mlpack open-source machine learning library and community

Marcus Edel
Institute of Computer Science,
Free University of Berlin
marcus.edel@fu-berlin.de

Abstract

mlpack is an open-source C++ machine learning library with an emphasis on speed and flexibility. Since its original inception in 2007, it has grown to be a large project implementing a wide variety of machine learning algorithms. This short paper describes the general design principles and discusses how the open- source community around the library functions.

1 Introduction

All modern machine learning research makes use of software in some way. Machine learning as a field has thus long supported the development of software tools for machine learning tasks, such as scripts that enable individual scientific research, software packages for small collaborations, and data processing pipelines for evaluation and validation. Some software packages are, or were, supported by large institutions and are intended for a wide range of users. Other packages like the `mlpack` machine learning library are developed by individual researchers or research groups and are typically used by smaller groups for more domain-specific purposes; and highly benefits from a community and ecosystem built around a shared foundation.

`mlpack` is a flexible and fast machine learning library, is free to use under the terms of the BSD open-source license, and accepts contributions from anyone. Since 2007, the project has grown to have nearly 120 individual contributors, has been downloaded over 50,000 times and is being used all around the world and in space as well [7].

A large part of this success owes itself to the vibrant community of developers and a continuously-growing ecosystem of tools, web services, stable well- developed packages that enable easier collaboration on software development, continuous testing and its participation in open-source initiatives such as Google Summer of Code (`https://summerofcode.withgoogle.com/`).

This short document aims to discuss the design of mlpack, as well as how the open-source community functions and growth. Section 2 discusses the general design principles of mlpack, providing an overview of its functionality. Section 3 discusses the open-source community of mlpack and Section 4 its participation in programs such as Google Summer of Code, and how someone new to the library might get involved and contribute. Lastly, Section 5 briefly discusses the future planned functionality and directions of mlpack.

2 Description of mlpack

`mlpack` is a flexible and fast machine learning library written in C++ that has bindings that allow use from the command-line and from Python, with support for other languages in active development. The `mlpack` library, which is meant to be the machine learning analog to the general-purpose LAPACK linear algebra library, strives to balance four overarching goals:

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

- implement fast, scalable machine learning algorithms;
- design an intuitive, consistent, and simple API for non-expert users;
- implement a variety of machine learning methods; and
- provide cutting-edge machine learning algorithms unavailable elsewhere.

In order to provide efficient implementations, `mlpack` uses C++ template metaprogramming to optimize mathematical expressions when the program is compiled. In addition, mlpack uses the fast Armadillo linear algebra library as it's core [9]. Inside of mlpack, a breadth of machine learning algorithms are available. A basic list is given below, but note that the list is not exhaustive, and to find an up-to-date list one should consult the mlpack website `http://www.mlpack.org/` or a recent paper detailing the design and structure of mlpack [3].

- **Classification:** logistic regression, softmax regression, deep neural networks, AdaBoost, perceptrons, Naive Bayes classifier, decision trees, decision stumps, Hoeffding trees
- **Regression:** linear regression, ridge regression, LARS, LASSO
- **Clustering:** k-means, DBSCAN, minimum spanning tree calculation
- **Data transformations:** PCA, kernel PCA, sparse coding, local coordinate coding, RADICAL, NMF
- **Distance-based tasks:** (approximate and exact) nearest neighbor search, furthest neighbor search, range search, max-kernel search, locality sensitive hashing
- **Optimization algorithms:** L-BFGS, SGD, simulated annealing, AdaGrad, ADAM, RMSprop, AdaMax, augmented Lagrangian solver, Parallel SGD (Hogwild!)
- Other: collaborative filtering/recommender systems, data preprocessing tools, neighborhood components analysis, SVD matrix decompositions, density estimation trees

3 Community description

Central to the success of `mlpack` is an open environment where anybody can contribute to the project. This model leads to an "organic" growth, where features are implemented by different people with different programming styles and interfaces. Thus, all development of mlpack is public and is primarily conducted on Github (`https://github.com/mlpack/mlpack/`) and on IRC (internet relay chat). Github is a centralized open-source development website that allows any user to contribute easily to some open-source software project. The website can be used to do many things—a user can ask for help using `mlpack`, report a bug inside of `mlpack`, submit new code, discuss ideas with `mlpack` maintainers, update or add documentation, or run automatic tests on `mlpack` code, among other things. Thanks to Github's standard interface and wide adoption inside the open-source community, this means that it is quite simple and straightforward for anyone interested in `mlpack` to perform any of these tasks.

Code is contributed to the `mlpack` package or modified through "pull requests" (via GitHub) that often contain several git commits. Pull requests may fix bugs, implement new features, or improve or modify the infrastructure that supports the development and maintenance of the package. Individual pull requests are generally limited to a single conceptual addition or modification to make code review tractable.

All Pull requests that modify or add code must be reviewed and approved by `mlpack` maintainers to ensure that the code meets the `mlpack` standards before they are merged into the codebase.

These standards include simple requirements like formatting and more complex standards like following the standardized APIs [3]. In addition, any new code must be submitted with tests, so that it can be known that the new code functions properly.

Typically the review process will include several comments from maintainers that need to be addressed by the contributor(s). When an agreement is reached between the maintainers and the contributors that the code is ready, then the code can be merged into the codebase. Since the project is careful to accept only high- quality code, this process can typically take several days to several weeks, depending on the complexity and quality of the original submission. An example of this process can be seen at `https://github.com/mlpack/mlpack/pull/1018`.

Because the Github pull request interface and process is standardized across all Github projects, it is straightforward for new contributors to get involved with `mlpack` and start getting their contributions

merged into the code. Since the mlpack project moved to Github in the fall of 2014, the number of contributors to the project has grown more rapidly than before.

As of version 3.0, mlpack contains 306130 lines of code, contributed by 116 unique contributors from around the world with over 19300 git commits.

4 Community Experiences

Another factor contributing to the growth of the project is mlpack's participation in the Google Summer of Code program (https://summerofcode.withgoogle.com); a global program that matches students up with open source, free software and technology-related organizations. The organizations provide mentors who act as guides through the entire process, from learning about the community to contributing code. With the goal of getting more students involved in and familiar with the open source community.

mlpack has been a part of Google Summer of Code (GSoC) for many years now, and each year it has been of tremendous benefit to our organization and community. Because of GSoC, many of us have been able to meet each other in person, develop lasting friendships, publish papers in academic conferences, and improve the functionality and quality of the library.

A the beginning of GSoC mlpack selects a number of promising students for acceptance into the program. These students each prepare a proposal for a project that they will complete over the course of the summer, with a detailed description and timeline. Those students who are selected are paid a stipend and a t-shirt by Google to work full-time on their project over the summer, and usually, at the end of the summer their work will be incorporated into the library. For mlpack, Summer of Code is an excellent teaching opportunity and allows us to work with incredibly talented students from all over the world.

Examples of previous improvements include an automatic benchmarking system [6], a collaborative filtering framework [1], implementation of different tree types [8], addition of new deep learning techniques [2, 4, 5], and others. Since 2013, mlpack has mentored 30 students from around the world.

The most important part of GSoC is the community around the project. This is why when students "join" the community before GSoC, we encourage them to join the mailing lists, introduce themselves, and join the IRC channel. IRC discussion is certainly not limited to technical discussion, and friendly banter is very helpful in building relationships. At the outset of GSoC, we hold a yearly IRC meeting to introduce students to the community, set expectations, provide some trivia about our organization, and so forth. Then, during the summer, we strongly encourage all communication to be public, either on the mailing lists or in the IRC channel.

We also encourage applicants to discuss their proposals with their potential mentors. This ensures that there is a clear understanding and interest in the project from both parties. Additionally, our mentors are required to get at least a weekly update of the student's progress and upcoming issues. If the mentor does not hear from the student, the mentor is responsible for contacting the student and inquiring what is leading to the lack of communication. The mentor will then, if necessary, try to work with the student to guarantee the success of the project. At the end, the mentors do their best to engage students and provide the most convenient atmosphere for the work on the project. This has been very effective in the past.

Also, during the last years, we have discovered that a major factor that influences whether or not a student will stay involved in the community is the personal ties developed during the program. To encourage a tighter connection with the community, we encourage our students to post questions and submit work in the same public forums that the rest of the community uses. This enables other members of our team to interact with the participants and increases the sense of community which is central to open source projects. We have had several students over the last 4 years who have continued development after the end of GSoC; also a lot of projects that GSoC students do can lead to further interesting work and in many cases, academic publications. We encourage our students to develop their work further if able and try to keep in touch by tagging them in relevant Github issues or inviting them to help with other work or papers. Most successful GSoC students are then asked if they want to be a mentor the following year. This seems to work well, this year, of our 5 mentors, 4 were previous GSoC students for mlpack.

5 Conclusion and Future Goals

We have introduced the open-source mlpack C++ machine learning library and discussed its general philosophy, as well as its open-source development and community building process. However, mlpack is under active development, so the library is far from complete or static. New versions are released typically every few months with enhanced functionality, bugfixes, and efficiency improvements. The next major release, that will include several projects from the Google Summer of Code program, is currently planned for late in 2018 with the following major functionality improvements and changes:

- **Unsupervised machine learning:** Generative adversarial networks, Variational Autoencoder
- **Collaborative filtering:** Neural Collaborative Filtering, Neighborhood-Based Collaborative Filtering
- **Metric learning:** Large margin nearest neighbor
- **Bindings:** Go, Julia

This list, of course, is non-comprehensive; for a more clear list of goals, the mlpack website (`http://www.mlpack.org/`) and the Github website (`https://github.com/mlpack/mlpack`) have more details both about the future goals of the project and the current status of the project.

References

[1] S. Agrawal, R.R. Curtin, S.K. Ghaisas, and M.R. Gupta. Collaborative filtering via matrix decomposition in mlpack. In *Proceedings of the ICML 2015 Workshop on Machine Learning Open Source Software*, 2015.

[2] Marcin Andrychowicz and Karol Kurach. Learning efficient algorithms with hierarchical attentive memory. *CoRR*, abs/1602.03218, 2016.

[3] Ryan R. Curtin and Marcus Edel. Designing and building the mlpack open-source machine learning library. *arXiv preprint arXiv:1708.05279*, 2017.

[4] Ryan R. Curtin, Marcus Edel, Mikhail Lozhnikov, Yannis Mentekidis, Sumedh Ghaisas, and Shangtong Zhang. mlpack 3: a fast, flexible machine learning library. *Journal of Open Source Software*, 3:726, 2018.

[5] M. Edel and J. Lausch. Capacity visual attention networks. In Christoph Benzmüller, Raul Rojas, and Geoff Sutcliffe, editors, *Second Global Conference on Artificial Intelligence (GCAI), Proceeedings*, pages 72–80, 2016.

[6] M. Edel, A. Soni, and Curtin R.R. An automatic benchmarking system. In *NIPS 2014 Workshop on Software Engineering for Machine Learning*, 2014.

[7] T. M. Hackett, S. G. Bilén, P. V. R. Ferreira, A. M. Wyglinski, and R. C. Reinhart. Implementation of a space communications cognitive engine. In *2017 Cognitive Communications for Aerospace Applications Workshop (CCAA)*, pages 1–7, June 2017.

[8] James McNames. A fast nearest-neighbor algorithm based on a principal axis search tree. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(9):964–976, September 2001.

[9] C. Sanderson and R.R. Curtin. Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1:1–2, 2016.