RcppMLPACK: R integration with MLPACK using Rcpp

Qiang Kou
qkou@umail.iu.edu

April 20, 2020

1 MLPACK

MLPACK[1] is a C++ machine learning library with emphasis on scalability, speed, and ease-of-use. It outperforms competing machine learning libraries by large margins, for detailed results of benchmarking, please refer to Ref[1].

1.1 Input/Output using Armadillo

MLPACK uses Armadillo as input and output, which provides vector, matrix and cube types (integer, floating point and complex numbers are supported)[2]. By providing Matlab-like syntax and various matrix decompositions through integration with LAPACK or other high performance libraries (such as Intel MKL, AMD ACML, or OpenBLAS), Armadillo keeps a good balance between speed and ease of use.

Armadillo is widely used in C++ libraries like MLPACK. However, there is one point we need to pay attention to: Armadillo matrices in MLPACK are stored in a column-major format for speed. That means observations are stored as columns and dimensions as rows. So when using MLPACK, additional transpose may be needed.

1.2 Simple API

Although MLPACK relies on template techniques in C++, it provides an intuitive and simple API. For the standard usage of methods, default arguments are provided.

The MLPACK paper[1] provides a good example: standard k-means clustering in Euclidean space can be initialized like below:

```
KMeans<> k();
```

Listing 1: k-means with all default arguments

If we want to use Manhattan distance, a different cluster initialization policy, and allow empty clusters, the object can be initialized with additional arguments:

```
KMeans<ManhattanDistance, KMeansPlusPlusInitialization, AllowEmptyClusters> k();
```

Listing 2: k-means with additional arguments

1.3 Available methods in MLPACK

Commonly used machine learning methods are all implemented in MLPACK. Besides, it contains a strong set of tree-building and tree-query routines.

Available methods in MLPACK:

- 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
- Neighbourhood Components Analysis (using LRSDP)
- RADICAL (Robust, Accurate, Direct ICA aLgorithm)
- Tree-based k-nearest-neighbours search and classifier
- Tree-based range search

For the details of each algorithm and usage, please visit the tutorial page of MLPACK (`http://www.mlpack.org/tutorial.html`).

2 RcppMLPACK

2.1 Rcpp and RcppArmadillo

As said above, MLPACK is a C++ library using Armadillo. Since we **Rcpp**[3] and **RcppArmadillo**[4], which can used to integrate C++ and Armadillo with R seamlessly, **RcppMLPACK** becomes something very natural.

2.2 k-means example

Here we continue the k-means example above. By using **RcppMLPACK**, a k-means method which can be called by R can be implemented like Listing 3. The interfere between R and C++ is handled by **Rcpp** and **RcppArmadillo**.

```cpp
#include "RcppMLPACK.h"

using namespace mlpack::kmeans;
using namespace Rcpp;

// [[Rcpp::export]]
List kmeans(const arma::mat& data, const int& clusters) {

    arma::Col<size_t> assignments;

    // Initialize with the default arguments.
    KMeans<> k;

    k.Cluster(data, clusters, assignments);

    return List::create(_["clusters"] = clusters,
                        _["result"]   = assignments);
}
```

Listing 3: k-means example

2.3 Using inline package

inline[5] package provides a complete wrapper around the compilation, linking, and loading steps. So all the steps can be done in an R session. There is no reason that **RcppMLPACK** doesn't support the inline compilation.

```cpp
library(inline)
library(RcppMLPACK)
code <- '
  arma::mat data = as<arma::mat>(test);
  int clusters = as<int>(n);
  arma::Col<size_t> assignments;
  mlpack::kmeans::KMeans<> k;
  k.Cluster(data, clusters, assignments);
  return List::create(_["clusters"] = clusters,
                      _["result"]   = assignments);
'
mlKmeans <- cxxfunction(signature(test="numeric", n ="integer"), code,
                        plugin="RcppMLPACK")
data(trees, package="datasets")
mlKmeans(t(trees), 3)
```

Listing 4: k-means example, inline version

As said above, MLPACK uses a column-major format of matrix, so when we pass data from R to MLPACK, a transpose may be needed.

2.4 RcppMLPACK.package.skeleton

The package also contains a `RcppMLPACK.package.skeleton()` function for people who want to use MLPACK code in their own package. It follows the structure of `RcppArmadillo.package.skeleton()`.

```
library(RcppMLPACK)
RcppMLPACK.package.skeleton("foobar")
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './foobar/Read-and-delete-me'.

Adding RcppMLPACK settings
 >> added Imports: Rcpp
 >> added LinkingTo: Rcpp, RcppArmadillo, BH, RcppMLPACK
 >> added useDynLib and importFrom directives to NAMESPACE
 >> added Makevars file with RcppMLPACK settings
 >> added Makevars.win file with RcppMLPACK settings
 >> added example src file using MLPACK classes
 >> invoked Rcpp::compileAttributes to create wrappers
```

Listing 5: RcppMLPACK.package.skeleton

By using `RcppMLPACK.package.skeleton()`, a package skeleton is generated and files are list below.

```
system("ls -R foobar")
foobar:
DESCRIPTION  man  NAMESPACE  R  Read-and-delete-me  src

foobar/man:
foobar-package.Rd

foobar/R:
RcppExports.R

foobar/src:
kmeans.cpp  Makevars  Makevars.win  RcppExports.cpp
```

Listing 6: RcppMLPACK.package.skeleton result

2.5 Processing files on disk

MLPACK contains two functions, `mlpack::data::Load()` and `mlpack::data::Save()`. They are used into load and save matrix from files. These can be really useful, since reading large files into R can be a nightmare. We can just pass the file names from R to C++, not read them first.

```
std::string inputFile = Rcpp::as<std::string>(input);
std::string outputFile = Rcpp::as<std::string>(output);
arma::mat data;
mlpack::data::Load(inputFile, data, true);
int clusters = as<int>(n);
arma::Col<size_t> assignments;
mlpack::kmeans::KMeans<> k;
k.Cluster(data, clusters, assignments);
arma::Mat<size_t> out = trans(assignments);
mlpack::data::Save(outputFile, out);
```

Listing 7: Reading and writing files on disk

2.6 Performance

Even without a performance testing, we are still sure the C++ implementations should be faster. A small wine data set from UCI data sets repository is used for benchmarking, you can download it from `https://archive.ics.uci.edu/ml/datasets/Wine`. **rbenchmark**[6] package is also used.

```
suppressMessages(library(rbenchmark))
res <- benchmark(mlKmeans(t(wine),3),
                 kmeans(wine,3),
                 columns=c("test", "replications", "elapsed",
                 "relative", "user.self", "sys.self"), order="relative")
```

Listing 8: Benchmarking script

Table 1: Benchmarking result

test	replications	elapsed	relative	user.self	sys.self
mlKmeans(t(wine), 3)	100	0.028	1.000	0.028	0.000
kmeans(wine, 3)	100	0.947	33.821	0.484	0.424

From benchmarking result, we can see that MLPACK version of k-means is 33-time faster than `kmeans()` in R. However, we should note that R returns more information than the clustering result and there are much more checking functions in R.

Acknowledgement

Very special thank you to developers of Rcpp and MLPACK. Also huge thanks to people on Rcpp-devel for the help. Testing and bugs reports are deeply welcome. If you have any questions, you can always find me by email (`qkou@umail.iu.edu`), or open issues on `https://github.com/thirdwing/RcppMLPACK`.

References

[1] Ryan Curtin, James Cline, Neil Slagle, William March, Parikshit Ram, Nishant Mehta, and Alexander Gray. MLPACK: A scalable C++ machine learning library. *The Journal of Machine Learning Research*, 14(1):801–805, 2013.

[2] Conrad Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

[3] Dirk Eddelbuettel and Romain Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 4 2011.

[4] Dirk Eddelbuettel and Conrad Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics & Data Analysis*, 71:1054–1063, 2014.

[5] Oleg Sklyar, Duncan Murdoch, Mike Smith, Dirk Eddelbuettel, and Romain François. *inline: inline C, C++, Fortran function calls from R*, 2013. R package version 0.3.13.

[6] Wacek Kusnierczyk. *rbenchmark: Benchmarking routine for R*, 2012. R package version 1.0.

A Modifications on original MLPACK library

To avoid the maintenance tasks, we try to minimize the modification on MLPACK. However, for the integration and pass `R CMD check`, there are some changes we have to make. There is no changes in the methods, all modifications are in utility classes and done by a script.

`Log` class

`Log` class provides four levels of log information. The logging functions are replaced with `Rcpp::Rcout` to redirect the output stream to R session.

`cli` class

`cli` class is used to parse the command line arguments. It relies on `boost::program_options` which requires additional linking. Since all arguments are passed from R, this class is removed.

`SaveRestoreUtility` class

`SaveRestoreUtility` class is used to store and restore MLPACK models from XML files. This class has been removed to avoid additional linking to libxml2.

`Timer` class

In many methods, `Timer` class is used for timing and print information into log. This class has been commented.