Statistical Gradient Boosting of Generalised Additive Model with Ridge Regression on GPU

Antoni Sieminski[1], Johnny MyungWon Lee[1]

[1]The University of Edinburgh
School of Mathematics

6th April 2023

THE UNIVERSITY of EDINBURGH
School of Mathematics

Outline

Generalised Additive Model

The Gaussian Generalised Additive Model (GAM) is defined as:

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2)$$
$$\boldsymbol{\mu} = \beta_0 + f_1(\boldsymbol{x}_1) + \ldots + f_p(\boldsymbol{x}_p)$$

which allows us to easily evaluate the effects of each variable $\boldsymbol{x}_j$ on the response $\boldsymbol{y}$.

Generalised Additive Model

The Gaussian Generalised Additive Model (GAM) is defined as:

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2)$$
$$\boldsymbol{\mu} = \beta_0 + f_1(\boldsymbol{x}_1) + \ldots + f_p(\boldsymbol{x}_p)$$

which allows us to easily evaluate the effects of each variable $\boldsymbol{x}_j$ on the response $\boldsymbol{y}$.

For an unknown $f(x_j)$, it is usually sensible to assume it's a smooth function, which can be represented as a linear combination of rank-reduced k basis functions $b_i(.)$ multiplied by their fitted coefficients γ_i:

$$f(\boldsymbol{x}_j) = \sum_{i=1}^{k} b_i(\boldsymbol{x}_j)\gamma_i = \mathbf{B}_j\boldsymbol{\gamma}_j$$

GAM – smooth functions

In order to enforce smoothness on $f_j(\boldsymbol{x}_j)$, the term $\lambda_j \boldsymbol{\gamma}_j^T \mathbf{S}_j \boldsymbol{\gamma}_j$ is typically added to the f_j's loss function, where $\mathbf{S}_j$ is the smoothing matrix and λ_j is the penalty term.

$$L = ||\boldsymbol{y} - \hat{\boldsymbol{\mu}}||^2 + \sum_{j=1}^{p} \lambda_j \boldsymbol{\gamma}_j^T \mathbf{S}_j \boldsymbol{\gamma}_j$$

GAM – smooth functions

In order to enforce smoothness on $f_j(\boldsymbol{x}_j)$, the term $\lambda_j \boldsymbol{\gamma}_j^T \mathbf{S}_j \boldsymbol{\gamma}_j$ is typically added to the f_j's loss function, where $\mathbf{S}_j$ is the smoothing matrix and λ_j is the penalty term.

$$L = ||\boldsymbol{y} - \hat{\boldsymbol{\mu}}||^2 + \sum_{j=1}^{p} \lambda_j \boldsymbol{\gamma}_j^T \mathbf{S}_j \boldsymbol{\gamma}_j$$

In our case we have chosen $\mathbf{S} = \mathbf{I}$, which results in a simple ridge penalty $\lambda_j ||\boldsymbol{\gamma}_j||^2$.

GAM – smooth functions

In order to enforce smoothness on $f_j(\boldsymbol{x}_j)$, the term $\lambda_j \boldsymbol{\gamma}_j^T \mathbf{S}_j \boldsymbol{\gamma}_j$ is typically added to the f_j's loss function, where $\mathbf{S}_j$ is the smoothing matrix and λ_j is the penalty term.

$$L = ||\boldsymbol{y} - \hat{\boldsymbol{\mu}}||^2 + \sum_{j=1}^{p} \lambda_j \boldsymbol{\gamma}_j^T \mathbf{S}_j \boldsymbol{\gamma}_j$$

In our case we have chosen $\mathbf{S} = \mathbf{I}$, which results in a simple ridge penalty $\lambda_j ||\boldsymbol{\gamma}_j||^2$. However, there are many other, more complex penalties to choose from. For example, a P-Spline smooths the function by setting $\mathbf{S} = \mathbf{D}^T\mathbf{D}$, which enforces smoothness by penalising the changes in differences of adjacent coefficients:

$$\boldsymbol{\gamma}^T \mathbf{S} \boldsymbol{\gamma} = \sum_{i+1}^{k-1} (\gamma_{i-1} - 2\gamma_i + \gamma_{i+1})^2$$

Gradient Boosting

- Gradient boosting is a fitting algorithm, which sequentially fits weak estimators (a.k.a. base learners) to the gradient of the loss function $\mathbf{g}^{[m]}$.
- The resulting model can avoid including unnecessary base learners and shrink estimates from the included ones to reduce the generalisation error of the model.
- In GAMs, the base learners are typically penalised regression models – each with a design matrix $\mathbf{B}_j$, smoothing matrix $\mathbf{S}_j$, and the smoothing parameter λ_j.

Gradient Boosting

- Gradient boosting is a fitting algorithm, which sequentially fits weak estimators (a.k.a. base learners) to the gradient of the loss function $\mathbf{g}^{[m]}$.
- The resulting model can avoid including unnecessary base learners and shrink estimates from the included ones to reduce the generalisation error of the model.
- In GAMs, the base learners are typically penalised regression models – each with a design matrix $\mathbf{B}_j$, smoothing matrix $\mathbf{S}_j$, and the smoothing parameter λ_j.

Gradient Boosting

- Gradient boosting is a fitting algorithm, which sequentially fits weak estimators (a.k.a. base learners) to the gradient of the loss function $\mathrm{g}^{[m]}$.
- The resulting model can avoid including unnecessary base learners and shrink estimates from the included ones to reduce the generalisation error of the model.
- In GAMs, the base learners are typically penalised regression models – each with a design matrix $\mathbf{B}_j$, smoothing matrix $\mathbf{S}_j$, and the smoothing parameter λ_j.

Smooth functions as base learners

At each iteration m, all K base learners are fitted to the residuals of the model $\mathbf{g}^{[m]}$:

$$\boldsymbol{\gamma}_j^{[m]} = (\mathbf{B}_j^T \mathbf{B}_j + \lambda_j \mathbf{S}_j)^{-1} \mathbf{B}_j^T \mathbf{g}^{[m]}$$

and the one with the lowest residual sum of squares is selected.

$$\text{learner index} = \arg\min_j ||\mathbf{g}^{[m]} - \mathbf{B}_j \boldsymbol{\gamma}_j^{(m)}||^2$$

The smoothing parameter λ_j is selected before the fitting process, such that the learner has some small number of effective degrees of freedom trace$(\mathbf{H}_j)$.

Smooth functions as base learners

At each iteration m, all K base learners are fitted to the residuals of the model $\mathbf{g}^{[m]}$:

$$\boldsymbol{\gamma}_j^{[m]} = (\mathbf{B}_j^T \mathbf{B}_j + \lambda_j \mathbf{S}_j)^{-1} \mathbf{B}_j^T \mathbf{g}^{[m]}$$

and the one with the lowest residual sum of squares is selected.

$$\text{learner index} = \arg\min_j ||\mathbf{g}^{[m]} - \mathbf{B}_j \boldsymbol{\gamma}_j^{(m)}||^2$$

The smoothing parameter λ_j is selected before the fitting process, such that the learner has some small number of effective degrees of freedom trace$(\mathbf{H}_j)$.

Smooth functions as base learners

At each iteration m, all K base learners are fitted to the residuals of the model $\mathbf{g}^{[m]}$:

$$\boldsymbol{\gamma}_j^{[m]} = (\mathbf{B}_j^T \mathbf{B}_j + \lambda_j \mathbf{S}_j)^{-1} \mathbf{B}_j^T \mathbf{g}^{[m]}$$

and the one with the lowest residual sum of squares is selected.

$$\text{learner index} = \arg\min_j ||\mathbf{g}^{[m]} - \mathbf{B}_j \boldsymbol{\gamma}_j^{(m)}||^2$$

The smoothing parameter λ_j is selected before the fitting process, such that the learner has some small number of effective degrees of freedom trace$(\mathbf{H}_j)$.

Initialisation of Gradient Boosting

Algorithm Initialise Gradient Boosted Ridge Regression

for $i = 1$ to K **do**
    $\lambda_i \leftarrow \underset{\lambda}{\mathrm{uniroot}}\left(edf(\mathbf{B}_i, \lambda) - \omega\right)$
    learner$_i \leftarrow \{\mathbf{B}_i; (\mathbf{B}_i^T \mathbf{B}_i + \lambda_i \mathbf{I})^{-1}\}$
end for

Iteration algorithm

Algorithm Iterate Gradient Boosted Ridge Regression

1: **for** $m = 1$ to m_{stop} **do**
2: $\quad \mathbf{g}^{[m]} \leftarrow \boldsymbol{y} - \boldsymbol{\mu}^{[m-1]}$
3: $\quad i \leftarrow \arg\min_i \text{RSS}(\text{learner}_i, \mathbf{g}^{[m]})$
4: $\quad \boldsymbol{\delta}_i^{[m]} \leftarrow \nu(\text{MatVecMul}(\mathbf{B}_i^T \mathbf{B}_i + \lambda_i \mathbf{I})^{-1}, \text{MatVecMul}(\mathbf{B}_i^T, \mathbf{g}^{[m]}))$
5: $\quad \boldsymbol{\gamma}_i^{[m]} \leftarrow \boldsymbol{\gamma}_i^{[m-1]} + \boldsymbol{\delta}_i^{[m]}$
6: $\quad \boldsymbol{\mu}^{[m]} \leftarrow \boldsymbol{\mu}^{[m-1]} + X_i \boldsymbol{\delta}_i^{[m]}$
7: **end for**

Computational considerations

In gradient boosting, finding λ for each learner and the associated $(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{S})^{-1}$ is relatively inexpensive as it can be done just once.

Operations that happen at each boosting iteration m are the main computational bottleneck and include the following matrix-vector multiplications:

1. $\mathbf{B}^T\mathbf{g}^{[m]}$, $O(np)$
2. $(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{S})^{-1}(\mathbf{B}^T\mathbf{g}^{[m]})$, $O(p^2)$
3. $\mathbf{B}\boldsymbol{\gamma}^{[m]}$, $O(np)$

These can be sped up significantly by parallel computing.

Computational considerations

In gradient boosting, finding λ for each learner and the associated $(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{S})^{-1}$ is relatively inexpensive as it can be done just once.

Operations that happen at each boosting iteration m are the main computational bottleneck and include the following matrix-vector multiplications:

1. $\mathbf{B}^T\mathbf{g}^{[m]}$, $O(np)$
2. $(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{S})^{-1}(\mathbf{B}^T\mathbf{g}^{[m]})$, $O(p^2)$
3. $\mathbf{B}\boldsymbol{\gamma}^{[m]}$, $O(np)$

These can be sped up significantly by parallel computing.

Matrix Vector Multiplication

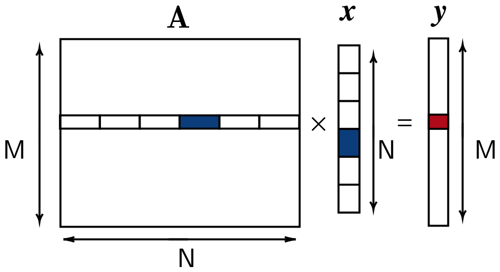
Algorithm Naive Implementation of Matrix Vector Multiplication

for $m = 1$ to M **do**
  $\boldsymbol{y}[i] \leftarrow 0$
  for $j = 1$ to N **do**
    $\boldsymbol{y}$[i] $+=$ $\mathbf{A}$[i][j] * $\boldsymbol{x}$[j]
  end for
end for

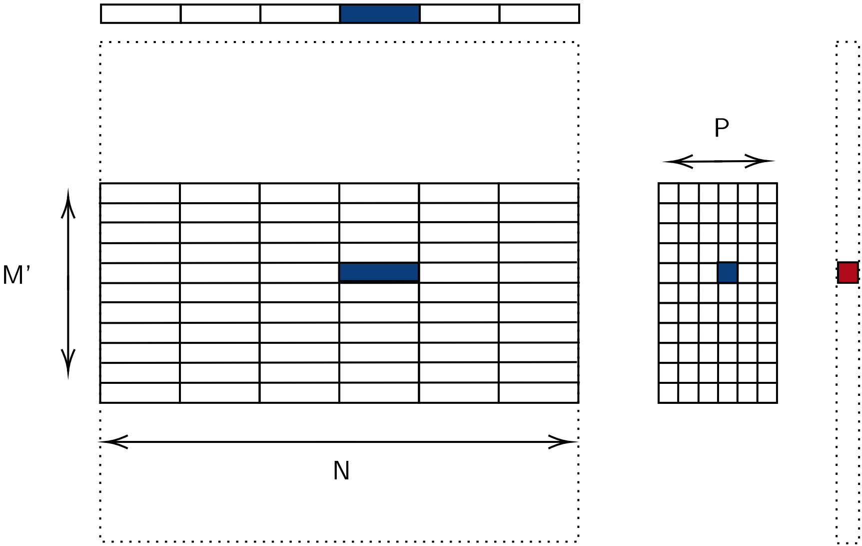
Matrix Vector Multiplication

Algorithm Naive Implementation of Matrix Vector Multiplication

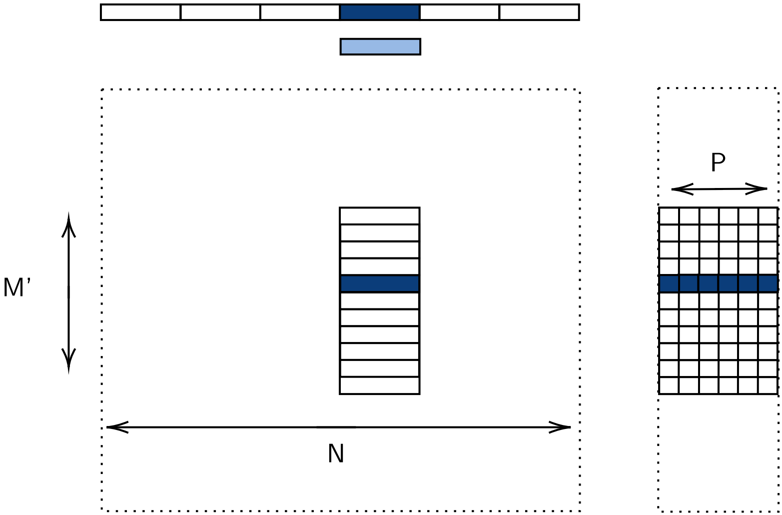
for $m = 1$ to M **do**
    $\boldsymbol{y}[i] \leftarrow 0$
    for $j = 1$ to N **do**
        $\boldsymbol{y}$[i] $+=$ $\mathbf{A}$[i][j] * $\boldsymbol{x}$[j]
    end for
end for



P Threads per Dot Product (Parallelisation)



P Threads per Row with Preloading of x_i



Data Generation

- Sample each explanatory variable $\boldsymbol{x}_j$ with different lengths of $\{10^3, 10^4, 10^5\}$
- $x_{ji} \sim U(0, 1)$

Data Generation

- Sample each explanatory variable $\boldsymbol{x}_j$ with different lengths of $\{10^3, 10^4, 10^5\}$
- $x_{ji} \sim U(0,1)$
- Randomly selected 20 explanatory variables from $\boldsymbol{X}$, $\{\boldsymbol{x}_j : j \in G | G \subset [1..100] \mid \#G = 20\}$.

Data Generation

- Sample each explanatory variable $\boldsymbol{x}_j$ with different lengths of $\{10^3, 10^4, 10^5\}$
- $x_{ji} \sim U(0, 1)$
- Randomly selected 20 explanatory variables from $\boldsymbol{X}$, $\{\boldsymbol{x}_j : j \in G | G \subset [1..100] \mid \#G = 20\}$.
- $\boldsymbol{\mu} = 7 + \sum_{j \in G} f_j(\boldsymbol{x}_j), \quad f_j(\boldsymbol{x}_j) = 10 \sin(2\pi \boldsymbol{x}_j)$
- Generate response variable with random noise with mean, μ_i and variance, $\sigma^2 = 10^{-3}$.

Data Generation

- Sample each explanatory variable $\boldsymbol{x}_j$ with different lengths of $\{10^3, 10^4, 10^5\}$
- $x_{ji} \sim U(0,1)$
- Randomly selected 20 explanatory variables from $\boldsymbol{X}$, $\{\boldsymbol{x}_j : j \in G | G \subset [1..100] \mid \#G = 20\}$.
- $\boldsymbol{\mu} = 7 + \sum_{j \in G} f_j(\boldsymbol{x}_j), \quad f_j(\boldsymbol{x}_j) = 10 \sin(2\pi \boldsymbol{x}_j)$
- Generate response variable with random noise with mean, μ_i and variance, $\sigma^2 = 10^{-3}$.
- Create B-spline basis matrices, $B_j(\boldsymbol{x}_j)$ with $\{16, 32, 64\}$ columns.

Simulation

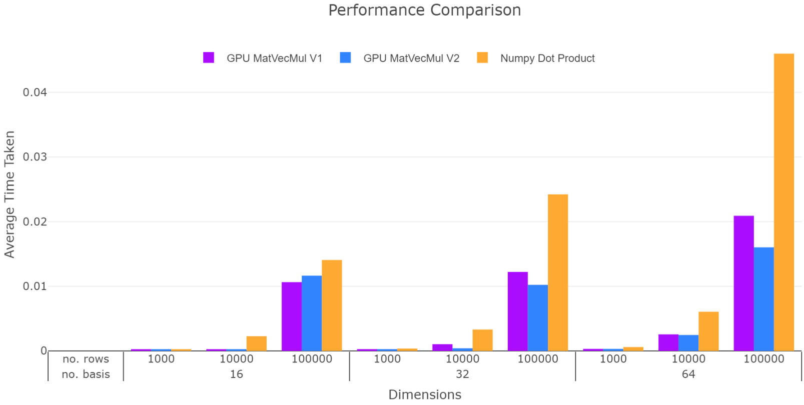


Figure: Comparison of Matrix Vector Multiplication Implementations

Reference

- Eilers, Paul HC, and Brian D. Marx. "Flexible smoothing with B-splines and penalties." Statistical science 11.2 (1996): 89-121.
- Hofner, Benjamin, et al. "Model-based boosting in R: a hands-on tutorial using the R package mboost." Computational statistics 29 (2014): 3-35.
- Tutz, Gerhard, and Harald Binder. "Generalized additive modeling with implicit variable selection by likelihood-based boosting." Biometrics 62.4 (2006): 961-971.
- Bainville, Eric. "GPU matrix-vector product." (2010).
- Sørensen, Hans Henrik Brandenborg. "High-performance matrix-vector multiplication on the GPU." Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29–September 2, 2011, Revised Selected Papers, Part I 17. Springer Berlin Heidelberg, 2012.

Decomposing penalised functions

The basis matrix $\mathbf{B}$ can be reparametrised as a sum of smooth and unpenalised components:

$$\mathbf{B}\boldsymbol{\gamma} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{B}$$
$$\boldsymbol{\gamma}^T\mathbf{S}\boldsymbol{\gamma} = \mathbf{B}^T\mathbf{I}\mathbf{B}$$

which means we can split functions $f_j(\boldsymbol{x}_j)$ into separate base learners for unpenalised $(\mathbf{X}\boldsymbol{\beta})$ and smooth components $(\mathbf{Z}\mathbf{b})$.
This allows $f_j(\boldsymbol{x}_j)$ to take the form of a linear or smooth function independently of the other improving model interpretability and performance.