STA 250 Lecture 8: Big Data

Minjie Fan

October 25, 2013

Contents

1 Introduction to Big Data

1.1 What is "big" data?

It depends on what you are trying to do with it!

- Large n and not large p (**our focus**).
- Large p and not large n.
- Large n and large p.
- Complex (non-rectangular) "big" data.

1.2 Scaling to "Big" Data

Naive approaches designed for traditional amounts of data do not tyically scale to "big" data. How to scale to big data then? Usually some combination of:

- Assuming that the data has inherently lower-dimensional structure
  - Sparsity
  - Conditional independence
- Fast algorithms
  - Parallelization
  - Typically linear time algorithms or better
- Methodology that avoids the need to fit the "full" data
  - Consensus Monte Carlo
  - Bag of Little Bootstraps (**our focus**)

1.3 Q & A

- For highly correlated data, we had better treat them jointly, e.g. Gibbs sampler.

1.4 Limitation of R

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.

1.5 What then for "big" data?

We can't read in data to memory, so what alternatives are there?

- File-backed data structures (i.e., data remains stored on disk, not memory) (**our focus**)
  - Examples: 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.

- Databases (**just a little bit**)
  - 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.
- Distributed File Systems (**our focus**)
  - 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: ”Big” Logistic Regression

On Gauss I have created an uncompressed 255Gb file containing data for fitting a ”big” logistic regression model (6m observations, 3k covariates).

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

To do this:

- Figure out how to work with that much data using bigmemory (or Python equivalent)
- Figure out how to obtain standard errors for parameter estimates in a scalable manner (**Algorithm**).

2.1 R Package: Bigmemory

Function 1: read.big.matrix

Description: write the contents of a big.matrix to a suitably-formatted ASCII file.

Usage:

```
goo <- read.big.matrix(infile, type="double", header=FALSE,
                       backingpath=datapath,
                       backingfile=backingfilename,
                       descriptorfile=descriptorfilename)
```

Function 2: attach.big.matrix

Description: attach the big.matrix

Usage:

```
attach.big.matrix(dget(descriptorfile),backingpath=datapath)
```

See details: http://cran.r-project.org/web/packages/bigmemory/bigmemory.pdf

2.2 Working with the data:

- We can fit "big regressions" with biglm.big.matrix or bigglm.big.matrix.
- We can still do the basics (they just might take a while!).

2.3 Code your own "bigmemory" in R/Python

We actually won't use any of the real functionality of the bigmemory suite of packages. All we really need is the ability to read arbitrary lines from a file without loading the full file into memory.

- load the file
- Read line-by-line until the desired line is reached
- Extract the data from the line

2.4 SE Estimates for $\hat{\beta}$

We can use the bootstrap talked during boot camp. For the logistic regression model, we have both X's and y's. When we resample points, we resample both x_i and y_i. This is sometimes called the paired bootstrap.

For the logistic regression problem, using $B = 500$:

1. Let $\hat{F}$ denote the empirical probability distribution of the data (i.e., placing mass 1/6000000 at each of the 6000000 data points)

2. Take a random sample of size 6000000 from $\hat{F}$ (with replacement). Call this a "bootstrap dataset", X_j^* for $j = 1, \cdots, 500$.

3. For each of the 500 bootstrap datasets, compute the estimate $\hat{\beta}_j^*$.

4. Use the standard deviation of $\{\hat{\beta}_1^*, \cdots, \hat{\beta}_{500}^*\}$ to approximate $SD(\hat{\beta})$.

2.5 The Bag of Little Bootstraps

For estimating $SD(\hat{\theta})$:

1. Let $\hat{F}$ denote the empirical probability distribution of the data (i.e., placing mass $1/n$ at each of the n data points)

2. Select s subsets of size b from the full data (i.e., randomly sample a set of b indices $I_j = \{i_1, \cdots, i_b\}$ from $\{1, 2, \cdots, n\}$ without replacement, and repeat s times).

3. For each of the s subsets ($j = 1, \cdots, s$):

    - Repeat the following steps r times ($k = 1, \cdots, r$):

        (a) Resample a bootstrap dataset $X_{j,k}^*$ of size n from subset j.

        (b) Compute and store the estimator $\hat{\theta}_{j,k}$

    - Compute the bootstrap SE of $\hat{\theta}$ based on the r bootstrap datasets for subset j i.e., compute:

    $$\xi_j^* = SD\{\hat{\theta}_{j,1}^*, \cdots, \hat{\theta}_{j,r}^*\}.$$

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

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

- How to select s? (Number of subsets)

- How to select b? (Size of subsets)

  Real key is b. From paper $b \approx n^{0.6}$ or $b \approx n^{0.7}$ works well.

- How to select r? (Number of bootstrap replicates per subset)

  r should be large enough for each of the s subsets. Typically, $r > s$. For example, if $rs = 500$, then $r = 50$ and $s = 10$.

The **gain** is that there are only (at most) b unique data points within each bootstrapped dataset. We have existing approaches to fit this kind of data with the same time cost as if there are b data points.