An Introduction to **Rclusterpp**

Michael Linderman

Rclusterpp version 0.1.3 as of December 6, 2011

Abstract

The **Rclusterpp** package provides alternative implementations for common geometric hierarchical clustering algorithms, e.g., average link clustering, that are optimized for large numbers of observations and efficient execution on modern multicore processors. **Rclusterpp** can be used directly from R as a replacement for `stats::hclust`, or as a linkable C++ library.

1 Introduction

Hierarchical clustering is a fundamental data analysis tool. However, the $O(n^2)$ memory footprint of commonly available implementations, such as `stats::hclust`, which maintain the dissimilarity matrix in memory (stored-distance) limit these implementations to tens of thousands of observations or less. In the motivating domain for this work, flow cytometry, datasets are hundreds of thousands or even millions of observations in size (but with low dimensionality, e.g., less than 30). In this and other similar contexts building out the complete distance matrix is not possible and alternative implementations with $O(n)$ memory footprint are needed.

The memory requirements of hierarchical clustering have motivated the development of alternative clustering algorithms that do not require the full dissimilarity matrix. Such algorithms are not the focus of **Rclusterpp**. Often, a much larger application is built and validated around a standard hierarchical clustering algorithm, e.g. average-link, and only later scaled to large datasets. In these cases, we wish to maintain the same algorithm but scale efficiently. Thus the goal for **Rclusterpp** is to provide efficient "stored data" implementations for common hierarchical clustering routines, e.g., single, complex, average and Ward's linkage, that scale to hundreds of thousands of observations while delivering identical results as the "stock" `stats::hclust` implementation.

As an example, the following two statements produce identical results:

```
> h <- hclust(dist(USArrests, method="euclidean"), method="average")
> r <- Rclusterpp.hclust(USArrests, method="average", distance="euclidean")
> # Check equality of the dedrogram tree and agglomeration heights
> identical(h$merge, r$merge) && all.equal(h$height, r$height)

[1] TRUE
```

however, in the latter, the memory footprint is on the order of $O(n)$ as opposed to $O(n^2)$, for n observations (ignoring the footprint of the data itself). The trade-off can but does not have to be increased computation. In such cases, **Rclusterpp** purposely trades time for space. Fortunately there exist algorithms for some commonly used clustering methods, specifically Ward's minimum variance method and single-link, that achieve optimal time complexity with only $O(n)$ space. **Rclusterpp** implements those more efficient algorithms when possible. Section 2 includes a summary of the complexity of each linkage method as implemented.

The computational demanding components of **Rclusterpp** are implemented in C++ using OpenMP[1] to take advantage of multi-core processors and multi-processor shared memory computers. Thus even when incurring additional computation costs to reduce the memory footprint **Rclusterpp** is faster than `stats::hclust`, and in cases, such as Ward's linkage, where no such trade-off exists, **Rclusterpp** can be faster than even the "fast" stored-distance clustering packages like **fastcluster**. Sample benchmark results are shown in Table 1.

[1]OpenMP is only enabled on Linux and OSX due to issues with the pthreads compatibility DLL on Windows

Table 1: Execution time averaged across 5 runs for various clustering implementations (including distance computation) for Ward's minimum variance method for $n \times 10$ input data measured on a quad-core 3.05 GHz Intel i7 950 server

Implementation	Exec. Time (s)	n
Rclusterpp	0.0006	
fastcluster	0.0008	100
hclust	0.0028	
Rclusterpp	0.0036	
fastcluster	0.0208	500
hclust	0.2606	
Rclusterpp	0.0092	
fastcluster	0.1252	1000
hclust	1.8012	
Rclusterpp	0.1814	
fastcluster	2.6246	5000
hclust	199.1894	

Table 2: Worst-case time and space complexities for the **Rclusterpp** stored-data implementation (not including the original $O(n * m)$ data footprint)

Method	Algorithm	Time Complexity	Space Complexity
Average	RNN	$O(n^3 * m)$	$O(n)$
Complete	RNN	$O(n^3 * m)$	$O(n)$
Ward	RNN	$O(n^2 * m)$	$O(n * m)$
Single	SLINK	$O(n^2 * m)$	$O(n)$

In some applications, such as the WGNCA [4] algorithm that also motivated this work, the distance matrix is already computed in a previous stage of the workflow and thus there is no advantage to be gained with stored-data approaches. However, memory footprint is still a concern. Those individuals who have attempted to cluster more than 46340 observations have discovered that R limits matrices to 2^31 elements or less. In these cases, it is desirable to perform all of the memory intensive components of the workflow on the "C++ side", where there is no such limit and where the implementor has more control over the creation of temporaries. **Rclusterpp** exposes its various clustering implementations as a templated library that can be linked against by other C++-backed R packages (modeled on the techniques used in the **Rcpp** package).

2 Stored-data Hierarchical Clustering

Rclusterpp currently implements a subset of the clustering methods and distance metrics provided by `stats::hclust`. Specifically, **Rclusterpp** currently supports the following linkage methods:

```
> Rclusterpp.linkageKinds()

[1] "ward"     "average"  "single"   "complete"
```

and the following distance metrics:

```
> Rclusterpp.distanceKinds()

[1] "euclidean" "manhattan" "maximum"   "minkowski"
```

The linkage methods are currently limited to reducible geometric methods that can implemented exactly using the *recursive nearest neighbor (RNN)* algorithm [1].

Table 2 shows the estimated worst-case time and space complexities [2] for the algorithms used in **Rclusterpp**. As described previously, Ward's and single-link are implemented with optimal time and space using RNN and SLINK [3] algorithms respectively. While average and complete-link trade increased time bounds, in exchange for reducing the memory footprint to $O(n)$ from $O(n^2)$.

Table 3: Execution time averaged across 5 runs for various clustering implementations (including distance computation) for average-link/euclidean distance clustering on $n \times 10$ input data measured on a quad-core 3.05 GHz Intel i7 950 server. `RclusterppDistance` is the stored-distance implementation and `Rclusterpp` is the stored data implementation.

Implementation	Exec. Time (s)	n
fastcluster	0.0008	
RclusterppDistance	0.0014	100
Rclusterpp	0.0014	
hclust	0.0028	
fastcluster	0.0182	
RclusterppDistance	0.0220	500
Rclusterpp	0.0376	
hclust	0.2354	
fastcluster	0.1306	
RclusterppDistance	0.1364	1000
Rclusterpp	0.1508	
hclust	1.7396	
fastcluster	2.4642	
RclusterppDistance	2.8008	5000
Rclusterpp	5.5344	
hclust	204.2906	

As shown previously, the `Rclusterpp.hclust` function has a very similar interface to `stats::hclust`, but will also accept a numeric matrix (instead of a `dist` object) and a distance metric. The return value is the same `hclust` object as produced by `stats::hclust`.

Since the underlying components of the clustering implementation, including the RNN implementation, linkage methods and distance functions, are all exposed as a templated C++ library, users can readily create derivative packages that implement custom clustering methodologies without starting from scratch. Section 3 has more information on how to work with the C++ library (in the context of stored-distance implementations, but the information is just as a applicable to stored-data). In addition, the interested user is pointed to the source for `hclust_from_data`, the `C++` function called by `Rclusterpp.hclust`, which is itself a consumer of the templated library.

3 Stored-distance Hierarchical Clustering (in C++)

`Rclusterpp.hclust` can be used as a limited-functionality replacement for `stats::hclust`, i.e., it will accept a `dist` object as input. However, as shown Table 3, in this usage **Rcpp** is often slower than **fastcluster** and other packages specifically optimized for this use case. Instead, **Rclusterpp**'s stored-distance functionality is intended for use as linkable C++ library.

Rclusterpp is modeled on the **Rcpp*** family of packages. **Rclusterpp** provides its own skeleton function, `Rclusterpp.package.skeleton`, which can be used to generate new packages that are setup to link against the **Rclusterpp** library. Alternately one can use the **inline** package to compile C++ code from within R. The **Rclusterpp** package includes an example "inline" function, shown below, which we will use that as our working example in this document.

```
> cat(readLines(system.file("examples","clustering.R",package="Rclusterpp")),sep="\n")
```

```
#!/usr/bin/env Rscript

library(Rclusterpp)
suppressMessages(require(inline))

basic_clustering <- function(data) {
  fx <- cxxfunction( signature(data = "matrix"), '
    using namespace Rclusterpp;
    using namespace Eigen;
    // Convert from R data structure to Eigen without any data copying (the "Map" part)
    MapNumericMatrix data_e(as<MapNumericMatrix>(data));

    // Compute the distance matrix. Note this can be bettter vectorized. Also
    // note we create a strictly lower matrix.
    Eigen::NumericMatrix data_d = Eigen::NumericMatrix::Zero(data_e.rows(), data_e.rows());
    for (ssize_t i=0; i<(data_e.rows()-1); i++) {
      for (ssize_t j=i+1; j<data_e.rows(); j++) {
        data_d(j, i) = norm( data_e.row(i) - data_e.row(j) );  // Euclidean distance
      }
    }
    StrictlyLowerNumericMatrix data_t = data_d.triangularView<Eigen::StrictlyLower>();

    typedef NumericCluster::plain cluster_type;
    ClusterVector<cluster_type> clusters(data_t.rows());  // Create cluster vector

    // Perform average link clustering via recursive nearest neighbor method
    cluster_via_rnn(
      average_link<cluster_type>(data_t, FromDistance),
      init_clusters(data_t, clusters)
    );

    // Return merge, height, etc. to R...
    return wrap( clusters );
  ',
  plugin = "Rclusterpp", verbose=FALSE)
  return(fx(data))
}

r <- basic_clustering(as.matrix(USArrests))
h <- hclust(dist(USArrests, method="euclidean"), method="average")

if (identical(r$merge, h$merge)) {
  message("Success! Clustering output is the same between Rclusterpp and stats::hclust ...")
} else {
  error("Failure! Clustering output doesn't match!")
}
```

Rclusterpp makes extensive use of **Rcpp** to build the interface between R and C++, and the **Eigen** library (via **RcppEigen**) for matrix and vector operations. A working knowledge of both libraries will be needed to effectively use **Rclusterpp** as this lower level.

Rclusterpp provides several convenience `typedef`s for working at the interface of Eigen and R, in this case, we use `MapNumericMatrix` to wrap, or "map", an Eigen `Matrix` around the R data pointer (and thus no copy is involved) for use with Eigen operators. We further create a `NumericMatrix` to store the distance matrix we will compute, and extract a reference to the strictly lower portion of that matrix for use in the clustering routine.

At present, **Rclusterpp** assumes the dissimilarities are in the strictly lower portion of the matrix, and will not work with other inputs.

Agglomerations are tracked in `ClusterVector` object. The user needs to select the appropriate cluster type for their problem. A templated type factory, `ClusterTypes` is provided to assist in this selection (`NumericCluster` is a convenience `typedef` for this factory for `NumericMatrix`).

```
typedef ClusterTypes<Rcpp::NumericMatrix::stored_type> NumericCluster;
NumericCluster::plain;  // Simplest cluster, used for stored-distance
NumericCluster::center; // Maintains cluster "center", used for Ward's linkage
NumericCluster::obs;    // Tracks obs in each cluster, used for Average, Complete...
```

Clustering is performed by specifying the clustering method, i.e., RNN, the linkage method and the initialized cluster vector. In this case we are performing stored-distance average link clustering using the distance matrix computed earlier. Note that the stored-distance linkage methods are implemented with

Lance-Williams update algorithm and are destructive to the strictly lower portion of the dissimilarity matrix.

At the completion of the clustering, the cluster vector will contain all of the agglomerations along with the agglomeration heights. **Rclusterpp** extends **Rcpp** with a `wrap` implementation that will translate that vector into a R list with the `merge`, `height` and `order` entries needed for the `hclust` object.

References

[1] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *Computer Journal*, 26:354–359, 1983.

[2] F. Murtagh. Complexities of hierarchic clustering algorithms: State of the art. *Computational Statistics Quarterly*, 1(2):101–113, 1984.

[3] R. Sibson. Slink: An optimally efficient algorithm for the single-link cluster method. *Computer Journal*, 16:30–34, 1973.

[4] B. Zhang and S. Horvath. A general framework for weighted gene co-expression network analysis. *Statistical applications in genetics and molecular biology*, 4:Article17, 2005. Zhang, Bin Horvath, Steve Stat Appl Genet Mol Biol. 2005;4:Article17. Epub 2005 Aug 12.