Lecture 16: Large-scale methods for data analysis

Rebecka Jörnsten, Mathematical Sciences

MSA220/MVE441 Statistical Learning for Big Data

16st May 2022

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Low-rank approximations for matrices

Low-rank approximations

- **Low-rank approximations** of matrices become very important to make large-scale data manageable

$$\underset{n\times p}{\mathbf{X}} \approx \underset{n\times q}{\mathbf{A}} \cdot \underset{q\times p}{\mathbf{B}}$$

Low-rank approximations

- **Low-rank approximations** of matrices become very important to make large-scale data manageable

$$\underset{n\times p}{\mathbf{X}} \approx \underset{n\times q}{\mathbf{A}} \cdot \underset{q\times p}{\mathbf{B}}$$

- Algorithms to determine $\mathbf{A}$ and $\mathbf{B}$ discussed in the lecture: **Low-rank SVD** and **low-rank NMF**

Low-rank approximations

- **Low-rank approximations** of matrices become very important to make large-scale data manageable

$$\underset{n\times p}{\mathbf{X}} \approx \underset{n\times q}{\mathbf{A}} \cdot \underset{q\times p}{\mathbf{B}}$$

- Algorithms to determine $\mathbf{A}$ and $\mathbf{B}$ discussed in the lecture: **Low-rank SVD** and **low-rank NMF**
- Works best if original data in $\mathbf{X}$ is approximately of rank $q \ll \min(n, p)$

Low-rank approximations

- **Low-rank approximations** of matrices become very important to make large-scale data manageable

$$\underset{n\times p}{\mathbf{X}} \approx \underset{n\times q}{\mathbf{A}} \cdot \underset{q\times p}{\mathbf{B}}$$

- Algorithms to determine $\mathbf{A}$ and $\mathbf{B}$ discussed in the lecture: **Low-rank SVD** and **low-rank NMF**
- Works best if original data in $\mathbf{X}$ is approximately of rank $q \ll \min(n, p)$
- $\mathbf{X}$ could be a really large data matrix, but it could also come from an intermediate calculation, e.g. a Gram matrix or a distance matrix

Low-rank approximations

- **Low-rank approximations** of matrices become very important to make large-scale data manageable

$$\underset{n \times p}{\mathbf{X}} \approx \underset{n \times q}{\mathbf{A}} \cdot \underset{q \times p}{\mathbf{B}}$$

- Algorithms to determine $\mathbf{A}$ and $\mathbf{B}$ discussed in the lecture: **Low-rank SVD** and **low-rank NMF**
- Works best if original data in $\mathbf{X}$ is approximately of rank $q \ll \min(n, p)$
- $\mathbf{X}$ could be a really large data matrix, but it could also come from an intermediate calculation, e.g. a Gram matrix or a distance matrix
- NMF and SVD are computationally efficient if either n or p are reasonably small to medium sized (computational complexity $O(n^2 p + p^3)$ for SVD)

What if both n and p are large?

Dimension reduction to the rescue

Assume for now that $\mathbf{X}$ actually has rank $q \ll \min(n, p)$. Then we could find an exact factorisation $\mathbf{X} = \mathbf{AB}$, e.g. using the SVD truncated after q terms.

Dimension reduction to the rescue

Assume for now that $\mathbf{X}$ actually has rank $q \ll \min(n, p)$. Then we could find an exact factorisation $\mathbf{X} = \mathbf{AB}$, e.g. using the SVD truncated after q terms.

SVD on the full matrix is too expensive, but can we **cheaply reduce** at least one of the dimensions?

Dimension reduction to the rescue

Assume for now that $\mathbf{X}$ actually has rank $q \ll \min(n, p)$. Then we could find an exact factorisation $\mathbf{X} = \mathbf{AB}$, e.g. using the SVD truncated after q terms.

SVD on the full matrix is too expensive, but can we **cheaply reduce** at least one of the dimensions?

Since $\mathbf{X}$ is assumed to have rank q, its image

$$\operatorname{Im}(\mathbf{X}) = \{\mathbf{y} : \mathbf{y} = \mathbf{X}\mathbf{w} \text{ for some } \mathbf{w} \in \mathbb{R}^p\}$$

is only q dimensional. Projecting the columns of $\mathbf{X}$ **at least approximately** to a q-dimensional space leaves the overall structure of the data intact.

Dimension reduction to the rescue

Assume for now that $\mathbf{X}$ actually has rank $q \ll \min(n, p)$. Then we could find an exact factorisation $\mathbf{X} = \mathbf{AB}$, e.g. using the SVD truncated after q terms.

SVD on the full matrix is too expensive, but can we **cheaply reduce** at least one of the dimensions?

Since $\mathbf{X}$ is assumed to have rank q, its image

$$\mathrm{Im}(\mathbf{X}) = \{\mathbf{y} : \mathbf{y} = \mathbf{X}\mathbf{w} \text{ for some } \mathbf{w} \in \mathbb{R}^p\}$$

is only q dimensional. Projecting the columns of $\mathbf{X}$ **at least approximately** to a q-dimensional space leaves the overall structure of the data intact.

But how do we choose the projection?

Dimension reduction through random projection (I)

Recall: To project the data onto the first principal component direction $\mathbf{r}_1$ it was enough to compute

$$\mathbf{p}_1 = \mathbf{X}\mathbf{r}_1.$$

Dimension reduction through random projection (I)

Recall: To project the data onto the first principal component direction $\mathbf{r}_1$ it was enough to compute

$$\mathbf{p}_1 = \mathbf{X}\mathbf{r}_1.$$

Let ω_i for $i = 1, \ldots, q$ be random vectors (e.g. with standard normal entries). Then, the vectors

$$\mathbf{y}_i = \mathbf{X}\omega_i$$

are called **random projections** and can be shown to be **linearly independent with high probability**.

Dimension reduction through random projection (I)

Recall: To project the data onto the first principal component direction $\mathbf{r}_1$ it was enough to compute

$$\mathbf{p}_1 = \mathbf{X}\mathbf{r}_1.$$

Let ω_i for $i = 1, \ldots, q$ be random vectors (e.g. with standard normal entries). Then, the vectors

$$\mathbf{y}_i = \mathbf{X}\omega_i$$

are called **random projections** and can be shown to be **linearly independent with high probability**.

This can be seen as a **cheap** and **approximate** way of exploring the range of $\mathbf{X}$.

Dimension reduction through random projection (I)

Recall: To project the data onto the first principal component direction $\mathbf{r}_1$ it was enough to compute

$$\mathbf{p}_1 = \mathbf{X}\mathbf{r}_1.$$

Let ω_i for $i = 1, \ldots, q$ be random vectors (e.g. with standard normal entries). Then, the vectors

$$\mathbf{y}_i = \mathbf{X}\omega_i$$

are called **random projections** and can be shown to be **linearly independent with high probability**.

This can be seen as a **cheap** and **approximate** way of exploring the range of $\mathbf{X}$.

Why is this a justifable strategy?

Johnson-Lindenstrauss lemma (1984)

Given $0 < \varepsilon < 1$ and an integer n let

$$q \geq \frac{4\log(n)}{\varepsilon^2/2 - \varepsilon^3/3}$$

be an integer. For every set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{R}^p$, there is a mapping $f : \mathbb{R}^p \to \mathbb{R}^q$ such that for any $\mathbf{x}_i, \mathbf{x}_j$

$$(1-\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \leq \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2^2 \leq (1+\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$$

Note: The result is **independent of p**.

Johnson-Lindenstrauss lemma (II)

For small ε the exact result is mainly of interest for $p \gg n$.

n	ε	$q_{\min}$
3	0.1	942
50	0.05	12951
	0.1	3354
	0.5	188
100	0.1	3948
1000	0.1	5921

Note: In practice, the dimension of the data is reduced to any useful dimension. However, be aware that the theoretical guarantees potentially are lost.

Random projection

There are multiple possibilities how the map f in the **Johnson-Lindenstrauss theorem** can be found.

Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ be a data matrix and q the target dimension.

- **Gaussian random projection:** Set

$$\Omega_{ij} \sim N\left(0, \frac{1}{q}\right) \quad \text{for} \quad i = 1, \ldots, p,\ j = 1, \ldots, q$$

- **Sparse random projection:** For a given $s > 0$ set

$$\Omega_{ij} = \sqrt{\frac{s}{q}} \begin{cases} -1 & \\ 0 & \text{with probability} \\ 1 & \end{cases} \begin{matrix} 1/(2s) \\ 1 - 1/s \\ 1/(2s) \end{matrix}$$

  for $i = 1, \ldots, p,\ j = 1, \ldots, q$ where often $s = 3$ or $s = \sqrt{p}$

then $\mathbf{Y} = \mathbf{X\Omega} \in \mathbb{R}^{n \times q}$ is a **random projection** for $\mathbf{X}$.

Random projections and the Johnson-Lindenstrauss lemma

Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ where $X^{(i,j)} \sim N(0, 1/\sqrt{p})$, $n = 3$, $\varepsilon = 0.1$ and Gaussian random projections onto the minimum JL dimension $q = 942$ were performed.

p	$(1-\varepsilon)\\|\mathbf{x}_i - \mathbf{x}_j\\|$	$\\|\mathbf{\Omega x}_i - \mathbf{\Omega x}_j\\|$	$(1+\varepsilon)\\|\mathbf{x}_i - \mathbf{x}_j\\|$
3	0.78	0.88	0.95
	1.12	1.22	1.37
	0.67	0.72	0.82
1000	1.26	1.40	1.54
	1.23	1.37	1.50
	1.25	1.35	1.52
15000	1.26	1.40	1.54
	1.27	1.42	1.56
	1.28	1.44	1.56

Dimension reduction through random projection (II)

Let $q < \min(n, p)$, $\mathbf{\Omega} \in \mathbb{R}^{p \times q}$ a **random projection matrix** and set $\mathbf{Y} = \mathbf{X\Omega}$.

Dimension reduction through random projection (II)

Let $q < \min(n, p)$, $\mathbf{\Omega} \in \mathbb{R}^{p \times q}$ a **random projection matrix** and set $\mathbf{Y} = \mathbf{X}\mathbf{\Omega}$.

A q-dimensional subspace of the range of $\mathbf{X}$ can be found by orthonormalising $\mathbf{Y}$ using e.g. the **QR-decomposition** (computational complexity $O(nq^2 - q^3/3)$)

$$\mathbf{Y} = \mathbf{Q}\mathbf{R}$$

where $\mathbf{Q} \in \mathbb{R}^{n \times q}$ has orthogonal columns and $\mathbf{R} \in \mathbb{R}^{q \times q}$ is upper-triangular.

Dimension reduction through random projection (II)

Let $q < \min(n, p)$, $\boldsymbol{\Omega} \in \mathbb{R}^{p \times q}$ a **random projection matrix** and set $\mathbf{Y} = \mathbf{X}\boldsymbol{\Omega}$.

A q-dimensional subspace of the range of $\mathbf{X}$ can be found by orthonormalising $\mathbf{Y}$ using e.g. the **QR-decomposition** (computational complexity $O(nq^2 - q^3/3)$)

$$\mathbf{Y} = \mathbf{Q}\mathbf{R}$$

where $\mathbf{Q} \in \mathbb{R}^{n \times q}$ has orthogonal columns and $\mathbf{R} \in \mathbb{R}^{q \times q}$ is upper-triangular.

Assuming $\mathbf{X}$ is approximately of rank q it can be shown that

$$\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{X}$$

where $\mathbf{Q}\mathbf{Q}^\top \in \mathbb{R}^{n \times n}$ is a **random orthogonal projection matrix** to a q-dimensional subspace of the range of $\mathbf{X}$.

Randomized low-rank SVD

Original goal: Apply SVD in cases where both n and p are large.

Idea: Determine an approximate low-dimensional basis for the range of $\mathbf{X}$ and perform the matrix-factorisation in the low-dimensional space.

- Using a random projection $\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^\top\mathbf{X} = \mathbf{Q}\mathbf{T}$
- Note that $\mathbf{T} \in \mathbb{R}^{q \times p}$ and q is small
- Calculate the SVD of $\mathbf{T} = \underset{q\times q}{\mathbf{U}_0} \cdot \underset{q\times q}{\mathbf{D}} \cdot \underset{q\times p}{\mathbf{V}^\top}$
- Set $\mathbf{U} = \mathbf{Q}\mathbf{U}_0 \in \mathbb{R}^{n \times q}$, then $\mathbf{X} \approx \mathbf{U}\mathbf{D}\mathbf{V}^\top$

The SVD of $\mathbf{X}$ can therefore be found by **random projection** into a q-dimensional subspace of the range of $\mathbf{X}$, performing **SVD in the lower-dimensional subspace** and subsequent **reconstruction** of the vectors into the original space.

Notes on randomized low-rank SVD

- In practice the matrix $\mathbf{X}$ will most-likely not have rank q but rather a continuous spectrum of eigenvalues that go towards zero
- Possible solutions:
  - **Oversampling**: Create a random projection matrix of size $p \times (q + k)$ where k is a small integer. Setting $k = 5$ or 10 is often enough in practice
  - **Power iterations:** Instead of $\mathbf{Y} = \mathbf{X\Omega}$ consider $\mathbf{Y} = (\mathbf{XX}^\top)^l \mathbf{X\Omega}$ for some integer l. This ensures that small eigenvalues of $\mathbf{X}$ are forced to zero and only large eigenvalues are dominant.

Notes on randomized low-rank SVD

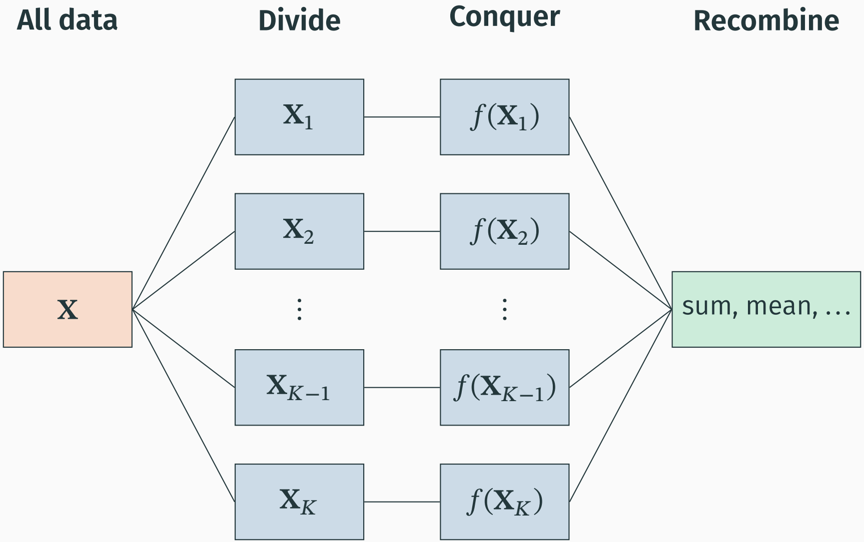
- In practice the matrix $\mathbf{X}$ will most-likely not have rank q but rather a continuous spectrum of eigenvalues that go towards zero
- Possible solutions:
  - **Oversampling**: Create a random projection matrix of size $p \times (q + k)$ where k is a small integer. Setting $k = 5$ or 10 is often enough in practice
  - **Power iterations:** Instead of $\mathbf{Y} = \mathbf{X\Omega}$ consider $\mathbf{Y} = (\mathbf{XX}^\top)^l \mathbf{X\Omega}$ for some integer l. This ensures that small eigenvalues of $\mathbf{X}$ are forced to zero and only large eigenvalues are dominant.
- The idea of randomized computation can be applied to other algorithms as well, e.g. PCA, eigenvalues, …

Notes on randomized low-rank SVD

- In practice the matrix $\mathbf{X}$ will most-likely not have rank q but rather a continuous spectrum of eigenvalues that go towards zero
- Possible solutions:
  - **Oversampling**: Create a random projection matrix of size $p \times (q + k)$ where k is a small integer. Setting $k = 5$ or 10 is often enough in practice
  - **Power iterations:** Instead of $\mathbf{Y} = \mathbf{X}\boldsymbol{\Omega}$ consider $\mathbf{Y} = (\mathbf{X}\mathbf{X}^\top)^l \mathbf{X}\boldsymbol{\Omega}$ for some integer l. This ensures that small eigenvalues of $\mathbf{X}$ are forced to zero and only large eigenvalues are dominant.
- The idea of randomized computation can be applied to other algorithms as well, e.g. PCA, eigenvalues, ...
- Implemented in R package *rsvd* or Python's *sklearn* (as *randomized_svd*)

Divide and conquer

Divide and conquer



Example: Divide and Conquer for linear regression

In linear regression, we want to find the regression coefficients $\widehat{\boldsymbol{\beta}}$, which can be calculated as

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

Example: Divide and Conquer for linear regression

In linear regression, we want to find the regression coefficients $\widehat{\beta}$, which can be calculated as

$$\widehat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Divide the data into K parts $\mathbf{X}_1, \ldots, \mathbf{X}_K$, such that $\mathbf{X}$ is the row concatenation of its parts.

Example: Divide and Conquer for linear regression

In linear regression, we want to find the regression coefficients $\widehat{\beta}$, which can be calculated as

$$\widehat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Divide the data into K parts $\mathbf{X}_1, \dots, \mathbf{X}_K$, such that $\mathbf{X}$ is the row concatenation of its parts.Then estimate (**conquer**)

$$\widehat{\beta}_k = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{y}_k$$

Example: Divide and Conquer for linear regression

In linear regression, we want to find the regression coefficients $\widehat{\beta}$, which can be calculated as

$$\widehat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Divide the data into K parts $\mathbf{X}_1, \dots, \mathbf{X}_K$, such that $\mathbf{X}$ is the row concatenation of its parts.Then estimate (**conquer**)

$$\widehat{\beta}_k = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{y}_k$$

To **recombine** the parts, consider that

$$\widehat{\beta} = \left(\sum_k \mathbf{X}_k^\top \mathbf{X}_k \right)^{-1} \left(\sum_k \mathbf{X}_k^\top \mathbf{X}_k \widehat{\beta}_k \right)$$

This means that $\widehat{\beta}_k$ and $\mathbf{X}_k^\top \mathbf{X}_k \in \mathbb{R}^{p \times p}$ have to be returned from each batch.

Example: Divide and Conquer for linear regression

In linear regression, we want to find the regression coefficients $\widehat{\beta}$, which can be calculated as

$$\widehat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Divide the data into K parts $\mathbf{X}_1, \dots, \mathbf{X}_K$, such that $\mathbf{X}$ is the row concatenation of its parts.Then estimate (**conquer**)

$$\widehat{\beta}_k = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{y}_k$$

To **recombine** the parts, consider that

$$\widehat{\beta} = \left(\sum_k \mathbf{X}_k^\top \mathbf{X}_k \right)^{-1} \left(\sum_k \mathbf{X}_k^\top \mathbf{X}_k \widehat{\beta}_k \right)$$

This means that $\widehat{\beta}_k$ and $\mathbf{X}_k^\top \mathbf{X}_k \in \mathbb{R}^{p \times p}$ have to be returned from each batch.

Note: Since $\mathrm{Cov}(\widehat{\beta}_k) = \sigma^2 (\mathbf{X}_k^\top \mathbf{X}_k)^{-1}$ the recombination is a weighted average of the batch estimates. Here, σ^2 is the variance of the residual error.

Example: Divide and Conquer for general estimation problems

In a **general estimation problem** (regression or MLE) there is often a need to solve the **score equation**

$$\sum_{l=1}^{n} \boldsymbol{\Psi}(y_l; \mathbf{x}_l, \theta) = \mathbf{0}$$

where y_l is a response, $\mathbf{x}_l$ a vector of predictors, and θ a vector of parameters.

Examples:

- Normal equations in linear regression $\sum_{l=1}^{n}(y_l - \mathbf{x}_l^\top \boldsymbol{\beta})\mathbf{x}_l = \mathbf{0}$
- Maximum likelihood estimation $\sum_{l=1}^{n} \frac{\partial \log f(y_l; \mathbf{x}_l, \theta)}{\partial \theta} = \mathbf{0}$

Advanced example (II)

To apply **Divide and Conquer** to this problem, divide the data into K subsets S_k and solve the subproblems

$$\mathbf{M}_k(\theta) = \sum_{l \in S_k} \Psi(y_l; \mathbf{x}_l, \theta) = \mathbf{0}$$

Per batch, the estimate is $\hat{\theta}_k$.

Compute

$$\mathbf{A}_k(\theta) := -\frac{\mathrm{d}\mathbf{M}_k(\theta)}{\mathrm{d}\theta} = -\sum_{l \in S_k} \frac{\partial \boldsymbol{\Psi}(y_l; \mathbf{x}_l, \theta)}{\partial \theta}$$

and use the 1st order Taylor expansion of $\mathbf{M}_k$ in $\hat{\theta}_k$ to get

$$\mathbf{M}_k(\theta) \approx \mathbf{A}_k(\hat{\theta}_k)\left(\theta - \hat{\theta}_k\right)$$

Advanced example (III)

Returning to the full problem of solving the score equation

$$\mathbf{0} = \sum_{l=1}^{n} \boldsymbol{\Psi}(y_l; \mathbf{x}_l, \theta) = \sum_{k=1}^{K} \mathbf{M}_k(\theta) \approx \sum_{k=1}^{K} \mathbf{A}_k(\hat{\theta}_k)\left(\theta - \hat{\theta}_k\right)$$

The solution to the approximation is then given by

$$\hat{\theta} = \left(\sum_{k=1}^{K} \mathbf{A}_k(\hat{\theta}_k)\right)^{-1} \left(\sum_{k=1}^{K} \mathbf{A}_k(\hat{\theta}_k)\hat{\theta}_k\right)$$

Note: For this approximation the per-batch covariance matrices $\mathbf{X}_k^\top \mathbf{X}_k$ are replaced by the matrices $\mathbf{A}_k(\hat{\theta}_k)$.

In case of the MLE example

$$\mathbf{A}_k(\hat{\theta}_k) = -\sum_{l \in S_k} \frac{\partial^2 \log f(y_l; \mathbf{x}_l, \theta)}{\partial \theta^2}$$

which is the **observed Fisher information**.

Sampling methods for big-n

Recap: Random Forests

Computational procedure:

1. Given training data $\mathbf{X} \in \mathbb{R}^{n \times p}$, do for $b = 1, \dots, B$
   1.1 Draw a **bootstrap sample of size n** from training data (with replacement)
   1.2 Grow a tree T_b until nodes are pure or reach minimal node size $n_{\min}$
      1.2.1 Randomly select m variables out of p variables
      1.2.2 Find best splitting variable among these m
      1.2.3 Split the node
2. For a new $\mathbf{x}$ predict

Regression: $\widehat{f}_{rf}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} T_b(\mathbf{x})$

Classification: Majority vote at $\mathbf{x}$ across trees

For big-n: In principal all trees can be grown in parallel. However, this requires B bootstrap samples of size n which can be infeasibly large in a big-n scenario.

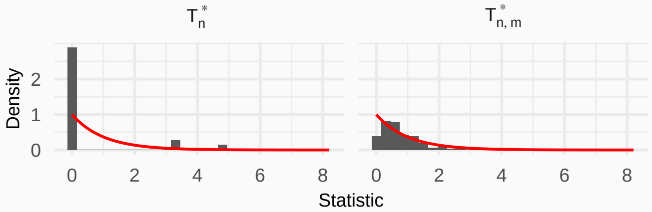
The m-out-of-n bootstrap

Instead of drawing a bootstrap sample of n samples with replacement (as in the standard bootstrap), a smaller sample of size $m < n$ is drawn *with replacement*.

- **Note:** If $m < n$ samples are drawn *without replacement*, then this is called **subsampling**.
- Surprisingly, the m-out-of-n bootstrap (moon bootstrap) works even in situations where the standard bootstrap fails
- For the theoretical guarantees to hold, it is required that when $m, n \to \infty$ then $m/n \to 0$
- $m = 2\sqrt{n}$ is a possible choice

Example: m-out-of-n bootstrap

- Let $x_1, \dots, x_n \sim \mathrm{Uniform}(0, \theta)$ and $\hat{\theta}_n = \max_i x_i$.
- Consider the statistics
  - $T_n = n(\theta - \hat{\theta}_n)$, the statistic to be approximated
  - $T_n^* = n(\hat{\theta}_n - \hat{\theta}_n^*)$ where $\hat{\theta}_n^* = \max_i x_i^*$ for a standard bootstrap sample $x_1^*, \dots, x_n^*$
  - $T_{n,m}^* = m(\hat{\theta}_n - \hat{\theta}_{n,m}^*)$ where $\hat{\theta}_{n,m}^* = \max_i x_i^*$ for a standard bootstrap sample $x_1^*, \dots, x_m^*$
- Simulated data with $n = 1000$, $m = 2\sqrt{1000} \approx 64$, $B = 10000$, and $\theta = 1$



The red line is the density of T_n given the true θ.

Bag of little bootstraps (BLB)

A **two-stage bootstrapping technique**

1. Draw K subsets of size $m < n$ from original data (with or without replacement)
2. For each subset
   2.1 Draw B set of weights $(n_1, \dots, n_m) \sim \mathrm{Multinomial}(n, 1/m)$ (**oversampling**)
   2.2 Estimate the statistic of interest from the B weighted samples
   2.3 Combine values of the statistic for each subset, e.g. by averaging
3. Recombine statistics from each subset, e.g. by averaging

This is known as the **bag of little bootstraps (BLB)** (Kleiner et al. 2014)

Notes on the BLB

- One of the computational burdens of the standard bootstrap is having to create resamples of size n
- The BLB circumvents that by resampling from a limited amount of samples and thereby being able to use weights instead of a full sample
- Typically $m \geq n^{\gamma}$ for $\gamma \in [0.5, 1]$ works well (e.g. for $\gamma = 0.6$: when $n = 10^6$ choose $m = 3982$)
- The BLB is easier to parallelise, since less data has to be propagated to each batch.
- Fits well within the **Divide and Conquer** framework

Random forests for big-n

Instead of the standard RF with normal bootstrapping, multiple strategies can be taken

- **Subsampling (once):** Take a subsample of size m and grow RF from there. Very simple to implement, but difficult to ensure that the subsample is representative.
- **m-out-of-n sampling:** Instead of standard bootstrapping, draw repeatedly m samples and grow a tree on each subsample. Recombine trees in the usual fashion.
- **BLB sampling:** Grow a forest on each subset by repeatedly oversampling to n samples.
- **Divide and Conquer:** Split original data in K parts and grow a random forest on each.

Subsampling for big-n

Leverage

Problem: Representativeness

How can we ensure that a subsample is still representative?

We need **additional information** about the samples. Consider the special case of linear regression and $n \gg p$.

Recall: For least squares predictions it holds that

$$\hat{\mathbf{y}} = \mathbf{X}\widehat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{H}\mathbf{y}$$

with the **hat-matrix** $\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$.

Specifically $\hat{\mathbf{y}}^{(i)} = \sum_{j=1}^{n} \mathbf{H}^{(i,j)}\mathbf{y}^{(j)}$, which means that $\mathbf{H}^{(i,i)}$ influences its own fitted values.

Element $\mathbf{H}^{(i,i)}$ is called the **leverage** of the i-th observation. Leverage captures if the observation i is close or far from the centre of the data in feature space.

Leveraging (I)

Goal: Subsample the data, but make the **more influential** data points, those with **high leverage**, more likely to be sampled.

Computational approach

- Weight sample i by
$$\pi_i = \frac{\mathbf{H}^{(i,i)}}{\sum_{j=1}^{n} \mathbf{H}^{(j,j)}}$$
- Draw a weighted subsample of size $m \ll n$
- Use the subsample to solve the regression problem

This procedure is called **Leveraging** (Ma and Sun, 2013).

Leveraging (II)

Problem: How to perform regression?

1. **Ordinary least squares:** Biased with regard to the full sample estimate, due to subsampling, but unbiased with respect to the true coefficients and generally small variance
2. **Weighted least squares:** Use the inverse sampling weights $1/\pi_i$ as weights during the regression. Unstable for very small weights, i.e. high variance. Weights can be stabilized by using

$$\tau_i = \alpha\pi_i + (1-\alpha)\frac{1}{n}$$

instead of π_i for α recommended at 0.8–0.9.

Leveraging (III)

Problem: How should the diagonal entries of the hat matrix be determined without having to solve the original regression problem?

Let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ be the SVD of the data matrix, then

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top = \mathbf{U}\mathbf{U}^\top$$

and therefore, with $\mathbf{u}_i$ being the i-th row of $\mathbf{U}$,

$$\mathbf{H}^{(i,i)} = \|\mathbf{u}_i\|_2^2$$

Using e.g. randomized SVD or other fast computational approaches, this is feasible for very large data.

Notes on leveraging

- **Pro:** Fast and simple approach to make subsampling more focused on the important samples
- **Pro:** Smaller datasets are easier to use computationally, but also visualisations get feasible again
- **Caveat:** Careful with outliers! These often have large leverage, but are misrepresentative of the actual shape of the data.

Take-home message

- Large-scale data brings its own challenges, many of which are computational
- Randomization can help to speed up classical algorithms in practice
- Divide and Conquer can help in $n \gg p$ and big-n scenarios; can be non-trivial to determine how to recombine
- Subsampling/clever bootstrapping can reduce the necessary computational load tremendously