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.

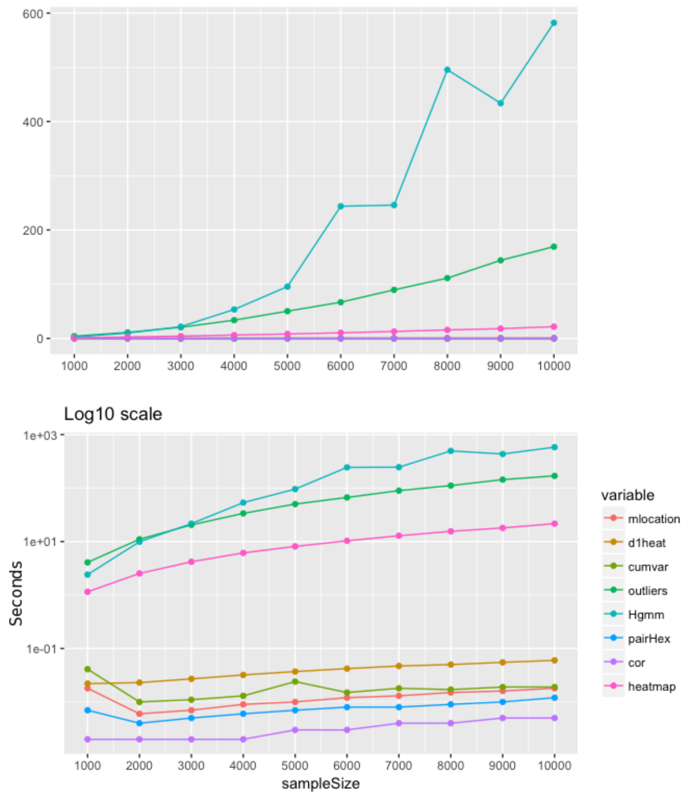


Figure 1: Timing plots for the functions used in meda. The bottom plot is the log scale of the top one. The data used consisted of 10,000 observations in 24 dimensions, sampled in chunks of inceasing size. The Hgmm function takes about 10 minutes on the largest sample used.

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.

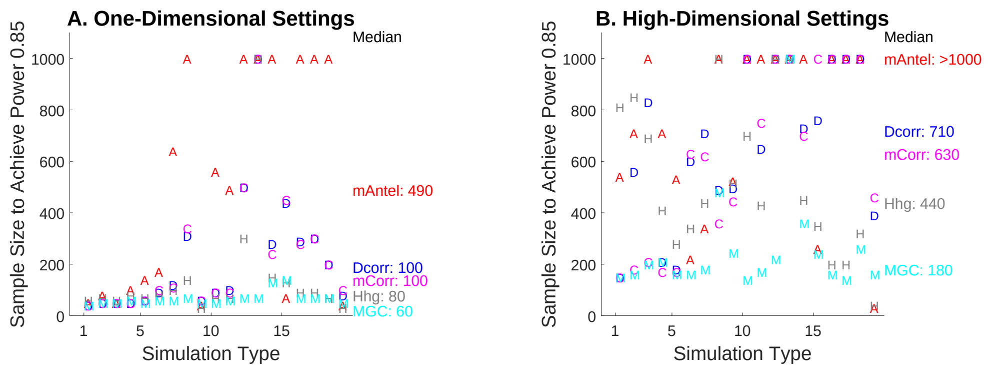


Figure 2: Sample size of different methods to achieve a power of 85% at type 1 error level 0.05, for the 20 different settings under 1-dimensional (A) and high-dimensional (B) settings. The x-axis is the simulation type, and y-axis shows the minimal sample size of each method to achieve the required 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 Multiscale Network Testing for Two-Graph

We invented multiscale network test via diffusion maps and MGC, and extends its utility into testing two graphs of the same node set with different edge sets. Assume two graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ are generated via a latent variable $\mathbf{u}_i = (u_{1i}\, u_{2i}\, \cdots\, u_{5i}) \in \mathbb{R}^5$ as follows:

$$\begin{aligned} u_{ki} &\overset{i.i.d.}{\sim} Unif(0,1), \quad i = 1,2,\ldots,n;\ k = 1,2,\ldots,5 \\ w_i &:= (1-u_{i1})^2, \quad i = 1,2,\ldots,n \\ A_{ij}^{(1)}|\mathbf{u}_i,\mathbf{u}_j &\sim Bernoulli\big(< \mathbf{u}_i/5, \mathbf{u}_j/5 > \big), \quad \forall i<j;\ i,j = 1,2,\ldots,n;\ \mathbf{u}_i,\mathbf{u}_j \in \mathbb{R}^5 \\ A_{ij}^{(2)}|w_i,w_j &\sim Bernoulli\big(< w_i, w_j > \big), \quad \forall i<j;\ i,j = 1,2,\ldots,n. \end{aligned} \tag{1}$$

That is, Each graph is generated by a random dot product graph (RDPG), and the underlying dependency is reflected via the quadratic function of one-dimensional latent variable; this implies both multi-dimensional and nonlinear relationship where MGC is preferred to other benchmarks in testing network dependency in nodal attributes.

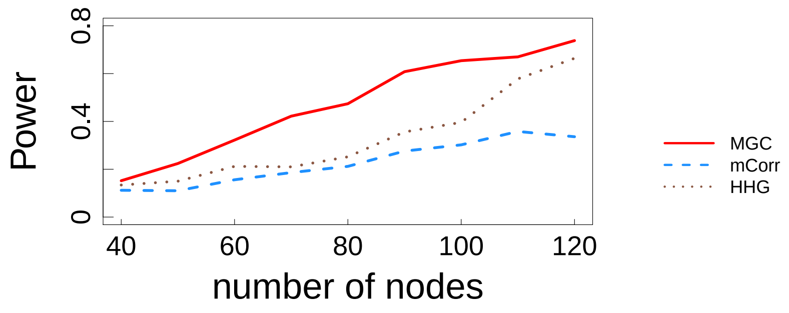


Figure 3: The power curve with respect to increasing number of nodes for the two-graph dependency testing simulation (Equation (1)). The proposed approach achieves higher power than other methods.

Figure 3 shows the testing power of MGC, mCorr, and HHG against the number of nodes n, all based on the diffusion maps, and it demonstrates that the proposed approach is able to achieve higher testing power under relatively small number of nodes. Note that if noise is included in the set-up, or the nonlinear relationship is more complex than quadratic, the proposed approach still enjoys the same advantage, i.e., the testing power converges to 1 faster than all other methods, though the actual number of nodes to achieve perfect power will likely increase under noisy and complex dependency.
The draft is submitted this month and available on arXiv.

2.4 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.

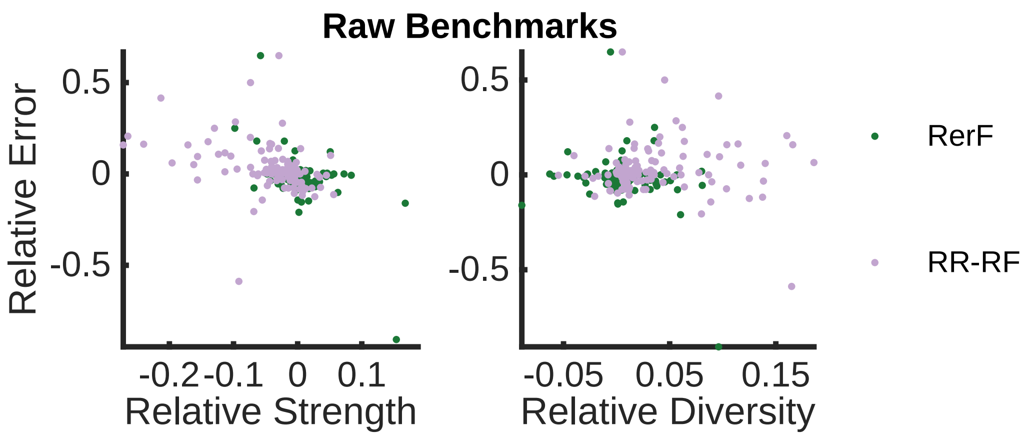


Figure 4: For each of the 114 benchmark datasets, the errors of RerF and RR-RF relative to RF were plotted against 1) the average strength of these algorithms relative to RF (left) and 2) the diversity of trees of these algorithms relative to RF. Tree strength is defined as the individual classification 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.

2.5 Vertex Screening

We are developing a vertex screening method to recover the signal subgraph. Specifically, we have m pairs of graph and label sampled independently from some distribution $F_{G,Y}$,

$$(G_1, Y_1), (G_2, Y_2), (G_3, Y_3), ..., (G_m, Y_m) \overset{i.i.d.}{\sim} F_{G,Y}.$$

It is often the case that the signal is sparse in G. That is to say, there is a small subgraph $G[S]$ induced by signal vertices S which contains all information about Y. The vertex screening method wants to recover the signal vertices S. It consists of three steps: feature extraction, computing distance correlations, and thresholding.

The first step is to extract a feature vector for each vertex in a graph. We use notation $\hat{X}_i[u, \cdot]$ to denote the feature extracted for vertex u in graph i where $i \in [m]$ and $u \in [n]$. A simple approach to obtain a feature vector is to set $\hat{X}_i[u]$ to the uth row of adjacency matrix A_i, that is $\hat{X}_i[u, \cdot] = A_i[u, \cdot]$. In this case, $\hat{X}_i[u, \cdot]$ is a vector in $\mathbb{R}^n$ which can be a high dimensional space. Alternatively, Adjacency Spectral Embedding could also extract a feature vector $\hat{X}_i[u, \cdot]$ which lies in $\mathbb{R}^d$. The second step computes a correlation between $\{\hat{X}_i[u, \cdot]\}_{i=1}^m$ and $\{Y_i\}_{i=1}^m$ for each $u \in V$. The correlation could be distance correlation (Dcorr) or multiscale generalized correlation (MGC). Let c_u be the correlation, that is

$$c_u = Dcorr(\{\hat{X}_i[u, \cdot]\}_{i=1}^m, \{Y_i\}_{i=1}^m) \text{ or } MGC(\{\hat{X}_i[u, \cdot]\}_{i=1}^m, \{Y_i\}_{i=1}^m).$$

The last step order c_us by their magnitudes. Then, we threshold the correlations by a critical value c. The vertices survive thresholding will be the estimated signal vertices $\hat{S}$, that is

$$\hat{S} = \{u \in V | c_u > c\}.$$

The estimated signal subgraph and the corresponding adjacency matrix $G[\hat{S}]$ and $A[\hat{S}]$. The Algorithm 1 describes the general procedure of vertex screening using adjacency vector as feature and MGC. A simulation experiment is shown in the Figure 5 which demonstrates that the vertex screening can significantly improve the graph classification performance for various sample size. We are currently working on theories of vertex screening and finishing the first draft.

Algorithm 1 Vertex Screening. Find the signal vertex estimate $\hat{S}$.

```
1: procedure Input $\{(A_i, Y_i)\}_{i=1}^m$ and $c \in [0, 1]$
2:     for $u = 1 : n$ do
3:         $c_u = MGC(\{\hat{X}_i[u, \cdot]\}_{i=1}^m, \{Y_i\}_{i=1}^m)$
4:     Output $\hat{S} = \{u | c_u > c\}$
```

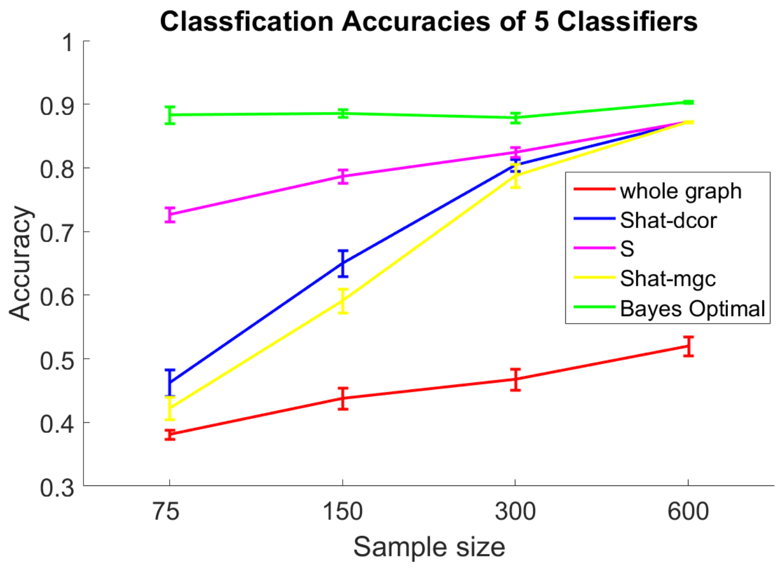


Figure 5: The classification accuracies of five approaches with their standard errors are shown. We generate graphs from 3 different inhomogeneous Erdos-Renyi model, then apply 5 classifiers to classify these graphs: Bayes optimal classifier (green), Bayes plugin on $G[S]$ (purple), Bayes plugin on $G[\hat{S}]$ with $\hat{S}$ estimated by Dcorr (blue), Bayes plugin on $G[\hat{S}]$ with $\hat{S}$ estimated by MGC (yellow), and Bayes plugin on G (red). The classifiers with vertex screening (blue and yellow) have significantly better classification performance compared to without screening (red), and are close to Bayes optimal (green) when given 600 graphs.

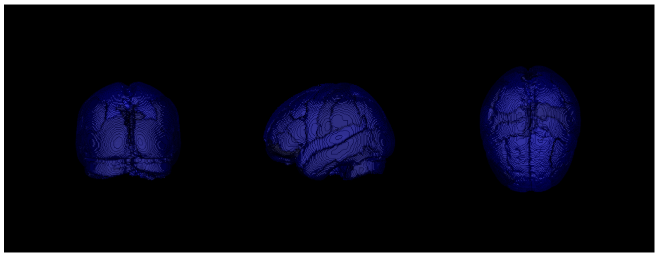
2.6 Joint Embedding

The latest draft is posted on arXiv and submitted for publication. The code can be found here. We are also working on making a R package with joint embedding included.

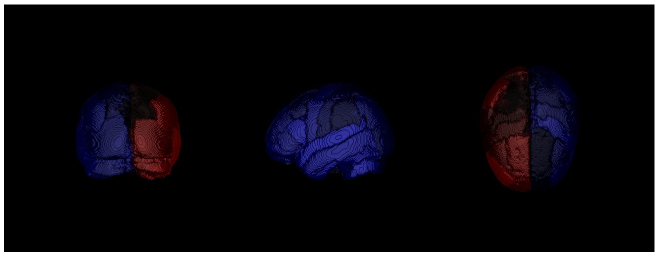
2.7 Law of Large Graphs

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. 6. 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.

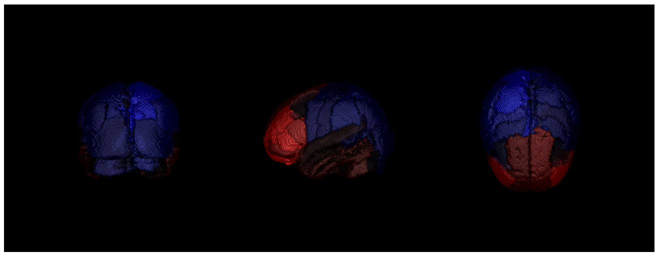
(a) 1st dimension



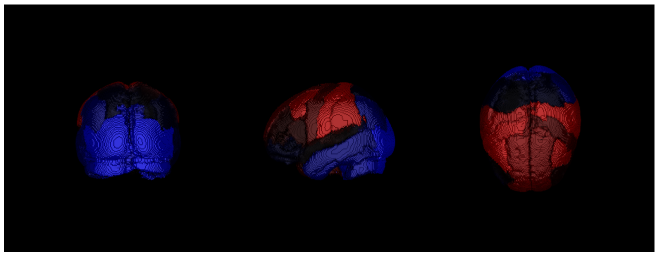
(b) 2nd dimension



(c) 3rd dimension



(d) 4th dimension



(e) 5th dimension

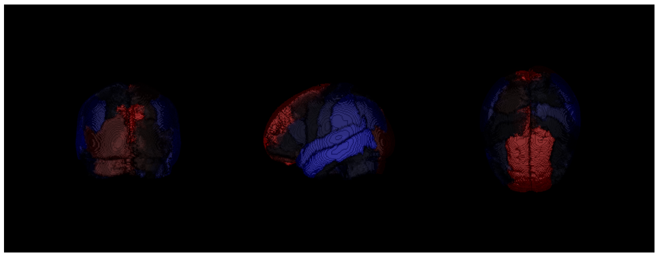


Figure 6: **Brain plots colored by the first 5 dimensions of $\hat{X}$ for the Desikan atlas respectively.** We plot the brain using the first 5 dimension of $\hat{X}$. From the figures, we can see the embeddings have its own neuro-meaning, for example there is a clear distinction of the left and right hemisphere as conveyed in the second dimension. Also, the first dimension provides an average level of the entire brain.

2.8 Robust Law of Large Graphs

In order to see a bias-variance tradeoff phenomenon with respect to the parameter q in the MLqE estimator, we change our simulation setting as following: We consider the 2-block SBM with respect to the exponential distributions parameterized by

$$B = \begin{bmatrix} 4 & 2 \\ 2 & 7 \end{bmatrix}, \qquad \rho = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}.$$

And let the contamination also be a 2-block SBM with the same structure parameterized by

$$B' = \begin{bmatrix} 9 & 6 \\ 6 & 13 \end{bmatrix}, \qquad \rho = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}.$$

Figure 7 shows the mean squared error in average by varying the parameter q in MLqE with fixed $n = 100$, $m = 20$ and $\epsilon = 0.1$ based on 1000 Monte Carlo replicates. Different types of lines represent the simulated MSE associated with four different estimators. From the figure, we can see that the ASE procedure takes advantage of the graph structure and improves the performance of the corresponding estimators independent of the selection of q. Moreover, within a proper range of q, the MLqE wins the bias-variance tradeoff and shows the robustness property compare to the MLE. And as q goes to 1, MLqE goes to the MLE as expected.

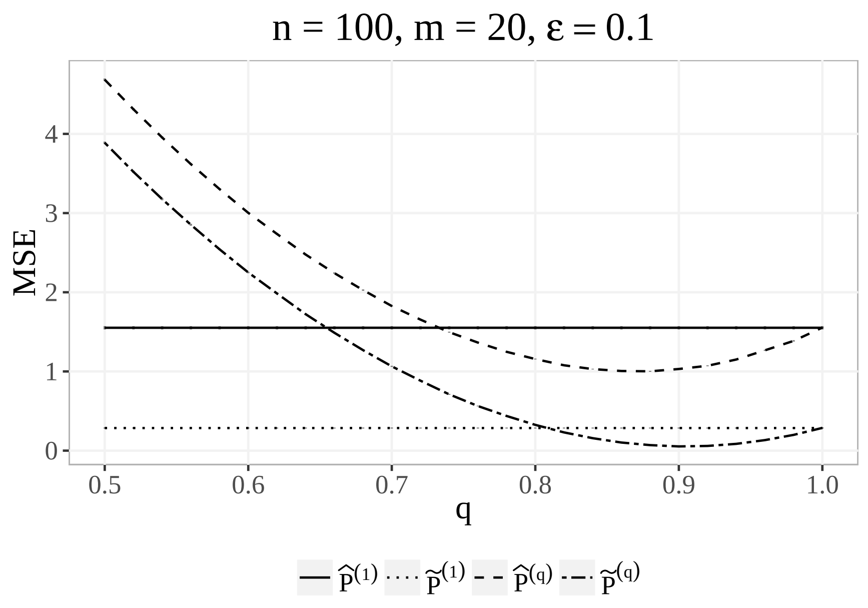
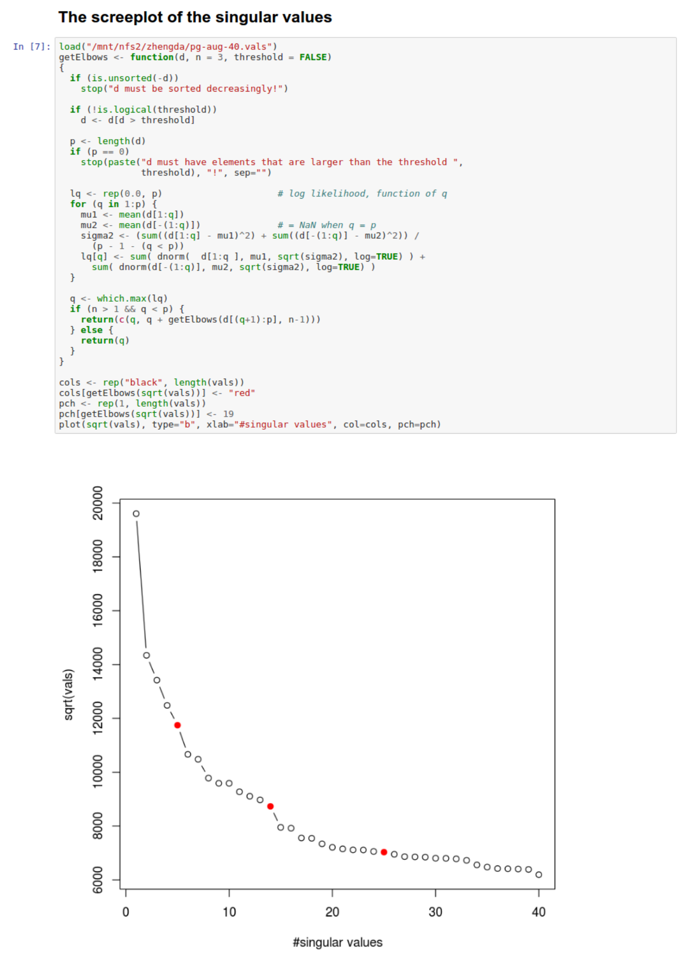


Figure 7: Mean squared error in average by varying the parameter q in MLqE with fixed $n = 100$, $m = 20$ and $\epsilon = 0.1$ based on 1000 Monte Carlo replicates. Different types of lines represent the simulated MSE associated with four different estimators. 1. ASE procedure takes advantage of the graph structure and improves the performance of the corresponding estimators independent of the selection of q; 2. Within a proper range of q, MLqE wins the bias-variance tradeoff and shows the robustness property compare to the MLE. Also as q goes to 1, MLqE goes to the MLE as expected.

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 8 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 8: 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

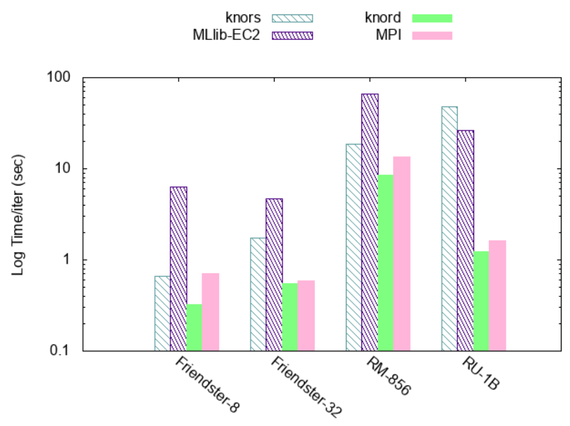
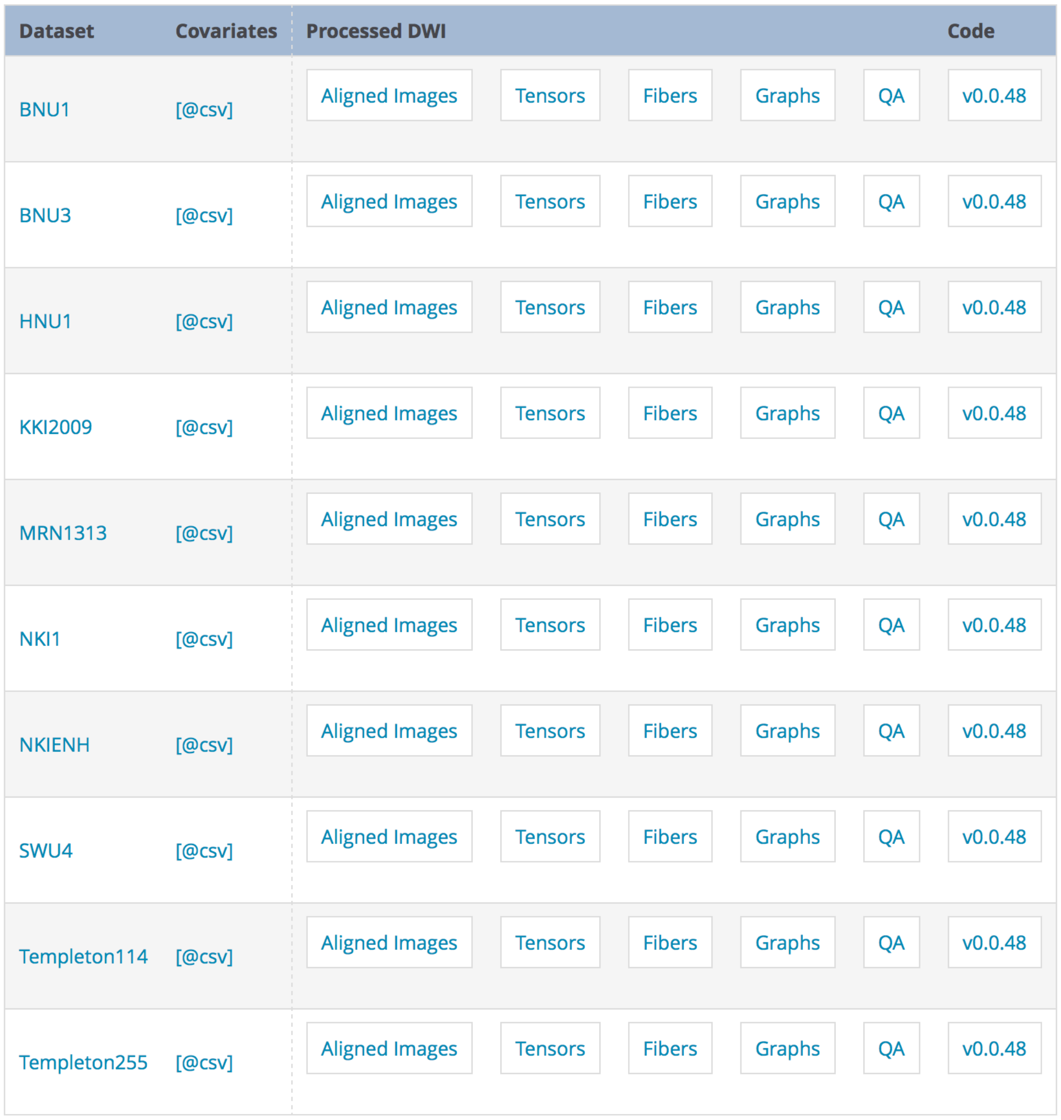


Figure 9: Performance comparison of **knors** to distributed packages. **knors** uses one i3.16xlarge machine with 32 physical cores. **knord**, MLlib-EC2 and MPI use 3 c4.8xlarge with a total of 48 physical cores for all datasets other than RU-1B where they use 8 c4.8xlarge with a total of 128 physical cores.

4 Data: What's in the Cloud



Dataset	Covariates	Processed DWI					Code
BNU1	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
BNU3	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
HNU1	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
KKI2009	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
MRN1313	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
NKI1	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
NKIENH	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
SWU4	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
Templeton114	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48
Templeton255	[@csv]	Aligned Images	Tensors	Fibers	Graphs	QA	v0.0.48

Figure 10: A current snapshot of the diffusion magnetic resonance imaging data resulting from the pipeline m2g for generating human connectomes at scale.

5 Scientific Pipelines: Infrastructure & Dataset Specific Progress

5.1 ndstore

With the release of Python 3.6 in December of last year, Python 3 has now reached sufficient maturity for us to begin migrating our infrastructure from Python 2.7 to Python 3.x. In addition to the large number of new features in the programming language that were introduced in Python 3, significant performance and stability improvements have also been introduced. We are particularly interested in using the much richer set of exceptions and increased performance for iterators present in Python 3. As a user-facing web service, NDStore must be both performant and stable, and return meaningful error messages to users when something goes wrong (e.g. bad user parameters, an internal system issue, or even a bug in the code).
To that end, we have been working with our collaborators at JHUAPL to move to a standardized Python 3 codebase for large (teravoxel) imaging volumes. We have successfully deployed a Python 3 endpoint, caching infrastructure, and some miscellaneous background processing modules (e.g for data ingest) in Amazon Web Services. We are now working to determine how best to unify our existing infrastructure with our new Python 3 infrastructure, with the overarching goal of maintaining a consistent interface across all of our assorted data types (e.g. electron and light microscopy, functional time-series imaging, magnetic resonance imaging) while minimizing duplication of effort between JHU and JHUAPL.
We have also spent considerable effort testing the data ingest client among JHU team members. We are hoping to deploy the ingest client to collaborators in the coming months. Allowing collaborators to ingest data with minimal JHU intervention increases the corpus of data we are able to collect, which provides more resources for training, running algorithms, or doing analysis.

5.2 ndviz

NeuroDataViz was rewritten to replace Leaflet entirely with Neuroglancer, an open-source web-based visualization tool from Google. Neuroglancer handles requesting data from a remote server and rendering data using WebGL. Whereas Leaflet requested 2D image tiles, Neuroglancer requests 3D image volumes. Although this results in a slight increase in request size, requesting 3D volumes allows Neuroglancer to render three canonical plane views (i.e. xy, xz, and yz) without requesting the data in triplicate as well as reducing latency when traveling in the z-direction. By replacing Leaflet with Neuroglancer we can add the same benefits to NDViz essentially "for free".

Our first major addition to the Neuroglancer code base involved refactoring the ?SliceView? functionality to support arbitrary data types. Neuroglancer manages data loading using functionality called ?SliceView?. Tiles are requested based on the viewport and chosen plane(s) (e.g. xy). Because 3D volumes can substantially increase the data size, both data loading and data unloading become equally important. The infrastructure in SliceView manages this entire process. However, SliceView was designed solely for image data. In conjunction with Jeremy Maitin-Shepard, who developed Neuroglancer, we refactored the SliceView code to support arbitrary data formats. We then added infrastructure to manage volumes containing not image data, but point data to create ?vector graphics? layers that can display points, lines, or polylines. We are using this point layer functionality to display vector fields derived from the feature point matches between image slices (see figure).

We are planning on contributed the SliceView modifications back to Neuroglancer to share with the neuroscience community at large and are currently in the process of finalizing and integrating those changes into the main Neuroglancer repository.

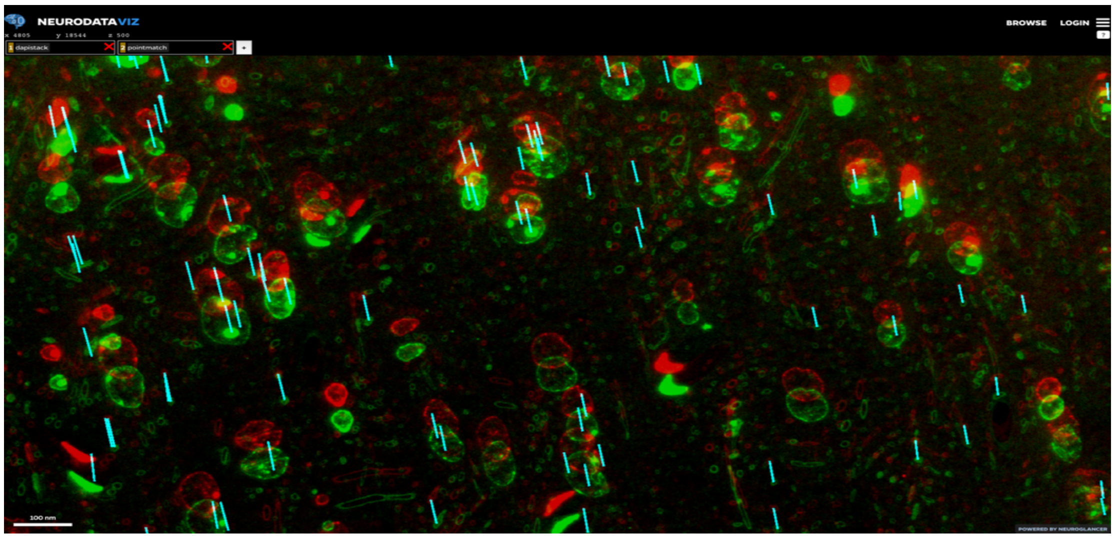


Figure 11: NeuroDataViz, powered by Neuroglancer with point matches rendered as 3D lines (in cyan) overlayed on two serial sections of Array Tomography data. Data collected by Forrest Collman et al at the Allen Insitute for Brain Science, Seattle WA.