Performance Evaluation of Minimum Average Deviance Estimation in High Dimensional Poisson Regression

REU Site: Interdisciplinary Program in High Performance Computing

Ely Biggs[1], Tessa Helble[2], George Jeffreys[3], Amit Nayak[4],
Graduate assistant: Elias Al-Najjar[2],
Faculty mentor: Kofi P. Adragni[2], Client: Andrew M. Raim[5]

[1]Department of Applied Mathematics, Wentworth Institute of Technology,
[2]Department of Mathematics and Statistics, UMBC,
[3]Department of Mathematics and Statistics, Rutgers University,
[4]Department of Mathematics, The George Washington University,
[5]U.S. Census Bureau

Technical Report HPCF–2015–21, hpcf.umbc.edu > Publications

Abstract

The second most expensive part of the 2010 Decennial Census was Address Canvassing (AdCan), a field operation to prepare the Master Address File (MAF) for census day. The MAF is a database of households in the United States maintained by the US Census Bureau and is used as a basis for the census and household surveys that it conducts. Motivated by the importance of the MAF and the cost of a large scale AdCan operation, there is an interest to use statistical methodologies to explain MAF errors discovered during canvassing. Ideally, statistical models could be used to predict future errors and assist with updating of the MAF. A major challenge in constructing a MAF error model is that important predictor variables associated with MAF errors are not known. Some recent works at Census Bureau have carried out variable selection using a collection of data sources, treating counts of errors per census block as the outcome. It may be possible to use dimension reduction methodologies to obtain count models with much lower dimensional predictors. Adragni et al. [4] proposed a methodology called Minimum Average Deviance Estimation (MADE), which is based on the concept of local regression and embeds sufficient dimension reduction of the predictors. MADE assumes a forward regression with the response variable following an exponential family distribution, such as Poisson for counts.

The goal of this project is to evaluate the performance of MADE on large data sets using simulations. We parallelized several snippets of the MADE source code to improve its performance and compare the speed up of these parallelized snippets with their serial alternatives. Simulated data sets with increasing dimensions are used to evaluate the run time. A limited stress test is performed to determine the extent of problem size that MADE can handle on maya, a high performance computing cluster at UMBC. These tests allow us to evaluate the capabilities of MADE to scale to large data sets, such as the AdCan modeling problem.

Keywords Sufficient dimension reduction, local linear regression, parallelization

AMS subject classifications (2010).
62 Statistics
68 Computer Science
90 Operations Research, Mathematical Programming

1 Introduction

The United States Census Bureau is an agency of the United States Department of Commerce that collects information concerning the population of the United States of America, Puerto Rico, and associated territories; its mission includes conducting a decennial census of the population and the American Community Survey. The Census Bureau maintains the Master Address File (MAF) to help support census and survey operations. The MAF is a database that contains an accurate, up to date inventory of all known living quarters in the United States, Puerto Rico and associated territories. The content of the MAF includes address information, Census geographic location codes, as well as source and history data[1] [2].

The content of the MAF is continuously updated to reflect the evolving landscape of living quarters. A major source of updates is the Delivery Sequence File provided by the United States Postal Service. In preparation for the 2010 Census, the Bureau conducted a nationwide Address Canvassing (AdCan) operation to ensure accuracy of the MAF. Updates to the MAF included addition of new habitable addresses (“adds”) and removal of addresses which were no longer habitable (“deletes”). There has been interest at the Bureau to develop statistical models for the add and delete outcomes using the wealth of data obtained from the 2010 AdCan operation. Ideally, if very accurate predictive models were available, they might be used to help reduce the amount of in-field canvassing needed in future operations.

Young et al. [13] and Raim and Gargano [14] study the prediction problem for counts of adds and deletes at the census block level. They use count regression models based on the zero-inflated Poisson and zero-inflated negative binomial distributions. A challenge encountered in both works is selection of covariates for the model. A number of data sources may be considered for this purpose, based on economy, geography, demographics, and so on.

[1] http://www2.census.gov/geo/pdfs/education/Uhl_CAS_2011.pdf
[2] https://www.census.gov/did/www/snacc/publications/MAF-Description.pdf

Within each data source, variables must be selected and coded into predictors. Interactions between predictors might also be important in excplaining the outcomes. In both of these previous works, the set of predictors selected for the model is fairly large, with some providing only a small contribution to the model. It would be desirable to reduce the predictor through formal dimension reduction methods if only a few dimensions contain most of the predictive power.

Adragni et al. [4] devise a general methodology called Minimum Average Deviance Estimation (MADE). The methodology is built upon the concept of sufficient dimension reduction while taking into account the distribution of the outcome. A reduction of the covariates is obtained using a local regression approach that helps capture possible linear and nonlinear trends. The reduction may be used to predict future outcomes.

MADE is developed for a class of distributions referred to as exponential family distributions. This class of distributions includes Gaussian, gamma, exponential, binomial, and Poisson distributions. MADE is reminiscent of Minimum Average Variance Estimation (MAVE) [2], which was designed for Gaussian outcomes. As with MAVE, MADE is also a kernel-based sufficient dimension reduction based on the estimation of the gradient of the conditional expectation $E(Y|X)$ by means of a local regression, where Y represents the response and X is the predictor.

The existing methodologies for sufficient dimension reduction can be grouped into three classes: moment-based, likelihood-based, and kernel-based methods. Examples of the moment-based sufficient dimension reduction include sliced inverse regression [5], sliced average variance estimation [6], inverse regression estimation [7], and directional regression [8]. Examples of the likelihood-based sufficient dimension reduction include principal fitted components [9] and likelihood acquired directions [10].

In this technical report, we focus on performance evaluation of MADE when the outcome is from a Poisson distribution. We evaluate the performance of a previously developed MADE R code by [4] on high performance computing hardware, and extend it by implementing parallel computing. Parallelization of MADE enables it to process larger data sets and decreases its run time.

The remainder of this report is organized as follows. In Section 2, we elaborate on the method and the statistical and computing tools that were used. In Section 3, we will discuss the graphs and results that we were able to attain from running the various simulations in serial and parallel. Finally, we provide some conclusions and recommendations for future work in Section 4.

2 Methodology

We assume that $Y \in \mathbb{R}$ is a response, X is a p-dimensional predictor, and $Y|X$ has a Poisson distribution with the density function

$$f(y \mid \mu(x)) = \frac{\mu(x)^y e^{-\mu(x)}}{y!}. \tag{2.1}$$

In the exponential family framework, the canonical parameter $\theta(X)$ is related to the mean function $\mu(X) = E(Y|X)$ through the link $\mu(X) = \exp\{\theta(X)\}$ so that (2.1) becomes

$$f(y \mid \theta(x)) = \exp\{y\theta(x) - \exp[\theta(x)]\}\frac{1}{y!}. \tag{2.2}$$

Let $(Y_i, X_i), i = 1, ..., n$ represent an independent sample from the distribution of (Y, X) so that $Y_i|X_i$ has the distribution (2.2). We do not assume that $\theta(X) = \beta^T X$ with a fixed β globally for all X. Instead, we assume that $\theta(X)$ is a continuous and smooth function of X, so that it can be approximated at any point X by the first order linear expansion

$$\theta(X_i) \approx \theta(X) + [\nabla\theta(X)]^T(X_i - X) \tag{2.3}$$

for any X_i in the neighborhood of X. Let $\alpha_X = \theta(X)$ and $\Gamma_X = \nabla\theta(X)$. The term Γ_X retains the core information that connects Y_i to X_i locally at X. As X varies in its sample space, the set $\{\Gamma_X\}$ describes a u-dimensional subspace $\mathcal{S}$ in $\mathbb{R}^p$ with $u \leq \min(n, p)$. Let $B \in \mathbb{R}^{p\times d}$ with $B^T B = I$ be a basis of $\mathcal{S}$ so that $\Gamma_X = B\gamma$ for some $\gamma = \gamma(X) \in \mathbb{R}^{u\times 1}$. Then, we have

$$\theta(X_i) = \theta(X) + \gamma_X^T B^T(X_i - X) + e(X, X_i), \tag{2.4}$$

where $e(X, X_i)$ is assumed negligible. Clearly, if B is known, then $\theta(X_i) = \theta(B^T X_i)$. Consequently, the distribution of $Y \mid X$ with probability density function provided by (2.2) is the same as the distribution of $Y \mid B^T X$. This implies that $B^T X$ is a sufficient dimension reduction of X according to the definition of Cook (2007).

Using (2.2) and (2.4), the local log-likelihood for a given $X \in \mathbb{R}^p$ in the Poisson setting can now be written as

$$L_X(\alpha, \gamma, \beta) = \sum_{i=1}^{n} w_i(B^T X)[y_i(\alpha + \gamma^T B^T(X_i - X)) - \exp\{\alpha + \gamma^T B^T(X_i - X)\} + \log(f_0(y_i))]. \tag{2.5}$$

Assuming that α_j and γ_j are known, minimizing the deviance as a function of B is equivalent to maximizing the objective function

$$Q(B) = \sum_{j=1}^{n}\sum_{i=1}^{n} w_i(B^T X_j)[y_i(\alpha_j + \gamma_j^T B^T(X_i - X_j)) - \exp\{\alpha_j + \gamma_j^T B^T(X_i - X_j)\}]. \tag{2.6}$$

While each sample point X_j has its own regression coefficients α_j and γ_j, they all share a common dimension reduction kernel matrix B. For a given B, the kernel weights are computed as

$$w_i(B^T X) = \frac{K_H(B^T(X_i - X))}{\sum_{j=1}^n K_H(B^T(X_j - X))}, \tag{2.7}$$

where $K_H(u) = |H|^{-1} K(H^{-1/2}u)$ and $K(u)$ denotes one of the usual multidimensional kernel density functions and the bandwidth H is a $d \times d$ symmetric and positive definite matrix [1]. These weights $w_1(B^T X), ..., w_n(B^T X)$ sum to 1. We use a Gaussian kernel mostly in this work, although the particular choice of kernel density is not required for the method.

The parameters of interest are α_j, γ_j, $j = 1, \cdots, n$, and B. The parameter spaces are respectively $\mathbb{R}, \mathbb{R}^d$, and the Stiefel manifold $S(d, p)$ which is the set of d-dimensional orthonormal spaces in $\mathbb{R}^p$. These parameters are to be estimated. MADE implements an algorithm that alternates between iterations of the Newton-Raphson method for α_j and γ_j and the Stiefel manifold optimization method for B until the required tolerance level is reached. Following is the full algorithm.

1. Provide an initial B and obtain the weights $w_{ij} = w_i(B^T X_j)$
2. Do until convergence
   (a) Fix B and estimate α_j and γ_j for $j = 1, ..., n$ using Newton-Raphson
   (b) Fix α_j and γ_j and the weights w_{ij} for $i, j = 1, ..., n$ and estimate B using the Stiefel manifold optimization
   (c) Update the weights $w_{ij} = w_i(B^T X_j)$

We consider the optimal choice of bandwidth for Gaussian kernel to be $H = hI$ with $h = n^{-1/(d+4)}$, although that choice may not be optimal for count data. The choice of h is yet to be further investigated. A cross-validation might be helpful selecting its value.

3 Performance Evaluation

We provide the description and several performance assessments of the parallel implementation of our codes on `maya`. We have looked at the distribution of run times across the different functions as well as the performance of the code under different hardware configurations, and also evaluated the performance with increasingly large datasets.

3.1 Implementing Parallel Functions in R

The original R codes for the estimation of the parameters α_j, γ_j, and B were provided by Adragni et al. [4]. The codes consist of several major functions; for example, `objfun` evaluates the objective function Q given current values of the parameters; `grad` computes the gradient of Q; `kern.weights` computes the $n \times n$ kernel weights matrix; `solve.xi.one` calculates the values of α_j and γ_j in the first step of each iteration using the Newton-Raphson method; `stiefel` solves for B on Stiefel manifold. A single iteration of the MADE optimization algorithm requires one or more calls to each of these functions. In several places, independent calculations are carried out one after another in a serial manner. For example, the n weights for a given B are calculated one at a time. In a parallel processing framework, these calculations can be distributed to a number of processes and carried out simultaneously. This has the potential to greatly reduce the computing time for large problems.

We proceeded to implement the initial R codes in parallel using the `snow` package on the cluster `maya`. Our efforts to parallelize MADE were concentrated on the four functions: `objfun, grad, solve.xi.one`, and `kern.weights`. The package `snow` is simple to use and contains parallel counterparts of `apply`-type functions which are available in R. The cluster `maya` is a 240-node distributed-memory cluster with high performance InfiniBand interconnect. It is hosted at the UMBC High Performance Computing Facility (HPCF), the community-based, interdisciplinary core facility for scientific computing and research on parallel algorithms at UMBC.

The function `objfun` is easy enough to parallelize as it works by calculating the value of a function for n different input values. The function `grad` is similarly easy to parallelize, except for the fact that the *for* loop, which we are replacing with a parallel `sapply` function, can only apply one function in parallel. The `sapply` function is good at doing one specific task, like calling a single function many times, while a *for* loop can do many different task within the loop many times. While the *for* loop may sound better since it can run different task, it fails because it is more difficult to implement in parallel. The same is true for the `kern.weights` function. Lastly, `solve.xi.one` is a little more complicated as it is called inside of a loop and does not itself contain any loops, so to get around that we created another function which is parallelized and itself calls `solve.xi.one.call`.

As an example of parallelizing a snippet of the MADE code, we look at the `objfun` function. This portion of the code calculates the value of a function for n different input values. Therefore, parallelizing this function means calculating the function value for several n at the same time, reducing the total run time.

The serial `objfun` uses a *for* loop to do more than one task per iteration. To work around this, it is possible to create a new function which has the same arguments found in the serial `objfun`'s *for* loop and to then have `sapply` call the new function we just created. Only then will `sapply` work because it is technically doing only one task; but in reality, `sapply` is calling a function, which does many tasks, multiple times.

The following is an example of the parallelization of the objective function. The original version of the objective function was written as

```
objfun <- function(B0){
   ff <- 0

   for (j in 1:n) {
      X.j <- matrix(X[j,], n, p, byrow = TRUE)
      theta.j <- a[j] + (X - X.j) %*% B0 %*% b[j,]
      ff <- ff + sum(We[,j] * (y*theta.j - exp(theta.j)))
   }
   return(-ff)
}
```

Now using `sapply`, the parallelized version becomes

```
objfun <- function(B0, Parallel = FALSE){
   ff <- 0

   if(Parallel){
      x.mat.vec <- apply(x, 1, matrix, n, p, byrow = TRUE)
      j <- seq(1, n)
      ff.vec <- sapply(j, call.objfun, B0, disp)
      ff <- sum(ff.vec)
      return(-ff)
   }else{
      for (j in 1:n) {
         X.j <- matrix(X[j,], n, p, byrow = TRUE)
         theta.j <- a[j] + (X - X.j) %*% B0 %*% b[j,]
         ff <- ff + sum(We[,j] * (y*theta.j - exp(theta.j)))
      }
      return(-ff)
   }
}
```

And the function `call.objfun` in the objective function is coded as

```
call.objfun <- function(j, B0)
```

```
{
   X.j <- matrix(X[j,], n, p, byrow = TRUE)
   theta.j <- a[j] + (X - X.j) %*% B0 %*% b[j,]
   ff.obv <- sum(We[,j] * (y*theta.j - exp(theta.j)))
   return (ff.obv)
}
```

Finally, we change `sapply` to `parSapply` to parallelize the code.

```
ff.vec <- parSapply(cluster, j, call.objfun, B0)
```

where `cluster` is a `snow` cluster object. These same concepts are used, in variation, to parallelize the other three functions.

3.2 Distribution of Time in MADE

We generated typical data on which we ran the algorithm. The pie chart in Figure 3.1 displays the percentage of time spent in each of the main functions within MADE. The `updateB` function took by far the most amount of time to run. The `solve.xi.one` function was the second most time consuming process within MADE. This is consistent with what was initially expected from how the MADE method was created. It should be noted that any of the snippets of code measured which were quick enough to have had an elapsed time near zero were omitted.

The function `updateB` consists almost solely of the `stiefel.optim` function. Figure 3.2 displays the percentage of time spent in each snippet of the `stiefel.optim` function. From here we see that the snippets which take the most time are the ones relating to the optimization itself as well as the last step of the algorithm.

3.3 MADE in Parallel

Figure 3.3 displays the run-time of the MADE function given the same data set for different combinations of nodes and processes per node. The point corresponding to one node and one process per node represents the serial version of the code.

It is evident that parallelizing the `objfun, grad, solve.xi.one`, and `kern.weights` functions within MADE provides a faster run-time than the serial version if the optimal combination of nodes and processes per node is utilized. However, the optimal number of nodes and processes per node was not what was expected. While two nodes performed better initially, the most optimal combination of nodes and processes per node was found to be one node and sixteen processes per node for the data set tested.

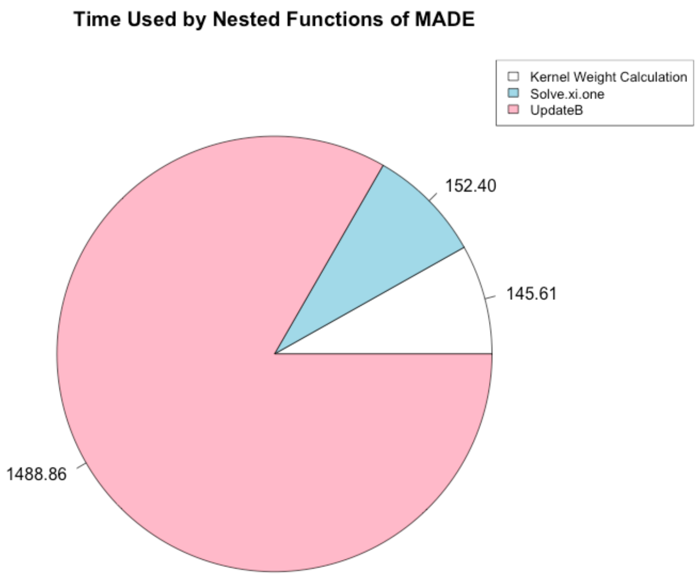

Figure 3.1: Distribution Run Time per Function

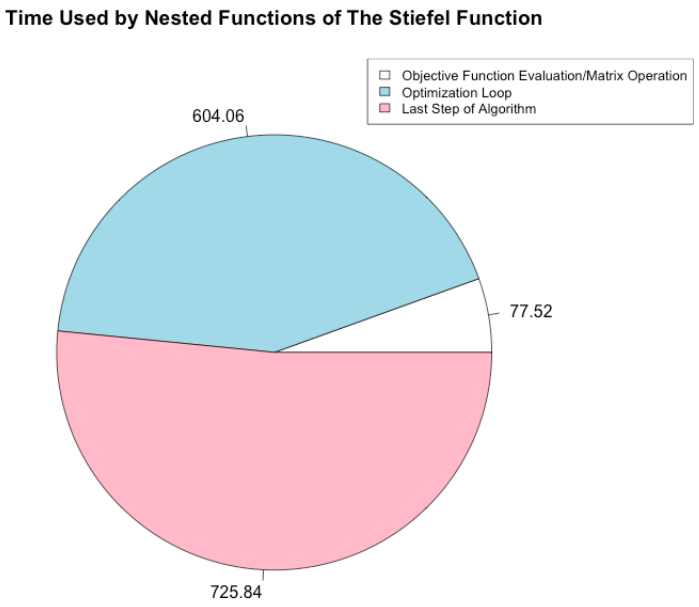

Figure 3.2: Distribution Run Time per Function

3.4 Upper Bound Dimension of the Dataset

Figure 3.4 demonstrates the run time for MADE using a data matrix of $n = 5000$ observations for $p = 100$ predictors and various combinations of nodes and processes per node. In terms of memory, datasets of dimension $n = 10000 \times p = 100$ were too large for the cluster maya

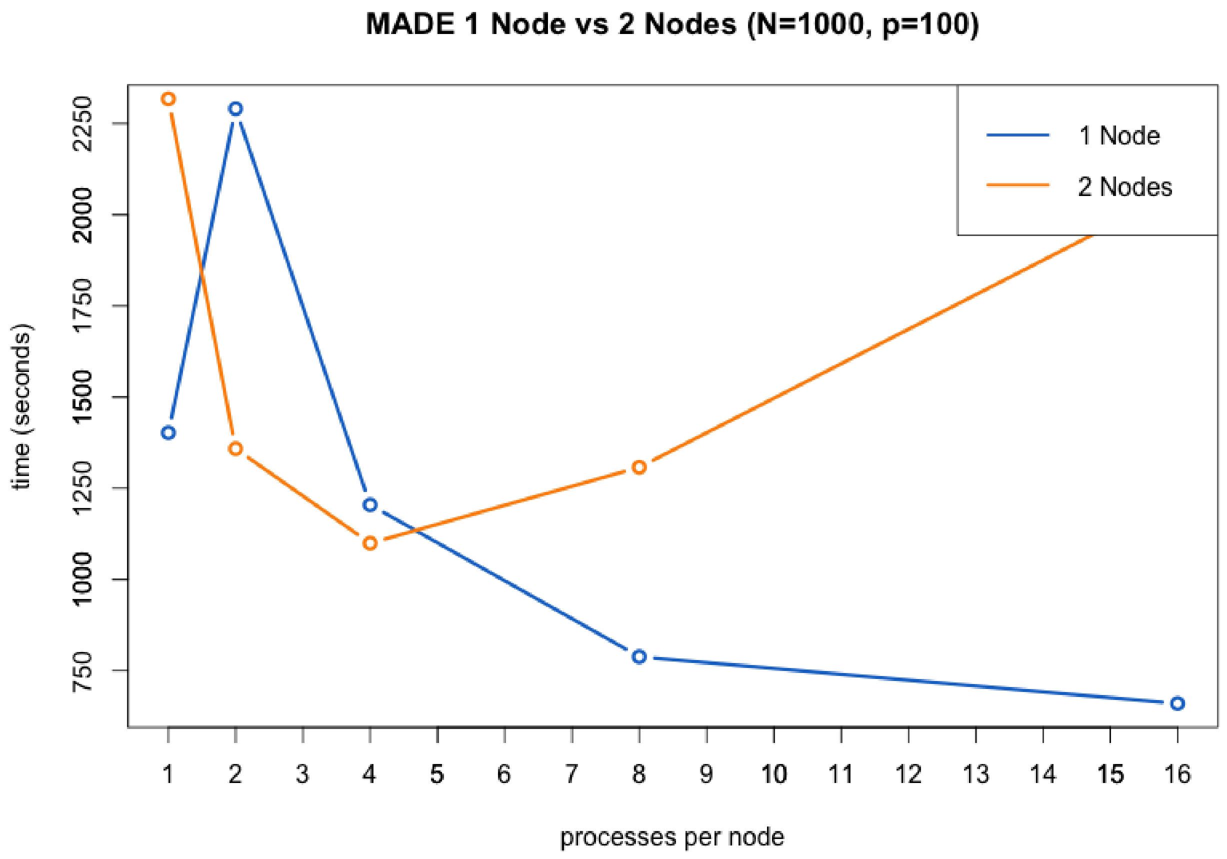

Figure 3.3: Run Time of MADE Functions.

to handle and the job would be automatically killed. It therefore appears that the upper bound of the data set is somewhere between 5000×100 and 10000×100.

Using the optimal number of nodes and processes per node (Section 3.2), one node and sixteen processes per node, a data set of 5000×100 in MADE was able to complete in approximately 13 hours.

4 Conclusions

This work analyzed the performance of an implementation of MADE and considered its ability to scale to large problems. We analyzed the run time of different components of the code, so that efforts to improve performance can focus on the most time consuming parts. As expected, the `updateB` function took by far the longest to run, followed by the `solve.xi.one` function. Within the `updateB` function, the two snippets which took the most time related to the optimization loop and the last step of the Stiefel optimization algorithm.

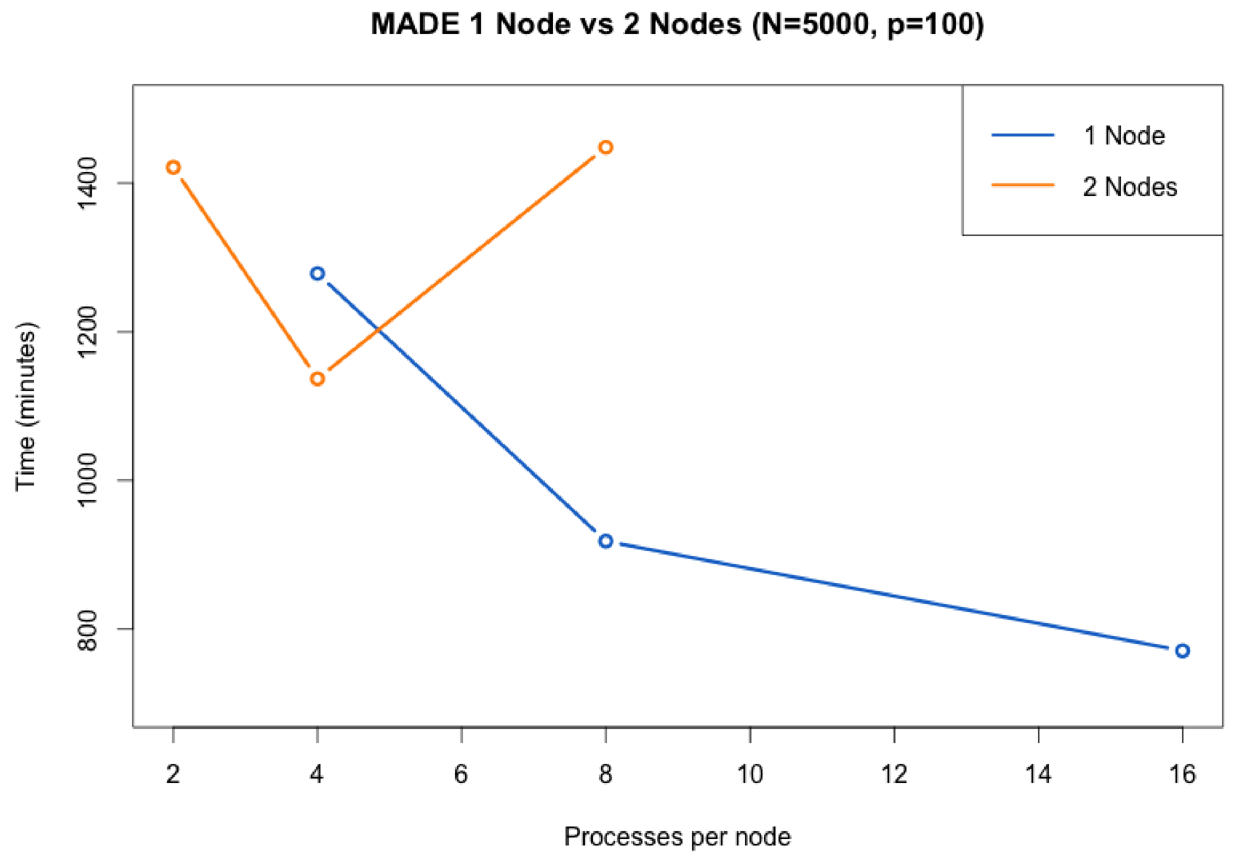

Figure 3.4: Performance for High Dimensional Data.

We also studied the combination of number of nodes and processes per node needed in order to find the optimal run time for MADE. It was determined that the optimal combination was one node and sixteen processes per node. This contradicts our initial expectation that the code would run more quickly using multiple nodes. A possible explanation for this could be that there is a great deal of time consuming node-to-node communication when utilizing multiple nodes, especailly for large data sets. Further examination of MADE and more simulations are required to analyze this phenomenon.

Finally, we attempted to determine an upper bound for the dimensionality of the data sets that our code could handle. By examining different sizes of data sets, a range for the upper bound was determined to be between a dataset of size 5000×100 and 10000×100. To run the code for data sets of higher dimensionality, the data set of predictors would need to be split and the code run several times.

While these results provide insight into the MADE implementation and its run time, there is still plenty of work left to be done. Ideally, we would like to compare the run times of MADE with its predecessor MAVE. It may also be of interest to examine MADE further and determine if it is possible to alter the code such that the run time decreases when utilizing multiple nodes. One possible method for alteration could be to write the entire function or snippets of the code in the C programming language.

References

[1] Simonoff, J.S. (1996), *Smoothing Methods in Statistics*, New York: Springer-Verlag.

[2] Yingcun Xia, Howell Tong, W. K. Li, and Li-Xing Zhu. An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society Series B*, 64(3):363-410, 2002

[3] RD Cook. *Regression Graphics*. New York: Wiley, 1998

[4] Adragni, KP; Raim, AM; Al-Najjar, Elias. Local Likelihood Acquired Directions for Sufficient Dimension Reduction via Generalized Linear Models. (Work in progress, 2015)

[5] KC Li. Sliced inverse regression for dimension reduction (with discussion). *Journal of the American Statistical Association*, 86:316-342, 1991

[6] RD Cook and S Weisberg. Discussion of sliced inverse regression by kc li. *Journal of the American Statistical Association*, 86:328-332, 1991

[7] RD Cook and L Ni. Sufficient dimension reduction via inverse regression: A minimum discrepancy approach. *Journal of the American Statistical Association*, 100:927-1010, 2005

[8] B Li and Wang S. On direction regression for dimension reduction. *Journal of American Statistical Association*, 102:997-1008, 2007

[9] RD Cook. Fisher lecture: Dimension reduction in regression. *Statistical Science*, 22(1): 1-26, 2007.

[10] RD Cook and L Forzani. Likelihood-based sufficient dimension reduction. *Journal of the American Statistical Association*, 104(485):197-208, 2009

[11] Luke Tierney and A. J. Rossini and Na Li and H. Sevcikova, snow: Simple Network of Workstations, 2013, R package version 0.3-13, http://CRAN.R-project.org/package=snow

[12] Ostrouchov, G., Chen, W.-C., Schmidt, D. and Patel, P. Programming with Big Data in R, 2012, http://r-pbd.org/

[13] Derek S. Young, Andrew M. Raim, and Nancy R. Johnson. Zero-inflated modeling for characterizing coverage errors of extracts from the U.S. Census Bureau's Master Address File, 2015. Submitted.

[14] Andrew M. Raim and Marissa N. Gargano. Selection of predictors to model coverage errors in the Master Address File, (In Progress, 2015).

5 Acknowledgements

These results were obtained as part of the REU Site: Interdisciplinary Program in High Performance Computing (hpcreu.umbc.edu) in the Department of Mathematics and Statistics at the University of Maryland, Baltimore County (UMBC) in Summer 2015. This program is funded by the National Science Foundation (NSF), the National Security Agency (NSA), and the Department of Defense (DOD),with additional support from UMBC, the Department of Mathematics and Statistics, the Center for Interdisciplinary Research and Consulting (CIRC), and the UMBC High Performance Computing Facility (HPCF). HPCF is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS-0821258 and CNS-1228778)and the SCREMS program (grant no. DMS-0821311),with additional substantial support from UMBC. Graduate assistant Elias Al-Najjar was supported during Summer 2015 by UMBC.