October 23, 2013
Transcribed by Hao Ji, with Nick's Templete

1 Overview of Big Data (Details in the Lecture Slides)

1.1 Types of "Big" Data

- Type 1: Large n and not large p
  Examples: Linear regression with $100M$ observations and 5000 covariates, website with large numbers of users, recording basic data on each user, etc.

- Type 2: Large p and not large n
  Examples: Linear Regression with $10K$ observations and $500K$ covariates, website with moderate number of users, tracking every action of each user, etc.

- Type 3: Large n and large p
  Examples: Linear regression with $100M$ observations and $50M$ covariates, website with large number of users, recording large number of actions per user (e.g. Google, Facebook and Yahoo), etc.

- Type 4: Complex (non-rectangular) "big" data
  Examples: Financial transactions (irregular), images, movies (e.g. brain scans), Wikipedia.

For the "big" data in this course, we will work with Type 1 above, namely large n and not large p.

1.2 Scaling to "Big" Data

Naive approaches designed for traditional amounts of data do not typically scale to big data. (e.g. invertibility of matrices, curse of dimensionality etc.) Usually we can use some combination of the following methods to scale to big data:

- Assuming that the data has inherently lower-dimensional structure:
  Sparsity, Conditional Independence(so that we can handle things separately)

- Fast algorithms:
  Parallelization and Typically linear time algorithms or better

- Methodology that avoids the need to fit the "full" data: Consensus Monte Carlo, Bag of Little Bootstraps

1.3 Accessing "Big" Data

Once our method is able to scale to big data, we have to find out a way to access the huge or complex data sets with practical ways.

- From the R documentation:
  There are limitations on the types of data that R handles well. Since all data being manipulated by R are resident in memory, and several copies of the data can be created during execution of a function, R is not well suited to extremely large data sets. Data objects that are more than a (few) hundred megabytes in size can cause R to run out of memory, particularly on a 32-bit operating system.

- What then for "big" data? (Alternatives to reading in data to memory)
  1. File-backed data structures(i.e. data remains stored on disk, not memory) Example: bigmemory (and other big* packages) See: http://www.bigmemory.org/
  Pros: Easy to use. Simple to understand, any language can mimic functionality.
  Cons: Requires "nice" data, burden on programmer to scale algorithms (parallelization etc.), doesn't scale as easily to data that cannot fit on disk.

  2. Databases:
  Relational Databases (e.g., SQL): Rigid structure, relational algebra operations (union, intersection, difference etc.).
  NoSQL Databases (e.g., CouchDB, MongoDB): Less structure than a relational database, less functionality, but typically faster data retrieval.

  3. Distributed File Systems:
  Example: Hadoop Distributed File System (HDFS). Data remains on disk, but DFS provides a full ecosystem for scaling to data across multiple machines.
  Pros: Scales to essentially arbitrarily large amounts of data (just add more machines).
  Cons: Harder to interact with data. More restrictive progamming paradigm (MapReduce).

2 Example of File-backed Data Structures: "Big" Logistic Regression

2.1 Basic Setting

On Gauss, we have a $255Gb$ file, with $6M$ observations and $3K$ covariates for logistic regression.

GOAL: Find standard errors for the parameter estimates of the logistic regression model.

To do this,
1. Figure out how to work with that much data using bigmemory (or Python equivalent);
2. Figure out how to obtain standard errors for parameter estimates in a scalable manner.

The codes for doing this can be found in Lecture Slides 8. Basically, we just need to create a file-backed big data set and attach the big data matrix in our program, then we are able to access to the data matrix by reading in arbitrary lines and extracting target data from the lines.

2.2 Model and Algorithm: Bags of Little Bootstraps

The model we are fitting is then:

$$Y_i|\beta \sim \text{Bin}\left(1, \frac{\exp\{x_i^T\beta\}}{1+\exp\{x_i^T\beta\}}\right), i = 6000000$$

want to obtain an element-wise SE (or CI) Estimate for $\widehat{\beta}$.

Review of Bootstrap

For small data sets, we can use Bootstrap to obtain SE for the coefficients of interests. The algorithm is summarized as follows:

- Figure out the empirical distribution $\widehat{F}_n$ based on the observed data
- Sample $\{X_i, Y_i\}_{i=1}^n$ with replacement from $\widehat{F}$ s times, to obtain s Bootstrap samples of size n:

$$\{\mathbf{X}_1^*, ..., \mathbf{X}_s^*\}$$

- For each Bootstrap sample, compute estimate of β: $\{\widehat{\beta}_1^*, ..., \widehat{\beta}_s^*\}$
- Estimate $SE(\widehat{\beta})$ by $SD(\{\widehat{\beta}_1^*, ..., \widehat{\beta}_s^*\})$.

This traditional Bootstrap method requires independent observations, and it is not scalable to the big data we have. To generate s samples of size n in our case is way too slow. Instead of considering the traditional Bootstrap, we consider the following algorithm.

Bags of Little Bootstraps

The algorithm is summarized as following:
For estimating SD($\widehat{\beta}$):

- Step 1:
  Let $\widehat{F}$ denote the empirical probability distribution of the data (i.e., placing mass $1/n$ at each of the n data points)
- Step 2:
  Select s subsets of size b from the full data (i.e. randomly sample a set of b indices $\{I\}_j = \{i_1, ..., i_b\}$ from $\{1, 2, ..., n\}$ without replacement, and repeat s times)
- Step 3:
  For each of the s subsets ($j = 1, ..., s$): Repeat the following steps r times ($k = 1, ..., r$):
  - Resample a bootstrap dataset $X_{j,k}^*$ of size n from subset j. (i.e., sample $(n_1, , n_b) \sim Multinomial(n, (1/b, , 1/b))$, where $(n_1, , n_b)$ denotes the number of times each data point of the subset occurs in the bootstrapped dataset.)
  - Compute and store the estimator $\widehat{\theta}_{j,k}$
  - Compute the bootstrap SE of $\widehat{\theta}$ based on the r bootstrap data sets for subset j i.e., compute:

$$\xi_j^* = SD\{\widehat{\theta}_{j,1}, ..., \widehat{\theta}_{j,1}\}$$

- Step 4:
  Average the s bootstrap SE's, $xi_1^*, ..., xi_s^*$s to obtain an estimate of $SD(\widehat{\theta})$ i.e.,

$$\widehat{SD(\widehat{\theta})} = \frac{1}{s}\sum_{j=1}^{s} \xi_j^*.$$

Things to think about:

- Note that although we sample $s \times r$ datasets of size n in this algorithm, the Bags of Little Bootstrap still works faster. The reason behind is that each data set has at most b distinct paired observations, and we can denote each dataset in an aggregate way. This kind of data representation property is the key to success. (Generally, linear models and generalized linear models have this property)

- How to choose s, b, r?
  Paper suggests that $b = n^{0.6}$ or $b = n^{0.7}$;
  We need a big enough r so that the Bootstrap samples resemble the s sampled datasets, but it is not worthwhile to make it massive;

- Utilize the array job capability of Gauss
  BLB suits the array job capability of Gauss. We can submit $s \times r$ jobs with 1 Boostrap dataset for each job with smart indices to make sure that each r datasets are sampled from one of the s subsets.