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

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

Some timing tests were conducted to determine which methods/functions contribute the most to processing time, see figure 1. The figure implies that attention should be given to `Hgmm` which is our current implmentation of hierarchical Gaussian mixture models. Other 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.

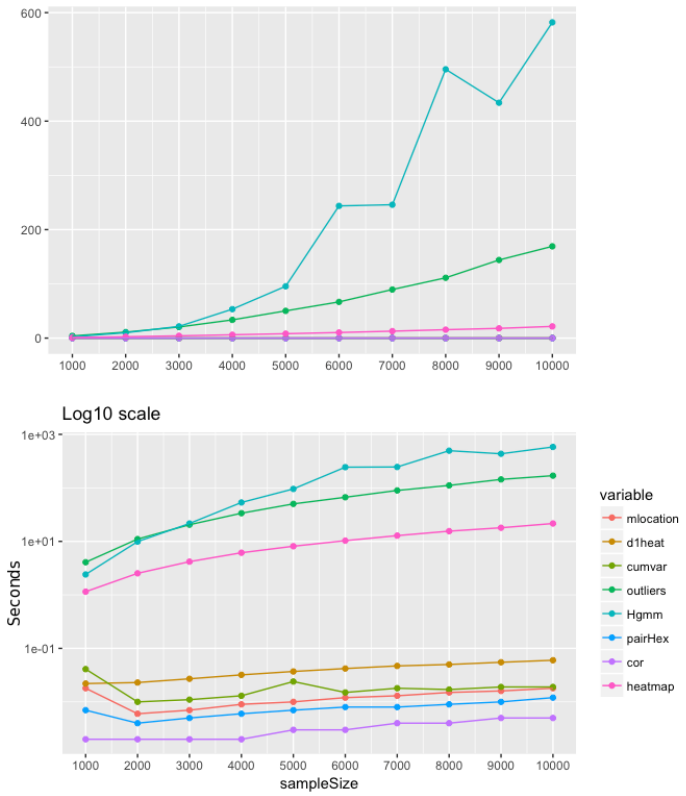


Figure 1: Timing plots for the functions used in meda. The bottom plot is the log scale of the top one. The data used consisted of 10,000 observations in 24 dimensions, sampled in chunks of inceasing size. The Hgmm function takes about 10 minutes on the largest sample used.

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 2.

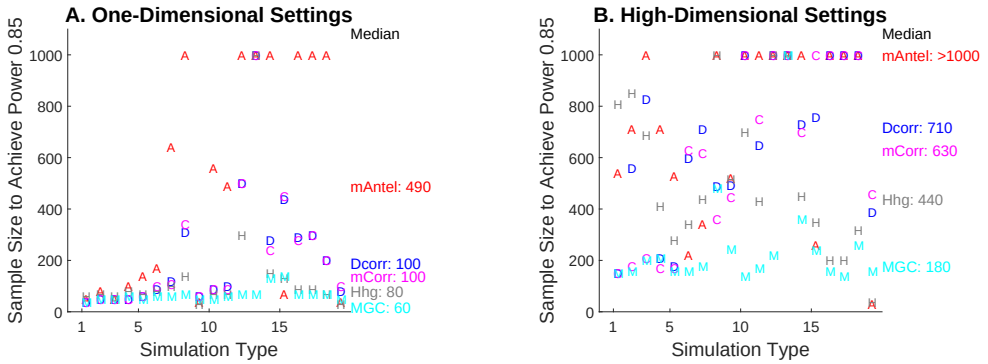


Figure 2: Sample size of different methods to achieve a power of 85% at type 1 error level 0.05, for the 20 different settings under 1-dimensional (A) and high-dimensional (B) settings. The x-axis is the simulation type, and y-axis shows the minimal sample size of each method to achieve the required testing power, the smaller the better. We bound the sample size to 1000 in visualization, with the median sample size for each method reported in the far right column. The results indicate that Mgc is a superior choice for finite-sample dependency testing, e.g., for the second best method Hhg, on the median it requires greater than twice the sample size of Mgc to achieve the same power.

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.

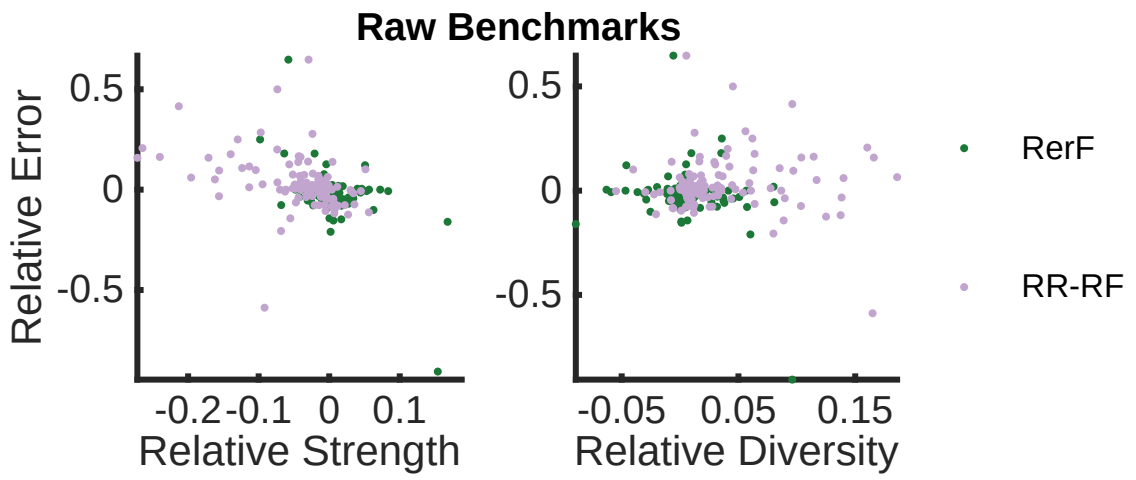


Figure 3: For each of the 114 benchmark datasets, the errors of RerF and RR-RF relative to RF were plotted against 1) the average strength of these algorithms relative to RF (left) and 2) the diversity of trees of these algorithms relative to RF. Tree strength is defined as the individual classification accuracy of a tree. Diversity is defined as the variance in predictions made by the individual trees.

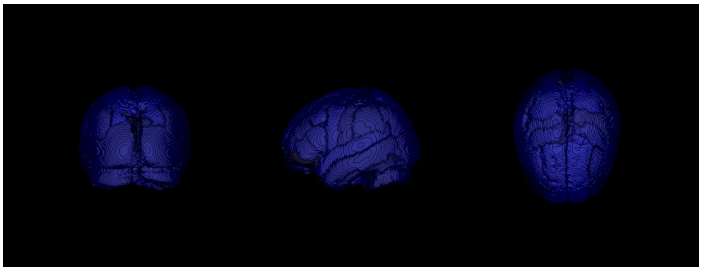
The R version of RerF is now functional and is an order of magnitude faster than the Matlab implementation. This version of RerF allows the user to specify the minimum size of a node and a parameter to tweak the rotation matrix. Additional basic functionality is being added to this tool including bagging,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

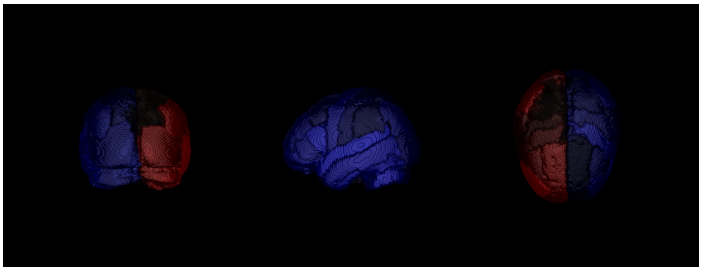
2.4 Law of Large Graphs

We note that low-rank methods can often be more easily interpreted. By representing a low-rank matrix in terms of the latent position, where each vertex is represented as a vector in $\Re^d$ and the entries of the matrix are given by the inner products of these vectors, one can analyze and visualize the geometry of these vectors in order to interpret how each vertex is behaving in the context of the larger graph. Now we take the CoRR dataset experiment as an example and consider the same sample of size $M = 5$ based on the Desikan atlas. Our estimator $\hat{P}$ is based on the estimated latent positions $\hat{X} \in \mathbb{R}^{N \times d}$, where $N = 70$ is the number of vertices and $d = 11$ is the dimension selected by the Zhu and Ghodsi's method. We color the brain using the first 5 dimensions of $\hat{X}$ as in Fig. 4. From the figures, we can see the embeddings have its own neuro-meaning, for example there is a clear distinction of the left and right hemisphere as conveyed in the second dimension. Also, the first dimension provides an average level of the entire brain. We are still exploring the interpretation other dimensions are providing.

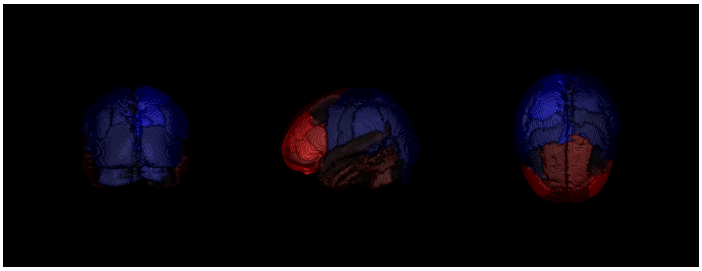
(a) 1st dimension



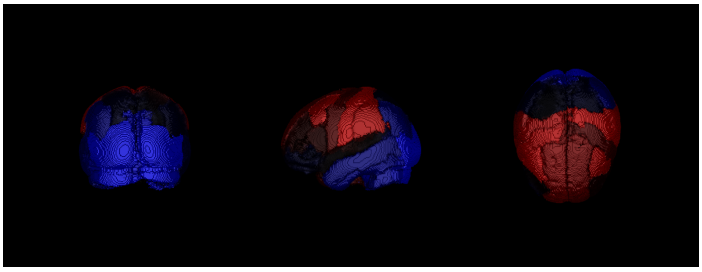
(b) 2nd dimension



(c) 3rd dimension



(d) 4th dimension



(e) 5th dimension

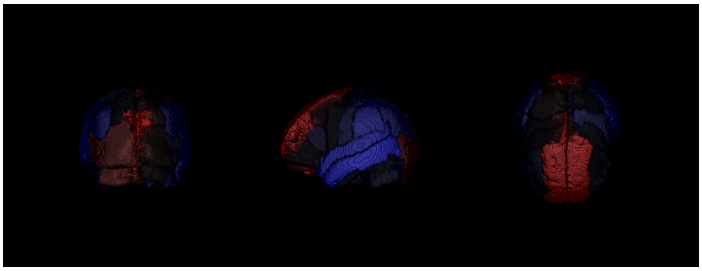


Figure 4: **Brain plots colored by the first 5 dimensions of $\hat{X}$ for the Desikan atlas respectively.** We plot the brain using the first 5 dimension of $\hat{X}$. From the figures, we can see the embeddings have its own neuro-meaning, for example there is a clear distinction of the left and right hemisphere as conveyed in the second dimension. Also, the first dimension provides an average level of the entire brain.

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 5 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)                    # log likelihood, function of q
  for (q in 1:p) {
    mu1 <- mean(d[1:q])
    mu2 <- mean(d[-(1:q)])             # = NaN when q = p
    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)
```

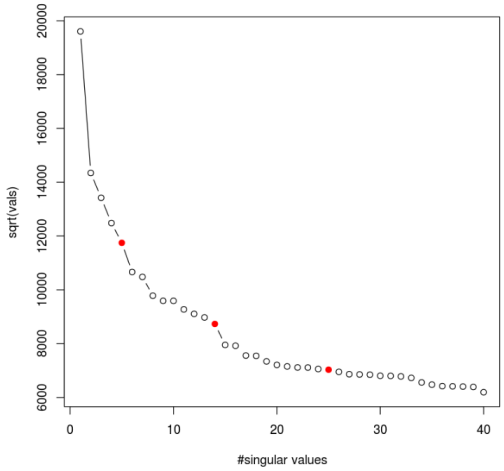


Figure 5: Generate a screeplot for singular values of a graph with billions of vertices.