BUSS6002 Data Science in Business

Week 8: Statistical Methodologies, Computing and Programming Techniques for Big Data - I

Dr. Jie Yin
Discipline of Business Analytics
University of Sydney Business School

Table of contents

BIG issues with BIG Data

A computational perspective of BIG Data

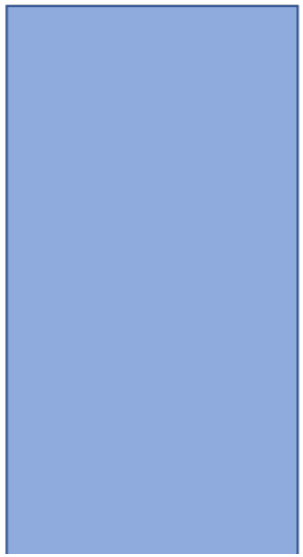
Big data vary in shape. They call for different approaches.

Wide Shape

Thousands / Millions of Variables
Hundreds of Samples

We have too many variables; prone to overfitting. Need to remove variables, or regularise, or both (Week 7)

Tall Shape



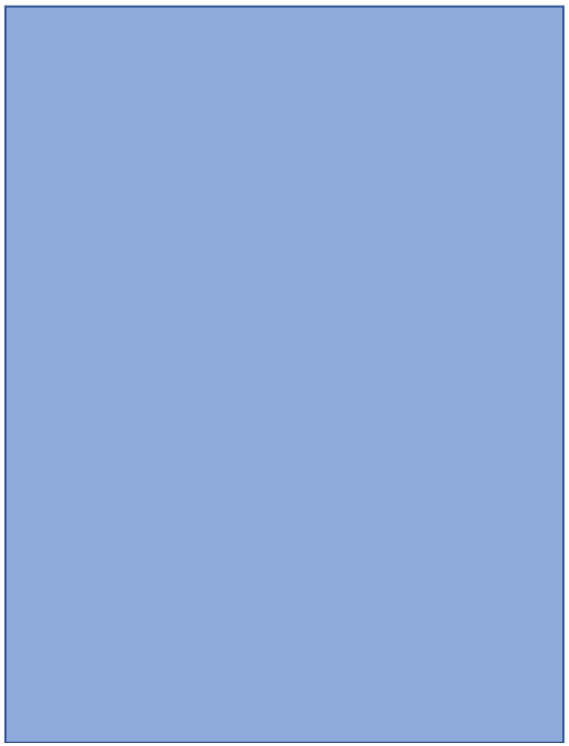
Tens / Hundreds of Variables
Thousands / Millions of Samples

Sometimes simple models (linear) don't suffice. We have enough samples to fit nonlinear models, but it is a big issue to deal with the large number of samples

A computational perspective of BIG Data

Big data vary in shape. They call for different approaches.

Wide and Tall Shape



Thousands / Millions of Variables
Millions to Billions of Samples

Exploit sparsity
Variable selection
Regularisation
Subsample rows
Divide-and-conquer
MapReduce

The focus of Week 8 & 9 is to deal with the BIG n!

Motivating example

- Suppose you have a BIG dataset of n customers $D = \{(y_i, x_i), i = 1, ..., n\}$.
- Suppose that a statistical model $p(y_i|x_i, \theta)$, like logistic regression model, has been developed. This gives you the log-likelihood function

$$\ell(\theta) = \log p(D|\theta) = \sum_{i=1}^{n} \log p(y_i|x_i, \theta)$$

- You use the Maximum Likelihood Method to estimate θ, i.e. you need to maximise $\ell(\theta)$.
- Suppose that it takes 0.1 second to compute each $\log p(y_i|x_i, \theta)$ in your desktop, $n = 20$ million. Usually, the MLE takes roughly 10 iterations to converge. How long would it take to estimate this model?
- Yes, almost a year!

Two BIG issues with BIG Data

- **Speed**: it might take years to estimate your model.
- **Computer's memory**: you can't load such a large dataset into your computer's memory (usually 8 or 16GB), while your dataset can be hundreds Gigabytes or more.

Note: Computer's memory (or RAM) is different from hard drive. RAM is like a person's brain and hard drive is like books in a library.

Two BIG issues with BIG Data

- Traditional statistical methodologies such as MLE and OLS are often slow. New scalable algorithms are necessary.
- Model fitting takes a lot longer. This might test our patience for model evaluation and selection.
- New computing and programming techniques are necessary for the memory issue.
- In this lecture, you will learn some statistical methodologies and programming techniques for Big Data.

Statistical methodologies for Big Data

Statistical methodologies for Big Data

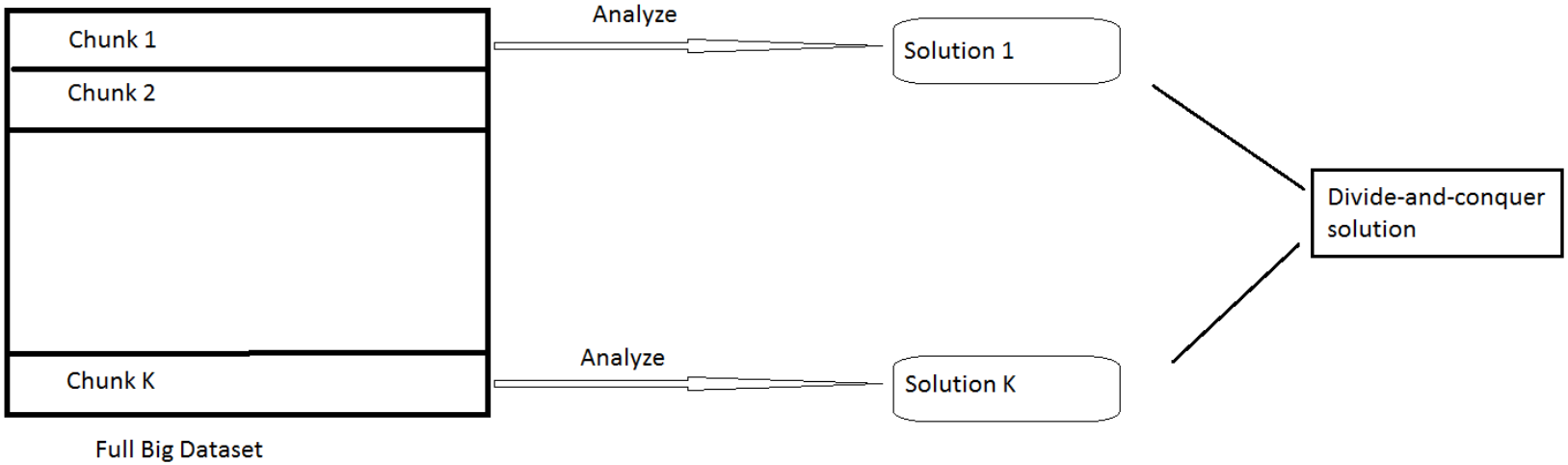
- Divide-and-conquer approach
- Subsampling approach

Divide-and-conquer approach

The main idea of the divide-and-conquer approach is:

- Divide the big dataset into smaller blocks
- Analyse each block separately. Often this is run in parallel in a multi-processor computer.
- Finally, combine the solutions from each block to form the final solution

Divide-and-conquer approach



Divide-and-conquer approach for linear regression

Consider the linear regression model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_i \\ \vdots \\ \epsilon_n \end{bmatrix}.$$

$$\widehat{\boldsymbol{\beta}}_{ls} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$\boldsymbol{X}$ is a matrix of size $n \times (p+1)$. If n is very large, then $\boldsymbol{X}$ is so large that it is impossible to compute $\boldsymbol{X}^T\boldsymbol{X}$ or it is close to being singular

Divide-and-conquer approach for linear regression

Let's divide $\boldsymbol{X}$ and $\boldsymbol{y}$ into K blocks

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \vdots \\ \boldsymbol{X}_K \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_K \end{bmatrix}.$$

Each block has roughly $n_k = n/K$ rows. K should be large enough so that each $\boldsymbol{X}_k$ can be loaded in computer's memory.

Suppose we have estimates $\widehat{\boldsymbol{\beta}}_k = (\boldsymbol{X}_k^T \boldsymbol{X}_k)^{-1} \boldsymbol{X}_k^T \boldsymbol{y}_k$ for each block.

How do we combine $\widehat{\boldsymbol{\beta}}_k$ to get the original $\widehat{\boldsymbol{\beta}}_{ls}$?

Divide-and-conquer approach for linear regression

Let $\boldsymbol{A}_k = \boldsymbol{X}_k^T \boldsymbol{X}_k$. Each matrix $\boldsymbol{A}_k$ is of size $(p+1) \times (p+1)$. If p is not too large, then it's feasible to compute and store $\boldsymbol{A}_k$ and $\widehat{\boldsymbol{\beta}}_k$.

Since we have

$$\boldsymbol{X}^T \boldsymbol{X} = \sum_{k=1}^{K} \boldsymbol{X}_k^T \boldsymbol{X}_k, \quad \boldsymbol{X}^T \boldsymbol{y} = \sum_{k=1}^{K} \boldsymbol{X}_k^T \boldsymbol{y}_k$$

Then

$$\widehat{\boldsymbol{\beta}}_{ls} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y} = (\sum_{k=1}^{K} \boldsymbol{X}_k^T \boldsymbol{X}_k)^{-1} \sum_{k=1}^{K} (\boldsymbol{X}_k^T \boldsymbol{X}_k)(\boldsymbol{X}_k^T \boldsymbol{X}_k)^{-1} \boldsymbol{X}_k^T \boldsymbol{y}_k$$

$$= (\sum_{k=1}^{K} \boldsymbol{A}_k)^{-1} \sum_{k=1}^{K} \boldsymbol{A}_k \widehat{\boldsymbol{\beta}}_k$$

Divide-and-conquer approach for linear regression

So we have the following algorithm

- **Divide:** Divide the data into K blocks
- **Analysis:** Compute $\boldsymbol{A}_k$ and $\widehat{\boldsymbol{\beta}}_k$, $k = 1, ..., K$ in parallel
- **Combine:** Compute

$$\widehat{\boldsymbol{\beta}}_{ls} = \left(\sum_{k=1}^{K} \boldsymbol{A}_k \right)^{-1} \sum_{k=1}^{K} \boldsymbol{A}_k \widehat{\boldsymbol{\beta}}_k$$

This algorithm solves both issues: speed (by parallel computing) and computer memory (by dividing the full dataset into smaller blocks). It never requires accessing to the full data $\boldsymbol{X}$ and $\boldsymbol{y}$.

Divide-and-conquer approach for other models

- The divide-and-conquer algorithm for linear regression has a very nice property: the divide-and-conquer solution is exactly equal to the full-data solution.
- Unfortunately, in general this property no longer holds for other models such as logistic regression.
- The reason is that the estimator $\widehat{\beta}$ in the logistic regression cannot be decomposed as $\widehat{\beta}_{ls}$ as in the linear regression.
- Simple solution: compute solutions for each block, then simply take the average to form the final solution.
- This final solution is in general different from the solution based on the full dataset.

Divide-and-conquer approach: example

Example. The data set "women_labour.xlsx" contains information on women's labour force participation of 750 women.

In-labour-force `inlf` is the response variable

$$\texttt{inlf} = \begin{cases} 1, & \text{if the female is in labour force} \\ 0, & \text{otherwise.} \end{cases}$$

The potential predictors are `nwifeinc` (income), `educ` (years of education), `exper` (years of experience), `expersq` (squared years of experience), `kidslt6` (number of kids less than 6-year-old) and `kidsge6` (number of kids more than 6-year-old).

A suitable model is logistic regression.

Divide-and-conquer approach: example

- This is not a big dataset, but we use it to demonstrate the idea.
- Want to see
  - if the divide-and-conquer solution $\widehat{\beta}_{d\&c}$ is close to the full-data solution $\widehat{\beta}_{full}$
  - the effect of the number of blocks, K

Divide-and-conquer approach: example

K	Euclidean difference $\|\widehat{\beta}_{full} - \widehat{\beta}_{d\&c}\|$
1	0
5	0.1779
10	0.5929
15	1.6021

- In general, when we have a cluster of computers with many CPUs, we want to use a big K to reduce the computation time
- But clearly, a bigger K leads to a bigger error between $\widehat{\beta}_{d\&c}$ and $\widehat{\beta}_{full}$.

Divide-and-conquer approach: theory

- The divide-and-conquer solution is in general different from the full-data solution.
- However, it has been proved recently that, if the size of each block is large enough relatively to the number of blocks K, then the difference is negligible.

$$n \to \infty$$

$$K \to \infty$$

$$\frac{K}{n} \to 0$$

E.g., $K = \frac{\sqrt{n}}{\log p}$

See, e.g., Battey, Fan, Liu, Lu and Zhu (2015). Distributed estimation and inference with statistical guarantees.

Subsampling approach

Subsampling approach

Basic idea

- Select randomly a subset S of data from the full dataset, then compute a solution based on this random subset. Repeat this step for R times
- Take an average of the R solutions to form the final solution - called the subsampling solution. Or update the solution sequentially in an iterative scheme (see later).

Mathematically,

- It can be proved that the subsampling solution converges to the full-data solution with probability 1 when R increases.
- This guarantees that no information in the full data is lost.

Subsampling approach

How to select a random subset S?

- Let $D = \{(x_1, y_1), ..., x_n, y_n)\}$ be the full dataset, n is large
- Introduce random variables $u_1, ..., u_n$ where

$$u_i = \begin{cases} 1, & \text{with probability } p \\ 0, & \text{with probability } 1 - p \end{cases}$$

  If $u_i = 1$ then (x_i, y_i) is included in the subset S
- On average, the size of a random subset is np.

Maximum Likelihood Estimation

Recall the logistic regression model

- Response/output variable y_i are binary: 0 and 1, Yes or No
- Want to explain/predict y_i based on a vector of input variables $\boldsymbol{x}_i = (x_{i1}, ..., x_{ip})^T$.
- Given $\boldsymbol{x}_i$, the distribution of y_i is Bernoulli with the success probability $p(x_i)$, where

$$p_1(\boldsymbol{x}_i) = \mathbb{P}(y_i = 1|\boldsymbol{x}_i) = \frac{\exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip})}{1 + \exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip})}$$

Maximum Likelihood Estimation

- The probability density function of y_i is

$$p(y_i|\boldsymbol{x}_i, \beta) = \begin{cases} p_1(\boldsymbol{x}_i), & y_i = 1 \\ 1 - p_1(\boldsymbol{x}_i), & y_i = 0 \end{cases} = p_1(\boldsymbol{x}_i)^{y_i}(1 - p_1(\boldsymbol{x}_i))^{1-y_i}$$

  So the likelihood function is

$$p(\boldsymbol{y}|X, \boldsymbol{\beta}) = \prod_{i=1}^{n} p_1(\boldsymbol{x}_i)^{y_i}(1 - p_1(\boldsymbol{x}_i))^{1-y_i}$$

- The log-likelihood (recall week 6's content)

$$\ell(\boldsymbol{\beta}) = \log p(\boldsymbol{y}|X, \boldsymbol{\beta}) = \sum_{i=1}^{n} \log p(y_i|\boldsymbol{x}_i, \beta)$$

$$= \sum_{i} (y_i \log p_1(\boldsymbol{x}_i) + (1 - y_i) \log(1 - p_1(\boldsymbol{x}_i)))$$

- MLE estimate of $\boldsymbol{\beta}$ is the value that maximises $\ell(\boldsymbol{\beta})$.

$$\widehat{\boldsymbol{\beta}} = \text{argmax}_{\beta}\{\ell(\boldsymbol{\beta})\}$$

Maximum Likelihood Estimation

The optimization problem

$$\widehat{\boldsymbol{\beta}} = \text{argmax}_{\beta}\{\ell(\boldsymbol{\beta})\}$$

is often solved by the Gradient Ascent method

- Initialization: start from some initial guess $\boldsymbol{\beta}^{(0)}$
- Iteration: Update until convergence

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + \alpha_t \nabla \ell(\boldsymbol{\beta}^{(t)}), \quad t = 0, 1, ...$$

Here

- $\nabla \ell(\boldsymbol{\beta})$ is the gradient vector of the function $\ell(\boldsymbol{\beta})$. Gradient vector points to the direction that the function increases
- α_t is a small number that controls the jump in that direction. Usually $\alpha_t = 1/(1+t)$.
- Try to implement this in Python.

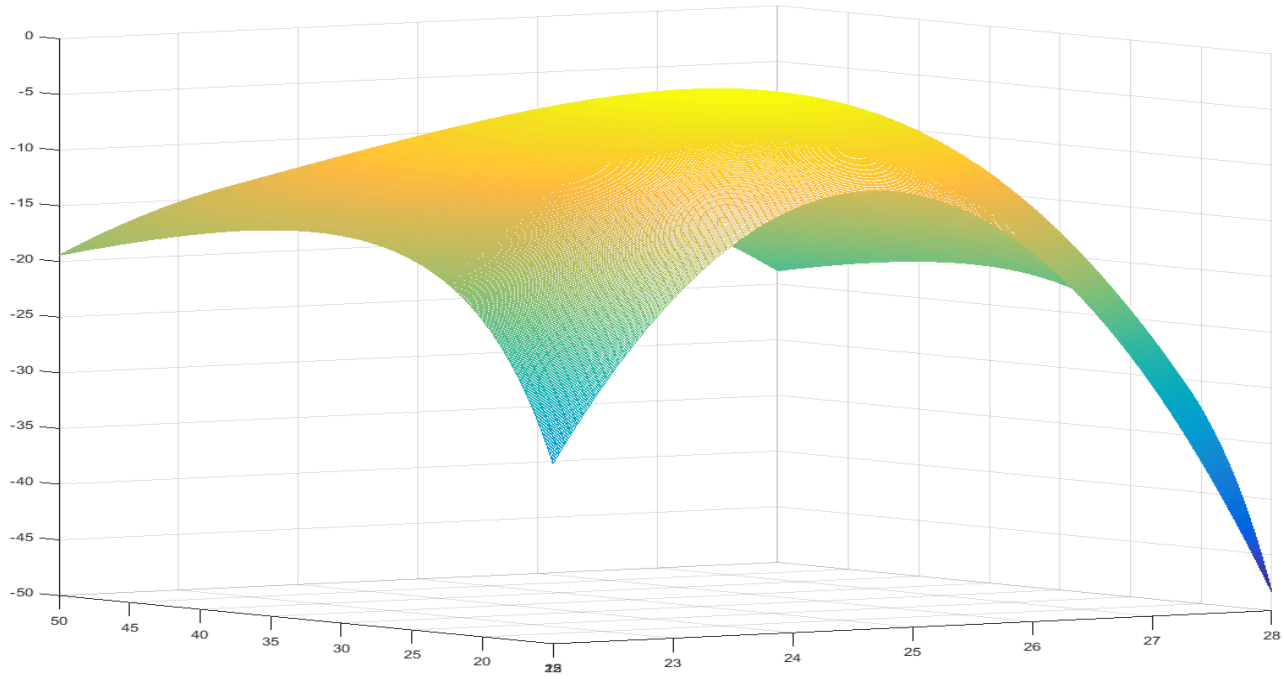


Figure: The picture shows the surface of the log-likelihood function $\ell(\beta)$. Gradient vector points to the direction that $\ell(\beta)$ increases. At the current value $\beta^{(t)}$, we move up the hill in the direction of $\nabla\ell(\beta^{(t)})$. Step size α_t controls the jump. If α_t is too large, we might jump over the hill to the other side. If α_t is too small, we might move too slowly.

Calculate the Gradient

$$\begin{aligned}\nabla \ell(\boldsymbol{\beta}) = \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \sum_{i=1}^{n} \nabla \log p(y_i | \boldsymbol{x}_i, \beta) \\ &= \sum_{i=1}^{n} \Big(y_i - p_1(\boldsymbol{x}_i) \Big) \boldsymbol{x}_i\end{aligned}$$

where

$$p_1(\boldsymbol{x}_i) = \mathbb{P}(y_i = 1 | \boldsymbol{x}_i) = \frac{\exp(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})}{1 + \exp(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})}$$

Subsampling Maximum Likelihood Estimation

- The main computational burden in this iteration procedure is computation of

$$\nabla \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \nabla \log p(y_i | \boldsymbol{x}_i, \boldsymbol{\beta})$$

- When n is huge, the iteration procedure might take years to run
- Subsampling MLE approach: compute this gradient using a random subset

$$\widehat{\nabla \ell(\boldsymbol{\beta})} = \sum_{(\boldsymbol{x}_i, y_i) \in S} \nabla \log p(y_i | \boldsymbol{x}_i, \beta)$$

Subsampling Maximum Likelihood Estimation*

The subsampling MLE algorithm

- Initialization: start from some initial guess $\boldsymbol{\beta}^{(0)}$
- Iteration:
  - Select a random subset S as above
  - Compute

$$\widehat{\nabla \ell(\boldsymbol{\beta})} = \sum_{(\boldsymbol{x}_i, y_i) \in S} \nabla \log p(y_i | \boldsymbol{x}_i, \boldsymbol{\beta})$$

  - Update

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + a_t \widehat{\nabla \ell(\boldsymbol{\beta}^{(t)})}, \quad t = 0, 1, ...$$

$\boldsymbol{\beta}^{(t)}$ is guaranteed to converge to the estimate based on the full dataset. That is, no information is lost although you don't use the whole dataset at any time

Subsampling Maximum Likelihood Estimation

Some further technical details

- Subset S must be selected at random. Every data point $(\boldsymbol{x}_i, y_i)$ must have a positive chance to be selected
- How fast the subsampling MLE converges depends heavily on the variance of the estimated gradient $\widehat{\nabla \ell(\boldsymbol{\beta})}$. Some variance reduction techniques are often needed.

Example

- We want to fit a logistic regression model to the data `InputX` and `OutputY`
- The sample size is $n = 1,000,000$. Not a very large dataset, but can be used to demonstrate the subsampling MLE

Example

```
import numpy as np
import pandas as pd

#load data

y_full = pd.read_csv('y_data.csv')
X_full = pd.read_csv('X_data.csv')
n_full = len(y_full)# the number of samples

With the above Python code, we have:
n_full= 1000000.
# p = 0.001, so 0.1% of full data is used in each iteration n_subset = p*n_full.
```

Example*

```
#Let's compare Subsampling MLE and full-data MLE

Estimate by subsampling MLE and full-data MLE:

    0.9979    0.9951
    0.5009    0.5035
   -1.9950   -1.9943

Running time by subsampling MLE and full-data MLE:

    1.2158   12.7295
```

The subsampling MLE estimate is very close to the full-data MLE estimate, but the subsampling is about 10 times faster than using the full data.

The difference is due to randomness, mathematically guaranteed to go to zero when increasing the number of iterations.

Women's labour example

The effect of subsample size, i.e. the size of the random subset S.
The numnber of repetitions $R = 1000$.

the size of S	Euclidean difference $\|\widehat{\beta}_{full} - \widehat{\beta}_{subsampling}\|$
10% of the full data	0.9463
40% of the full data	0.0946
60% of the full data	0.0624

Women's labour example

The effect of R, the size of S is fixed at 10% of the full data

R	Euclidean difference $\|\widehat{\beta}_{full} - \widehat{\beta}_{subsampling}\|$
500	1.0546
1000	0.9851
5000	0.7001

Divide-and-conquer vs subsampling

Different goals with these two approaches

- Divide-and-conquer
  - Work out analytically how parameter estimates from each split should be combined to produce the estimate you would have obtained had you used the full sample
- Subsampling
  - Each subsample to provide a good estimate of the parameter of interest.
  - Simple aggregation with means

Recall...

Two main problems with Big Data

- Speed: it typically takes a very long time to perform an analysis on the entire data
- Computer's memory: cannot load a large dataset into the computer's memory.

We have considered two solutions to the first issue: divide-and-conquer and data subsampling. Actually, these two approaches also provide a solution to the second issue as they don't require access to the entire full dataset.

We now look at the second issue from a programming point of view.

Computing techniques for Big Data

The computer's memory issue

- When the computer processes a computational task, it loads data into the so-called RAM (random access memory). RAM is not big, usually 8-16GB
- If you turn off your computer or Python session, all the data in RAM will be deleted.
- RAM not just stores your data, but many other things.
- So when your data is big, you can't load the entire dataset into RAM
- You can store data as big as you want into hard-drive devices, but not RAM.
- Think of hard-drive as books - where we can keep the entire human knowledge, and think of RAM as your brain's memory.
- We need special programming techniques to deal with this problem

The computer's memory issue

- A programming technique for dealing with this problem is MapReduce.
- Precisely speaking MapReduce is a programming model or a programming approach for dealing with big data. It's not a software or programming language
- You might have heard of Hadoop or Spark. These are software that implement MapReduce, typically on multiple-server structures.

MapReduce

- MapReduce is a programming technique for dealing with large datasets that do not fit in the computer memory
- Key idea: Divide the dataset into smaller subsets (chunks) so that each chunk can be loaded into the computer memory. Each chunk is loaded into RAM and analysed separately. The results are then combined to form the final result.
- The entire large dataset is still physically stored in the hard-drive.
- So MapReduce fits perfectly well in divide-and-conquer and subsampling approaches as they both analyse data subsets separately.