

NeuroData SIMPLEX Report: April 2017

The following report documents the progress made by the labs of PI Joshua T. Vogelstein and Co-PIs Randal Burns and Carey Priebe at Johns Hopkins University towards goals set by the DARPA SIMPLEX grant.

Contents

1 Bibliography	2
2 Statistical Theory and Methods	3
2.1 meda	3
2.2 Multiscale Generalized Correlation (MGC)	4
2.3 Randomer Forest	5
3 Scalable Algorithm Implementations	7
3.1 FlashX	7
3.2 knor: K-means NUMA Optimized Routines	8

1 Bibliography

Talks

- [1] T. Tomita, “Roflmao: Robust Oblique Forests with Linear Matrix Operations,” SIAM International Conference on Data Mining, Apr 2017, Contributed talk.

Conferences

- [1] T. Tomita, “Roflmao: Robust Oblique Forests with Linear Matrix Operations,” SIAM International Conference on Data Mining, Apr 2017, Contributed poster.

2 Statistical Theory and Methods

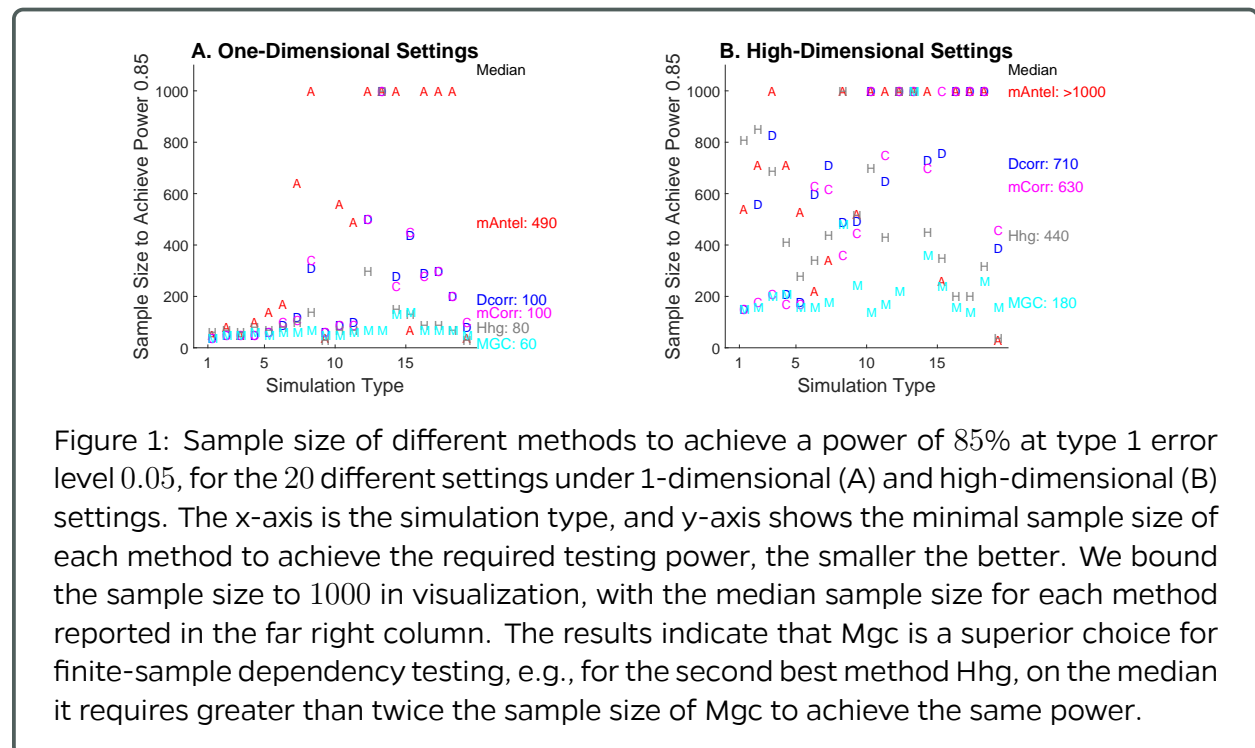
2.1 meda @JesseLP

Work on MEDA this month has been mostly devoted to creating a docker container and shifting the code base to be cloud friendly. A development version is now available on [Docker Hub](#). This will allow processing of data with meda to be deployed in the cloud.

2.2 Multiscale Generalized Correlation (MGC)

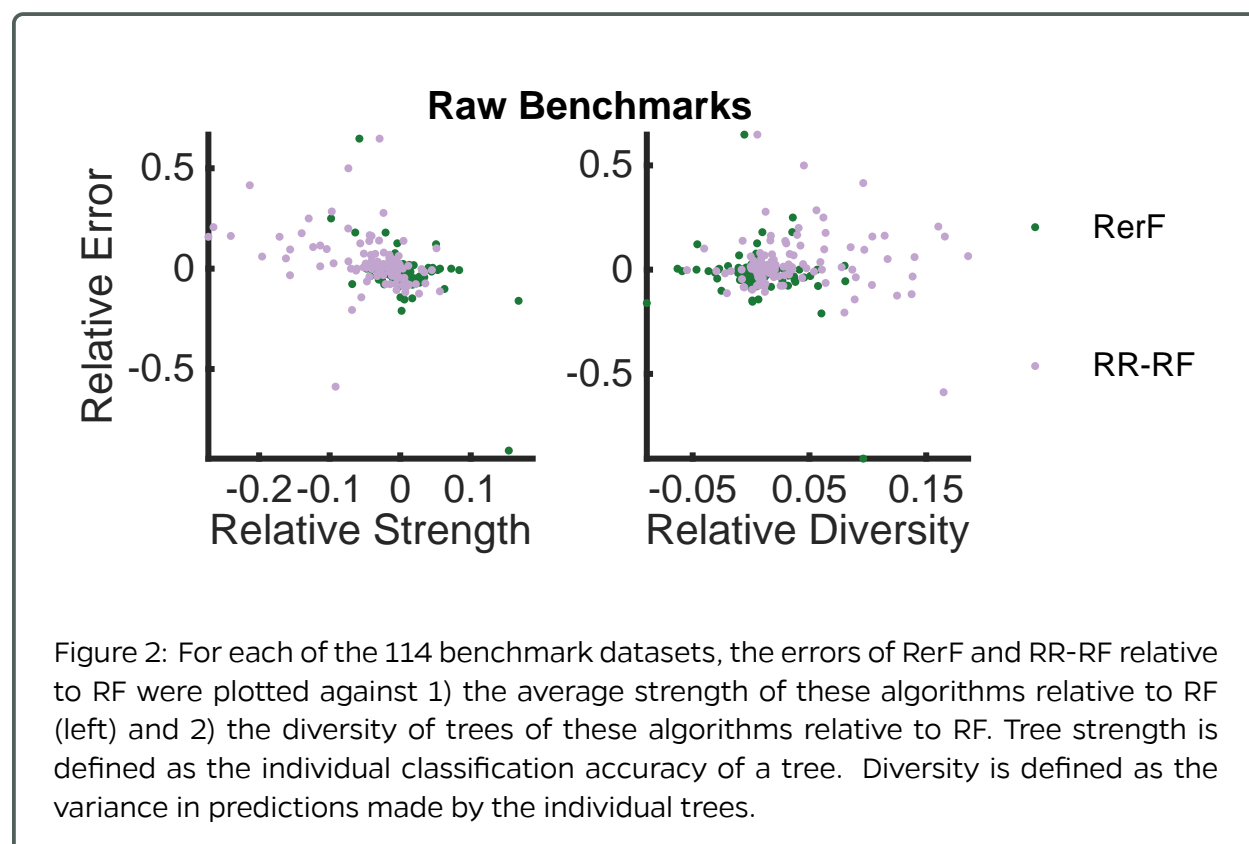
We developed the Multiscale Generalized Correlation method to better detect associations between two datasets X and Y . We demonstrate that Oracle MGC is a consistent test statistic (power converge to 1 as sample size increases) under standard regularity conditions, is equivalently to the global correlation under linear dependency (i.e., each observation X_i is a linear transformation of Y_i), and can be strictly better than the global correlation under common nonlinear dependencies.

In practice, MGC often achieves the same testing power using much less sample size than its competitors, which is important for data collection purpose. This is shown in Figure 1.



2.3 Randomer Forest (RerF)

Previously, we had demonstrated that RerF tends to outperform other tree ensemble algorithms on synthetic datasets as well as a large suite of benchmark datasets. Our current effort is to understand when we can expect RerF to win and when we can expect it to lose. As a first step, we have made scatter plots of classification error against two metrics known to be influential in the performance of ensembles: 1) strength of individual weak learners and 2) diversity of weak learners. Below, the plots suggest that strength seems to have a stronger correlation with classification performance than does diversity. RR-RF tends to have lower average tree strength than both RF and RerF. RerF tends to have higher average tree strength than RF.



The R version of RerF is now functional and is an order of magnitude faster than the Matlab implementation. This version of RerF allows the user to specify the minimum size of a node and a parameter to tweak the rotation matrix. Additional basic functionality is being added to this tool including bagging, out-of-bag error reporting, max tree depth, and pruning.

The R version of Rerf, **R-Rerf**, has been tested on a 400Mb artificial dataset. The training time on this data set is about 3.5 minutes/tree using 1 core, with the training time scaling linearly with the number of cores added. The increased memory requirements of the multicore implementation are higher than anticipated though – requiring 8 times the size of the input data per core. To reduce the memory requirements we are testing depth first vs breadth first tree growing methods.

3 Scalable Algorithm Implementations

3.1 FlashX

This month, we improve FlashR deployment to simplify the use of FlashR in a production environment. First, we integrate FlashR into Jupyter Notebook. As such, users can analyze large datasets stored on a server, simply using a Web browser. Figure 3 shows an example of using this environment to compute singular value decomposition on a graph with billions of vertices and generate the screeplot for singular values. In addition, we deploy FlashR in docker to enable FlashR to run in various environments easily, such as Macbook, Windows laptops and clouds.

The screeplot of the singular values

```
In [7]: load("/mnt/nfs2/zhengda/pg-aug-40.vals")
getElbows <- function(d, n = 3, threshold = FALSE)
{
  if (is.unsorted(-d))
    stop("d must be sorted decreasingly!")
  if (!is.logical(threshold))
    d <- d[d > threshold]
  p <- length(d)
  if (p == 0)
    stop(paste("d must have elements that are larger than the threshold ",
              threshold, "-!", sep=""))
  lq <- rep(0.0, p)
  for (q in 1:p) {
    # log likelihood, function of q
    mu1 <- mean(d[1:q])
    mu2 <- mean(d[-(1:q)])
    sigma2 <- (sum((d[1:q] - mu1)^2) + sum((d[-(1:q)] - mu2)^2)) /
      (p - 1 - (q < p))
    lq[q] <- sum(dnorm(d[1:q], mu1, sqrt(sigma2), log=TRUE)) +
      sum(dnorm(d[-(1:q)], mu2, sqrt(sigma2), log=TRUE))
  }
  q <- which.max(lq)
  if (n > 1 && q < p) {
    return(c(q, q + getElbows(d[(q+1):p], n-1)))
  } else {
    return(q)
  }
}
cols <- rep("black", length(vals))
cols[getElbows(sqrt(vals))] <- "red"
pch <- rep(1, length(vals))
pch[getElbows(sqrt(vals))] <- 19
plot(sqrt(vals), type="b", xlab="#singular values", col=cols, pch=pch)
```

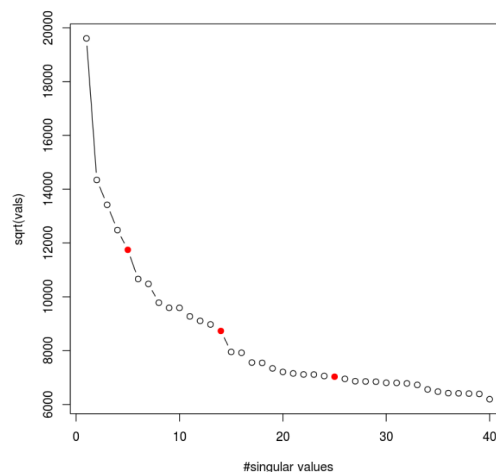


Figure 3: Generate a screeplot for singular values of a graph with billions of vertices.

3.2 knor: K-means NUMA Optimized Routines

As part of our commitment to readily available tools we migrated **knor** to the FlashX organization <https://github.com/flashxio> to increase visibility. Furthermore, we extended the capability of **knor** by providing the option of using **knor** as an initialization tool for other clustering algorithms. The intention is to allow users to seamlessly add **knor** to their workflow. We pushed a new release named african jallof to provide this capability:

<https://github.com/flashxio/knor/releases>.

We benchmarked our Semi-external memory routine, **knors** in the cloud using Amazon EC2 using a single 32 core i3.16xlarge machine with 8 SSDs on Amazon EC2 compared to **knord**, MLlib and an optimized MPI routine running in a cluster. We run **knors** with 48 threads, with extra parallelism coming from symmetric multiprocessing. We show comparable to in-memory performance and outperform MLlib with much less hardware resources. We added these results the paper which shall be published in HPDC '17.

Table 1: The datasets under evaluation in this study. Friendster is a social network. We obtain the graph containing friend relationships and compute Eigenvalues and Eigenvectors.

Data Matrix	n	d	Size
Friendster-8 eigenvectors	66M	8	4GB
Friendster-32 eigenvectors	66M	32	16GB
Rand-Multivariate (RM-856M)	856M	16	103GB
Rand-Multivariate (RM-1B)	1.1B	32	251GB

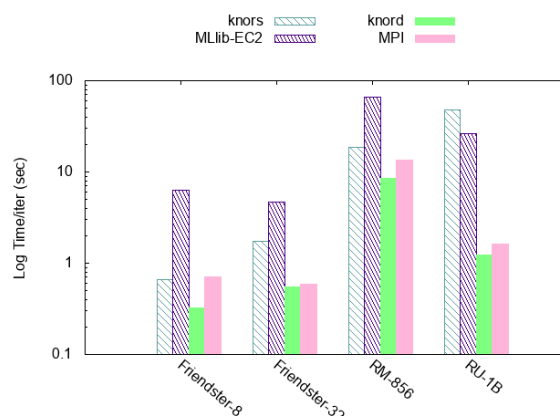


Figure 4: Performance comparison of **knors** to distributed packages. **knors** uses one i3.16xlarge machine with 32 physical cores. **knord**, MLlib-EC2 and MPI use 3 c4.8xlarge with a total of 48 physical cores for all datasets other than RU-1B where they use 8 c4.8xlarge with a total of 128 physical cores.