainly make sense and some algorithms will perform better than others.

The ability to easily test and compare the performance of algorithms on given problems was one of the factors that motivated the development of $\mathcal{MLC}$++. In many cases, accuracy is not the sole measure of performance. Some domains may require comprehensibility, compactness, fast deployment, incremental learning, etc. Whatever the performance measure (Fayyad, Piatetsky-Shapiro & Smyth 1996), the ability to try different algorithms and see their result is extremely useful.

Although there are many rules of thumb for choosing learning algorithms, we believe the best method is to try several of them and see the results (Kohavi, Sommerfield & Dougherty 1997).

$\mathcal{MLC}$++ can serve three types of users:

1. System integrators, developers of tools, and developers of vertical applications can use the libraries to integrate machine learning technology into products and solutions. MineSet™ is the best example of this use. $\mathcal{MLC}$++ is integrated into a commercial data mining product sold by Silicon Graphics [link to section D2.2.5] (Silicon Graphics 1998, Brunk, Kelly & Kohavi 1997), which provides access to some of the algorithms in $\mathcal{MLC}$++ through a graphical user interface (GUI). In addition, it provides database access, transformations, and visualizations.
2. Machine learning researchers developing learning algorithms can use the library itself, modify it, add routines, and build hybrid algorithms.
3. Machine learning researchers and power users comparing learning algorithms can use the $\mathcal{MLC}$++ utilities to compare performance of different algorithms.

Users of type one and two use the library itself. Users of type three use the utilities built on top of $\mathcal{MLC}$++, which give a command-line interface to the underlying functions.

D2.1.2.2 Short History of $\mathcal{MLC}$++

The development of $\mathcal{MLC}$++ started at Stanford University in the summer of 1993 and continued there for two years. The original library was public domain. Since

late 1995 the distribution and support have moved to Silicon Graphics (Kohavi & Sommerfield 1995). Development of $\mathcal{MLC}$++ continues in the analytical data mining group at Silicon Graphics. The original sources are still available as public domain; the enhanced sources are also available off the web, although their use is restricted to research use only.

Over 15 students worked on the project at Stanford, several over multiple quarters. Today several people at Silicon Graphics work on $\mathcal{MLC}$++ development and maintenance as part of the MineSet product.

The $\mathcal{MLC}$++ mailing list has over 800 subscribers. Over 1600 unique sites have downloaded $\mathcal{MLC}$++ source code.

D2.1.2.X $\mathcal{MLC}$++ Algorithms

While $\mathcal{MLC}$++ is useful for writing new algorithms, most users simply use it to test different learning algorithms. Pressure from reviewers to compare new algorithms with others led us to also interface induction algorithms written by other people. $\mathcal{MLC}$++ provides a uniform interface for these algorithms, termed *external inducers*.

The following induction algorithms were implemented in $\mathcal{MLC}$++:

A constant predictor based on majority, a decision table (Kohavi & Sommerfield 1998), ID3 [link to section C5.1.3] (Quinlan 1986), Lazy decision trees (Friedman, Kohavi & Yun 1996), nearest-neighbor [link to section C5.1.6] (Dasarathy 1990), Naive-Bayes [link to section C5.1.5] (Domingos & Pazzani 1997), 1R (Holte 1993), OODG (Kohavi 1995*b*), Option Decision Trees (Kohavi & Kunz 1997), Perceptron [link to section C5.1.8] (Hertz, Krogh & Palmer 1991), and Winnow (Littlestone 1988).

The following external inducers are interfaced by $\mathcal{MLC}$++:

C4.5 and C4.5-rules [link to section C5.1.3] Quinlan (1993), C5.0, CART (Breiman, Friedman, Olshen & Stone 1984), CN2 [link to section C5.2] (Clark & Boswell 1991), IB (Aha 1992), Neural Network: Aspirin/MIGRAINES [link to section C5.1.8] (Hertz et al. 1991), OC1 (Murthy, Kasif & Salzberg 1994), PEBLS (Cost & Salzberg 1993), Ripper (Cohen 1995), and T2 (Auer, Holte & Maass 1995).

Because algorithms are encapsulated as C++ objects in $\mathcal{MLC}$++, we were able to build useful wrappers. A *wrapper* is an algorithm that treats another algorithm as a black box and acts on its output. Once an algorithm is written in $\mathcal{MLC}$++, a wrapper may be applied to it with no extra work.

The most important wrappers in $\mathcal{MLC}$++ are performance estimators, feature selectors, and ensemble creators. Performance estimators apply any of a range of methods, including holdout, cross-validation, bootstrap (Kohavi 1995*a*), learning curves, and ROC Curves (Provost & Fawcett 1997) to evaluate the performance of an inducer. Feature selection methods run a search based on performance estimation using the inducer itself to determine which attributes in the database are useful for learning. The wrapper approach to feature selection automatically tailors the feature set to the inducer being run (Kohavi & John 1997). Ensemble methods create different models and then combine their votes. $\mathcal{MLC}$++ includes bagging, boosting, and several variants (Bauer & Kohavi n.d., Breiman 1996, Freund & Schapire 1996).

In addition to the above, $\mathcal{MLC}$++ supports a discretization wrapper/filter, which pre-discretizes the data, allowing algorithms that do not support continuous features (or those that do not handle them well) to work properly. A parameter optimization wrapper allows tuning the parameters of an algorithm automatically based on a search in the parameter space that optimizes the performance estimate of an inducer using different parameters.

$\mathcal{MLC}$++ Software Architecture

$\mathcal{MLC}$++ is a class library for development of machine learning algorithms. Its software architecture is particularly important as it is the foundation of the entire project.

Coding standards and safety $\mathcal{MLC}$++ defines a set of coding standards and conventions. Because $\mathcal{MLC}$++ is a common framework for machine learning and data mining research and because research results can easily be ruined by software bugs, the $\mathcal{MLC}$++ coding standards promote safety over efficiency. All major classes in the library contain intensive integrity checks and $\mathcal{MLC}$++ programs always abort at the first sign of an error. While this method may be inappropriate for real-time systems, it helps guarantee the correctness of the

algorithms. $\mathcal{MLC}$++ also incorporates an optional fast mode which deactivates many of the checks.

Platforms $\mathcal{MLC}$++ is a cross-platform library. Version 2.01 has been released for SGI IRIX, Windows NT (using the Microsoft Visual C++ compiler), and gnu g++. The g++ version may be used on most UNIX platforms including Solaris and Linux. All compilation is handled by a cross-platform compilation wrapper which selects the options most appropriate for the target platform. This wrapper script is designed to be extensible so that new platforms may be added with ease.

MCore The $\mathcal{MLC}$++ library is built around a set of foundation classes known as MCore. MCore provides common classes such as Arrays, Hash Tables, and Link Lists which are used heavily within $\mathcal{MLC}$++. It is a template-based library similar in spirit to the Standard Template Library yet focusing more on code safety.

Central learning classes The library contains an extensive set of machine learning support classes. These classes come in two varieties. The first set forms an API for incorporating algorithms into $\mathcal{MLC}$++. Algorithms writing to this API gain the full benefits of $\mathcal{MLC}$++: for example, an algorithm written as an $\mathcal{MLC}$++ classifier can immediately be run through the cross validation and feature selection wrappers. The API covers classification, regression, clustering, automatic binning, and search algorithms. Algorithms written outside of the library may be wrapped by *external inducers* which use the API. Once enclosed in an $\mathcal{MLC}$++ wrapper, the *external inducer* gains most of the benefits of $\mathcal{MLC}$++. For example, we can use our own bootstrap accuracy estimation to compare results from C5.0 and our internal decision trees. The second set of classes form the building blocks for data mining and machine learning algorithms. Code for computing statistics, information theoretic measures (such as entropy), and basic numerics is included in this set. We also provide a set of data handling and simple transformation classes. Simple operations such as removing attributes and discretization and all handled within $\mathcal{MLC}$++.

Wrappers The class-based architecture of $\mathcal{MLC}$++ facilitates rapid development of hybrid classification algorithms. Wrapper algorithms are particularly easy to develop in $\mathcal{MLC}$++. $\mathcal{MLC}$++ wrappers use the concept of an Inducer, which is a generic model-building class. To build a wrapper, one creates a subclass

of Inducer which incorporates another Inducer class within it. This method allows the wrapper to use any algorithm implemented within the library's API. The library also provides a generic Categorizer class which represents a model. Hybrid models may be constructed by embedding Categorizer classes within each other. For example, the NBTree model (Kohavi 1996) embeds a Naive Bayes models within a decision tree.

Persistent Models All models (classifiers, regressors, and clustering models) written in $\mathcal{MLC}$++ may be saved to disk in a common persistent format. These models may be read back from disk for deployment or further modification. The $\mathcal{MLC}$++ library provides support tools and APIs for reading and writing these files. New models may be made persistent by following the library guidelines. Hybrid models are automatically supported by the mechanism once all internal parts are supported. The persistent model files are stored in an extensible ASCII format.

Evaluation A main feature of $\mathcal{MLC}$++ is providing an extensible framework for data mining research. A large portion of this framework is a common set of evaluation classes. These classes support evaluation methods like holdout, cross validation, bootstrap, and learning curves, and may be applied to any algorithm written as a subclass of the $\mathcal{MLC}$++ Inducer. External inducers such as C5.0 may be evaluated with no extra work once they are plugged into $\mathcal{MLC}$++. The evaluation framework is also used as the basis for evaluating nodes in the search space for search algorithms like feature subset search. The framework is designed to be extensible so that new evaluation methods may be added and immediately applied to the full range of algorithms.

Testing architecture To guarantee correctness of the code base, $\mathcal{MLC}$++ includes approximately 40,000 lines of automated testing code. Each module in the code is fully tested under a range of conditions. The tests are calibrated so that they produce identical output on different platforms as much as possible. Tests are run in both fast mode (where most integrity checks are disabled for speed), and two different levels of debugging.

Utilities $\mathcal{MLC}$++ contains a set of machine learning utilities, written using the library. These utilities perform tasks such as induction of models, scoring of records, and evaluation of the model building process. We also provide utilities for association rule generation, discretization, and clustering. Aside from being useful in themselves, these applications provide good examples on the use

of $\mathcal{MLC}$++. The majority of users interested in $\mathcal{MLC}$++ have downloaded the $\mathcal{MLC}$++ utilities, which are available off the web as a standalone package for several platforms.

Integration with MineSet

$\mathcal{MLC}$++ integrates with MineSet(Silicon Graphics 1998, Brunk et al. 1997) to provide the following features:

Database access $\mathcal{MLC}$++ itself can only read data from flat files in the C4.5/UCI format (C. Blake & Merz 1998). However, when coupled with MineSet, $\mathcal{MLC}$++ is capable of taking input directly from Oracle, Informix, and Sybase databases.

Apply model/model deployment MineSet provides a scoring module using $\mathcal{MLC}$++ to apply classifiers stored in the $\mathcal{MLC}$++ Persistent Classifier format. Any stored classifier supported by MineSet may be applied using this method. Furthermore, while $\mathcal{MLC}$++ algorithms operate only on data loaded into memory, MineSet provides a model deployment capability which runs directly from disk.

Transformations While $\mathcal{MLC}$++ itself provides only limited data transformations, a much more extensive set is available through MineSet. $\mathcal{MLC}$++ itself handles attribute removal (projection), binning, and simple feature construction. MineSet adds aggregation, more binning options, filtering, more advanced feature construction, and an integrated ability to apply saved models to the data. If it is connected to a database, MineSet can also read its data through an SQL query, allowing more complex operations such as joins.

Mining tool options $\mathcal{MLC}$++ contains an extensible system for handling the large number of options needed to run a data mining algorithm. Options are passed to $\mathcal{MLC}$++ from MineSet through option files. While the contents of these files are mostly determined by the MineSet user interface, the user may add additional options using a special file called .mineset-classopt. Using this file gives advanced MineSet users access to the full range of $\mathcal{MLC}$++ options, producing greater flexibility in using MineSet.

$\mathcal{MLC}$++ Visualization

$\mathcal{MLC}$++ interfaces with two visualization packages: MineSet and GraphViz.

$\mathcal{MLC}$++ generates MineSet visualizations for its decision tree, naive bayes, decision table, clustering, and association rule algorithms. The visualizations are generated in an ASCII format which may be read directly by MineSet. Visualizers may be launched automatically if they are installed on the $\mathcal{MLC}$++ target platform.

$\mathcal{MLC}$++ also generates graph visualizations of all graph-based algorithms using Graphviz (Dot/Dotty) from AT&T (Ellson, Gansner, Koutsofios & North 1998). These visualizations include decision trees, decision graphs, and visual representations of the search spaces used by the search engine.

D2.1.2.X Acknowledgment

The $\mathcal{MLC}$++ project could not have started without the support of Nils Nilsson and Yoav Shoham at Stanford University. $\mathcal{MLC}$++ was partly funded by ONR grants N00014-94-1-0448, N00014-95-1-0669, and NSF grant IRI-9116399. After the summer of 1995, Silicon Graphics Inc. provided continued support for $\mathcal{MLC}$++ as part of the MineSet project.

References

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

Auer, P., Holte, R. & Maass, W. (1995), Theory and applications of agnostic PAC-learning with small decision trees, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann.

Bauer, E. & Kohavi, R. (n.d.), 'An empirical comparison of voting classification algorithms: Bagging, boosting, and variants', *Machine Learning* p. To appear.

Breiman, L. (1996), 'Bagging predictors', *Machine Learning* **24**, 123–140.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984), *Classification and Regression Trees*, Wadsworth International Group.

Brunk, C., Kelly, J. & Kohavi, R. (1997), MineSet: an integrated system for data mining, *in* D. Heckerman, H. Mannila, D. Pregibon & R. Uthurusamy, eds, 'Proceedings of the third international conference on Knowledge Discovery and Data Mining', AAAI Press, pp. 135–138.
`http://mineset.sgi.com`.

C. Blake, E. K. & Merz, C. (1998), 'UCI repository of machine learning databases'.
`http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Clark, P. & Boswell, R. (1991), Rule induction with CN2: Some recent improvements, *in* Y. Kodratoff, ed., 'Proceedings of the fifth European conference (EWSL-91)', Springer Verlag, pp. 151–163.
`http://www.cs.utexas.edu/users/pclark/papers/newcn.ps`.

Cohen, W. W. (1995), Fast effective rule induction, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann.

Cost, S. & Salzberg, S. (1993), 'A weighted nearest neighbor algorithm for learning with symbolic features', *Machine Learning* **10**(1), 57–78.

Dasarathy, B. V. (1990), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, California.

Domingos, P. & Pazzani, M. (1997), 'Beyond independence: Conditions for the optimality of the simple Bayesian classifier', *Machine Learning* **29**(2/3), 103–130.

Ellson, J., Gansner, E., Koutsofios, E. & North, S. (1998), Graphviz.
`http://www.research.att.com/sw/tools/graphviz`.

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996), From data mining to knowledge discovery: An overview, *in* 'Advances in Knowledge Discovery and Data Mining', AAAI Press and the MIT Press, chapter 1, pp. 1–34.

Freund, Y. & Schapire, R. E. (1996), Experiments with a new boosting algorithm, *in* L. Saitta, ed., 'Machine Learning: Proceedings of the Thirteenth National Conference', Morgan Kaufmann, pp. 148–156.

Friedman, J., Kohavi, R. & Yun, Y. (1996), Lazy decision trees, *in* 'Proceedings of the Thirteenth National Conference on Artificial Intelligence', AAAI Press and the MIT Press, pp. 717–724.

Hertz, J., Krogh, A. & Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation*, Addison Wesley.

Holte, R. C. (1993), 'Very simple classification rules perform well on most commonly used datasets', *Machine Learning* **11**, 63–90.

Kohavi, R. (1995*a*), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 1137–1143. `http://robotics.stanford.edu/~ronnyk`.

Kohavi, R. (1995*b*), Wrappers for Performance Enhancement and Oblivious Decision Graphs, PhD thesis, Stanford University, Computer Science department. STAN-CS-TR-95-1560,
http://robotics.Stanford.EDU/~ronnyk/teza.ps.Z.

Kohavi, R. (1996), Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid, *in* 'Proceedings of the Second International Conference on Knowledge Discovery and Data Mining', pp. 202–207. Available at
`http://robotics.stanford.edu/users/ronnyk`.

Kohavi, R. & John, G. H. (1997), 'Wrappers for feature subset selection', *Artificial Intelligence* **97**(1-2), 273–324.
`http://robotics.stanford.edu/users/ronnyk`.

Kohavi, R. & Kunz, C. (1997), Option decision trees with majority votes, *in* D. Fisher, ed., 'Machine Learning: Proceedings of the Fourteenth International Conference', Morgan Kaufmann Publishers, Inc., pp. 161–169. Available at
`http://robotics.stanford.edu/users/ronnyk`.

Kohavi, R. & Sommerfield, D. (1995), MLC++ utilities. `www.sgi.com/Technology/mlc`.

Kohavi, R. & Sommerfield, D. (1998), Targeting business users with decision table classifiers, *in* R. Agrawal, P. Stolorz & G. Piatetsky-Shapiro, eds, 'Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining', AAAI Press, pp. 249–253.

Kohavi, R., Sommerfield, D. & Dougherty, J. (1997), 'Data mining using $\mathcal{MLC}$++: A machine learning library in C++', *International Journal on Artificial Intelligence Tools* **6**(4), 537–566.
`http://www.sgi.com/Technology/mlc`.

Littlestone, N. (1988), 'Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm', *Machine Learning* **2**, 285–318.

Mitchell, T. M. (1982), 'Generalization as search', *Artificial Intelligence* **18**, 203–226. Reprinted in Shavlik and Dietterich (eds.) *Readings in Machine Learning*.

Murthy, S. K., Kasif, S. & Salzberg, S. (1994), 'A system for the induction of oblique decision trees', *Journal of Artificial Intelligence Research* **2**, 1–33.

Provost, F. & Fawcett, T. (1997), Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions, *in* D. Heckerman, H. Mannila, D. Pregibon & R. Uthurusamy, eds, 'Proceedings of the third international conference on Knowledge Discovery and Data Mining', AAAI Press.

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

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California.

Schaffer, C. (1994), A conservation law for generalization performance, *in* 'Machine Learning: Proceedings of the Eleventh International Conference', Morgan Kaufmann, pp. 259–265.

Silicon Graphics (1998), *MineSet User's Guide*, Silicon Graphics, Inc. http://mineset.sgi.com.

Wolpert, D. H. (1994), The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework, *in* D. H. Wolpert, ed., 'The Mathematics of Generalization', Addison Wesley.