Méthodes pour l'apprentissage de données massives

Nathalie Villa-Vialaneix
http://www.nathalievilla.org

en collaboration avec Fabrice Rossi

Journées d'Études en Statistique

2-7 octobre 2016 - Fréjus

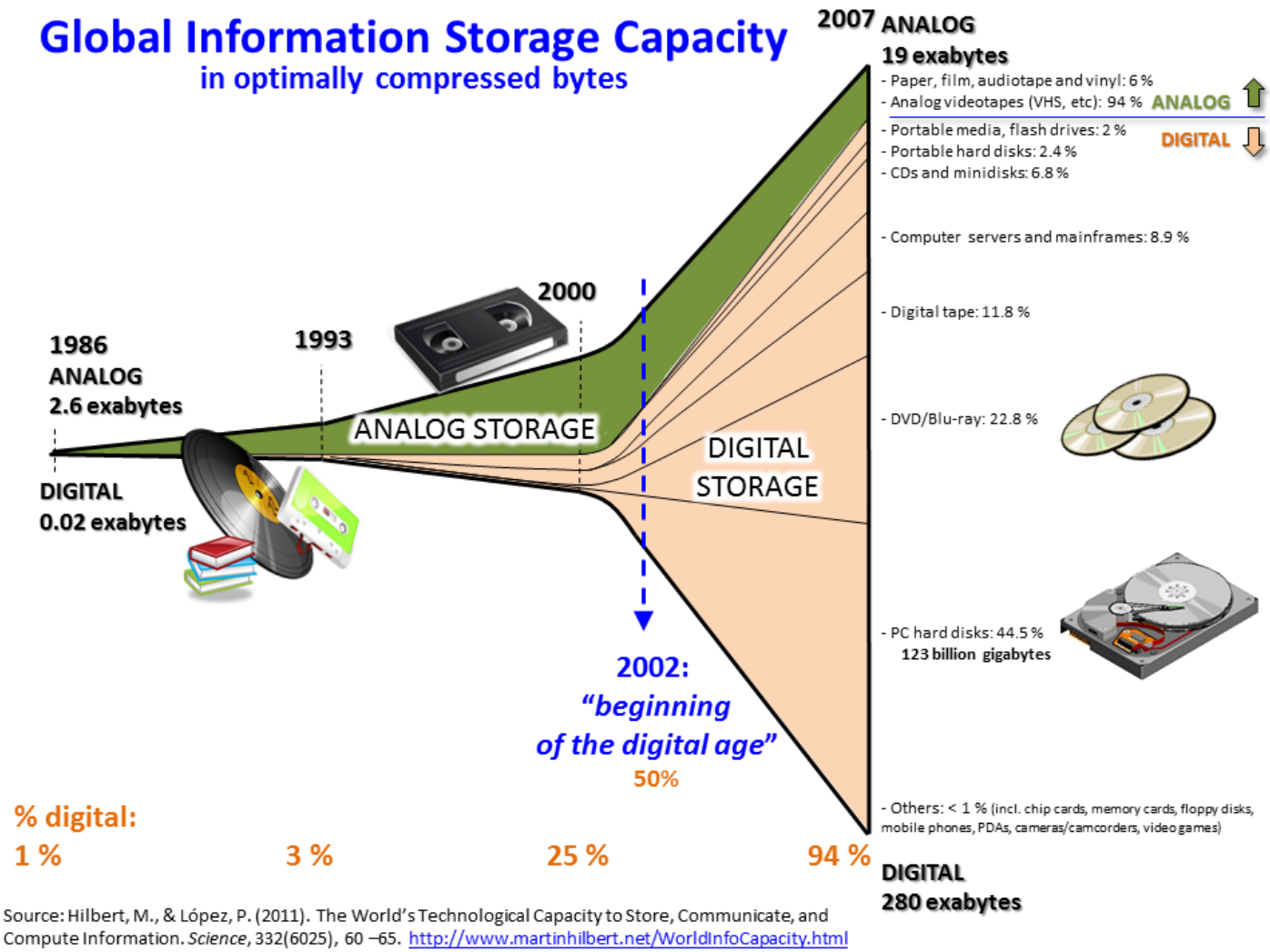
Apprentissage statistique et données massives

Sommaire

Sommaire

Big Data?

Reference to the fast and recent increase of worldwide data storage:

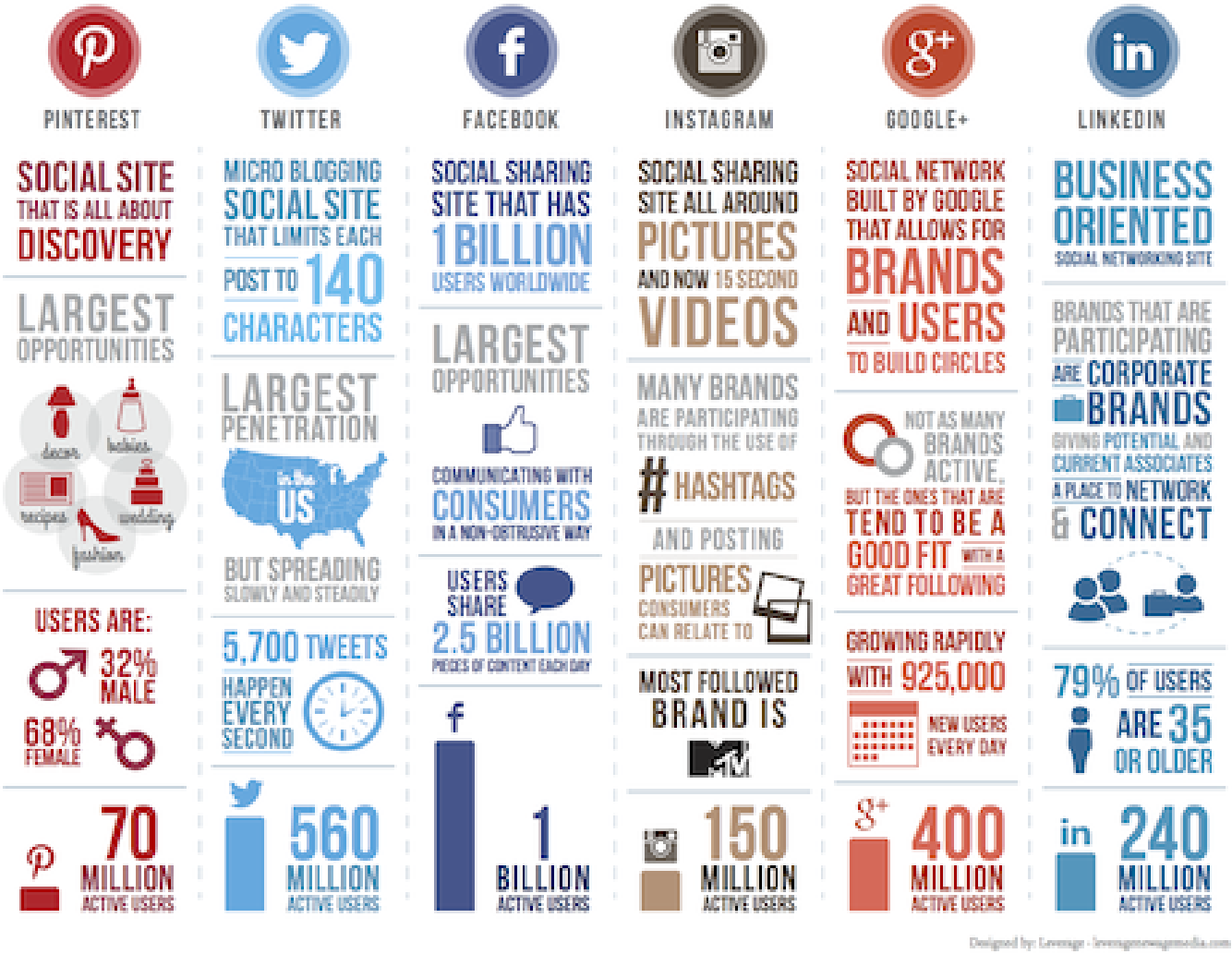


Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

Image taken from Wikipedia Commons, CC BY-SA 3.0, author: Myworkforwiki. exabyte ~ 10^{18} bits

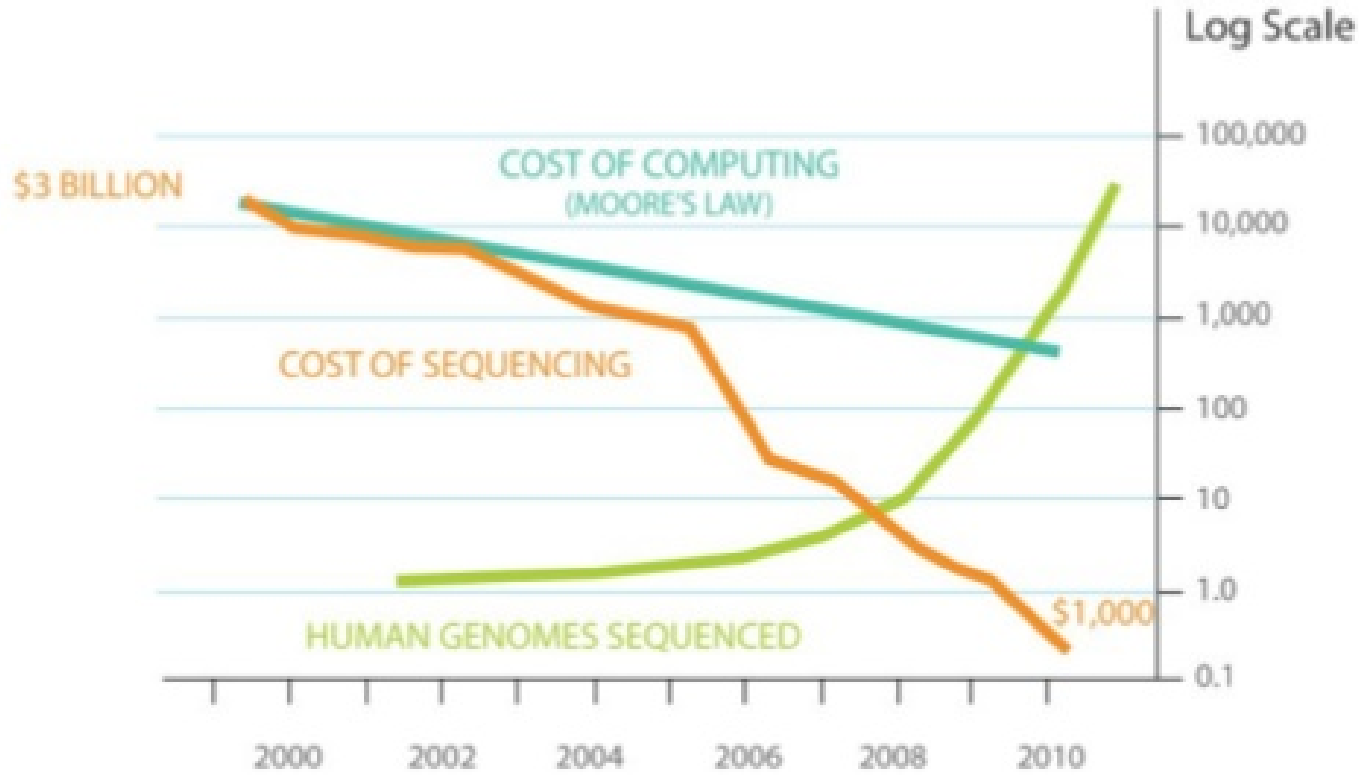
Big Data?

Reference to the fast and recent increase of worldwide data storage:



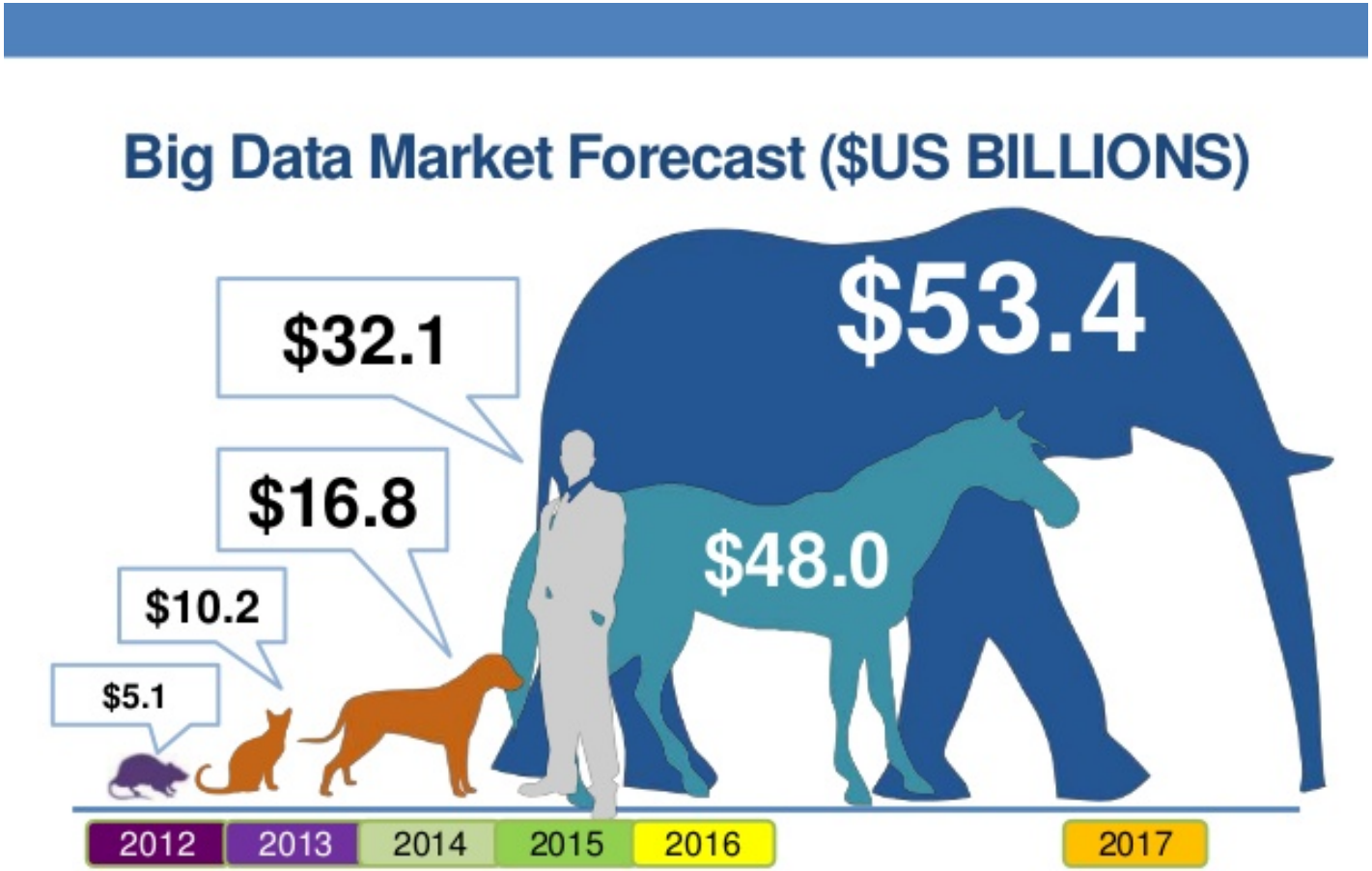
Big Data?

Reference to the fast and recent increase of worldwide data storage:



Big Data?

Reference to the fast and recent increase of worldwide data storage:



Big Data?

Standard sizes (in bits):

- 42×10^6: for the complete work of Shakespeare
- 6.4×10^9: capacity of human genome (2 bits/pb)
- 4.5×10^{16}: capacity of HD space in Google server farm in 2004
- 2×10^{17}: storage space of Megaupload when it was shut down (2012)
- 2.4×10^{18}: storage space of facebook data warehouse in 2014, with an increase of 0.6×10^{15} / day
- 1.2×10^{20} storage space of Google data warehouse in 2013

Source: https://en.wikipedia.org/wiki/Orders_of_magnitude_(data)

Big Data?

The 3V:

- **V**olume: amount of data
- **V**elocity: speed at which new data is generated
- **V**ariety: different types of data (text, images, videos, networks...)

Why are Big Data seen as an opportunity?

- economic opportunity: advertisements, recommendations, . . .
- social opportunity: better job profiling
- find new solutions to existing problems: open data websites with challenges or publication of re-use `https://www.data.gouv.fr`, `https://ressources.data.sncf.com` or `https://data.toulouse-metropole.fr`

When should we consider data as “big”?

We deal with Big Data when:

- data are at google scale (rare)
- data are big compared to our computing capacities

When should we consider data as “big”?

We deal with Big Data when:

- data are at google scale (rare)
- data are big compared to our computing capacities

[R Core Team, 2015, Kane et al., 2013]

R is not well-suited for working with data structures larger than about 10–20% of a computer’s RAM. Data exceeding 50% of available RAM are essentially unusable because the overhead of all but the simplest of calculations quickly consumes all available RAM. Based on these guidelines, we consider a data set large if it exceeds 20% of the RAM on a given machine and massive if it exceeds 50%.

When should we consider data as “big”?

We deal with Big Data when:

- data are at google scale (rare)
- data are big compared to our computing capacities ... and depending on what we need to do with them

[R Core Team, 2015, Kane et al., 2013]

R is not well-suited for working with data structures larger than about 10–20% of a computer’s RAM. Data exceeding 50% of available RAM are essentially unusable because the overhead of all but the simplest of calculations quickly consumes all available RAM. Based on these guidelines, we consider a data set large if it exceeds 20% of the RAM on a given machine and massive if it exceeds 50%.

Big Data and Statistics

Evolution of problems posed to statistics:

- "small n, small p" problems
- large dimensional problems: $p > n$ or $p \gg n$
- big data problems: n is very large

Big Data and Statistics

Evolution of problems posed to statistics:

- "small n, small p" problems
- large dimensional problems: $p > n$ or $p \gg n$
- big data problems: n is very large

[Jordan, 2013]: scale statistical methods originally designed to deal with small n and take advantage of parallel or distributed computing environments to deal with large/big n while ensuring good properties (consistency, good approximations...).

Big Data and Statistics

Evolution of problems posed to statistics:

- "small n, small p" problems
- large dimensional problems: $p > n$ or $p \gg n$
- big data problems: n is very large

[Jordan, 2013]: scale statistical methods originally designed to deal with small n and take advantage of parallel or distributed computing environments to deal with large/big n while ensuring good properties (consistency, good approximations...).
This requires a closer cooperation between statisticians and computer scientists.

Purpose of this presentation

What we will discuss

Standard approaches used to scale statistical methods with examples of applications to learning methods discussed in previous presentations.

Purpose of this presentation

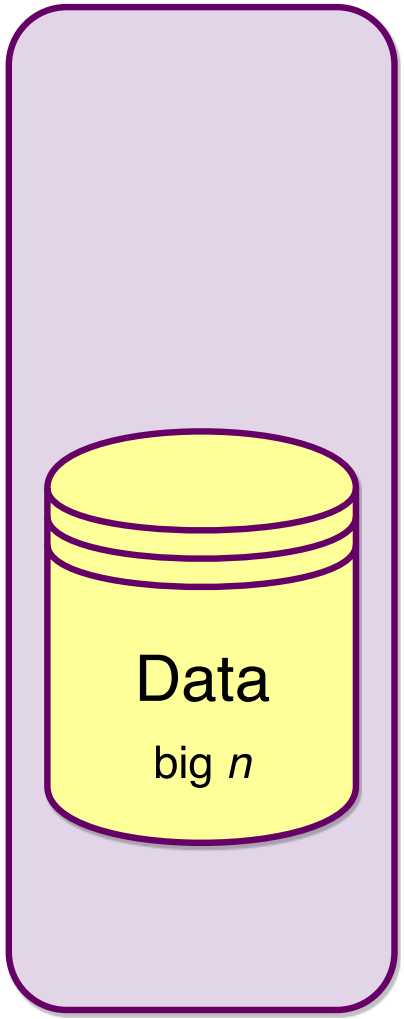
What we will discuss

Standard approaches used to scale statistical methods with examples of applications to learning methods discussed in previous presentations.

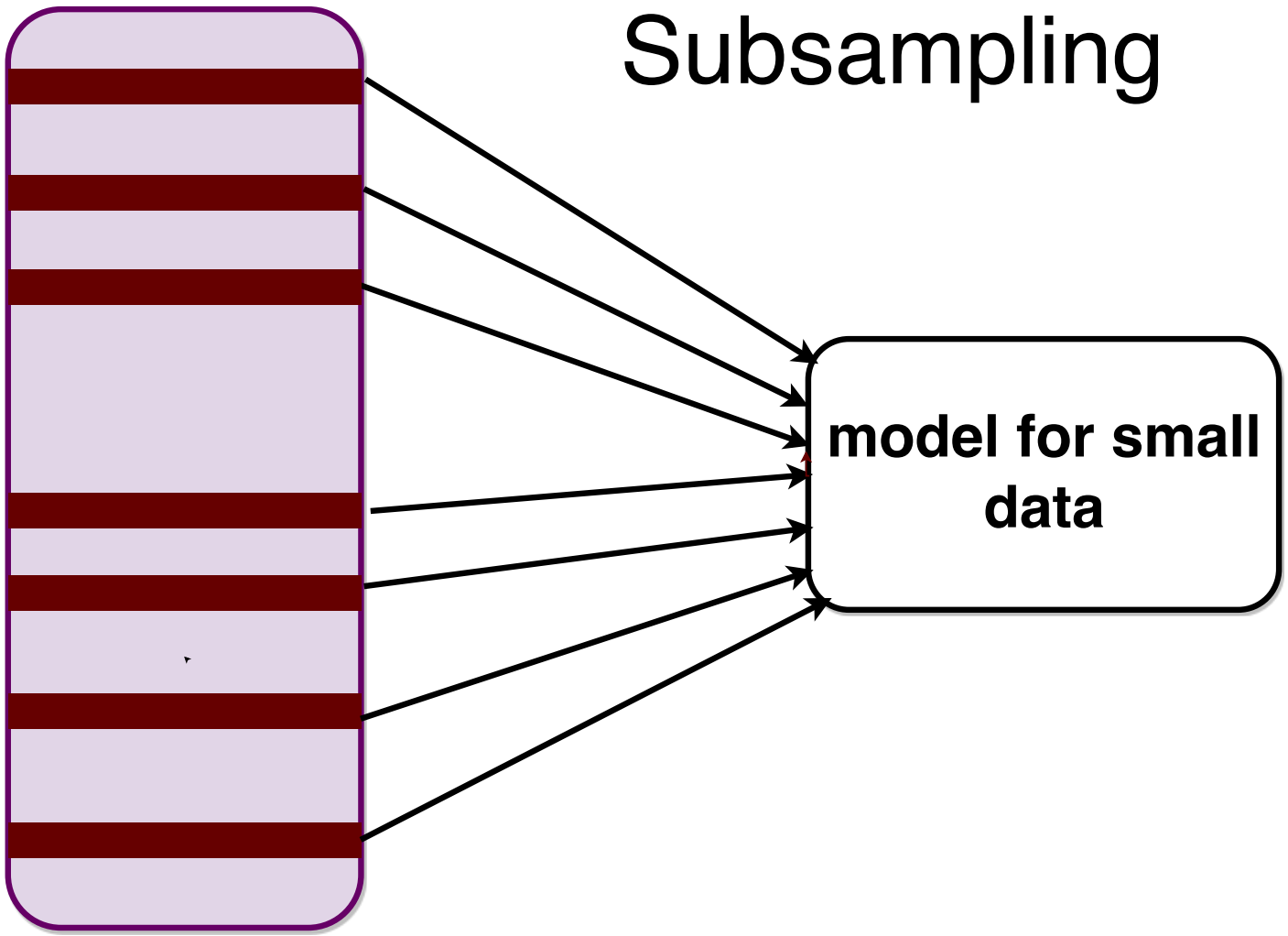
What we will not discuss

Practical implementations on various computing environments or programs in which these approaches can be used.

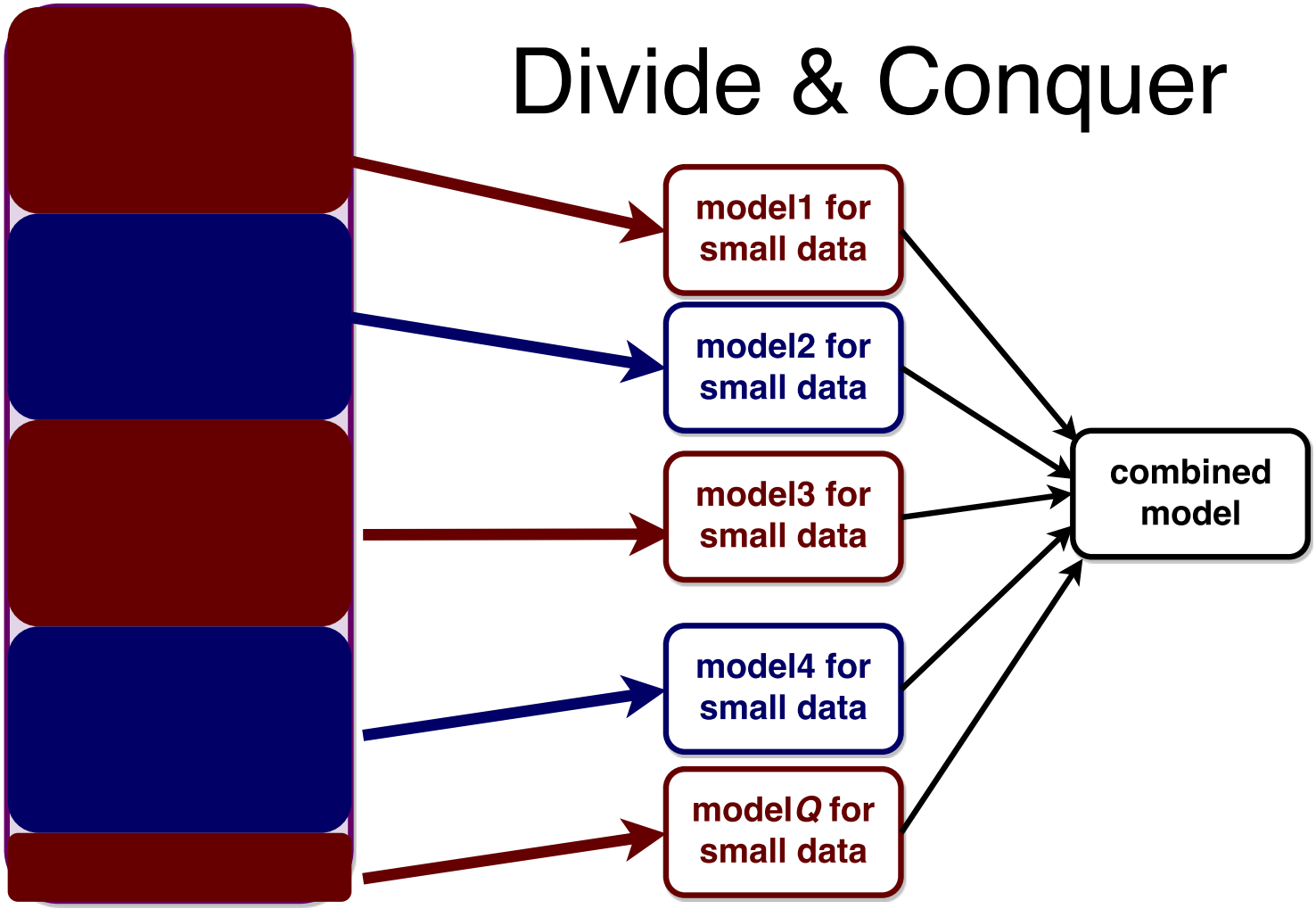
Organization of the talk



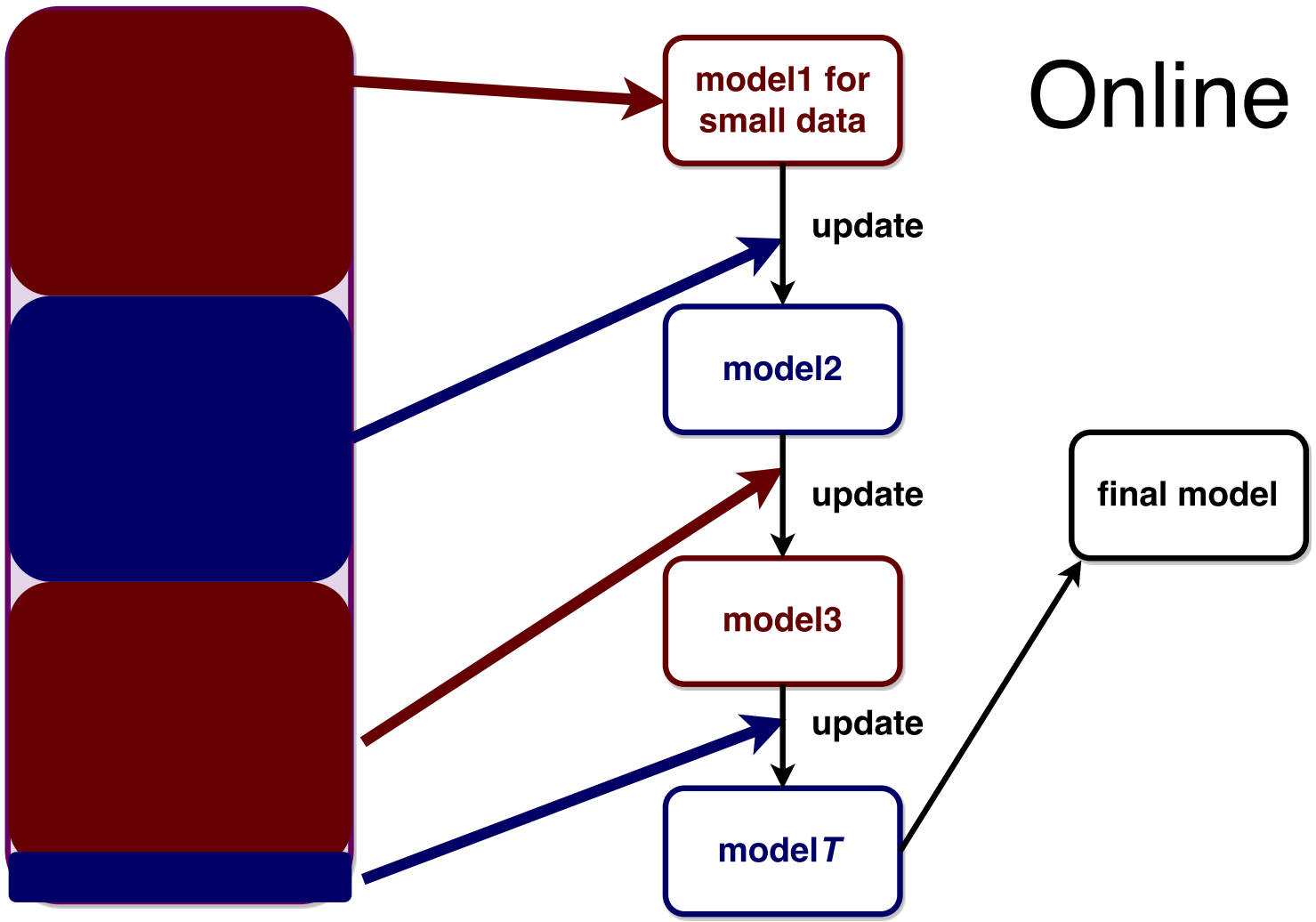
Organization of the talk



Organization of the talk



Organization of the talk



Sommaire

Overview of BLB

[Kleiner et al., 2012, Kleiner et al., 2014]

- method used to scale any bootstrap estimation
- consistency result demonstrated for a bootstrap estimation

Overview of BLB

[Kleiner et al., 2012, Kleiner et al., 2014]

- method used to scale any bootstrap estimation
- consistency result demonstrated for a bootstrap estimation

Here: we describe the approach in the simplified case of bagging (illustration for random forest)

Overview of BLB

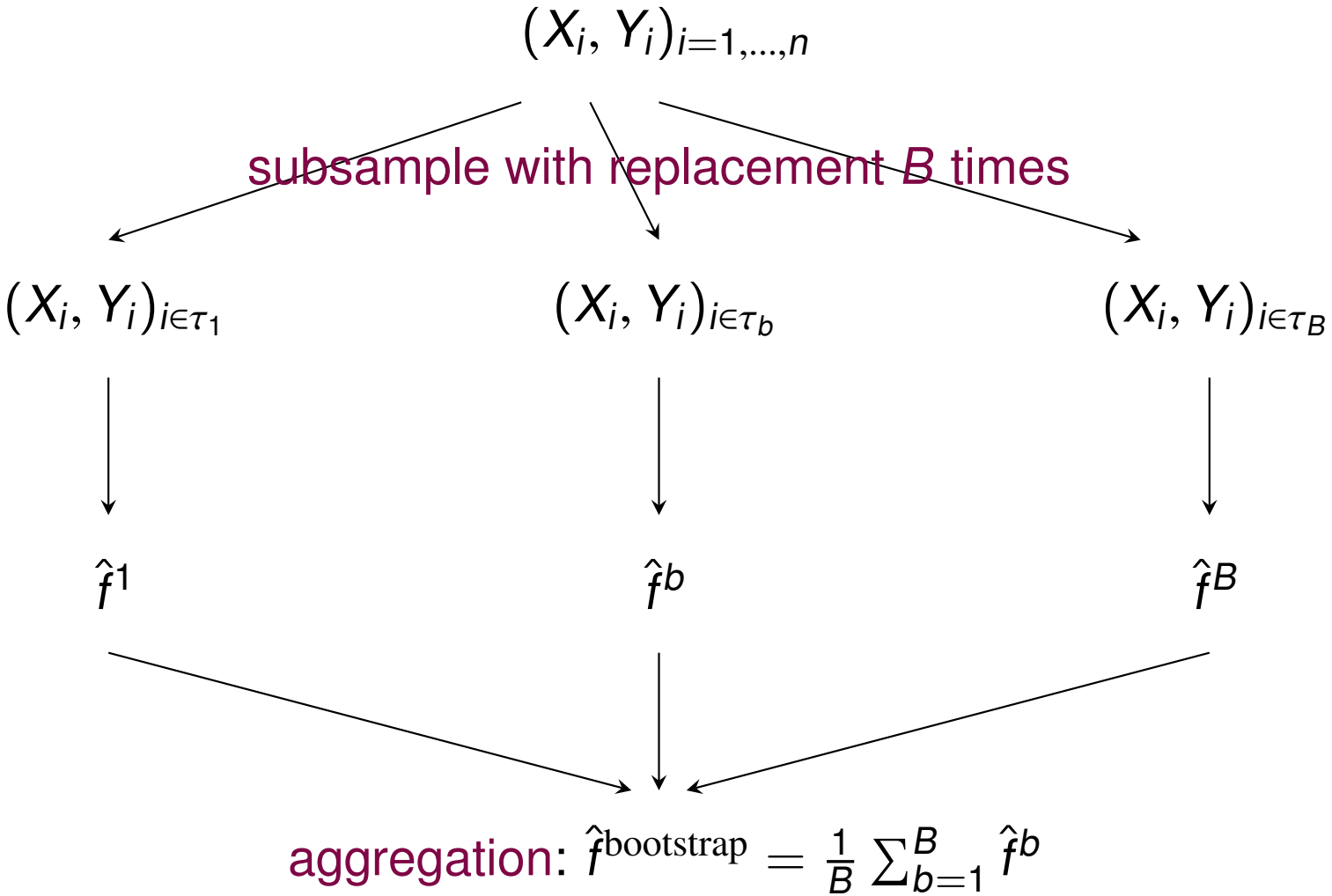
[Kleiner et al., 2012, Kleiner et al., 2014]

- method used to scale any bootstrap estimation
- consistency result demonstrated for a bootstrap estimation

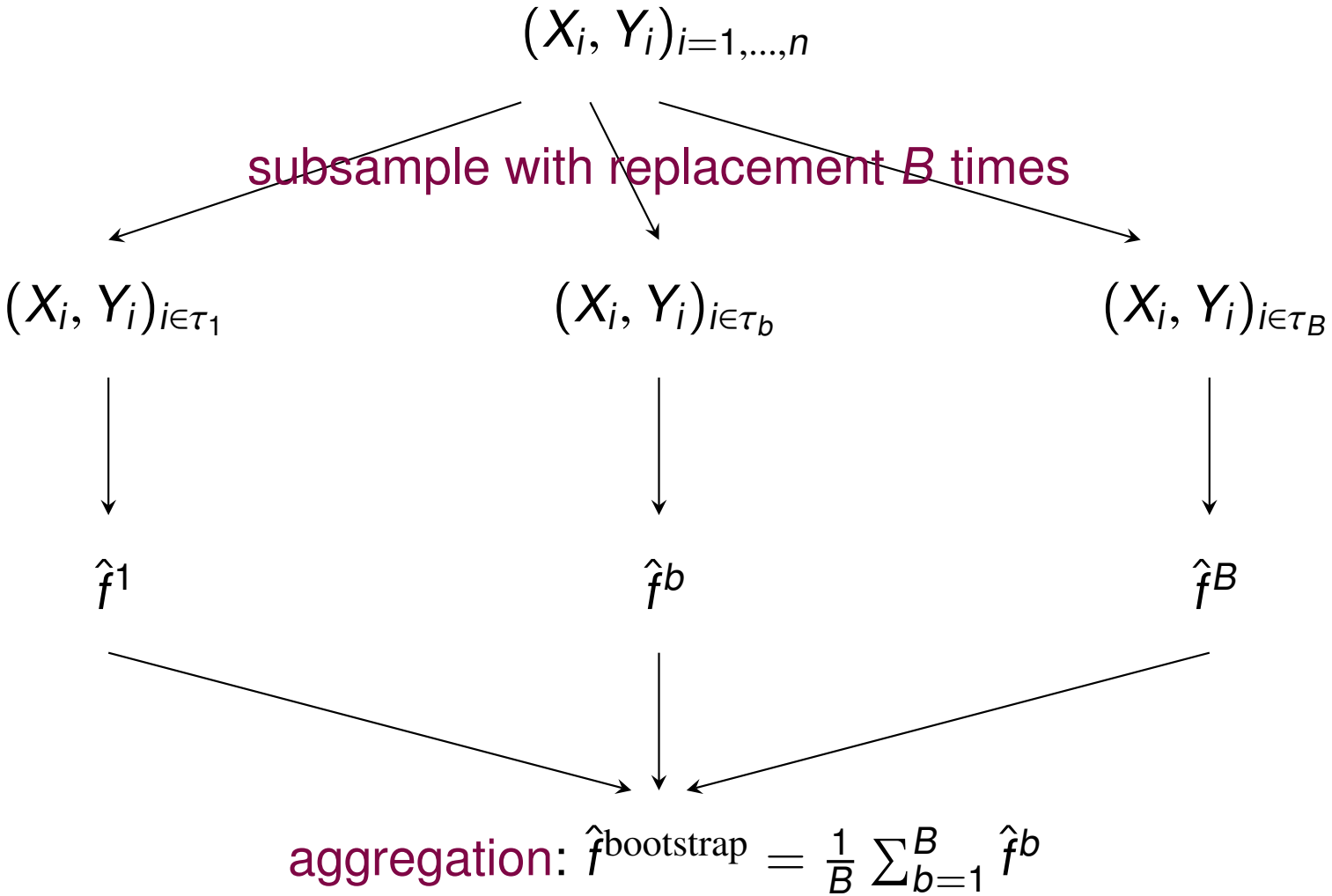
Here: we describe the approach in the simplified case of bagging (illustration for random forest)

Framework: $(X_i, Y_i)_{i=1,\dots,n}$ a learning set. We want to define a predictor of $Y \in \mathbb{R}$ from X given the learning set.

Standard bagging



Standard bagging



Advantage for Big Data: Bootstrap estimators can be learned in parallel.

Problem with standard bagging

When n is big, the number of different observations in τ_b is $\sim 0.63n \Rightarrow$ still BIG!

Problem with standard bagging

When n is big, the number of different observations in τ_b is $\sim 0.63n \Rightarrow$ still BIG!

First solution...: **[Bickel et al., 1997]** propose the "m-out-of-n" bootstrap: bootstrap samples have size m with $m \ll n$

Problem with standard bagging

When n is big, the number of different observations in τ_b is $\sim 0.63n \Rightarrow$ still BIG!

First solution...: **[Bickel et al., 1997]** propose the "m-out-of-n" bootstrap: bootstrap samples have size m with $m \ll n$
But: The quality of the estimator strongly depends on m!

Problem with standard bagging

When n is big, the number of different observations in τ_b is $\sim 0.63n \Rightarrow$ still BIG!

First solution...: **[Bickel et al., 1997]** propose the "m-out-of-n" bootstrap: bootstrap samples have size m with $m \ll n$
But: The quality of the estimator strongly depends on m!

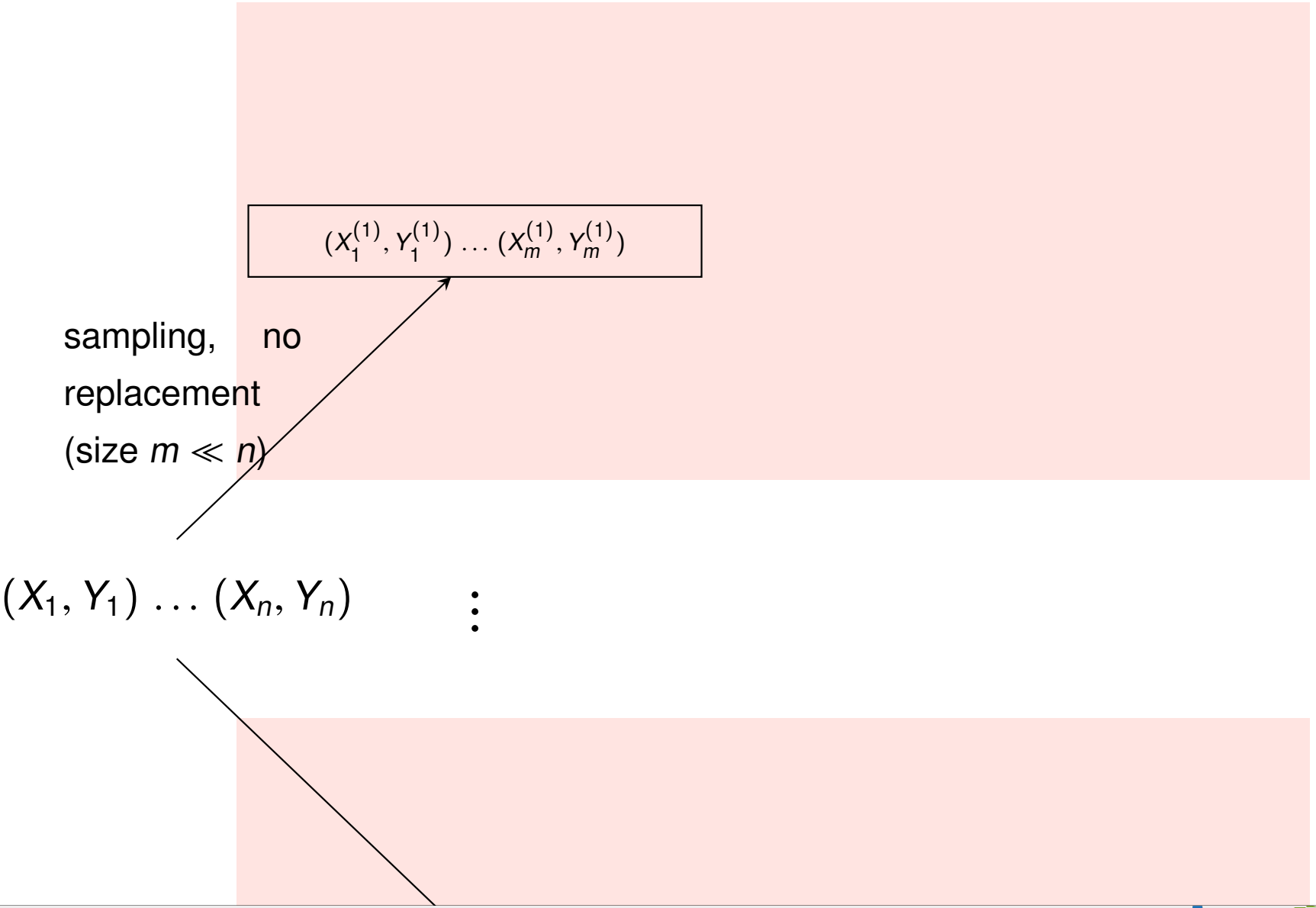
Idea behind BLB

Use bootstrap samples having size n but with a very small number of different observations in each of them.

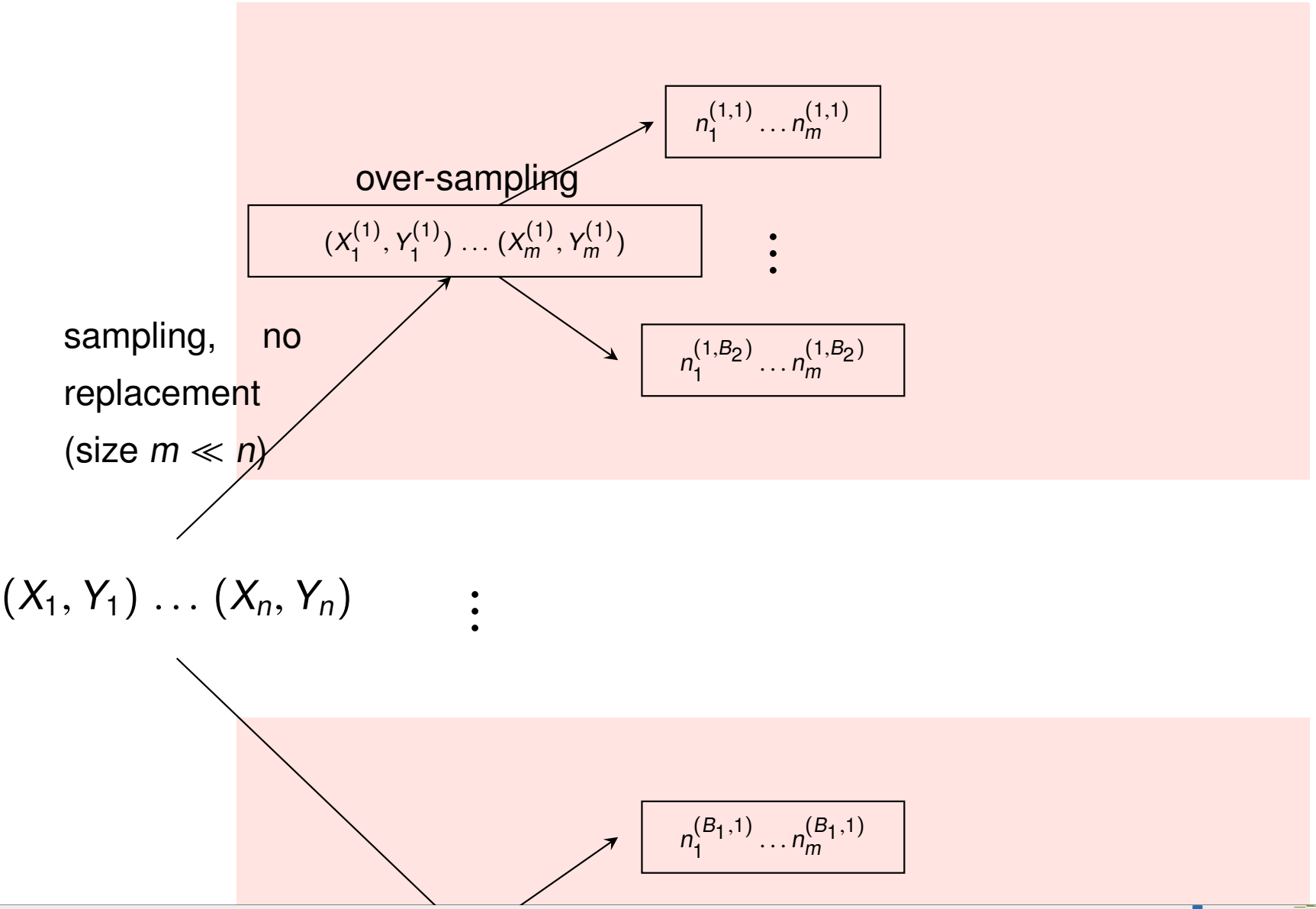
Presentation of BLB

$(X_1, Y_1) \ldots (X_n, Y_n)$

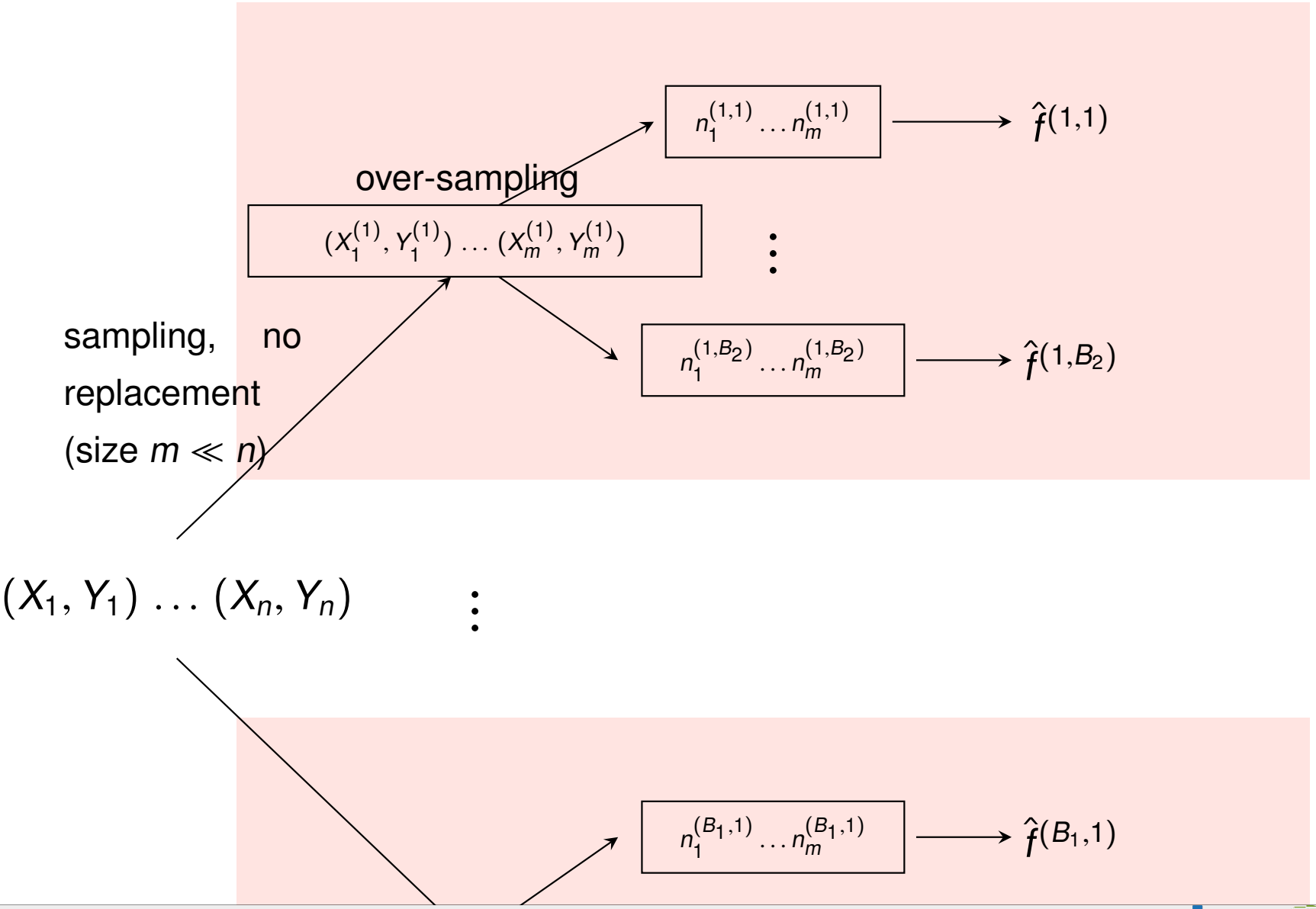
Presentation of BLB



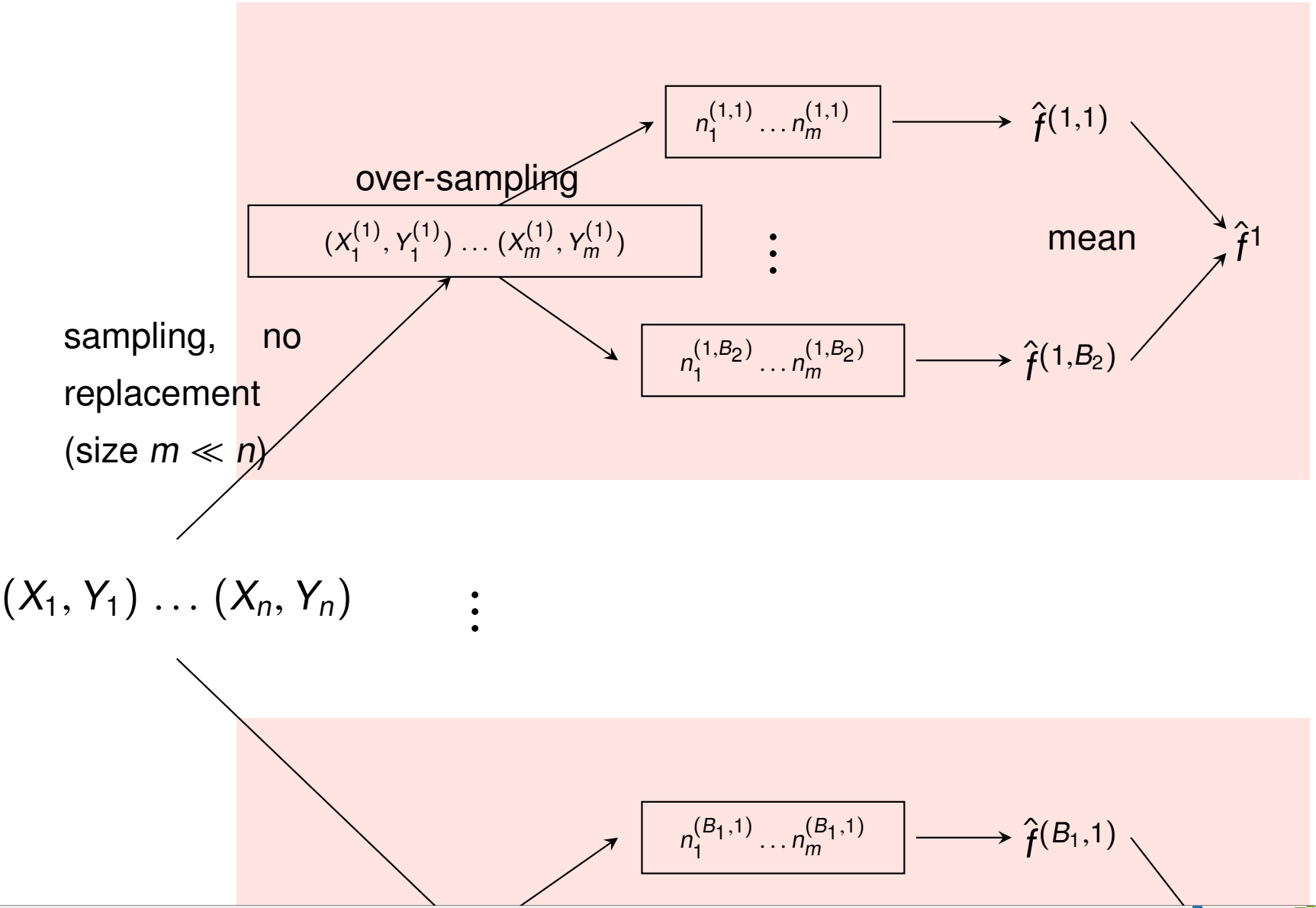
Presentation of BLB



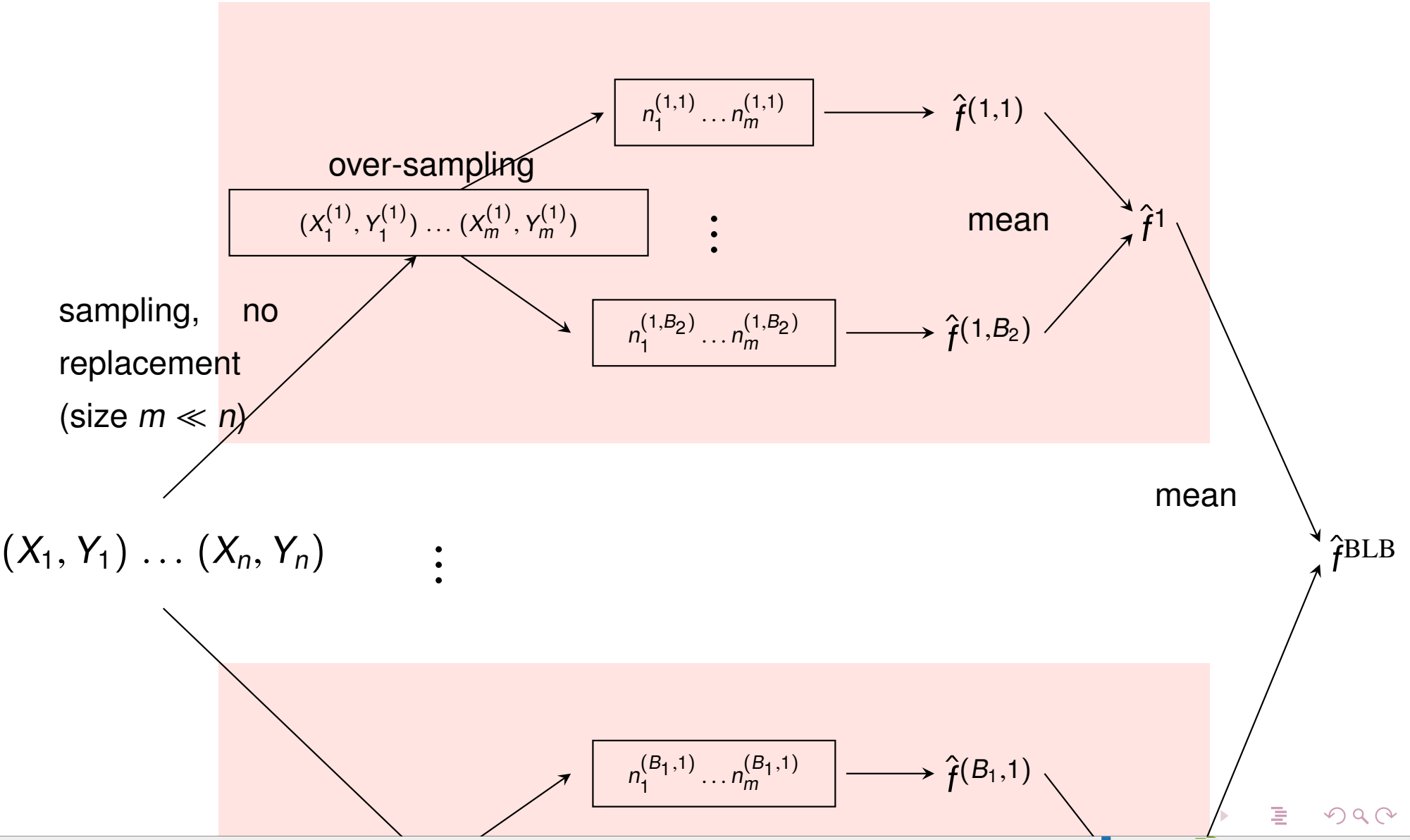
Presentation of BLB



Presentation of BLB



Presentation of BLB



What is over-sampling and why is it working?

BLB steps:

1. create B_1 samples (without replacement) of size $m \sim n^\gamma$ (with $\gamma \in [0.5, 1]$: for $n = 10^6$ and $\gamma = 0.6$, typical m is about 4000, compared to 630 000 for standard bootstrap

What is over-sampling and why is it working?

BLB steps:

1. create B_1 samples (without replacement) of size $m \sim n^\gamma$ (with $\gamma \in [0.5, 1]$

2. for every subsample τ_b, repeat B_2 times:
   - over-sampling: affect weights $(n_1, \ldots, n_m)$ simulated as $\mathcal{M}\left(n, \frac{1}{m}\mathbb{1}_m\right)$ to observations in τ_b

What is over-sampling and why is it working?

BLB steps:

1. create B_1 samples (without replacement) of size $m \sim n^\gamma$ (with $\gamma \in [0.5, 1]$

2. for every subsample τ_b, repeat B_2 times:
   - over-sampling: affect weights $(n_1, \ldots, n_m)$ simulated as $\mathcal{M}\left(n, \frac{1}{m}\mathbb{1}_m\right)$ to observations in τ_b
   - estimation step: train an estimator with weighted observations (if the learning algorithm allows a genuine processing of weights, computational cost is low because of the small size of m)

What is over-sampling and why is it working?

BLB steps:

1. create B_1 samples (without replacement) of size $m \sim n^\gamma$ (with $\gamma \in [0.5, 1]$

2. for every subsample τ_b, repeat B_2 times:
   - over-sampling: affect weights $(n_1, \ldots, n_m)$ simulated as $\mathcal{M}\left(n, \frac{1}{m}\mathbb{1}_m\right)$ to observations in τ_b
   - estimation step: train an estimator with weighted observations

3. aggregate by averaging

What is over-sampling and why is it working?

BLB steps:

1. create B_1 samples (without replacement) of size $m \sim n^\gamma$ (with $\gamma \in [0.5, 1]$

2. for every subsample τ_b, repeat B_2 times:
   - over-sampling: affect weights $(n_1, \ldots, n_m)$ simulated as $\mathcal{M}\left(n, \frac{1}{m}\mathbb{1}_m\right)$ to observations in τ_b
   - estimation step: train an estimator with weighted observations

3. aggregate by averaging

Remark: Final sample size $(\sum_{i=1}^m n_i)$ is equal to n (with replacement) as in standard bootstrap samples.

Large scale kernel methods

Given a training set $(X_i, Y_i)_{i=1,\dots,n}$ ($X \in \mathcal{X}$ and $Y \in \mathbb{R}$ or $\{0, \dots, K-1\}$), kernel methods (such as SVM) are based on the definition of a kernel: $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is symmetric ($K(x, x') = K(x', x)$) and positive ($\sum_{i,i'=1}^{N} \alpha_i \alpha_{i'} K(x_i, x_{i'}) \geq 0$).

Large scale kernel methods

Given a training set $(X_i, Y_i)_{i=1,\dots,n}$ ($X \in \mathcal{X}$ and $Y \in \mathbb{R}$ or $\{0, \dots, K-1\}$), kernel methods (such as SVM) are based on the definition of a kernel: $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

This is equivalent to embed $\mathcal{X}$ into an (implicit) Hilbert space in which standard (often linear) statistical methods can be used with the kernel trick:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$$

Large scale kernel methods

Given a training set $(X_i, Y_i)_{i=1,\dots,n}$ ($X \in \mathcal{X}$ and $Y \in \mathbb{R}$ or $\{0, \dots, K-1\}$), kernel methods (such as SVM) are based on the definition of a kernel: $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

This is equivalent to embed $\mathcal{X}$ into an (implicit) Hilbert space in which standard (often linear) statistical methods can be used with the kernel trick:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$$

Standard complexity (number of elementary operations) of kernel learning methods: $\mathcal{O}(n^2)$ or even $\mathcal{O}(n^3)$ (ex: K-PCA)

Low rank approximation solutions

Aim: approximate $\mathbf{K} = (K(X_i, X_{i'})_{i,i'=1,\ldots,n}$ with a low rank matrix (matrix with rank $k \ll n$.

Low rank approximation solutions

Aim: approximate $\mathbf{K} = (K(X_i, X_{i'})_{i,i'=1,\ldots,n}$ with a low rank matrix (matrix with rank $k \ll n$.

Then, use the approximation to train your predictor and correct it to "re-scale" it to n. Typical computational cost: $O(nk^2)$.

Sketch of Nyström approximation

[Williams and Seeger, 2000, Drineas and Mahoney, 2005]

- Pick at random m observations in $\{1, \ldots, m\}$ (without loss of generality suppose that the first m ones have been chosen).

Sketch of Nyström approximation

[Williams and Seeger, 2000, Drineas and Mahoney, 2005]

- Pick at random m observations in $\{1, \ldots, m\}$ (without loss of generality suppose that the first m ones have been chosen).

- Re-write:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}^{(m)} & \mathbf{K}^{(m,n-m)} \\ \mathbf{K}^{(n-m,m)} & \mathbf{K}^{(n-m,n-m)} \end{bmatrix} \quad \text{et} \quad \mathbf{K}^{(n,m)} = \begin{bmatrix} \mathbf{K}^{(m)} \\ \mathbf{K}^{(n-m,m)} \end{bmatrix},$$

with $\mathbf{K}^{(n-m,m)} = (\mathbf{K}^{(m,n-m)})^\top$, and use $\mathbf{K}^{(m)}$ instead of $\mathbf{K}$.

Approximate spectral decomposition of **K**

Notations:

$\mathbf{K}$	$\mathbf{K}^{(m)}$
eigenvectors: $(v_j)_{j=1,\dots,n}$	eigenvectors: $(v_j^{(m)})_{j=1,\dots,m}$
eigenvalues: $(\lambda_j)_{j=1,\dots,n}$ (positive, decreasing order)	eigenvalues: $(\lambda_j^{(m)})_{j=1,\dots,m}$ (positive, decreasing order)

Approximate spectral decomposition of **K**

Notations:

$\mathbf{K}$	$\mathbf{K}^{(m)}$
eigenvectors: $(v_j)_{j=1,\dots,n}$	eigenvectors: $(v_j^{(m)})_{j=1,\dots,m}$
eigenvalues: $(\lambda_j)_{j=1,\dots,n}$ (positive, decreasing order)	eigenvalues: $(\lambda_j^{(m)})_{j=1,\dots,m}$ (positive, decreasing order)

$$\forall\, j = 1, \dots, m, \qquad \mu_j \simeq \frac{n}{m}\mu_j^{(m)} \qquad \text{and} \qquad v_j \simeq \sqrt{\frac{m}{n}}\frac{1}{\mu_j^{(m)}}\mathbf{K}^{(n,m)}v_j^{(m)}$$

Approximate spectral decomposition of $\mathbf{K}$

Notations:

$\mathbf{K}$	$\mathbf{K}^{(m)}$
eigenvectors: $(v_j)_{j=1,\dots,n}$	eigenvectors: $(v_j^{(m)})_{j=1,\dots,m}$
eigenvalues: $(\lambda_j)_{j=1,\dots,n}$ (positive, decreasing order)	eigenvalues: $(\lambda_j^{(m)})_{j=1,\dots,m}$ (positive, decreasing order)

$$\forall\, j = 1, \dots, m, \qquad \mu_j \simeq \frac{n}{m}\mu_j^{(m)} \qquad \text{and} \qquad v_j \simeq \sqrt{\frac{m}{n}}\frac{1}{\mu_j^{(m)}}\mathbf{K}^{(n,m)}v_j^{(m)}$$

- complexity of the direct calculation: $\mathcal{O}(n^3)$
- complexity of the approximate solution: $\mathcal{O}(m^3) + \mathcal{O}(nm^2)$

Approximate spectral decomposition of **K**

Notations:

$\mathbf{K}$	$\mathbf{K}^{(m)}$
eigenvectors: $(v_j)_{j=1,\dots,n}$	eigenvectors: $(v_j^{(m)})_{j=1,\dots,m}$
eigenvalues: $(\lambda_j)_{j=1,\dots,n}$ (positive, decreasing order)	eigenvalues: $(\lambda_j^{(m)})_{j=1,\dots,m}$ (positive, decreasing order)

$$\forall\, j = 1, \dots, m, \qquad \mu_j \simeq \frac{n}{m}\mu_j^{(m)} \qquad \text{and} \qquad v_j \simeq \sqrt{\frac{m}{n}}\frac{1}{\mu_j^{(m)}}\mathbf{K}^{(n,m)}v_j^{(m)}$$

Remark: When the rank of $\mathbf{K}$ is $< m$, the approximation is exact.

Approximate spectral decomposition of $\mathbf{K}$

Notations:

$\mathbf{K}$	$\mathbf{K}^{(m)}$
eigenvectors: $(v_j)_{j=1,\dots,n}$	eigenvectors: $(v_j^{(m)})_{j=1,\dots,m}$
eigenvalues: $(\lambda_j)_{j=1,\dots,n}$ (positive, decreasing order)	eigenvalues: $(\lambda_j^{(m)})_{j=1,\dots,m}$ (positive, decreasing order)

$$\forall\, j = 1, \dots, m, \qquad \mu_j \simeq \frac{n}{m}\mu_j^{(m)} \qquad \text{and} \qquad v_j \simeq \sqrt{\frac{m}{n}}\frac{1}{\mu_j^{(m)}}\mathbf{K}^{(n,m)}v_j^{(m)}$$

Notations: $\mu^{(n,m)} = \frac{n}{m}\mu_j^{(m)}$ and $v_j^{(n,m)} = \sqrt{\frac{m}{n}}\frac{1}{\mu_j^{(m)}}\mathbf{K}^{(n,m)}v_j^{(m)}$

Low rank approximation of $\mathbf{K}$

A rank k (for $k \leq m$) approximation of $\mathbf{K}$ is obtained with:

$$\widetilde{\mathbf{K}} = \mathbf{K}^{(n,m)} \left(\mathbf{K}_k^{(m)}\right)^+ \mathbf{K}^{(m,n)}$$

in which $\mathbf{K}_k^{(m)}$ is the best rank k approximation of $\mathbf{K}^{(m)}$ and $\left(\mathbf{K}_k^{(m)}\right)^+$ is its pseudo inverse.

Low rank approximation of $\mathbf{K}$

A rank k (for $k \leq m$) approximation of $\mathbf{K}$ is obtained with:

$$\widetilde{\mathbf{K}} = \mathbf{K}^{(n,m)} \left(\mathbf{K}_k^{(m)}\right)^{+} \mathbf{K}^{(m,n)}$$

in which $\mathbf{K}_k^{(m)}$ is the best rank k approximation of $\mathbf{K}^{(m)}$ and $\left(\mathbf{K}_k^{(m)}\right)^{+} = \sum_{j=1}^{k} \left(\mu_j^{(m)}\right)^{-1} v_j^{(m)} \left(v_j^{(m)}\right)^{\top}$.

Low rank approximation of $\mathbf{K}$

A rank k (for $k \leq m$) approximation of $\mathbf{K}$ is obtained with:

$$\widetilde{\mathbf{K}} = \sum_{j=1}^{k} \mu_j^{(n,m)} v_j^{(n,m)} \left(v_j^{(n,m)} \right)^\top$$

Low rank approximation of $\mathbf{K}$

A rank k (for $k \leq m$) approximation of $\mathbf{K}$ is obtained with:

$$\widetilde{\mathbf{K}} = \sum_{j=1}^{k} \mu_j^{(n,m)} v_j^{(n,m)} \left(v_j^{(n,m)}\right)^\top$$

Quality of the approximation; [Cortes et al., 2010, Bach, 2013]

With probability at least $1 - \delta$,

$$\|\widetilde{\mathbf{K}} - \mathbf{K}\|_2 \leq \|\mathbf{K}_k - \mathbf{K}\|_2 + \frac{n}{\sqrt{m}} \max_{i=1,\ldots,n} \mathbf{K}_{ii} \left(2 + \log\left(\frac{1}{\delta}\right)\right)$$

where $\mathbf{K}_k$ is the best rank k approximation of $\mathbf{K}$ and $\mathbf{K}_{ii}$ are diagonal terms in $\mathbf{K}$.

Application to kernel methods

Example of the kernel ridge regression [Saunders et al., 1998]

KRR is a kernel regression method with simply performs a linear regression in the feature space, regularized by L_2 norm:

$$\arg\min_{w \in \mathcal{H}} \sum_{i=1}^{n} \left(Y_i - \langle w, \phi(x_i) \rangle_{\mathcal{H}} \right)^2 + \frac{\lambda}{2} \|w\|_{\mathcal{H}}^2$$

Application to kernel methods

Example of the kernel ridge regression [Saunders et al., 1998]

KRR is a kernel regression method with simply performs a linear regression in the feature space, regularized by L_2 norm:

$$\underset{w \in \mathcal{H}}{\arg\min} \sum_{i=1}^{n} (Y_i - \langle w, \phi(x_i) \rangle_{\mathcal{H}})^2 + \frac{\lambda}{2} \|w\|_{\mathcal{H}}^2$$

Solution:

$$\hat{f} : \; x \in \mathcal{X} \to \langle w, \phi(x_i) \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i K(X_i, x)$$

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = (\mathbf{K} + \lambda \mathbb{I}_n)^{-1} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

Application to kernel methods

Example of the kernel ridge regression [Saunders et al., 1998]

KRR is a kernel regression method with simply performs a linear regression in the feature space, regularized by L_2 norm:

$$\arg\min_{w \in \mathcal{H}} \sum_{i=1}^{n} (Y_i - \langle w, \phi(x_i) \rangle_{\mathcal{H}})^2 + \frac{\lambda}{2} \|w\|_{\mathcal{H}}^2$$

Solution:

$$\hat{f} : \; x \in \mathcal{X} \to \langle w, \phi(x_i) \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i K(X_i, x)$$

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = (\mathbf{K} + \lambda \mathbb{I}_n)^{-1} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

The inverse of $(\mathbf{K} + \lambda \mathbb{I}_n)$ is computationally very costly!

Application to kernel methods

Example of the kernel ridge regression [Saunders et al., 1998]

Nyström approximation: replace $\mathbf{K}$ by its Nyström rank k approximation...
α has the explicit form:

$$\tilde{\alpha} = \frac{1}{\lambda}\left(\mathbf{Y} - \mathbf{V}_k\left(\lambda \mathbb{I}_n + \Lambda_k \mathbf{V}_k^\top \mathbf{V}_k\right)^{-1} \Lambda_k \mathbf{V}_k^\top \mathbf{Y}\right),$$

Application to kernel methods

Example of the kernel ridge regression [Saunders et al., 1998]

Nyström approximation: replace $\mathbf{K}$ by its Nyström rank k approximation...
α has the explicit form:

$$\tilde{\alpha} = \frac{1}{\lambda}\left(\mathbf{Y} - \mathbf{V}_k\left(\lambda \mathbb{I}_n + \Lambda_k \mathbf{V}_k^\top \mathbf{V}_k\right)^{-1} \Lambda_k \mathbf{V}_k^\top \mathbf{Y}\right),$$

Quality of the prediction is also controlled in this approximation (see **[Cortes et al., 2010, Bach, 2013]**).
Similar methods exist to use Nyström approximation in SVM.

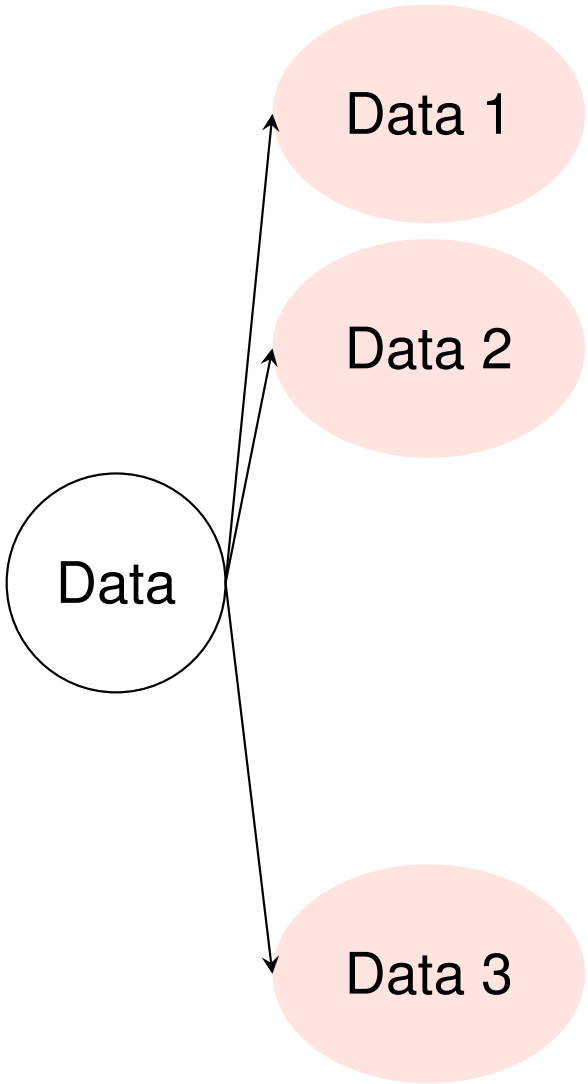
Sommaire

Overview of Map Reduce

Map Reduce is a generic method to deal with massive datasets stored on a distributed filesystem.

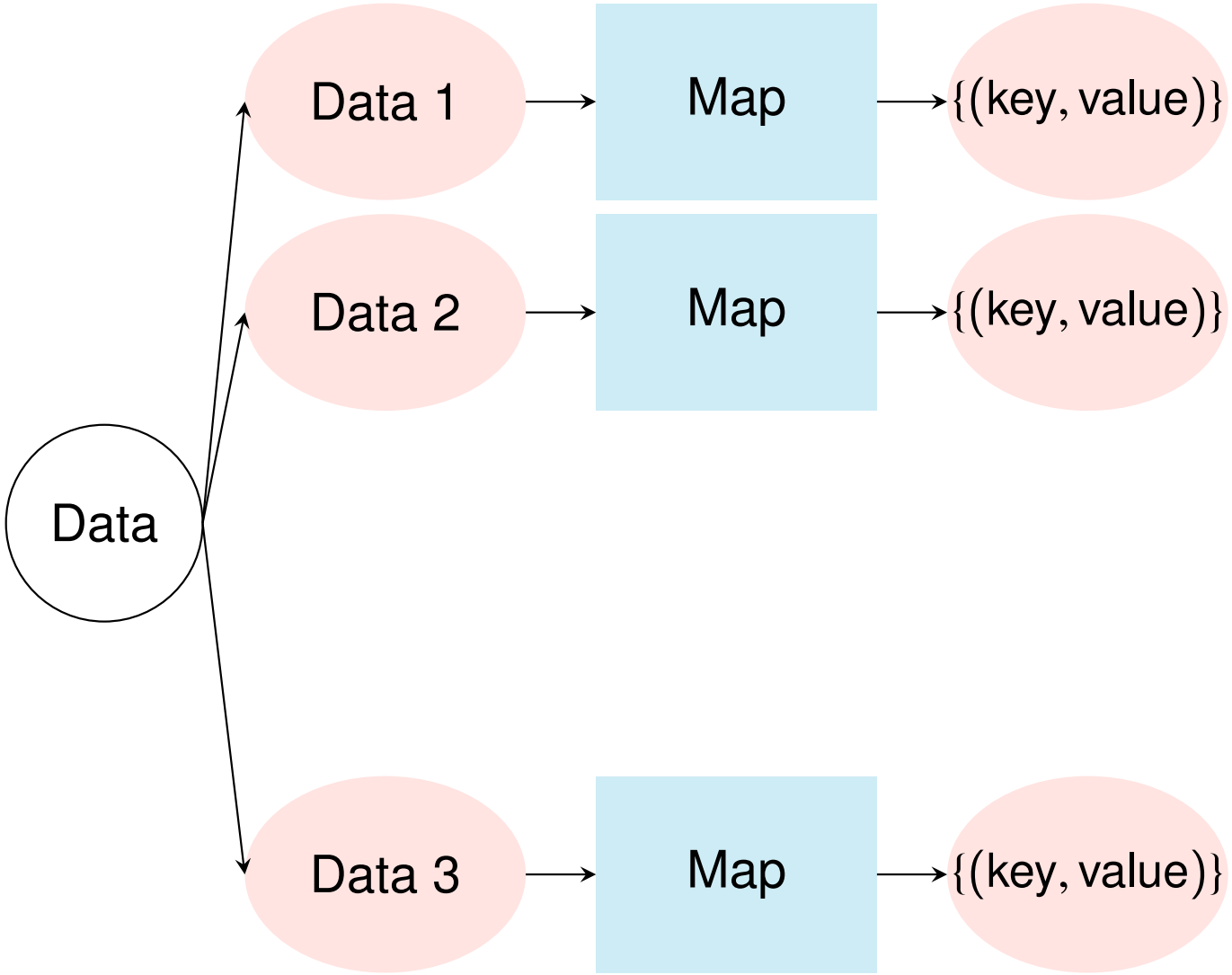
It has been developped by Google™ **[Dean and Ghemawat, 2004]** (see also **[Chamandy et al., 2012]** for example of use at Google).

Overview of Map Reduce



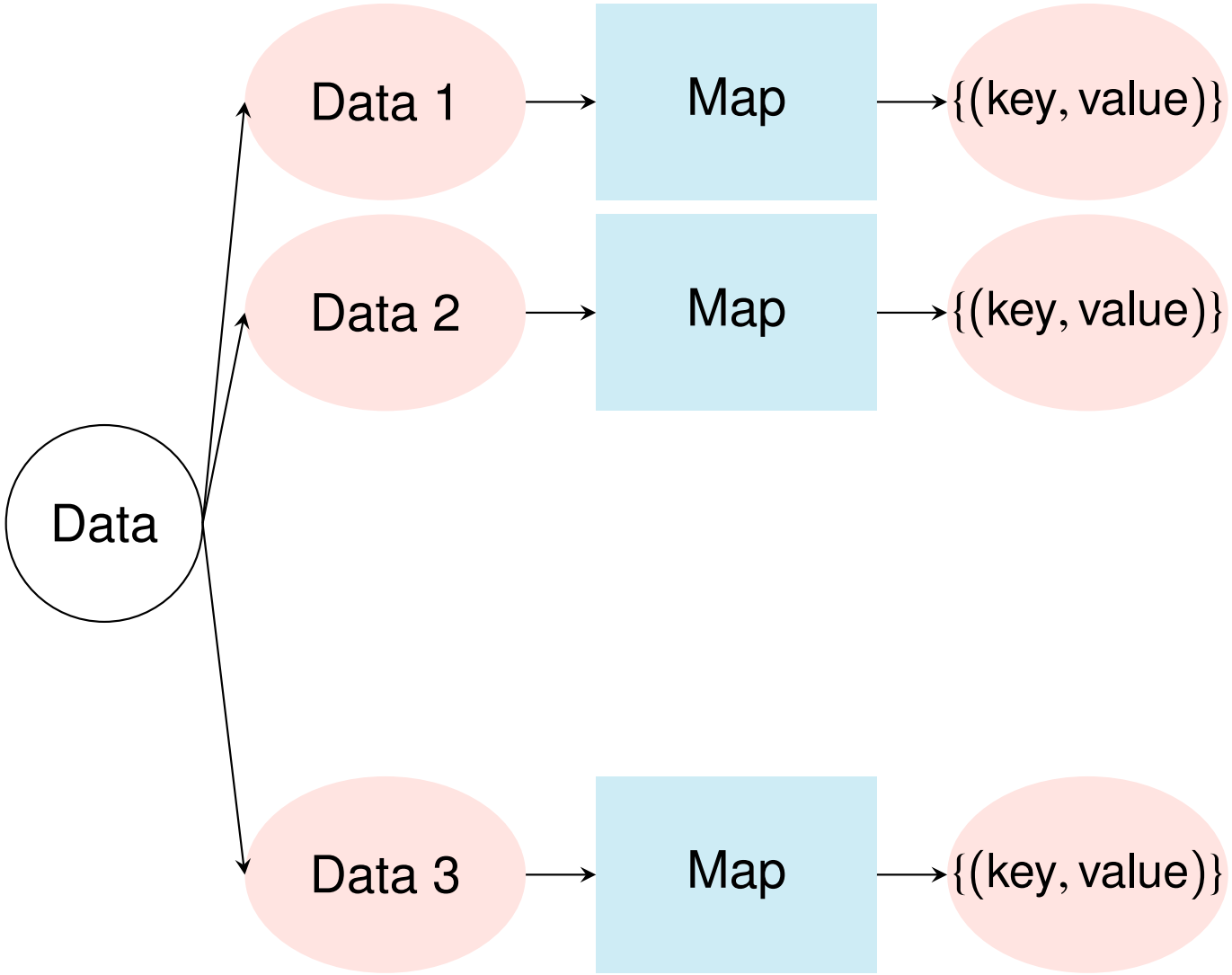
The data are broken into several bits.

Overview of Map Reduce



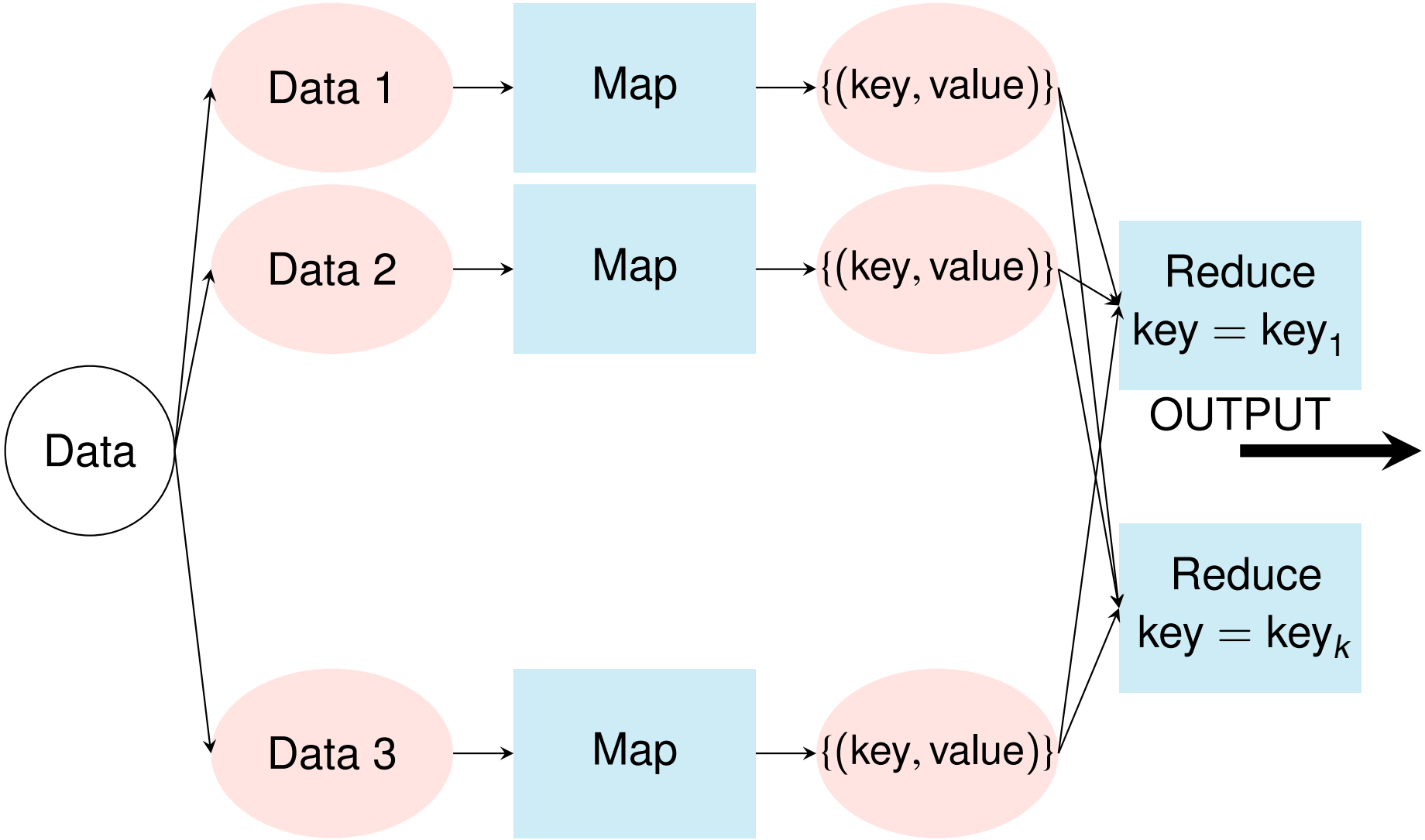
Each bit is processed through ONE map step and gives pairs {(key, value)}.

Overview of Map Reduce



Map jobs must be independent! Result: indexed data.

Overview of Map Reduce



Each key is processed through ONE reduce step to produce the output.

Map Reduce in practice

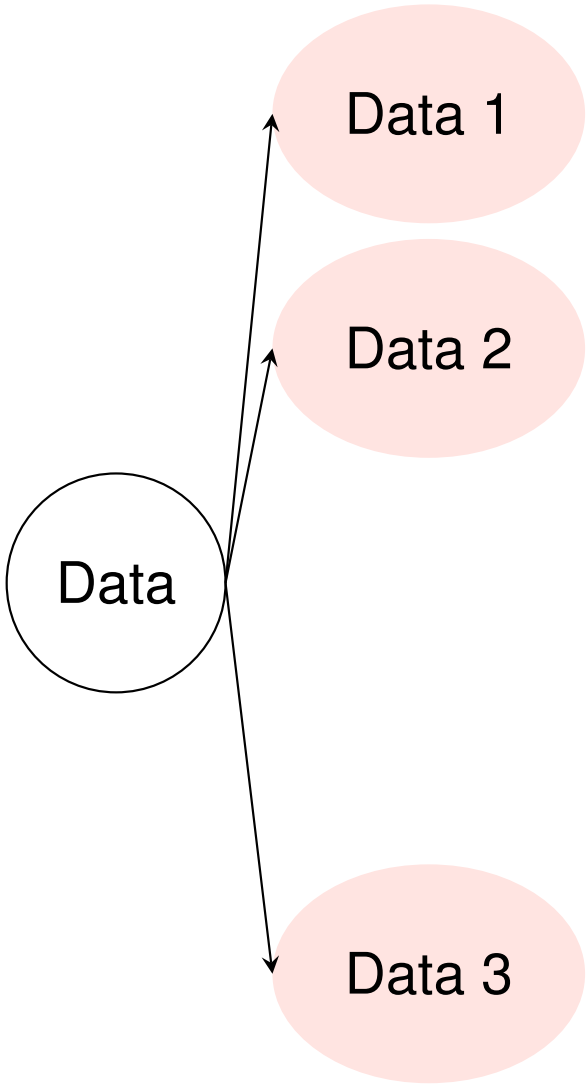
(stupid) Case study: A huge number of sales identified by the shop and the amount.

```
shop1,25000
shop2,12
shop2,1500
shop4,47
shop1,358
...
```

Question: Extract the total amount per shop.

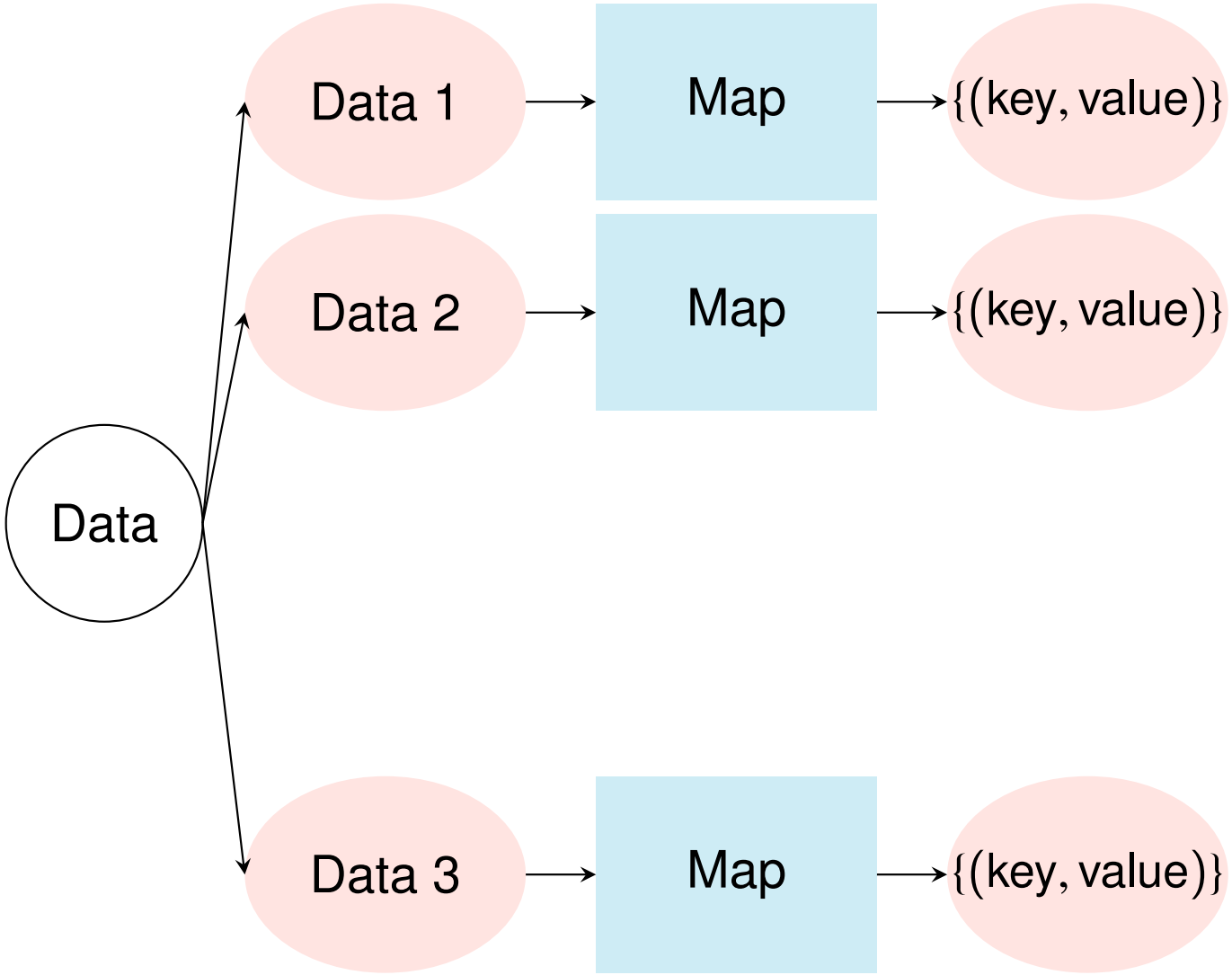
- Standard way (sequential)
  - the data are read sequentially;
  - a vector containing the values of the current sum for every shop is updated at each line.
- Map Reduce way (parallel)...

Map Reduce for an aggregation framework



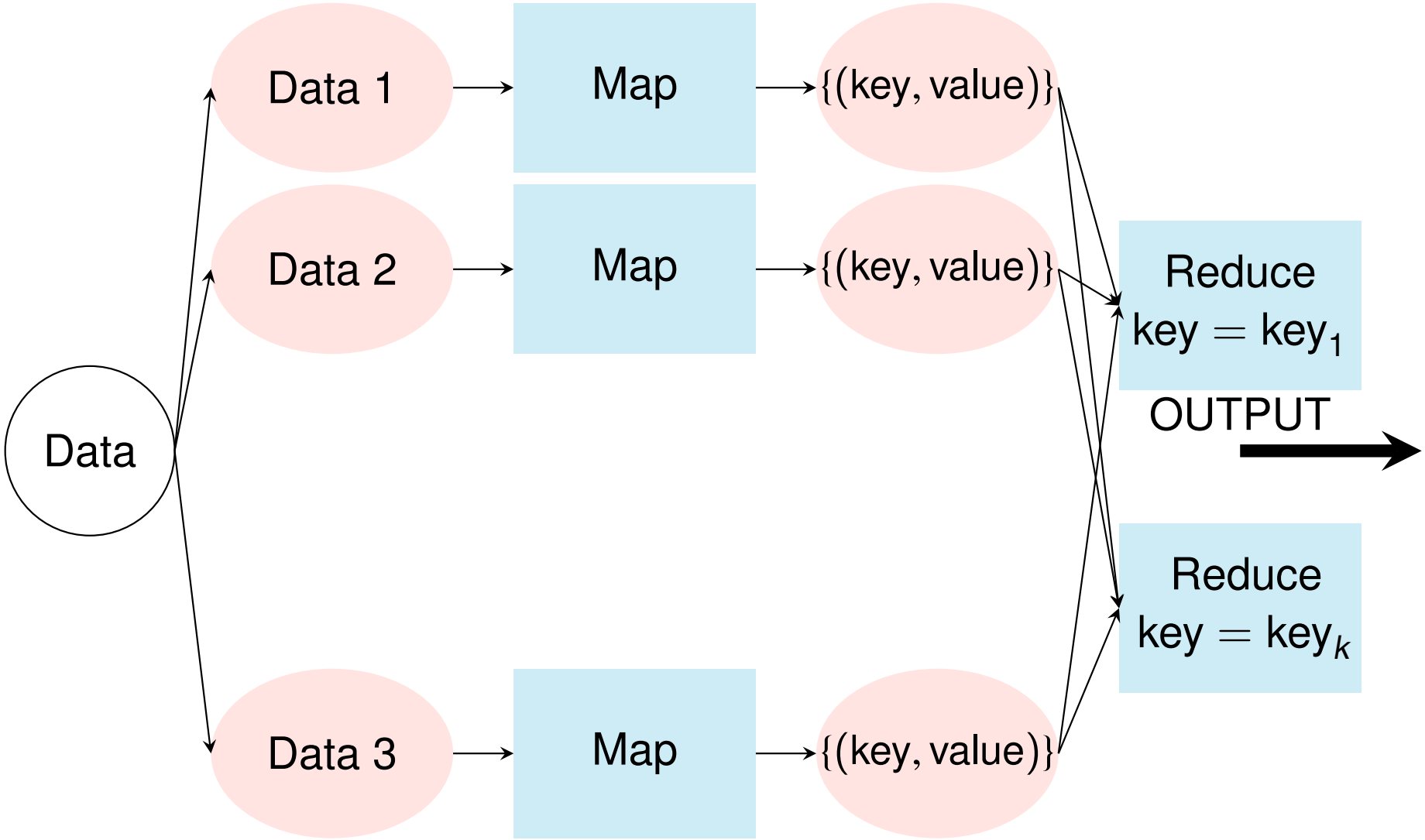
The data are broken into several bits.

Map Reduce for an aggregation framework



Map step: reads the line and outputs a pair key=shop and value=amount.

Map Reduce for an aggregation framework



Reduce step: for every key (*i.e.*, shop), compute the sum of values.

In practice, Hadoop framework

Apache Hadoop: open-source software framework for Big Data programmed in Java. It contains:

- a distributed file system (data seem to be accessed as if they were on a single computer, though distributed on several storage units);
- a map-reduce framework that takes advantage of data locality.

It is divided into: Name Nodes (typically two) that manage the file system index and Data Nodes that contain a small portion of the data and processing capabilities.

In practice, Hadoop framework

Apache Hadoop: open-source software framework for Big Data programmed in Java. It contains:

- a distributed file system (data seem to be accessed as if they were on a single computer, though distributed on several storage units);
- a map-reduce framework that takes advantage of data locality.

It is divided into: Name Nodes (typically two) that manage the file system index and Data Nodes that contain a small portion of the data and processing capabilities.
Data inside HDFS are not indexed (unlike SQL data for instance) but stored as simple text files (*e.g.*, comma separated) $\Rightarrow$ queries cannot be performed simply.

In practice, Hadoop framework

Apache Hadoop: open-source software framework for Big Data programmed in Java. It contains:

- a distributed file system (data seem to be accessed as if they were on a single computer, though distributed on several storage units);
- a map-reduce framework that takes advantage of data locality.

It is divided into: Name Nodes (typically two) that manage the file system index and Data Nodes that contain a small portion of the data and processing capabilities.

Data inside HDFS are not indexed (unlike SQL data for instance) but stored as simple text files (*e.g.*, comma separated) $\Rightarrow$ queries cannot be performed simply.

Advantages/drawback: Hadoop is designed to realize tasks on a very large number of computers ("data at Google scale"): Map tasks are made locally to speed the processing. But this advantage is lost when computation tasks are intensive on moderately large datasets (which fits in a single computer).

Application of MR to statistical learning methods

Learning problem: (X, Y) st $X \in \mathcal{X}$ and $Y \in \mathbb{R}$ (regression) or $Y \in \{1, \ldots, K-1\}$ (classification)

... that has to be learned from the observations $(X_i, Y_i)_{i=1,\ldots,n}$

...with n very large.

Standard approach for methods based on a sommation over n [Chu et al., 2010]

When a classification method is based on a sommation of the form

$$\sum_{i=1}^{n} F(X_i, Y_i)$$

it is easily addressed under the MR framework:

- data are split between Q bits sent to each map job;
- a map job computes a partial sommation $\sum_{i \in \text{current bit}} F(X_i, Y_i)$;
- the reducer then sums up intermediate results to get the final result.

Example: linear model

Framework:

$$Y = \beta^T X + \epsilon$$

in which β is estimated by solving $\Sigma_n \hat{\beta} = \Gamma_n$ with $\Sigma_n = \frac{1}{n} \sum_{i=1}^{n} X_i X_i^\top$ and $\Gamma_n = \frac{1}{n} \sum_{i=1}^{n} X_i Y_i$.

Example: linear model

Framework:

$$Y = \beta^T X + \epsilon$$

in which β is estimated by solving $\Sigma_n \hat{\beta} = \Gamma_n$ with $\Sigma_n = \frac{1}{n} \sum_{i=1}^{n} X_i X_i^\top$ and $\Gamma_n = \frac{1}{n} \sum_{i=1}^{n} X_i Y_i$.

MR for linear model

1. Map step: $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r), $n_r = \text{Card}\tau_r$, $\sigma_n^r = \sum_{i \in \tau_r} X_i X_i^\top$ and $\gamma_n^r = \sum_{i \in \tau_r} X_i Y_i$ (key is equal to 1 for every output)
2. Reduce step (only one task): $n = \sum_{r=1}^{Q} n_r$, $\Sigma_n = \frac{\sum_{r=1}^{Q} \sigma_n^r}{n}$, $\Gamma_n = \frac{\sum_{r=1}^{Q} \gamma_n^r}{n}$ and finally, $\hat{\beta} = \Sigma_n^{-1} \Gamma_n$

Example: linear model

Framework:

$$Y = \beta^T X + \epsilon$$

in which β is estimated by solving $\Sigma_n \hat{\beta} = \Gamma_n$ with $\Sigma_n = \frac{1}{n} \sum_{i=1}^{n} X_i X_i^\top$ and $\Gamma_n = \frac{1}{n} \sum_{i=1}^{n} X_i Y_i$.

MR for linear model

1. Map step: $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r), $n_r = \text{Card}\tau_r$, $\sigma_n^r = \sum_{i \in \tau_r} X_i X_i^\top$ and $\gamma_n^r = \sum_{i \in \tau_r} X_i Y_i$ (key is equal to 1 for every output)
2. Reduce step (only one task): $n = \sum_{r=1}^{Q} n_r$, $\Sigma_n = \frac{\sum_{r=1}^{Q} \sigma_n^r}{n}$, $\Gamma_n = \frac{\sum_{r=1}^{Q} \gamma_n^r}{n}$ and finally, $\hat{\beta} = \Sigma_n^{-1} \Gamma_n$

Remark: This approach is strictly equivalent to estimating the linear model from the whole dataset directly.

A more tricky problem: penalized linear model

New framework: minimize penalized least squares

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \beta^\top X_i\right)^2 + \lambda\,\text{pen}(\beta)$$

where, $\lambda \in \mathbb{R}^+$ and (usually)

- $\text{pen}(\beta) = \|\beta\|_2^2 = \sum_{j=1}^{p}\beta_j^2$ (ridge regularization **[Tikhonov, 1963]**);
- $\text{pen}(\beta) = \|\beta\|_1 = \sum_{j=1}^{p}|\beta_j|$ (LASSO **[Tibshirani, 1996]**)

A more tricky problem: penalized linear model

New framework: minimize penalized least squares

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \beta^\top X_i\right)^2 + \lambda\,\text{pen}(\beta)$$

where, $\lambda \in \mathbb{R}^+$ and (usually)

- $\text{pen}(\beta) = \|\beta\|_2^2 = \sum_{j=1}^{p}\beta_j^2$ (ridge regularization [Tikhonov, 1963]);
- $\text{pen}(\beta) = \|\beta\|_1 = \sum_{j=1}^{p}|\beta_j|$ (LASSO [Tibshirani, 1996])

The approach of simply summing the different quantities obtained in the different Map tasks is not valid anymore as explained in [Chen and Xie, 2014]

A more tricky problem: penalized linear model

New framework: minimize penalized least squares

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \beta^\top X_i\right)^2 + \lambda\,\text{pen}(\beta)$$

where, $\lambda \in \mathbb{R}^+$ and (usually)

- $\text{pen}(\beta) = \|\beta\|_2^2 = \sum_{j=1}^{p}\beta_j^2$ (ridge regularization **[Tikhonov, 1963]**);
- $\text{pen}(\beta) = \|\beta\|_1 = \sum_{j=1}^{p}|\beta_j|$ (LASSO **[Tibshirani, 1996]**)

The approach of simply summing the different quantities obtained in the different Map tasks is not valid anymore as explained in **[Chen and Xie, 2014]** $\Rightarrow$ solution involves weighting the different samples $(\tau_r)_{r=1,\ldots,Q}$ to obtain asymptotic equivalence when $Q = n^\delta$ for $0 \leq \delta \leq 1/2$.

MR implementation of random forest

A Map/Reduce implementation of random forest is included in Mahout (Apache scalable machine learning library) which works as **[del Rio et al., 2014]**:

- data are split between Q bits sent to each Map job;
- a Map job train a random forest with a small number of trees in it;
- there is no Reduce step (the final forest is the combination of all trees learned in the Map jobs).

MR implementation of random forest

A Map/Reduce implementation of random forest is included in Mahout (Apache scalable machine learning library) which works as **[del Rio et al., 2014]**:

- data are split between Q bits sent to each Map job;
- a Map job train a random forest with a small number of trees in it;
- there is no Reduce step (the final forest is the combination of all trees learned in the Map jobs).

Note that this implementation is not equivalent to the original random forest algorithm because the forests are not built on bootstrap samples of the original data set.

Drawbacks of MR implementation of random forest

- Locality of data can yield to biased random forests in the different Map jobs $\Rightarrow$ the combined forest might have poor prediction performances

Drawbacks of MR implementation of random forest

- Locality of data can yield to biased random forests in the different Map jobs $\Rightarrow$ the combined forest might have poor prediction performances

- OOB error cannot be computed precisely because Map job are independent. A proxy of this quantity is given by the average of OOB errors obtained from the different Map tasks $\Rightarrow$ again this quantity must be biased due to data locality.

MR-RF in practice: case study [Genuer et al., 2015]

15,000,000 observations generated from: Y with $P(Y=1)=P(Y=-1)=0.5$ and the conditional distribution of the $(X^{(j)})_{j=1,\ldots,7}$ given $Y=y$

- with probability equal to 0.7, $X^{(j)} \sim \mathcal{N}(jy,1)$ for $j \in \{1,2,3\}$ and $X^{(j)} \sim \mathcal{N}(0,1)$ for $j \in \{4,5,6\}$;
- with probability equal to 0.3, $X^{j} \sim \mathcal{N}(0,1)$ for $j \in \{1,2,3\}$ and $X^{(j)} \sim \mathcal{N}((j-3)y,1)$ for $j \in \{4,5,6\}$;
- $X^{7} \sim \mathcal{N}(0,1)$.

MR-RF in practice: case study [Genuer et al., 2015]

15,000,000 observations generated from: Y with $P(Y=1)=P(Y=-1)=0.5$ and the conditional distribution of the $(X^{(j)})_{j=1,\ldots,7}$ given $Y=y$

- with probability equal to 0.7, $X^{(j)} \sim \mathcal{N}(jy,1)$ for $j \in \{1,2,3\}$ and $X^{(j)} \sim \mathcal{N}(0,1)$ for $j \in \{4,5,6\}$;
- with probability equal to 0.3, $X^{j} \sim \mathcal{N}(0,1)$ for $j \in \{1,2,3\}$ and $X^{(j)} \sim \mathcal{N}((j-3)y,1)$ for $j \in \{4,5,6\}$;
- $X^{7} \sim \mathcal{N}(0,1)$.

Comparison of subsampling, BLB, MR with well distributed data within Map jobs and with Map jobs dealing with (mostly) data from one of the two submodels.

Discussion on MR-RF on a simulation study

Method	Comp. time	BDerrForest	errForest	errTest
sampling 10%	3 min	4.622e(-3)	4.381e(-3)	4.300e(-3)
sampling 1%	9 sec	4.586e(-3)	4.363e(-3)	4.400e(-3)
sampling 0.1%	1 sec	5.600e(-3)	4.714e(-3)	4.573e(-3)
sampling 0.01%	0.3 sec	4.666e(-3)	5.957e(-3)	5.753e(-3)
BLB-RF 5/20	1 min	4.138e(-3)	4.294e(-3)	4.267e(-3)
BLB-RF 10/10	3 min	4.138e(-3)	4.278e(-3)	4.267e(-3)
MR-RF 100/1	2 min	1.397e(-2)	4.235 e(-3)	4.006e(-3)
MR-RF 100/10	2 min	8.646e(-3)	4.155e(-3)	4.293e(-3)
MR-RF 10/10	6 min	8.501e(-3)	4.290e(-3)	4.253e(-3)
MR-RF 10/100	21 min	4.556e(-3)	4.249e(-3)	4.260e(-3)
MR x-biases 100/1	3 min	3.504e(-3)	1.010e(-1)	1.006e(-1)
MR x-biases 100/10	3 min	2.082e(-3)	1.010e(-1)	1.008e(-1)

Discussion on MR-RF on a simulation study

Method	Comp. time	BDerrForest	errForest	errTest
sampling 10%	3 min	4.622e(-3)	4.381e(-3)	4.300e(-3)
sampling 1%	9 sec	4.586e(-3)	4.363e(-3)	4.400e(-3)
sampling 0.1%	1 sec	5.600e(-3)	4.714e(-3)	4.573e(-3)
sampling 0.01%	0.3 sec	4.666e(-3)	5.957e(-3)	5.753e(-3)
BLB-RF 5/20	1 min	4.138e(-3)	4.294e(-3)	4.267e(-3)
BLB-RF 10/10	3 min	4.138e(-3)	4.278e(-3)	4.267e(-3)
MR-RF 100/1	2 min	1.397e(-2)	4.235 e(-3)	4.006e(-3)
MR-RF 100/10	2 min	8.646e(-3)	4.155e(-3)	4.293e(-3)
MR-RF 10/10	6 min	8.501e(-3)	4.290e(-3)	4.253e(-3)
MR-RF 10/100	21 min	4.556e(-3)	4.249e(-3)	4.260e(-3)
MR x-biases 100/1	3 min	3.504e(-3)	1.010e(-1)	1.006e(-1)
MR x-biases 100/10	3 min	2.082e(-3)	1.010e(-1)	1.008e(-1)

- all methods provide satisfactory results except MR when locality biases are introduced
- average OOB error over the Map forests can be a bad approximation of true OOB error (sometimes optimistic, sometimes pessimistic)

Another MR implementation of random forest

... using Poisson bootstrap **[Chamandy et al., 2012]** which is based on the fact that (for large n):

$$\text{Binom}\left(n, \frac{1}{n}\right) \simeq \text{Poisson}(1)$$

Another MR implementation of random forest

... using Poisson bootstrap **[Chamandy et al., 2012]** which is based on the fact that (for large n):

$$\text{Binom}\left(n, \frac{1}{n}\right) \simeq \text{Poisson}(1)$$

1. Map step $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r). $\forall\, i \in \tau_r$, generate B random i.i.d. random variables from Poisson(1) n_i^b ($b = 1, \ldots, B$).

Another MR implementation of random forest

... using Poisson bootstrap **[Chamandy et al., 2012]** which is based on the fact that (for large n):

$$\text{Binom}\left(n, \frac{1}{n}\right) \simeq \text{Poisson}(1)$$

1. Map step $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r). $\forall\, i \in \tau_r$, generate B random i.i.d. random variables from Poisson(1) n_i^b ($b = 1, \ldots, B$).
   Output: (key, value) are $(b,\ (i,\ n_i^b))$ for all pairs (i, b) st $n_i^b \neq 0$ (indices i st $n_i^b \neq 0$ are in bootstrap sample number b n_i^b times);

Another MR implementation of random forest

... using Poisson bootstrap **[Chamandy et al., 2012]** which is based on the fact that (for large n):

$$\text{Binom}\left(n, \frac{1}{n}\right) \simeq \text{Poisson}(1)$$

1. Map step $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r). $\forall\, i \in \tau_r$, generate B random i.i.d. random variables from Poisson(1) n_i^b ($b = 1, \ldots, B$).
   Output: (key, value) are $(b,\ (i,\ n_i^b))$ for all pairs (i, b) st $n_i^b \neq 0$ (indices i st $n_i^b \neq 0$ are in bootstrap sample number b n_i^b times);
2. Reduce step proceeds bootstrap sample number b: a tree is built from indices i st $n_i^b \neq 0$ repeated n_i^b times.

Another MR implementation of random forest

... using Poisson bootstrap **[Chamandy et al., 2012]** which is based on the fact that (for large n):

$$\text{Binom}\left(n, \frac{1}{n}\right) \simeq \text{Poisson}(1)$$

1. Map step $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r). $\forall\, i \in \tau_r$, generate B random i.i.d. random variables from Poisson(1) n_i^b ($b = 1, \ldots, B$).
   Output: (key, value) are $(b,\ (i,\ n_i^b))$ for all pairs (i, b) st $n_i^b \neq 0$ (indices i st $n_i^b \neq 0$ are in bootstrap sample number b n_i^b times);
2. Reduce step proceeds bootstrap sample number b: a tree is built from indices i st $n_i^b \neq 0$ repeated n_i^b times.
   Output: A tree... All trees are collected in a forest.

Another MR implementation of random forest

... using Poisson bootstrap **[Chamandy et al., 2012]** which is based on the fact that (for large n):

$$\text{Binom}\left(n, \frac{1}{n}\right) \simeq \text{Poisson}(1)$$

1. Map step $\forall\, r = 1, \ldots, Q$ (chunk of data τ_r). $\forall\, i \in \tau_r$, generate B random i.i.d. random variables from Poisson(1) n_i^b ($b = 1, \ldots, B$).
   Output: (key, value) are $(b,\ (i,\ n_i^b))$ for all pairs (i, b) st $n_i^b \neq 0$ (indices i st $n_i^b \neq 0$ are in bootstrap sample number b n_i^b times);
2. Reduce step proceeds bootstrap sample number b: a tree is built from indices i st $n_i^b \neq 0$ repeated n_i^b times.
   Output: A tree... All trees are collected in a forest.

Closer to using RF directly on the entire dataset But: every Reduce job should deal with approximately $0.63 \times n$ different observations... (only the bootstrap part is simplified)

Sommaire

Online learning framework

Data stream: Observations $(X_i, Y_i)_{i=1,\dots,n}$ have been used to obtain a predictor $\hat{f}_n$

New data arrive $(X_i, Y_i)_{i=n+1,\dots,n+m}$: How to obtain a predictor from the entire dataset $(X_i, Y_i)_{i=1,\dots,n+m}$?

Online learning framework

Data stream: Observations $(X_i, Y_i)_{i=1,\dots,n}$ have been used to obtain a predictor $\hat{f}_n$

New data arrive $(X_i, Y_i)_{i=n+1,\dots,n+m}$: How to obtain a predictor from the entire dataset $(X_i, Y_i)_{i=1,\dots,n+m}$?

- Naive approach: re-train a model from $(X_i, Y_i)_{i=1,\dots,n+m}$
- More interesting approach: update $\hat{f}_n$ with the new information $(X_i, Y_i)_{i=n+1,\dots,n+m}$

Online learning framework

Data stream: Observations $(X_i, Y_i)_{i=1,\dots,n}$ have been used to obtain a predictor $\hat{f}_n$

New data arrive $(X_i, Y_i)_{i=n+1,\dots,n+m}$: How to obtain a predictor from the entire dataset $(X_i, Y_i)_{i=1,\dots,n+m}$?

- Naive approach: re-train a model from $(X_i, Y_i)_{i=1,\dots,n+m}$
- More interesting approach: update $\hat{f}_n$ with the new information $(X_i, Y_i)_{i=n+1,\dots,n+m}$

Why is it interesting?

- computational gain if the update has a small computational cost (it can even be interesting to deal directly with big data which do not arrive in stream)
- storage gain

Online learning framework

Data stream: Observations $(X_i, Y_i)_{i=1,\dots,n}$ have been used to obtain a predictor $\hat{f}_n$
New data arrive $(X_i, Y_i)_{i=n+1,\dots,n+m}$: How to obtain a predictor from the entire dataset $(X_i, Y_i)_{i=1,\dots,n+m}$?

- Naive approach: re-train a model from $(X_i, Y_i)_{i=1,\dots,n+m}$
- More interesting approach: update $\hat{f}_n$ with the new information $(X_i, Y_i)_{i=n+1,\dots,n+m}$

Why is it interesting?

- computational gain if the update has a small computational cost (it can even be interesting to deal directly with big data which do not arrive in stream)
- storage gain

Additional remark: Restricted to stationnary problems (as opposed to "concept drift"

Simple example: online k-NN

Standard k-NN: in the regression framework,

$$\hat{f}_n(x) = \sum_{i=1}^{n} W_i^{k-\text{ppv}}(X_{1\ldots n}; x) Y_i$$

with $W_i^{k-\text{ppv}}(X_{1\ldots n}; x) = \frac{1}{k}$ if X_i is one of the k-nearest neighbors of x in $X_{1\ldots n}$ and 0 otherwise.

Simple example: online k-NN

Standard k-NN: in the regression framework,

$$\hat{f}_n(x) = \sum_{i=1}^{n} W_i^{k-\text{ppv}}(X_{1\dots n}; x) Y_i$$

with $W_i^{k-\text{ppv}}(X_{1\dots n}; x) = \frac{1}{k}$ if X_i is one of the k-nearest neighbors of x in $X_{1\dots n}$ and 0 otherwise.

What has to be stored/updated for online k-NN?

- $\mathcal{N}^n(x) \in \{1, \dots, n\}^n$: k-nearest neighbors of x (stored by increasing distance to allow for an easy online update)

Simple example: online k-NN

Standard k-NN: in the regression framework,

$$\hat{f}_n(x) = \sum_{i=1}^{n} W_i^{k-\text{ppv}}(X_{1\dots n}; x) Y_i$$

with $W_i^{k-\text{ppv}}(X_{1\dots n}; x) = \frac{1}{k}$ if X_i is one of the k-nearest neighbors of x in $X_{1\dots n}$ and 0 otherwise.

What has to be stored/updated for online k-NN?

- $\mathcal{N}^n(x) \in \{1, \dots, n\}^n$: k-nearest neighbors of x (stored by increasing distance to allow for an easy online update)
- $d^n(x) = (d(x, X_i))_{i \in \mathcal{N}^n(x)}$

Simple example: online k-NN

Standard k-NN: in the regression framework,

$$\hat{f}_n(x) = \sum_{i=1}^{n} W_i^{k-\text{ppv}}(X_{1\dots n}; x) Y_i$$

with $W_i^{k-\text{ppv}}(X_{1\dots n}; x) = \frac{1}{k}$ if X_i is one of the k-nearest neighbors of x in $X_{1\dots n}$ and 0 otherwise.

What has to be stored/updated for online k-NN?

- $\mathcal{N}^n(x) \in \{1, \dots, n\}^n$: k-nearest neighbors of x (stored by increasing distance to allow for an easy online update)
- $d^n(x) = (d(x, X_i))_{i \in \mathcal{N}^n(x)}$
- $\hat{f}_n(x)$

A new observation arrives: (X_{n+1}, Y_{n+1})...

1. compute $d(X_{n+1}, x)$
2. $d(X_{n+1}, x) > d_1^n(x) \Rightarrow$ nothing

$\mathcal{N}^n(x)$

5	1	3	2

$d^n(x)$

1.4	1.7	2.3	2.4

$d(X_{n+1}, x) = 2$

A new observation arrives: (X_{n+1}, Y_{n+1})...

1. compute $d(X_{n+1}, x)$
2. $d(X_{n+1}, x) > d_2^n(x) \Rightarrow$ nothing

$\mathcal{N}^n(x)$

5	1	3	2

$d^n(x)$

1.4	1.7	2.3	2.4

$d(X_{n+1}, x) = 2$

A new observation arrives: (X_{n+1}, Y_{n+1})...

1. compute $d(X_{n+1}, x)$
2. $d(X_{n+1}, x) < d_j^n(x)$ for a $j < k \Rightarrow$ update $\mathcal{N}^n(x)$ and $d^n(x)$

$\mathcal{N}^{n+1}(x)$

5	1	$n+1$	3

$\mathcal{N}^-_{n,n+1}(x) = \{2\}$

$d^{n+1}(x)$

1.4	1.7	2	2.3

$d(X_{n+1}, x)$

A new observation arrives: (X_{n+1}, Y_{n+1})...

1. compute $d(X_{n+1}, x)$
2. $d(X_{n+1}, x) < d_j^n(x)$ for a $j < k \Rightarrow$ update $\mathcal{N}^n(x)$ and $d^n(x)$ (can be repeated several times $\Rightarrow \mathcal{N}^-_{n,n+1}(x)$ and $\mathcal{N}^+_{n,n+1}(x)$)
3. update $\hat{f}_n(x)$:

$$\hat{f}_{n+1}(x) = \hat{f}_n(x) + \frac{1}{k}\left[\sum_{i \in \mathcal{N}^+_{n,n+1}(x)} Y_i - \sum_{i \in \mathcal{N}^-_{n,n+1}(x)} Y_i\right]$$

$\mathcal{N}^{n+1}(x)$

5	1	$n+1$	3

$\mathcal{N}^-_{n,n+1}(x) = \{2\}$

$d^{n+1}(x)$

1.4	1.7	2	2.3

$d(X_{n+1}, x)$

A new observation arrives: (X_{n+1}, Y_{n+1})...

1. compute $d(X_{n+1}, x)$
2. $d(X_{n+1}, x) < d_j^n(x)$ for a $j < k \Rightarrow$ update $\mathcal{N}^n(x)$ and $d^n(x)$ (can be repeated several times $\Rightarrow \mathcal{N}^-_{n,n+1}(x)$ and $\mathcal{N}^+_{n,n+1}(x)$)
3. update $\hat{f}_n(x)$:

$$\hat{f}_{n+1}(x) = \hat{f}_n(x) + \frac{1}{k}\left[\sum_{i \in \mathcal{N}^+_{n,n+1}(x)} Y_i - \sum_{i \in \mathcal{N}^-_{n,n+1}(x)} Y_i\right]$$

Costs for m updates:

- update: m (distances), $m \times k$ comparisons and k for the update itself
- storage: 2 vectors with length k and a real number

Much smaller than the cost of direct k-NN on $(X_i)_{i=1,\dots,n+m}$.

Kernel ridge regression [Saunders et al., 1998]

$$\underset{w\in\mathcal{H}}{\arg\min} \sum_{i=1}^{n} (Y_i - \langle w_n, \phi(X_i)\rangle_{\mathcal{H}})^2 + \frac{\lambda}{2}\|w\|_{\mathcal{H}}^2$$

Solution:

$$\hat{f}_N : \; x \in \mathcal{X} \to \langle w_n, \phi(X_i)\rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i K(X_i, x)$$

with

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \left(\mathbf{K}^{(n)} + \lambda \mathbb{I}_n\right)^{-1} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

Kernel ridge regression [Saunders et al., 1998]

$$\underset{w\in\mathcal{H}}{\arg\min} \sum_{i=1}^{n} \left(Y_i - \langle w_n, \phi(X_i)\rangle_{\mathcal{H}}\right)^2 + \frac{\lambda}{2}\|w\|_{\mathcal{H}}^2$$

Solution:

$$\hat{f}_N : \; x \in \mathcal{X} \to \langle w_n, \phi(X_i)\rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i K(X_i, x)$$

with

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \left(\mathbf{K}^{(n)} + \lambda \mathbb{I}_n\right)^{-1} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

Main problem: updating $\mathbf{Q}^{(n)} = \left(\mathbf{K}^{(n)} + \lambda \mathbb{I}_n\right)$ online

Online update of $\mathbf{Q}^{(n)}$

[Engel et al., 2003, van Vaerenbergh et al., 2012]

Writing:

$$\mathbf{K}^{(n+1)} = \begin{pmatrix} \mathbf{K}^{(n)} & \mathbf{k}_{n+1} \\ \mathbf{k}_{n+1}^\top & K(X_{n+1}, X_{n+1}) \end{pmatrix}$$

for $\mathbf{k}_{n+1} = (K(X_{n+1}, X_i))_{i=1,\ldots,n}$

Online update of $\mathbf{Q}^{(n)}$

[Engel et al., 2003, van Vaerenbergh et al., 2012]

Writing:

$$\mathbf{K}^{(n+1)} = \begin{pmatrix} \mathbf{K}^{(n)} & \mathbf{k}_{n+1} \\ \mathbf{k}_{n+1}^\top & K(X_{n+1}, X_{n+1}) \end{pmatrix}$$

for $\mathbf{k}_{n+1} = (K(X_{n+1}, X_i))_{i=1,\dots,n}$

Gives:

$$\mathbf{Q}^{(n+1)} = \frac{1}{\gamma_{n+1}} \begin{pmatrix} \gamma_{n+1}\mathbf{Q}^{(n)} + a_{n+1}a_{n+1}^\top & -a_{n+1} \\ -a_{n+1}^\top & 1 \end{pmatrix}$$

gives $a_{n+1} = \mathbf{Q}^{(n)}\mathbf{k}_{n+1}$ and $\gamma_{n+1} = K(X_{n+1}, X_{n+1}) + \lambda - \mathbf{k}_{n+1}^\top a_{n+1}$.

Online update of $\mathbf{Q}^{(n)}$

[Engel et al., 2003, van Vaerenbergh et al., 2012]

$$\mathbf{a}_{n+1}^\top \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \mathbf{k}_{n+1}^\top \mathbf{Q}^{(n)} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \mathbf{k}_{n+1}^\top \boldsymbol{\alpha}^{(n)} = \hat{Y}_{n+1}$$

with $\hat{Y}_{n+1} = \hat{f}_{n+1}(X_{n+1})$.

Online update of $\mathbf{Q}^{(n)}$

[Engel et al., 2003, van Vaerenbergh et al., 2012]

$$\mathbf{a}_{n+1}^\top \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \mathbf{k}_{n+1}^\top \mathbf{Q}^{(n)} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \mathbf{k}_{n+1}^\top \boldsymbol{\alpha}^{(n)} = \hat{Y}_{n+1}$$

with $\hat{Y}_{n+1} = \hat{f}_{n+1}(X_{n+1})$.

Online solution:

$$\hat{f}_{n+1} : \; x \in \mathcal{X} \to \sum_{i=1}^{n+1} \alpha_i^{(n+1)} K(X_i, x)$$

with

$$\begin{pmatrix} \alpha_1^{(n+1)} \\ \vdots \\ \alpha_{n+1}^{(n+1)} \end{pmatrix} = \begin{pmatrix} \alpha^{(n)} - \frac{1}{\gamma_{n+1}} a_{n+1} e_{n+1} \\ \frac{1}{\gamma_{n+1}} e_{n+1} \end{pmatrix}.$$

for $e_{n+1} = Y_{n+1} - \hat{Y}_{n+1}$.

In summary

Comparison of computational costs:

- update: $\mathcal{O}(n^2)$
- direct solution: $\mathcal{O}(n^3)$

In summary

Comparison of computational costs:

- update: $\mathcal{O}(n^2)$
- direct solution: $\mathcal{O}(n^3)$

Can be improved by using a dictionnary $\text{Dico}_n \subset \{1, \ldots, n\}$, for $m \ll n$ with

$$\hat{f}_n(x) = \sum_{i \in \text{Dico}_n} \alpha_i^{(n)} K(X_i, x)$$

with an online update of the dictionnary

[Engel et al., 2003, van Vaerenbergh et al., 2012].

Framework of online bagging

$$\hat{f}_n = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_n^b$$

in which

- $\hat{f}_n^b$ has been built from a bootstrap sample in $\{1, \ldots, n\}$
- we know how to update $\hat{f}_n^b$ with new data online

Framework of online bagging

$$\hat{f}_n = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_n^b$$

in which

- $\hat{f}_n^b$ has been built from a bootstrap sample in $\{1, \ldots, n\}$
- we know how to update $\hat{f}_n^b$ with new data online

Question: Can we update the bootstrap samples online when new data $(X_i, Y_i)_{i=n+1,\ldots,n+m}$ arrive?

Online bootstrap using Poisson bootstrap

1. generate weights for every bootstrap samples and every new observation: $n_i^b \sim \text{Poisson}(1)$ for $i = n+1, \ldots, n+m$ and $b = 1, \ldots, B$

Online bootstrap using Poisson bootstrap

1. generate weights for every bootstrap samples and every new observation: $n_i^b \sim \text{Poisson}(1)$ for $i = n+1, \ldots, n+m$ and $b = 1, \ldots, B$

2. update $\hat{f}_n^b$ with the observations X_i such that $n_i^b \neq 0$, each repeated n_i^b times

Online bootstrap using Poisson bootstrap

1. generate weights for every bootstrap samples and every new observation: $n_i^b \sim \text{Poisson}(1)$ for $i = n+1, \ldots, n+m$ and $b = 1, \ldots, B$

2. update $\hat{f}_n^b$ with the observations X_i such that $n_i^b \neq 0$, each repeated n_i^b times

3. update the predictor:

$$\hat{f}_{n+m} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_{n+m}^b.$$

Application: online PRF

In Purely Random Forest, the trees are generated independently from the data. It is described by:

- $\forall\, b = 1, \ldots, B$, $\hat{f}_n^b$: PR tree for bootstrap sample number b
- $\forall\, b = 1, \ldots, B$, for all terminal leaf l in $\hat{f}_n^b$, $\text{obs}_n^{b,l}$ is the number of observations in $(X_i)_{i=1,\ldots,n}$ which falls in leaf l and $\text{val}_n^{b,l}$ is the average Y for these observations (regression framework)

Application: online PRF

In Purely Random Forest, the trees are generated independently from the data. It is described by:

- $\forall\, b = 1, \ldots, B$, $\hat{f}_n^b$: PR tree for bootstrap sample number b
- $\forall\, b = 1, \ldots, B$, for all terminal leaf l in $\hat{f}_n^b$, $\text{obs}_n^{b,l}$ is the number of observations in $(X_i)_{i=1,\ldots,n}$ which falls in leaf l and $\text{val}_n^{b,l}$ is the average Y for these observations (regression framework)

Online update with Poisson bootstrap:

- $\forall\, b = 1, \ldots, B$, $\forall\, i \in \{n+1, \ldots, n+m\}$ st $n_i^b \neq 0$ and for the terminal leaf l of X_i:

$$\text{val}_i^{b,l} = \frac{\text{val}_{i-1}^{b,l} \times \text{obs}_{i-1}^{b,l} + n_i^b \times Y_i}{\text{obs}_{i-1}^{b,l} + n_i^b}$$

(online update of the mean...)

Application: online PRF

In Purely Random Forest, the trees are generated independently from the data. It is described by:

- $\forall\, b = 1, \ldots, B$, $\hat{f}_n^b$: PR tree for bootstrap sample number b
- $\forall\, b = 1, \ldots, B$, for all terminal leaf l in $\hat{f}_n^b$, $\text{obs}_n^{b,l}$ is the number of observations in $(X_i)_{i=1,\ldots,n}$ which falls in leaf l and $\text{val}_n^{b,l}$ is the average Y for these observations (regression framework)

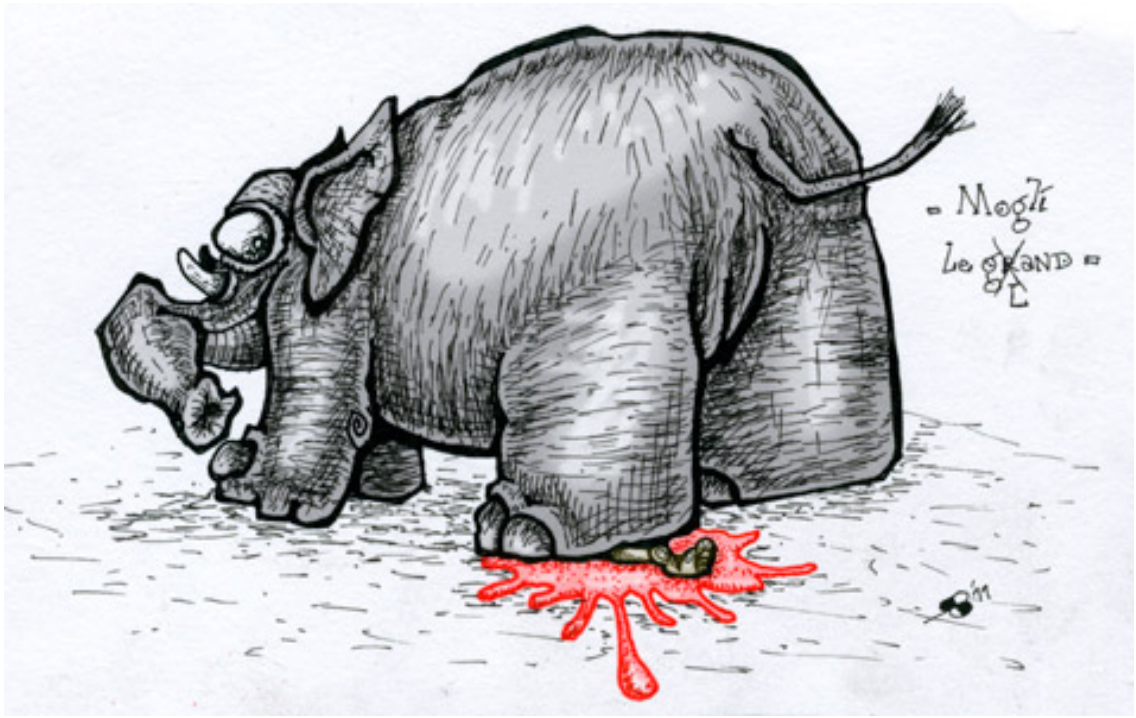
Online update with Poisson bootstrap:

- $\forall\, b = 1, \ldots, B$, $\forall\, i \in \{n+1, \ldots, n+m\}$ st $n_i^b \neq 0$ and for the terminal leaf l of X_i:

$$\text{val}_i^{b,l} = \frac{\text{val}_{i-1}^{b,l} \times \text{obs}_{i-1}^{b,l} + n_i^b \times Y_i}{\text{obs}_{i-1}^{b,l} + n_i^b}$$

  (online update of the mean...)
- $\text{obs}_i^{b,l} = \text{obs}_{i-1}^{b,l} + n_i^b$

Have you survived to Big Data?



Questions?

References

Bach, F. (2013).

Sharp analysis of low-rank kernel matrix approximations.
Journal of Machine Learning Research, Workshop and Conference Proceedings, 30:185–209.

Bickel, P., Götze, F., and van Zwet, W. (1997).

Resampling fewer than *n* observations: gains, losses and remedies for losses.
Statistica Sinica, 7(1):1–31.

Chamandy, N., Muralidharan, O., Najmi, A., and Naidu, S. (2012).

Estimating uncertainty for massive data streams.
Technical report, Google.

Chen, X. and Xie, M. (2014).

A split-and-conquer approach for analysis of extraordinarily large data.
Statistica Sinica, 24:1655–1684.

Chu, C., Kim, S., Lin, Y., Yu, Y., Bradski, G., Ng, A., and Olukotun, K. (2010).

Map-Reduce for machine learning on multicore.
In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems (NIPS 2010)*, volume 23, pages 281–288, Hyatt Regency, Vancouver, Canada.

Cortes, C., Mohri, M., and Talwalkar, A. (2010).

On the impact of kernel approximation on learning accuracy.
Journal of Machine Learning Research, Workshop and Conference Proceedings, 9:113–120.

Dean, J. and Ghemawat, S. (2004).

MapReduce: simplified data processing on large clusters.
In *Proceedings of Sixth Symposium on Operating System Design and Implementation (OSDI 2004)*.

del Rio, S., López, V., Beniítez, J., and Herrera, F. (2014).

On the use of MapReduce for imbalanced big data using random forest.
Information Sciences, 285:112–137.

Drineas, P. and Mahoney, M. (2005).
On the Nyström method for approximating a Gram matrix for improved kernel-based learning.
Journal of Machine Learning Research, 6:2153–2175.

Engel, Y., Mannor, S., and Meir, R. (2003).
The kernel recursive least squares algorithm.
IEEE Transactions on Signal Processing, 52(8):2165–6176.

Genuer, R., Poggi, J., Tuleau-Malot, C., and Villa-Vialaneix, N. (2015).
Random forests for big data.
Preprint arXiv:1511.08327. Submitted for publication.

Jordan, M. (2013).
On statistics, computation and scalability.
Bernoulli, 19(4):1378–1390.

Kane, M., Emerson, J., and Weston, S. (2013).
Scalable strategies for computing with massive data.
Journal of Statistical Software, 55(14).

Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. (2012).
The big data bootstrap.
In *Proceedings of 29th International Conference on Machine Learning (ICML 2012)*, Edinburgh, Scotland, UK.

Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. (2014).
A scalable bootstrap for massive data.
Journal of the Royal Statistical Society: Series B (Statistical Methodology), 76(4):795–816.

R Core Team (2015).
R: A Language and Environment for Statistical Computing.
R Foundation for Statistical Computing, Vienna, Austria.

Saunders, G., Gammerman, A., and Vovk, V. (1998).
Ridge regression learning algorithm in dual variables.

In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, pages 515–521, Madison, Wisconsin, USA.

Tibshirani, R. (1996).

Regression shrinkage and selection via the lasso.
Journal of the Royal Statistical Society, series B, 58(1):267–288.

Tikhonov, A. (1963).

Solution of incorrectly formulated problems and the regularization method.
Soviet mathematics - Doklady, 4:1036–1038.

van Vaerenbergh, S., Lázaro-Gredilla, M., and Santamaría, I. (2012).

Kernel recursive least-squares tracker for time-varying regression.
IEEE Transactions on Neural Networks and Learning Systems, 23(8):1313–1326.

Williams, C. and Seeger, M. (2000).

Using the Nyström method to speed up kernel machines.
In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems (Proceedings of NIPS 2000)*, volume 13, Denver, CO, USA. Neural Information Processing Systems Foundation.