DMDR: R Package for “Distributed Mean Dimension Reduction Through Semi-parametric Approaches”

Xie, Zhongming

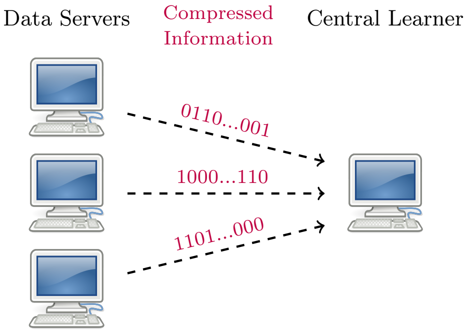
Nov 26, 2024

Available at `https://github.com/Chia202/DMDR`

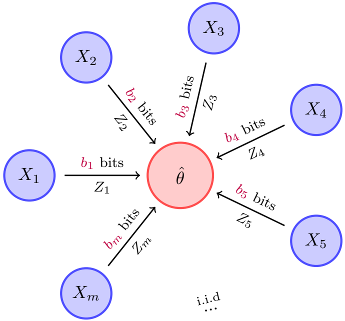
Content

- Distributed Learning
- Mean Dimension Reduction
- Algorithm 1 (Dense Solution)
- Algorithm 2 (Sparse Solution)
- Experiments
- Conclusion

Distributed Learning



(a) Distributed learning network



(b) Distributed protocol

(a) An illustration of a distributed learning network. (b) An illustration of distributed protocol.

Credit: Cai and Wei (2020)

Examples of Distributed Learning

- Estimating population mean

$$\hat{\mu} = \frac{1}{n} \sum_{j=1}^{m} \bar{X}_j$$

- Estimating linear regression coefficients

$$\hat{\beta} = \left(X_1^T X_1 + \cdots + X_m^T X_m \right)^{-1} \left(X_1^T Y_1 + \cdots + X_m^T Y_m \right)$$

- Stochastic gradient descent

$$\beta_{t+1} = \beta_t - \eta \frac{1}{m} \nabla f_i(\beta_t)$$

Mean Dimension Reduction

We assume that for covariates $\mathbf{x} \in \mathbb{R}^p$ and response $Y \in \mathbb{R}$, there exists $\beta \in \mathbb{R}^{p \times d}$ (usually $d < p$) such that

$$\mathbb{E}\left(Y|\mathbf{x}\right) = \mathbb{E}\left(Y|\beta^T\mathbf{x}\right)$$

It is equivalent that for some unknown function $m(\cdot)$,

$$Y = m(\beta^T\mathbf{x}) + \varepsilon, \quad \mathbb{E}\left(\varepsilon|\mathbf{x}\right) = 0$$

Our goal is to estimate the column space of β denoted by $S_{\mathbb{E}(Y|\mathbf{x})}$.
We may assume that the upper $d \times d$ submatrix of β is identity matrix and the rest $(p - d) \times d$ elements are free.

Mean Dimension Reduction

Define the inverse variance function as $w(\mathbf{x}) = \left(\mathbb{E}\left(\epsilon^2|\mathbf{x}\right)\right)^{-1}$ and the gradient of m as $\mathbf{m}_1\left(\beta^T\mathbf{x}\right) = \partial m(\beta^T\mathbf{x})/\partial\left(\beta^T\mathbf{x}\right)$.
Ma and Zhu (2014) showed that β satisfies

$$\mathbb{E}\left(\mathbf{S}\{\mathbf{x}, Y, \alpha, w(\mathbf{x})\}\right) \stackrel{\text{def}}{=} \mathbb{E}\left(\left\{Y - m\left(\mathbf{x}^T\alpha\right)\right\} w(\mathbf{x})\widetilde{\mathbf{x}}(\alpha)\right) = 0$$

where

$$\widetilde{\mathbf{x}}(\alpha) \stackrel{\text{def}}{=} \text{vecl}\left\{\left(\mathbf{x} - \frac{\mathbb{E}\left\{\mathbf{x}w(\mathbf{x}) \mid \mathbf{x}^T\alpha\right\}}{\mathbb{E}\left\{w(\mathbf{x}) \mid \mathbf{x}^T\alpha\right\}}\right)\mathbf{m}_1^T\left(\mathbf{x}^T\alpha\right)\right\} \in \mathbb{R}^{(p-d)d\times 1}.$$

We call $\mathbf{S}(\mathbf{x}, Y, \alpha, w(\mathbf{x}))$ the score function. Solving the sample version of the above equation gives us the estimate of β.

Newton-Raphson Method

The gradient of $\mathbb{E}\left(\mathbf{S}\{\mathbf{x}, Y, \alpha, w(\mathbf{x})\}\right)$ with respect to α is

$$-\mathbf{H}(\alpha) \overset{\text{def}}{=} -\mathbb{E}\left(w(\mathbf{x})\{\widetilde{\mathbf{x}}(\alpha)\}\{\widetilde{\mathbf{x}}(\alpha)\}^T\right).$$

Starting with an initial value $\beta^{(0)}$, the Newton-Raphson iteration proceeds as

$$\operatorname{vecl}\left(\beta^{(t+1)}\right) \overset{\text{def}}{=} \operatorname{vecl}\left(\beta^{(t)}\right) + \left\{\mathbf{H}\left(\beta^{(t)}\right)\right\}^{-1} E\left(\mathbf{S}\left\{\mathbf{x}, Y, \beta^{(t)}, w(\mathbf{x})\right\}\right).$$

To recast the above iteration in a least-squares framework, define

$$\widetilde{Y}(\alpha) \stackrel{\text{def}}{=} \{\widetilde{\mathbf{x}}(\alpha)\}^T \operatorname{vecl}(\alpha) + \left\{Y - m\left(\mathbf{x}^T \alpha\right)\right\}.$$

Newton-Raphson iteration can equivalently be written as

$$\operatorname{vecl}\left(\beta^{(t+1)}\right) = \left\{\mathbf{H}\left(\beta^{(t)}\right)\right\}^{-1} E\left(w(\mathbf{x})\widetilde{\mathbf{x}}\left(\beta^{(t)}\right)\widetilde{Y}\left(\beta^{(t)}\right)\right),$$

which minimizes the following weighted least squares loss:

$$\operatorname{vecl}\left(\beta^{(t+1)}\right) = \arg\min_{\alpha} E\left(\left\{\widetilde{Y}\left(\beta^{(t)}\right) - \widetilde{\mathbf{x}}\left(\beta^{(t)}\right)^T \operatorname{vecl}(\alpha)\right\}^2 w(\mathbf{x})\right).$$

Distributed Mean Dimension Reduction

Assume we have m machines with n observations each, denote by
$\left\{ \left(\mathbf{x}_{i,j}, Y_{i,j} \right), i = 1, \ldots, n, j = 1, \ldots, m \right\}$.
The idea is to

- Estimate score function at each machine.
- Average the score function across all machines.
- On the first machine, compute Hessian matrix $\mathbf{H}_1(\beta)$.
- Update β using the average score function and Hessian matrix.

Algorithm 1: Dense Solution

Estimate the following quantities at each j-th machine:

1. $\hat{m}_j(\mathbf{x}_{kj}^T\alpha), \hat{\mathbf{m}}_{1j}(\mathbf{x}_{kj}^T\alpha) =$

$$\arg\min_{b_{kj},\mathbf{b}_{kj}} \sum_{i=1,i\neq k}^{n} \left\{Y_{ij} - b_{kj} - \left(\mathbf{x}_{ij}^T\alpha - \mathbf{x}_{kj}^T\alpha\right)\mathbf{b}_{kj}\right\}^2 K_{h_1}\left(\mathbf{x}_{ij}^T\alpha - \mathbf{x}_{kj}^T\alpha\right)$$

2. Inverse variance function $\hat{w}_j(\mathbf{x}_{kj}) =$

$\sum_{i=1,i\neq k}^{n} K_h\left(\mathbf{x}_{ij}^T - \mathbf{x}_{kj}^T\right) / \sum_{i=1,i\neq k}^{n} K_h\left(\mathbf{x}_{ij}^T - \mathbf{x}_{kj}^T\right)\left(Y_{ij} - \hat{m}_j\left(\mathbf{x}_{ij}^T\alpha\right)\right)^2$

Algorithm 1: Dense Solution

3. Compute the conditional expectation of $w(\mathbf{x})$ and $\mathbf{x}w(\mathbf{x})$:

$$\hat{\mathbb{E}}_j\left(\hat{w}(\mathbf{x}_{kj})|\mathbf{x}_{kj}^T\alpha\right) = \frac{\sum_{i=1,i\neq k}^{n} K_{h_2}\left(\mathbf{x}_{ij}^T\alpha - \mathbf{x}_{kj}^T\alpha\right)\hat{w}_j(\mathbf{x}_{ij})}{\sum_{i=1,i\neq k}^{n} K_{h_2}\left(\mathbf{x}_{ij}^T\alpha - \mathbf{x}_{kj}^T\alpha\right)}$$

$$\hat{\mathbb{E}}_j\left(\mathbf{x}_{kj}\hat{w}_j(\mathbf{x}_{kj})|\mathbf{x}_{kj}^T\alpha\right) = \frac{\sum_{i=1,i\neq k}^{n} K_{h_3}\left(\mathbf{x}_{ij}^T\alpha - \mathbf{x}_{kj}^T\alpha\right)\left\{\mathbf{x}_{ij}\hat{w}_j(\mathbf{x}_{ij})\right\}}{\sum_{i=1,i\neq k}^{n} K_{h_3}\left(\mathbf{x}_{ij}^T\alpha - \mathbf{x}_{kj}^T\alpha\right)}$$

4. Accordingly, compute

$$\hat{\tilde{\mathbf{x}}}_j(\alpha) = \operatorname{vecl}\left\{\left[\mathbf{x}_{kj} - \frac{\hat{\mathbb{E}}_j\left(\mathbf{x}_{kj}\hat{w}_j(\mathbf{x}_{kj})|\mathbf{x}_{kj}^T\alpha\right)}{\hat{\mathbb{E}}_j\left(\hat{w}(\mathbf{x}_{kj})|\mathbf{x}_{kj}^T\alpha\right)}\right]\hat{\mathbf{m}}_{1j}^T(\mathbf{x}_{kj}^T\alpha)\right\}$$

Algorithm 1: Dense Solution

5. Compute score function locally, and average across all machines:

$$\hat{\mathbf{S}}_j\left\{\mathbf{x}_{kj}, Y_{kj}, \alpha, \hat{w}_j(\mathbf{x}_{kj})\right\} = \left\{Y_{kj} - \hat{m}_j\left(\mathbf{x}_{kj}^T\alpha\right)\right\}\hat{w}_j(\mathbf{x}_{kj})\hat{\tilde{\mathbf{x}}}_j(\alpha)$$

$$\hat{\mathbb{E}}_j\left[\mathbf{S}\left\{\mathbf{x}, Y, \alpha, w(\mathbf{x})\right\}\right] = \frac{1}{n}\sum_{k=1}^{n}\hat{\mathbf{S}}_j\left\{\mathbf{x}_{kj}, Y_{kj}, \alpha, \hat{w}_j(\mathbf{x}_{kj})\right\}$$

6. Compute the Hessian matrix at the first machine:

$$\hat{\mathbf{H}}_1(\alpha) = \frac{1}{n}\sum_{k=1}^{n}\hat{w}_j(\mathbf{x}_{kj})\hat{\tilde{\mathbf{x}}}_j(\alpha)\hat{\tilde{\mathbf{x}}}_j(\alpha)^T$$

7. Upate β using the following formula:

$$\beta^{(t+1)} = \beta^{(t)} + \left(\hat{\mathbf{H}}_1\left(\beta^{(t)}\right)\right)^{-1}\frac{1}{m}\sum_{j=1}^{m}\hat{\mathbb{E}}_j\left[\mathbf{S}\left\{\mathbf{x}, Y, \beta^{(t)}, w(\mathbf{x})\right\}\right]$$

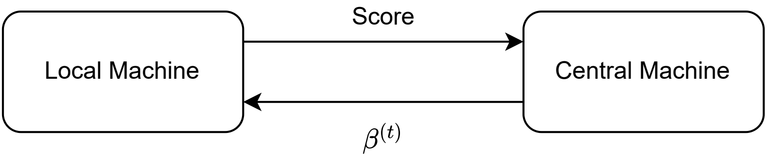
Algorithm 1: Dense Solution

Using the Adam optimizer to mitigate the gradient explosion. The update rule is

$$m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1)\nabla f(\beta^{(t)})$$

$$v^{(t+1)} = \beta_2 v^{(t)} + (1-\beta_2)\left(\nabla f(\beta^{(t)})\right)^2$$

$$\beta^{(t+1)} = \beta^{(t)} + \eta\frac{m^{(t+1)}}{\sqrt{v^{(t+1)}+\epsilon}}$$



Implementation of Algorithm 1

```
mat dmdrDense(data) {
  // Whiten data
  ...
  // Iterate machines
#pragma omp parallel for
for (machine in machines) {
  // Call estimateScore();
  // Collect scores;
}
  // Adam optimizer
  grad=H^(-1)*mean(scores);
  m=b1*m+(1-b1)*grad;
  v=b2*v+(1-b2)*grad^2;
  beta=beta+m/sqrt(v);
  return beta;
}
```

```
estimateScore(Local data) {
  // Estimate m, m1;
  // Estimate w;
  // Estimate E(w|x), E(xw|x);
  // Estimate tilde X;
  // Estimate score;
  return score;
}
```

Algorithm 2: Sparse Solution

Recall that algorithm 1 has a equivalent least squares formulation.

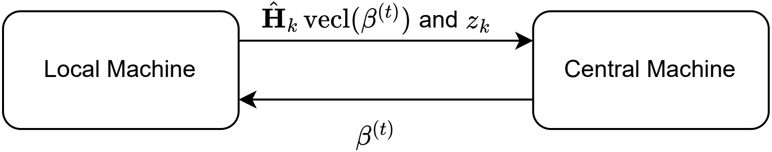
$$
\begin{aligned}
\mathcal{L}_N(\alpha) &= \frac{1}{2N}\sum_{i=1}^{n}\sum_{k=1}^{m}\left\{\widehat{Y}_{i,k}\left(\beta^{(t)}\right) - \widehat{\tilde{\mathbf{x}}}_{i,k}\left(\beta^{(t)}\right)^T \operatorname{vecl}(\alpha)\right\}^2 \tilde{w}\left(\mathbf{x}_{i,k}\right) \\
&\equiv \frac{1}{2}\operatorname{vecl}(\alpha-\beta^{(t)})^T \hat{\mathbf{H}} \operatorname{vecl}(\alpha-\beta^{(t)}) + \operatorname{vecl}(\alpha-\beta^{(t)})^T(\hat{\mathbf{H}}\operatorname{vecl}(\beta^{(t)}) - \mathbf{z}_N) \\
&= \frac{1}{2}\operatorname{vecl}(\alpha-\beta^{(t)})^T \hat{\mathbf{H}}_1 \operatorname{vecl}(\alpha-\beta^{(t)}) + \operatorname{vecl}(\alpha-\beta^{(t)})^T(\hat{\mathbf{H}}\operatorname{vecl}(\beta^{(t)}) - \mathbf{z}_N) \\
&\quad + \frac{1}{2}\operatorname{vecl}(\alpha-\beta^{(t)})^T\left(\hat{\mathbf{H}} - \hat{\mathbf{H}}_1\right)\operatorname{vecl}(\alpha-\beta^{(t)}) \\
&\approx \frac{1}{2}\operatorname{vecl}(\alpha-\beta^{(t)})^T \hat{\mathbf{H}}_1 \operatorname{vecl}(\alpha-\beta^{(t)}) + \operatorname{vecl}(\alpha-\beta^{(t)})^T(\hat{\mathbf{H}}\operatorname{vecl}(\beta^{(t)}) - \mathbf{z}_N) \\
&\equiv \frac{1}{2}\operatorname{vecl}(\alpha)^T(\hat{\mathbf{H}}_1)\operatorname{vecl}(\alpha) + \operatorname{vecl}(\alpha)^T\left((\hat{\mathbf{H}} - \hat{\mathbf{H}}_1)\operatorname{vecl}(\beta^{(t)}) - \mathbf{z}_N\right)
\end{aligned}
$$

where $\mathbf{z}_N = \frac{1}{m}\sum_{k=1}^{m} z_k = \frac{1}{m}\sum_{k=1}^{m}\frac{1}{n}\sum_{i=1}^{n}\hat{w}_k(\mathbf{x}_{ik})\hat{\tilde{\mathbf{x}}}_{ik}(\beta^{(t)})\hat{Y}_{ik}(\beta^{(t)})$.

Algorithm 2: Sparse Solution

- Estimate $\hat{H}_k \operatorname{vecl}(\beta^{(t)})$ and $\mathbf{z}_k$ at each machine.
- Average $\hat{H}_k \operatorname{vecl}(\beta^{(t)})$s and $\mathbf{z}_k$s at the first machine.
- Update β by solving

$$\beta^{(t+1)} = \arg\min_{\alpha} \left(\frac{1}{2} \operatorname{vecl}(\alpha)^T \hat{\mathbf{H}}_1 \operatorname{vecl}(\alpha) + \operatorname{vecl}(\alpha)^T \left[\left(\hat{\mathbf{H}} - \hat{\mathbf{H}}_1 \right) \operatorname{vecl}(\beta^{(t)}) - \mathbf{z}_N \right] + \lambda \|\alpha\|_1 \right).$$



Implementation of Algorithm 2

```
mat dmdrSparse(data) {
  // Whiten data
  ...
  // Iterate machines
#pragma omp parallel for
for (machine in machines) {
  // Call estimateSparse();
  // Collect H * beta and z;
}
// Call Lasso solver;
beta = LassoSolver(H1, (H - H1)
     * beta + z);

return beta;
}
```

```
List estimateSparse(Local data)
{
  // Estimate m, m1;
  // Estimate w;
  // Estimate E(w|x), E(xw|x);
  // Estimate tilde X;
  // Estimate Hessian H;
  // Estimate z;
  return List(H * beta, z);
}
```

```
mat LassoSolver(H, coef) {
  // Coordinate descent;
}
```

Experiments

We set $p = 16$, $d = 2$, and $\beta = (\beta_1, \beta_2)$, where:

$$\beta_1 = (1, 0, 0, 1, \underbrace{0, \ldots, 0}_{p-4})^T \quad \text{and} \quad \beta_2 = (0, 1, 1, -1, \underbrace{0, \ldots, 0}_{p-4})^T.$$

Example 1: Generate $\mathbf{x} \sim N_p(0, \Sigma)$, where $\Sigma_{ij} = 0.5^{|i-j|}$, and

$$Y = \mathbf{x}^T \beta_1 + \mathbf{x}^T \beta_2 + \varepsilon, \qquad \varepsilon \sim N(0, 2^2)$$

Example 2: Generate $\mathbf{x} \sim U[-2, 2]$ and

$$Y = \frac{\mathbf{x}^T \beta_1}{0.5 + \left(1.5 + \mathbf{x}^T \beta_2\right)^2} + \varepsilon, \qquad \varepsilon \sim N(0, e^{\mathbf{x}_1})$$

Use trace correlation to evaluate the estimator: $r = \text{trace}(P_\beta P_{\hat{\beta}})/d \in [0, 1]$.

Simulation Results

Results of algorithms for different (m, n) configurations.

Example	Algorithm	$(m, n) = (5, 500)$		$(m, n) = (25, 100)$	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Dense	0.4491	0.0060	0.4578	0.0049
	Sparse	0.5775	0.0002	0.6236	0.0040
2	Dense	0.4009	0.0048	0.3770	0.0025
	Sparse	0.4292	0.0005	0.6306	0.0039

Parallelization vs. Serial Execution

Performance: Repeating Algorithm 2 thirty times with ten machines.

Example	Algorithm 2	Min	Mean	Median	Max
1	Serial	62.3605	78.4520	78.8680	81.2772
	Parallel	9.8350	10.3798	10.3288	11.6317
2	Serial	65.6747	78.6136	78.6649	82.9928
	Parallel	20.5469	21.3434	21.1136	22.8305

Conclusion

- We implement two algorithms for distributed mean dimension reduction.
- The first algorithm based on Newton-Raphson method yields dense solution.
- The second algorithm based on least squares and LASSO yields sparse solution.
- Using Adam optimizer to matigate gradient explosion.
- Using `OpenMP` to parallelize the computation.

Thank you!