Workshop contribution MLJ

Diego Arenas
School of Computer Science
University of St Andrews
KY16 9SX, Scotland, UK
da60@st-andrews.ac.uk

Edoardo Barp
Centre for Complexity Science
University of Warwick
Coventry, United Kingdom
e.g.barp@warwick.ac.uk

Gergö Bohner
Mathematics Institute and Department of Statistics
University of Warwick
Warwick, United Kingdom
Gergo.Bohner@warwick.ac.uk

Valentin Churvay
Massachusetts Institute of Technology
Cambridge, MA
vchuravy@MIT.EDU

Franz Kiraly
Department of Statistical Science
University College London
WC1E 6BT, London, United Kingdom
f.kiraly@ucl.ac.uk

Thibaut Lienart
Department of Statistics
University of Oxford
Oxford, United Kingdom
lienart@stats.ox.ac.uk

Sebastian Vollmer
Mathematics Institute and Department of Statistics
University of Warwick
Warwick, United Kingdom
svollmer@turing.ac.uk

Mike Innes
Julia Computing, Inc.
mike.j.innes@gmail.com

Bernd Bischl
Department of Statistics
LMU
Ludwigstrasse 33, D80539 Munich
bernd.bischl@stat.uni-muenchen.de

Abstract

MLJ[1] is a machine learning toolbox for Julia with the aim to provide a coherent interface to run and compare machine learning models. The software should support basic supervised learning framework implementing key parts of workflow for most important modelling pipelines including tuned, ensembles, and deep learning/neural networks (via interfacing). Further development will extend to cover advanced ML tasks and workflows. This project is building an MLR type machine learning toolbox in Julia.

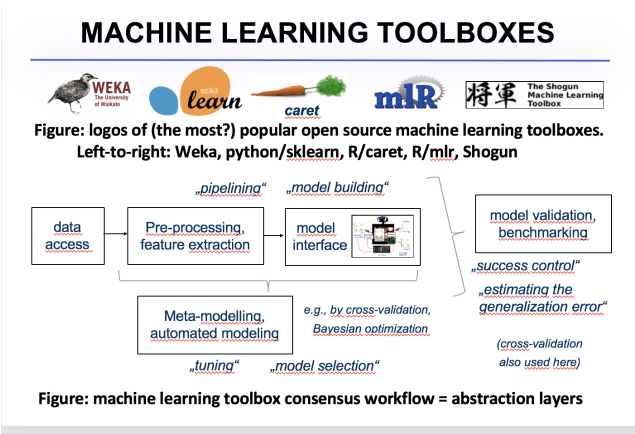
Introduction

The challenge of bringing projects from research to real-world impact is spinning out of control. Ideas need to reach clusters and supercomputers for large-scale data, be deployed to the cloud for

[1] https://github.com/alan-turing-institute/mlj

Preprint. Work in progress.

Figure 1



real-time analysis and be built on by other industrial and academic projects. Ad-hoc solutions by fragmented research groups using different languages create sprawling codebases and myriad tools that impair collaboration and innovation. Meanwhile, the well-oiled machines of Google (Deepmind), Facebook and others loom large over academic initiatives.

The Julia language offers a compelling solution: A single system (no need for specialist wrapping C++ code - single code base for end-user) that can meet all of these demands at once, cutting down the innovation cycle by months or years.

Julia can easily tie into pre-existing solution written in other languages with interfaces to C, Fortran, C++, Python, R, and Matlab as well as HTTP, Database, and ZeroMQ libraries and vice versa. This allows industrial adopters to gradually phase Julia into large complex software applications, without requiring significant, up-front development costs.

Julia has been taken up by scientists since it allows researchers to focus on the mathematical and algorithmic details while reducing implementation time. Examples impacting fundamental research include:

- Mixed mode automatic differentiation (Jarrett Revels, MIT)
- JuMP (Ian Dunning and Miles Lubin, MIT), mathematical optimizations
- DifferentialEquations.jl (Chris Rackauckas, UC Irvine)
- Flux a state of the art deep learning library.

The ML landscape outside deep learning is very fragmented in Julia.

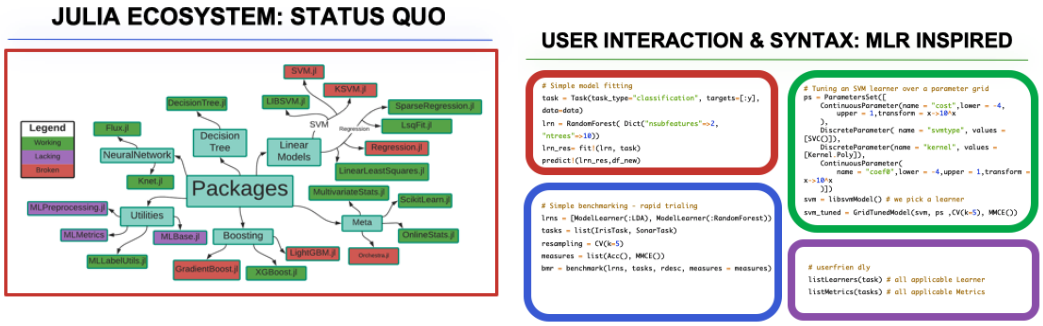
In contrast to the well established ML tools boxes such as SkLearn and MLR. see **Figure 1**. Many packages have core functionality missing, and it is hard for a practitioner to do an out of the box analysis **Figure 2 (a)**.

2. MLJ This project aims to fill this gap by connecting third-party projects through wrappers, benchmarking, workflows. We are aiming for it to be practical and useful in a short time - by working closely with the user community.

The steps this project has taken are as follows

1. Four MSc students the University of Warwick - reviewing and contributing too many existing packages, **Figure 2**. Julia-Machine-Learning-Review.
2. One of the students created a prototype MLJ-proto
3. A summer project lead to its current form `https://github.com/alan-turing-institute/mlj`

The current status of the project allows to interface with machine libraries written in Julia by creating a wrapper file. A `fit()` and `predict()` functions must be implemented in the wrapper file. These



(a) Review of extant machine learning functionality and meta-functionality "Meta" category = existing ML toolbox meta-packages in Julia.

(b) Example of user interaction and syntax of MLJ.

Figure 2: A figure with two subfigures

functions will call and interact with the inner functions of the interfaced library. MLJ has implemented a basic grid search functionality for hyper-tuning. The grid search function handles discrete and continuous parameters. **Figure 2 (b)** illustrates user experience.

3. Road map For the next six months.

1. **API** = simple and transparent description of ML job.
   (a) *Directions*→ enrich the descriptive possibility.
2. **Model fitting Interface** = interacting with the many (potentially primitive) Julia libraries that fit specific models (e.g. DecisionTree.jl, SparseRegression.jl, . . .) as well as consider general backend (e.g. Optim.jl for arbitrary linear regression with composite penalty etc).
   (a) *Directions* → enrich the interface + drive improvement of solvers.
3. **Model evaluation and metrics**
   (a) *Possible contributions* → add standard evaluation + metrics.
   (b) *Possible contributions* → use automatic differentiation libraries for hyperparameter tuning.
4. **Scheduling** = for a given level, build a list of jobs.
   (a) *Possible contributions* → enrich how copies of models are generated for a CV setting
   (b) *Possible contributions* → how to deal with models that require some form of ordering (e.g. pre-processing before an ensemble model)
   (c) *Possible contributions* → have a way to estimate roughly the computational effort for given tasks so that some primitive load balancing can be done.

4. Further

- **Integration with OpenML.org**
- **Preprocessing, hyperparameter optimisation** ⇒ **scheduling** = for non-trivial preprocessing and hp optimisation schemes, some models need to be trained first then new hyperparameters generated, this introduces dependencies, and a dependency graph should be considered (e.g. via Dagger.jl)
- **Dealing with arbitrary architectures** = if we have access to both a variety of CPUs as well as GPUs then potentially some compute nodes are faster than others, and this should be taken into account. Also, not all tasks can be done everywhere.
- **Fault tolerance** = what if one of the job fails?
- **Privacy issues** = can be built differential privacy - multiparty computations.

References

[1] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, & Zachary M. Jones. mlr: Machine Learning in R. *Journal of Machine Learning Research*. **17**(170):1-5.

[2] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. SIGKDD Explorations 15(2), pp 49-60, 2013

[3] Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.