The Journal of Open Source Software

mlpack 4: a fast, header-only C++ machine learning library

Ryan R. Curtin [1], **Marcus Edel** [2], **Omar Shrit** [1], **Shubham Agrawal** [1], **Suryoday Basak** [3], **James J. Balamuta** [4], **Ryan Birmingham** [5], **Kartik Dutt** [6], **Dirk Eddelbuettel** [7], **Rishabh Garg** [8], **Shikhar Jaiswal** [9], **Aakash Kaushik** [1], **Sangyeon Kim** [10], **Anjishnu Mukherjee** [11], **Nanubala Gnana Sai** [1], **Nippun Sharma** [8], **Yashwant Singh Parihar** [12], **Roshan Swain** [1], **and Conrad Sanderson** [13]

1 Independent Researcher **2** Collabora Ltd **3** Pennsylvania State University **4** Departments of Statistics and Informatics, University of Illinois, Urbana-Champaign **5** Emory University **6** Delhi Technological University **7** Department of Statistics, University of Illinois, Urbana-Champaign **8** Indian Institute of Technology Mandi **9** Microsoft Research India **10** NAVER WEBTOON AI **11** George Mason University **12** Department of Computer Science and Engineering, IIT Bombay **13** Data61/CSIRO and Griffith University

DOI: 10.21105/joss.05026

Software

- Review
- Repository
- Archive

Editor: Øystein Sørensen

Reviewers:

- @sandeep-ps
- @zhangjy-ge

Submitted: 25 November 2022
Published: 31 January 2023

Summary

For over 15 years, the mlpack machine learning library has served as a “swiss army knife'' for C++-based machine learning (Curtin et al., 2013). Its efficient implementations of common and cutting-edge machine learning algorithms have been used in a wide variety of scientific and industrial applications. This paper overviews mlpack 4, a significant upgrade over its predecessor (Curtin et al., 2018). The library has been significantly refactored and redesigned to facilitate an easier prototyping-to-deployment pipeline, including bindings to other languages (Python, Julia, R, Go, and the command line) that allow prototyping to be seamlessly performed in environments other than C++.

Statement of Need

The use of machine learning has become ubiquitous in almost every scientific discipline and countless commercial applications (Carleo et al., 2019; Jordan & Mitchell, 2015). There is one important commonality to virtually all of these applications: machine learning is often computationally intensive, due to the large number of parameters and large amounts of training data. This was the main motivator for the original development of mlpack in the C++ language, which allows for efficient close-to-the-metal implementations (Curtin et al., 2013).

But speed is not everything: development and deployment of applications that use machine learning can also be significantly hampered if the overall process is too difficult or unwieldy (Lavin et al., 2022; Paleyes et al., 2020). Furthermore, deployment environments often have computational or engineering constraints that make a full-stack Python solution infeasible (Fischer et al., 2020). As such, it is important that lightweight and easy-to-deploy machine learning solutions are available. This has motivated our refactoring and redesign of mlpack 4: we pair efficient implementations with easy and lightweight deployment, making mlpack suitable for a wide range of deployment environments. A more complete set of motivations can be found in the mlpack vision document (mlpack community, 2021).

mlpack is a general-purpose machine learning library, targeting both academic and commercial use; for instance, data scientists who need efficiency and ease of deployment, or, e.g., by researchers who need flexibility and extensibility. While there are other machine learning libraries intended to be used from C++, many, such as FAISS (Johnson et al., 2019) and FLANN (Muja & Lowe, 2009), are limited to a few specific algorithms, instead of a full range of machine learning algorithms, like mlpack provides. dlib-ml (King, 2009), on the other hand, does provide a broad toolkit of machine learning algorithms, but its extensibility is somewhat limited as it does not use policy-based design (Alexandrescu, 2001) to provide arbitrary user-defined behavior, and the range of machine learning algorithms provided is smaller than mlpack's.

Functionality

The library contains a wide variety of machine learning algorithms, some of which are new to mlpack 4. The list of algorithms includes linear regression, logistic regression, random forests, furthest-neighbor search (Curtin & Gardner, 2016), accelerated k-means variants (Curtin, 2017), kernel density estimation (Lee & Gray, 2008), and fast max-kernel search (Curtin & Ram, 2014). There is also a module for deep neural networks, which has implementations of numerous layer types, activation functions, and reinforcement learning applications. Details of the available functionality are provided in the online mlpack documentation. The efficiency of these implementations has been shown in various works (Curtin et al., 2013; Fang & Chau, 2016) using mlpack's benchmarking system (Edel et al., 2014).

The algorithms are available via automatically-generated bindings to Python, R, Go, Julia, and the command line. Each of these bindings has a unified interface across the languages; for example, a model trained in Python can be used from Julia or C++ (or any other language with mlpack bindings). The bindings are available in each language's package manager, as well as system-level package managers such as `apt` and `dnf`. Furthermore, ready-to-use Docker containers with the environment fully configured are available on DockerHub, and an interactive C++ notebook interface via the xeus-cling project is available on BinderHub.

Once a user has developed a machine learning workflow in the language of their choice, deployment is straightforward. The mlpack library is now header-only, and directly depends only on three libraries: Armadillo (Sanderson & Curtin, 2016), ensmallen (Curtin et al., 2021), and cereal. When using C++, the only linking requirement is to an efficient implementation of BLAS and LAPACK (required via Armadillo). This significantly eases deployment; a standalone C++ application with only a BLAS/LAPACK dependency is easily deployable to many environments, including standard Linux-based Docker containers, Windows environments, and resource-constrained embedded environments. To this end, mlpack's build system now also contains a number of tools for cross-compilation support, including the ability to easily statically link compiled programs (important for some deployment environments).

Major Changes

Below we detail a few of the major changes present in mlpack 4. For a complete and exhaustive list (including numerous bug fixes and new techniques), the `HISTORY.md` file (distributed with mlpack) can be consulted.

Removed dependencies. In accordance with the vision document (mlpack community, 2021), the majority of the refactoring and redesign work focused on reducing dependencies and compilation overhead. This has motivated the replacement of the Boost C++ libraries, upon which mlpack previously depended, with lightweight alternatives including cereal for serialization. The entire neural network module was refactored to avoid the use of Boost (amounting to an almost complete rewrite). This effort was rewarded handsomely: with mlpack 3, a simple program would often require several gigabytes of memory just for compilation. After refactoring and removing dependencies, compilation generally requires just a few hundred megabytes of memory, and is often an order of magnitude faster.

Interactive notebook environments. mlpack can be used in a Jupyter notebook environment (Kluyver et al., 2016) via the xeus-cling project. This is demonstrated interactively on the mlpack homepage. Examples of C++ notebooks can be found in the mlpack examples repository, and these can easily be run on BinderHub.

New bindings and enhanced availability. Support for the Julia (Bezanson et al., 2017), Go (Pike, 2012), and R languages (R Core Team, 2022; Singh Parihar et al., 2022) has been added via mlpack's automatic binding system. These bindings can be used by installing mlpack from the language's package manager (`Pkg.jl`, `go get`, `install.packages('mlpack')`). Furthermore, since mlpack's reduced dependency footprint has significantly simplified the deployment process, mlpack's Python dependencies are now available for numerous architectures both on PyPI and in `conda-forge`.

Cross-compilation support and build system improvements. mlpack's build configuration now supports easy cross-compilation, for instance via toolchains such as buildroot. By specifying a few flags, a user may produce a working mlpack setup for a variety of embedded systems. This required the implementation of a dependency auto-downloader, which is capable of downloading OpenBLAS and compiling (if necessary) for the target architecture. The auto-downloader can also be enabled and used for any situation, thus easing installation and deployment.

Acknowledgements

Development of mlpack is community-led. It is the product of hard work by over 220 individuals (at the time of writing). We are also indebted to people that have provided bug reports over the years. The development has been supported by Google, via a decade-long participation the Google Summer of Code program, and also by NumFOCUS, which fiscally sponsors mlpack.

References

Alexandrescu, A. (2001). *Modern C++ Design: Generic programming and design patterns applied*. Addison-Wesley.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98.

Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., & Zdeborová, L. (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, *91*(4), 045002.

Curtin, R. R. (2017). A dual-tree algorithm for fast k-means clustering with large k. *SIAM International Conference on Data Mining (SDM '17)*, 300–308.

Curtin, R. R., Cline, J. R., Slagle, N. P., March, W. B., Ram, P., Mehta, N. A., & Gray, A. G. (2013). MLPACK: A scalable C++ machine learning library. *The Journal of Machine Learning Research*, *14*(1), 801–805.

Curtin, R. R., Edel, M., Lozhnikov, M., Mentekidis, Y., Ghaisas, S., & Zhang, S. (2018). Mlpack 3: A fast, flexible machine learning library. *Journal of Open Source Software*, *3*(26), 726.

Curtin, R. R., Edel, M., Prabhu, R. G., Basak, S., Lou, Z., & Sanderson, C. (2021). The ensmallen library for flexible numerical optimization. *Journal of Machine Learning Research*, *22*, 1–6.

Curtin, R. R., & Gardner, A. B. (2016). Fast approximate furthest neighbors with data-dependent candidate selection. *International Conference on Similarity Search and Applications (SISAP 2016)*, 221–235.

Curtin, R. R., & Ram, P. (2014). Dual-tree fast exact max-kernel search. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *7*(4), 229–253.

Edel, M., Soni, A., & Curtin, R. R. (2014). An automatic benchmarking system. *NIPS Workshop on Software Engineering for Machine Learning*.

Fang, D., & Chau, D. H. (2016). M3: Scaling up machine learning via memory mapping. *International Conference on Management of Data*, 2249–2250.

Fischer, L., Ehrlinger, L., Geist, V., Ramler, R., Sobiezky, F., Zellinger, W., Brunner, D., Kumar, M., & Moser, B. (2020). AI System Engineering—Key Challenges and Lessons Learned. *Machine Learning and Knowledge Extraction*, *3*(1), 56–83.

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, *7*(3), 535–547.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, *10*, 1755–1758.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., & others. (2016). Jupyter Notebooks—a publishing format for reproducible computational workflows. *Proceedings of the 20th International Conference on Electronic Publishing*, 87–90.

Lavin, A., Gilligan-Lee, C. M., Visnjic, A., Ganju, S., Newman, D., Ganguly, S., Lange, D., Baydin, A. G., Sharma, A., Gibson, A., & others. (2022). Technology readiness levels for machine learning systems. *Nature Communications*, *13*(1), 1–19.

Lee, D., & Gray, A. G. (2008). Fast high-dimensional kernel summations using the Monte Carlo Multipole Method. *Advances in Neural Information Processing Systems*, *21*, 929–936.

mlpack community. (2021). *mlpack: A vision for an efficient prototype-to-deployment machine learning library*. https://www.mlpack.org/papers/vision.pdf.

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *Proceedings of the 2009 International Conference on Computer Vision Theory and Applications (VISAPP '09)*, 331–340.

Paleyes, A., Urma, R.-G., & Lawrence, N. D. (2020). Challenges in deploying machine learning: A survey of case studies. *ACM Computing Surveys (CSUR)*.

Pike, R. (2012). Go at Google: Language design in the service of software engineering. *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, 5–6.

R Core Team. (2022). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Sanderson, C., & Curtin, R. R. (2016). Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, *1*(2), 26.

Singh Parihar, Y., Curtin, R. R., Eddelbuettel, D., Balamuta, J., & others. (2022). *mlpack: "Rcpp" integration for the "mlpack" library*. https://CRAN.R-project.org/package=mlpack