MLPACK: A Scalable C++ Machine Learning Library

Ryan R. Curtin
gth671b@mail.gatech.edu

James R. Cline
james.cline@gatech.edu

Neil P. Slagle
nslagle3@gatech.edu

Matthew L. Amidon
mamidon@gatech.edu

Alexander G. Gray
agray@cc.gatech.edu

School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA 30318

Abstract

MLPACK is a new, state-of-the-art, scalable C++ machine learning library, which will be released in early December 2011. Its aim is to make large-scale machine learning possible for novice users by means of a simple, consistent API, while simultaneously exploiting C++ language features to provide maximum performance and maximum flexibility for expert users. MLPACK provides many cutting-edge algorithms, including—but not limited to—tree-based k-nearest-neighbors, fast hierarchical clustering, MVU (maximum variance unfolding) with LRSDP, RADICAL (ICA), and HMMs. Each of these algorithms is highly flexible and configurable, and all of them can be configured to use sparse matrices for datasets. Benchmarks are shown to prove that MLPACK scales more effectively than competing toolkits. Future plans for MLPACK include support for parallel algorithms, support for disk-based datasets, as well as the implementation of a large collection of cutting-edge algorithms. More information on MLPACK is available at the project's website, which is located at `http://www.mlpack.org`.

1 Introduction and Goals

While there are many machine learning libraries available to the public for an assortment of machine learning tasks, there are very few libraries targeted at the average user that focus on scalability. For instance, the popular WEKA toolkit [7] focuses on ease of use but does not scale well. On the other end of the continuum, the focus of the distributed Apache Mahout [12] library is scalability using larger, more overhead-intensive setups such as clusters and powerful servers; the average user is unlikely to have this. In addition, many other libraries implement only a few methods; for instance, libsvm [3] and the Tilburg Memory-Based Learner (TiMBL) [5] are highly capable and scalable software but each only implements one method. A user must learn a new API for each method.

MLPACK, which is meant to be the machine learning analog to the general-purpose LAPACK linear algebra library, aims to bridge the gap between scalability and ease of use. The library is written in C++ using the Armadillo matrix library [10]. MLPACK uses templated-based optimization techniques to eliminate unnecessary copying of datasets and optimize mathematical expressions in ways the compiler's optimizer cannot. In addition, MLPACK exploits the generic programming features of C++. This allows a user to easily customize machine learning methods to their own purposes without incurring any performance penalty. To our knowledge, MLPACK is the only machine learning library to exploit these techniques to their full potential.

In addition, the consistent, intuitive interface of MLPACK minimizes the learning time necessary for either an undergraduate student or a machine learning expert. Unlike many other scientific software packages, complete and comprehensible documentation is a high priority for MLPACK, and each method that MLPACK implements provides detailed reference.

The development team of MLPACK strives to balance four overarching goals in the design of the library:

- Implement scalable, fast machine learning algorithms
- Design an intuitive, consistent, and simple API for users who are not C++ gurus
- Implement as large a collection as possible of machine learning methods
- Provide cutting-edge machine learning algorithms that no other library does

This paper is meant to serve as a quick introduction to the features MLPACK provides, its simple and extensible API, and the superior performance and scalability of the library. The ways in which each of the four design goals are achieved are analyzed.

2 Scalable, Fast Machine Learning Algorithms

The most important feature of MLPACK is the scalability of the machine learning algorithms it implements; this scalability is achieved mostly by the use of C++. C++ is not the most popular choice of language for most machine learning implementations; instead, researchers generally choose simpler languages such as MATLAB, C, FORTRAN, and sometimes Java or Python. While each of those languages is modern and supports modern featuers, none are as complex as C++.

C++ is a highly complex language which requires training and expertise to fully understand and utilize. Unlike most other languages used in machine learning, C++ supports generic programming via templates. Although Java also supports generics, the C++ template mechanism is far more extensible and useful.

In the late 1990s and early 2000s, shortly after the introduction of templates into C++, the field of template metaprogramming emerged, culminating, in part, with Alexandrescu's seminal *Modern C++ Design* [2] in 2001. The main appeal of template metaprogramming is the ability to move parts of computation that are regularly performed at runtime into compile-time. However, this concept has not gained significant traction in the field of machine learning software, even ten years after the introduction of template metaprogramming.

MLPACK's underlying matrix library, Armadillo, uses lazy evaluation to avoid both unnecessary copies and unnecessary operations [10]. To illustrate this, consider the following very simple expression (assuming `x`, `y`, `z`, and `w` are square matrices of the same size):

```
w = x + (y + z)
```

If implemented as written in MATLAB, this will first compute `(y + z)`, store it in a temporary matrix, then add `x` to the temporary matrix and store the result in `w`. Clearly, for massive datasets, the creation of an unnecessary temporary matrix causes both memory and performance issues. The Armadillo library, relying on lazy evaluation using template metaprogramming, is able to avoid this entirely by optimizing operation performance at compile-time. The vast majority of machine learning libraries do not support this type of compile-time optimization.

Another drawback of most commonly-used scientific languages is suboptimal support of user-defined metrics and kernels. For instance, Shogun [11] provides arbitrary distance metric support via inheritance and virtual functions, but the use of virtual functions or other dynamic bindings incurs a performance penalty for the lookup of the dynamic binding [8]. MLPACK, instead, uses static polymorphism—an extension of the Curiously Recurring Template Pattern [4]—as well as policy-based design [2]. The performance penalty of the dynamic binding is eliminated by resolving the binding entirely at compilation time.

The use of policy-based design allows incredible flexibility. MLPACK supports entirely arbitrary, user-defined distance metrics or kernel functions. More importantly, through this paradigm ML-

PACK is able to support the use of both dense and sparse matrices anywhere in each of the algorithms it implements. No other machine learning library to date provides this functionality.

3 A Consistent, Simple API

The use of complex template techniques in C++ generally incurs complex and unwieldy APIs, forcing a user to read through countless pages of documentation to understand how to use the code. However, because a goal of MLPACK is to provide an intuitive and simple API, these issues can be avoided. This is possible, in part, by providing default parameters which can be left unspecified. For instance, the Neighborhood Components Analysis metric learning method can be created very easily (assuming `data` is an Armadillo matrix containing the dataset):

```
NCA nca(data);
```

However, an expert user could run NCA in a non-Euclidean metric space, using a custom optimizer, on sparse double-precision floating point input data:

```
NCA<ACustomMetric, ACustomOptimizer, arma::SpMat<double> > nca(data);
```

Each MLPACK method is designed similarly, and importantly, each MLPACK method has a consistent API. A user can move from one method to another, expecting to interact with the new method in the same way. This consistent API is a main reason that MLPACK's API can be called intuitive.

In addition, stringent documentation is a key requirement for an algorithm to be accepted into MLPACK. This comprehensive documentation allows a novice user to simply run the basic algorithm and an expert user to customize the algorithm to their specific needs.

4 A Large Collection of Machine Learning Methods

MLPACK implements commonly used machine learning algorithms as well as algorithms for which no other implementation is available. It provides a set of core routines, including several optimizers, such as L-BFGS and the Nelder-Mead method. MLPACK also contains a strong set of tree-building and tree query routines, making MLPACK's implementations of neighbor-search problems and general n-body problems [6] very powerful. As mentioned earlier, MLPACK even supports sparse matrices for any of its methods through the policy-based design C++ paradigm [2]; this innovation is novel to the machine learning software community.

Each machine learning method provides a well-documented command-line interface for any researcher who does not wish to extend MLPACK's code but instead use the provided methods. By using the library in this manner, a user of MLPACK does not even need to know C++ or any programming language. On the other hand, for an expert researcher, MLPACK allows arbitrary distance metrics and kernel functions for all of its methods, among its many customizable features.

Below is a list of the machine learning methods which will be available, along with some key features for each of them, at the time of the library's release in December 2011:

- Fast Hierarchical Clustering (Euclidean Minimum Spanning Trees)
- Gaussian Mixture Models (trained via EM)
- Hidden Markov Models (training, prediction, and classification)
- Kernel Principal Components Analysis
- K-Means clustering
- LARS/Lasso Regression
- Least-squares Linear Regression
- Maximum Variance Unfolding (using LRSDP)
- Naive Bayes Classifier
- Neighborhood Components Analysis (using LRSDP)
- RADICAL (Robust, Accurate, Direct ICA aLgorithm)

Dataset	MLPACK	WEKA	Shogun	MATLAB	mlpy	sklearn
1000x10	**0.078s**	0.271s	0.132s	0.166s	0.179s	0.341s
3162x10	**0.267s**	1.065s	1.093s	0.796s	0.974s	0.916s
10000x10	**1.332s**	4.734s	11.890s	4.026s	9.961s	3.549s
31622x10	**7.270s**	27.890s	120.320s	17.631s	116.965s	15.213s
100000x10	**47.350s**	171.313s	1357.910s	110.570s	1621.045s	75.039s
316227x10	**253.541s**	789.317s	$>$ 9000.000s	588.967s	$>$ 9000.000s	363.691s
1000000x10	**1423.949s**	*failure*	*failure*	2603.316s	$>$ 9000.000s	1550.720s
10000x31	**6.932s**	45.240s	13.980s	26.990s	13.688s	45.582s
10000x100	**18.075s**	192.548s	27.251s	82.214s	29.039s	198.953s
10000x316	**53.500s**	2957.581s	65.020s	245.471s	69.850s	732.694s
10000x1000	**174.704s**	*failure*	178.715s	750.660s	201.081s	2533.120s

Table 1: k-NN benchmarks.

- Tree-based k-nearest-neighbors search and classifier
- Tree-based range search

In addition, many methods are currently in development and will be released in the future.

5 Benchmarks

To demonstrate the efficiency of the algorithms implemented in MLPACK, the running time of k-nearest-neighbors is compared with other libraries. Only one algorithm is tested for the sake of conciseness, but when the library is released, full benchmarks for each algorithm will be available. The benchmarks shown here were run on a modest consumer-grade workstation containing an AMD Phenom II X6 1100T processor clocked at 3.3 GHz and 8 GB of RAM.

The libraries being compared are MLPACK, WEKA [7], the MATLAB `knnsearch()` routine, the Shogun Toolkit [11], the Python package mlpy [1], and the Python package scikit.learn ('sklearn') [9]. Several randomly generated, uniformly distributed datasets of varying sizes were used for this benchmark. The computation time of k-NN with $k = 5$ for each library and each dataset size is given in Table 1. The listed computational time includes the time taken to load the datasets (CSV) and save the results.

MLPACK is faster than every competitor for every case tested here. It is clear that MLPACK scales more gracefully and effectively than any of its competitors for k-nearest-neighbors. Benchmarks for other methods (which cannot be shown here due to space constraints) give similar results.

6 Future Plans

In spite of the favorable benchmarks shown here, these are by no means the best benchmarks MLPACK will ever achieve. MLPACK has an active development team of seven core developers, as well as several contributors. Because MLPACK is an open-source project, contributions from any outside source are welcome – as well as feature requests and bug reports. This means that the performance of MLPACK algorithms, as well as the extensibility of MLPACK algorithms and the breadth of algorithms MLPACK implements, are all certain to improve.

For the first release of MLPACK, parallelism was unable to be implemented elegantly, but experimental parallel MLPACK code is currently in testing. Work is currently ongoing into how to implement parallelism while both maintaining a simple API and not incurring large, reverse-incompatible API changes. Parallel algorithms are expected to be available in the next major release of MLPACK. Other useful planned features include the ability to use on-disk databases, instead of requiring the dataset to be loaded entirely into RAM.

Many additional algorithms are either planned or in development for MLPACK and will be released in the future. These algorithms include dimensionality reduction techniques such as linear discriminant analysis (LDA) and locally linear embedding (LLE), source separation methods such

as non-negative matrix factorization (NMF) and information maximization (Infomax) ICA, kernel density estimation methods, and a variety of other algorithms.

7 Conclusion

We have shown that MLPACK is a state-of-the-art scalable C++ machine learning library which leverages the powerful C++ concept of generic programming to give excellent performance on large datasets. The four goals which MLPACK development adheres to requires that in addition to being very fast, the code itself must be easy for a novice user to understand and use.

MLPACK supports a wide range of machine learning methods, each of which is highly configurable. For instance, a user can write an arbitrary distance metric or kernel function and use it with MLPACK methods. A user can also choose to use sparse or dense matrices for their datasets at will, which is flexibility other machine learning libraries do not offer.

In the future, MLPACK aims to support parallelism while maintaining a simple API, as well as publishing new state-of-the-art machine learning algorithms regularly. Effective interaction with the machine learning community through the channels of MLPACK's open-source development model, found at `http://www.mlpack.org`, enables the library to be shaped to the needs of the community, allowing MLPACK to be an effective tool in advancing the field of machine learning.

Acknowledgements

A full list of developers and researchers – other than the authors – who have contributed significantly to MLPACK: Sterling Peet, Vlad Grantcharov, Ajinkya Kale, Bill March, Dongryeol Lee, Nishant Mehta, Parikshit Ram, Chip Mappus, Hua Ouyang, Long Quoc Tran, and Noah Kauffman.

References

References

[1] Davide Albanese, Giuseppe Jurman, Roberto Visintainer, and Stefano Merler. mlpy: Machine Learning PYThon. October 2011. Project homepage at `http://mlpy.fbk.eu/`.

[2] Andrei Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Professional, Indianapolis, February 2001.

[3] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] James Coplien. *C++ Gems*. Cambridge University Press & Sigs Books, 1996.

[5] Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. TiMBL: Tilburg Memory Based Learner, version 4.0, Reference Guide. Technical Report 01-04, ILK, 2001.

[6] Alexander G. Gray and Andrew W. Moore. 'N-Body' problems in Statistical Learning. In *Advances in Neural Information Processing Systems 14*, volume 4, pages 521–527. Citeseer, 2001.

[7] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.

[8] ISO/IEC. Technical Report on C++ Performance. Technical report, February 2006.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical report, NICTA, 2010.

[11] Soeren Sonnenburg, Gunnar Raetsch, Sebastian Henschel, Christian Widmer, Jonas Behr, Alexander Zien, Fabio de Bona, Alexander Binder, Christian Gehl, and Vojtech Franc. The SHOGUN Machine Learning Toolbox. *Journal of Machine Learning Research*, 11:1799–1802, June 2010.

[12] The Apache Foundation. Apache Mahout. October 2011. Project homepage and software download at `http://lucene.apache.org/mahout/`.