XGBoost

Knowledge sharing session

Rostislav Stoyanov

May 28, 2025

Recap

- Boosting as an "additive" expansion:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{1}$$

  - $\beta_m; m = 1, 2, \ldots, M$ are called expansion coefficients (model weights).
  - $b(x; \gamma) \in \mathbb{R}$ are called basis functions. These are functions of multivariate argument x, with parameters γ
  - Essentially, a linear combination of simple (function) estimators.

Recap
Example-exponential basis

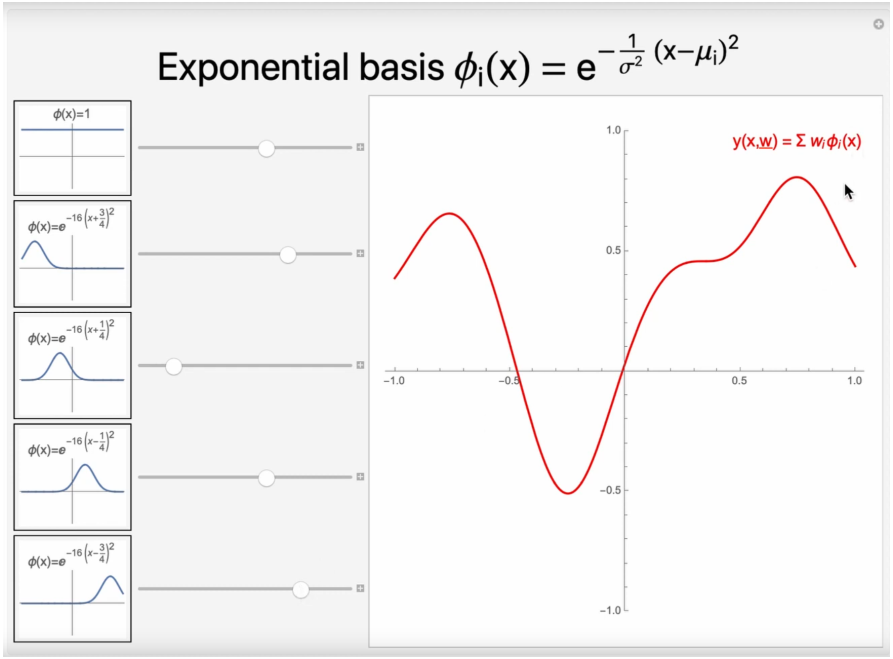
Exponential basis $\phi_i(x) = e^{-\frac{1}{\sigma^2}(x-\mu_i)^2}$
$\phi(x)=1$
$\phi(x)=e^{-16\left(x+\frac{3}{4}\right)^2}$
$\phi(x)=e^{-16\left(x+\frac{1}{4}\right)^2}$
$\phi(x)=e^{-16\left(x-\frac{1}{4}\right)^2}$
$\phi(x)=e^{-16\left(x-\frac{3}{4}\right)^2}$
$y(x,\underline{w}) = \Sigma\, w_i\phi_i(x)$
1.0
0.5
−0.5
−1.0
−1.0
−0.5
0.5
1.0

Figure. Linear combination of exponential basis functions [1]

[1]Source: `https://www.youtube.com/watch?v=sfLm09wqSVk`

Recap

Additive modelling

- Want to learn the basis functions, instead of prespecifying them.
- Optimize jointly (Eq.2), however this is usually not feasible:

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i; \gamma_m\right)\right). \tag{2}$$

  - Might be possible to use SGD, when differentiation w.r.t to γs and βs is possible.
- However, there is an alternative approach if we can solve the problem of fitting a single basis function fast:

$$\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, \beta b\left(x_i; \gamma\right)\right). \tag{3}$$

Recap
Forward Stagewise Additive Modeling

- Greedily fit one function at a time without adjusting previous functions.
- At each step fit the function and coefficient which is most optimal to add to the current expansion. Use a loss function L to decide which function is optimal.

Algorithm Forward Stagewise Additive Modeling [Has+09]

```
1: Initialize $f_0(x) = 0$.
2: for $m = 1, \dots, M$ do
3:     Compute
           $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(x_i\right) + \beta b\left(x_i; \gamma\right)\right).$$
4:     Set $f_m(x) = f_{m-1}(x) + \beta_m b\left(x; \gamma_m\right)$.
5: end for
```

- The hard part is now to find the current basis function (or model in our case) to fit.
- Similarly to gradient descent, attempt to find the locally best step direction.

Recap

Gradient boosting

- Want to minimize with respect to f:

$$J(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

- The issue here is that "f" is arbitrary and we want to take gradient in respect w.r.t to f.
- Notice that $J(f)$ depends on f only at the given training points.
- So we can denote

$$\mathbf{f} = \{f(x_1), f(x_2), \dots, f(x_N)\}^T$$

  and then rewrite $J(f)$:

$$J(f) = \sum_{i=1}^{N} L(y_i, \mathbf{f})$$

Recap
Gradient boosting

- Perform gradient descent on:

$$J(f) = \sum_{i=1}^{N} L(y_i, \mathbf{f})$$

- The gradient direction at $\mathbf{f} = \mathbf{f}_{m-1}$ is $\mathbf{g}_m \in \mathbf{R}^N$ with components:

$$g_{im} = \left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)} \right]_{f(x_i) = f_{m-1}(x_i)}$$

- $-\mathbf{g}_m \in \mathbb{R}^N$ is the direction in which we need to update our training predictions.
- Issue: doesn't generalize, we need predictions on more than $\mathbf{f}$.

RECAP
GRADIENT BOOSTING

- We have calculated the step direction $-\mathbf{g}_m$.
- Find a model (basis function) that approximates the direction (the negative gradient).
  - The components of the gradient are called "pseudo-residuals". These will be the targets for our model.
- Approximate using a least squares regression problem over $\mathcal{B}$, where $\mathcal{B}$ is the space from which we will choose our basis functions(model/hypothesis):

$$f_m = \min_{b \in \mathcal{B}} \sum_{i=1}^{N} (-g_{im} - b(x_i))^2$$

Recap

Step size

- Only thing left is to choose the step size. We have two options.
- Perform a line search:

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(x_i\right) + \beta b\left(x_i; \gamma\right)\right).$$

- Choose a fixed step size, called shrinkage parameter.
  - $\beta = 1$ is considered full step.
  - Choose $\beta \in [0, 1]$ - hyperparameter for our final model.

Recap
Gradient boosting

Algorithm Gradient boosting [Fri01]

```
1: $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, \rho\right)$
2: for $m = 1, \ldots, M$ do
3:     $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, \ldots, N$
4:     $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} \left[\tilde{y}_i - \beta h\left(\mathbf{x}_i; \mathbf{a}\right)\right]^2$
5:     $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{m-1}\left(\mathbf{x}_i\right) + \rho h\left(\mathbf{x}_i; \mathbf{a}_m\right)\right)$
6:     $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h\left(\mathbf{x}; \mathbf{a}_m\right)$
7: end for
```

Recap

GBM example

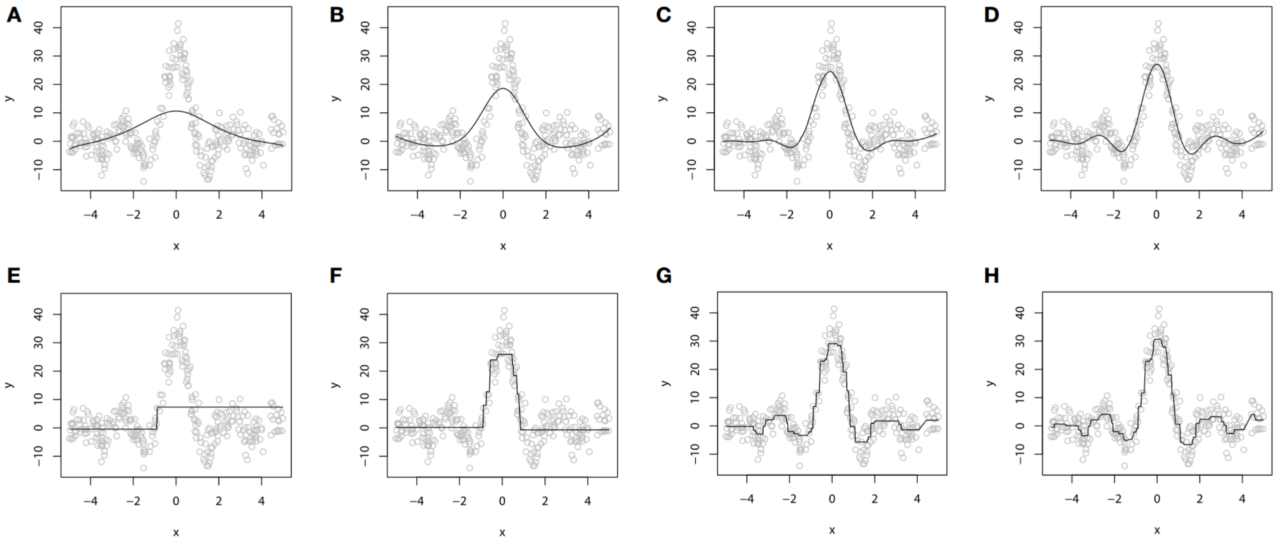

Figure. P-Spline GBM model for different numbers of boosts: (A) M = 1; (B) M = 10; (C) M = 50; (D) M = 100. Decision-tree based GBM model for different numbers of boosts: (E) M = 1; (F) M = 10; (G) M = 50; (H) M = 100. [NK13]

Recap
GBM modifications

- Some variations have been made to gradient boost algorithm to prevent overfitting:
  - Stochastic Gradient Boosting – choosing random subset of data for computing gradient step.
  - Feature sub-sampling – similarly to random forest we can choose a subset of features for each boosting round. Prevents overfitting even more than row sub-sampling.

Recap

Overfitting

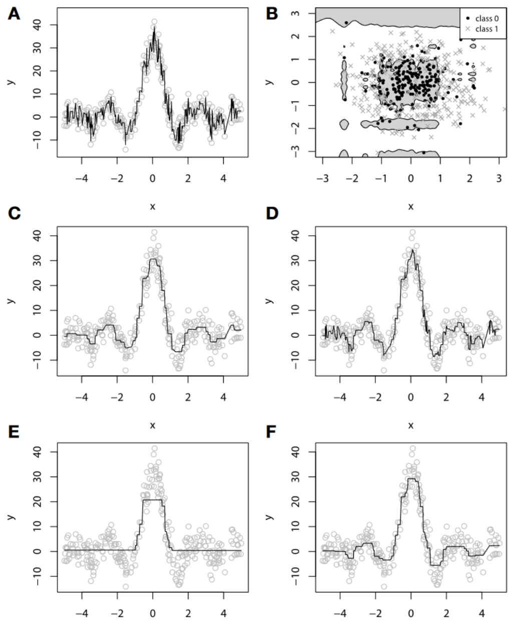

Figure. Examples of overfitting in GBMs on: (A) regression task; (B) classification task. Demonstration of fitting a decision-tree GBM to a noisy sinc(x) data: (C) M = 100, α = 1; (D) M = 1000, α =1; (E) M = 100, α = 0.1; (F) M = 1000, α = 0.1. [NK13]

Recap

Impact of shrinkage

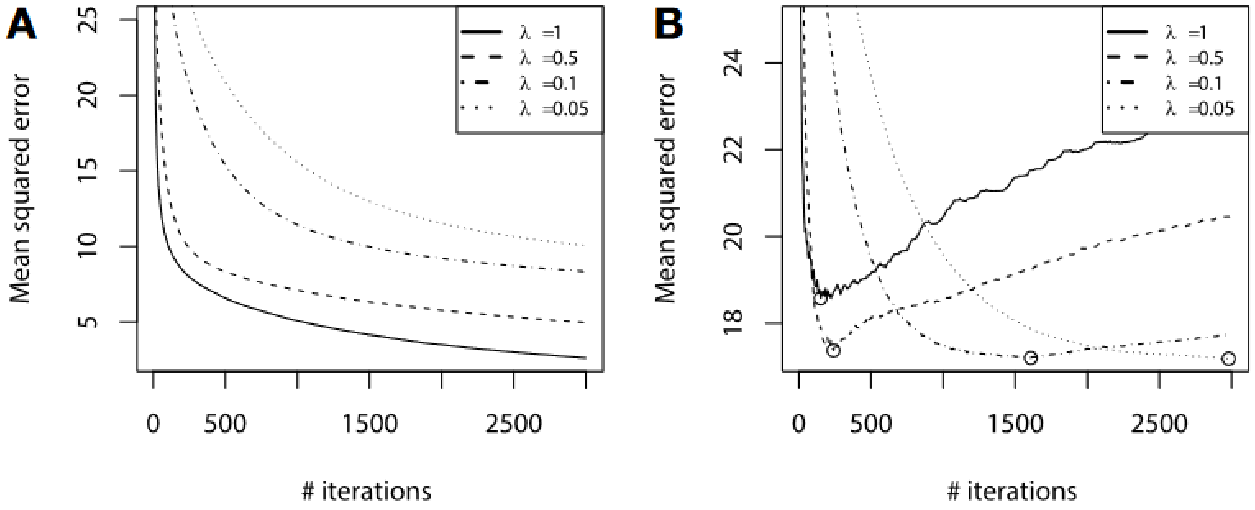

Figure. Error curves for GBM fitting on sinc(x) data: (A) training set error; (B) validation set error. [NK13]

Part I

XGBoost

- XGBoost uses a modified version of the gradient boosting algorithm.
    - Instead of gradient descent, XGBoost uses Newton's method for finding the step direction. This means that at each step m the booster will be trying to minimize:

      $$-\frac{g_m(x)}{h_m(x)},$$

      where g_m is the gradient as previously defined and h_m is the Hessian:

      $$h_{im} = \left[\frac{\partial^2 L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)^2}\right]_{f(x_i)=f_{m-1}(x_i)}$$

    - The loss function is extended with additional regularization penalty.
- Various other optimizations are also made to split-finding, parallelization, etc.

Notation

- $\mathcal{F}$ denotes the space of regression trees, or more formally: $\mathcal{F} = \{f(x) = w_{q(x)}\}$, where $q : \mathbb{R}^m \mapsto T, W \in \mathbb{R}^{\mathbb{T}}$;
- By q, the structure of a tree is denoted – that is the mapping of a sample to a corresponding leaf (more specifically the index of the leaf). T is the number of leaves in the tree;
- f_k denotes a tree structure q and the corresponding leaf weights w, where w_j is the weight of the j-th leaf;
- $\hat{y}_i$ is the ensemble model output for the i-th sample input.

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \;\; f_k \in \mathcal{F} \,. \tag{4}$$

- Training dataset - $\mathcal{D} = \{(x_i, y_i), x_i \in \mathbb{R}^m, y_i \in R\}$

Loss function

- Need to train the K trees (the f_k functions). Do it by minimizing the regularized objective:

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{5}$$

- l is a twice differentiable convex loss function.
- Ω is a regularization term, that is defined by:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||_2^2 + \alpha||w||_1 \tag{6}$$

  - γ punishes tree with higher number of leaves (and consequently more levels).
  - λ is the strength of the L2 regularization term.
  - α, that controls the L1 regularization, which is not included in the original paper [CG16].

Loss function

Taylor approximation

- Uses Taylor series for function approximation. Gives us an approximation of a function around a point.
- For a single-variate function:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \cdots$$

- For a function of two variables:

$$\begin{aligned} f(x,y) \approxeq f(x_0,y_0) &+ (x - x_0)\frac{\partial f}{\partial x}\Big|_{(x_0,y_0)} + (y - y_0)\frac{\partial f}{\partial y}\Big|_{(x_0,y_0)} + \\ &+ \frac{1}{2}\Big((x - x_0)^2\frac{\partial f^2}{\partial x^2}\Big|_{(x_0,y_0)} + 2(x - x_0)(y - y_0)\frac{\partial f^2}{\partial xy}\Big|_{(x_0,y_0)} + (y - y_0)^2\frac{\partial f^2}{\partial y^2}\Big|_{(x_0,y_0)}\Big) + \ldots \end{aligned} \tag{7}$$

Loss function

- Iterative procedure at each timestep t, the tree that most improves the ensemble is taken, which translates to taking the tree, which most minimizes the objective:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_k \Omega(f_k), \tag{8}$$

- Approximate $\mathcal{L}^{(t)}$ around $(y_i, \hat{y}_i^{(t-1)})$ using 7:

$$\mathcal{L}^{(t)} \approxeq \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_m(x_i) f_t(x_i) + \frac{1}{2} h_m(x_i) f_t^2(x_i)] + \sum_k \Omega(f_k) \ , \tag{9}$$

where:

$$g_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f^{(m-1)}(x)}$$

$$h_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=f^{(m-1)}(x)}$$

Loss function

- Simplify by removing constant terms:

$$\mathcal{L}^{(t)} \approxeq \sum_{i=1}^{n} [g_m(x_i) f_t(x_i) + \frac{1}{2} h_m(x_i) f_t^2(x_i)] + \sum_k \Omega(f_k) \tag{10}$$

- We can also rewrite 10:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \frac{1}{2} h_m(x_i)^2 [(-\frac{g_m(x_i)}{h_m(x_i)}) - f_t(x_i)]^2 + \sum_k \Omega(f_t)$$

- At each step we are trying to solve a weighted least-squares regression problem, with our model minimizing the Newton step.

Tree splitting criteria

- XGBoost also uses the loss function to help find optimal tree splits.
- $q(x_i)$ maps x_i to a leaf and w_j is the prediction for leaf j. Rewrite 10:

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n}[g_m(x_i)f_t(x_i) + \frac{1}{2}h_m(x_i)f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_m(x_i))w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \ ,\end{aligned} \tag{11}$$

where $I_j = \{i|q(x_i) = j\}$ is the instance set of a leaf j.

Tree splitting criteria

- For simplicity let:
  - $\sum_{i \in I_j} g_m(x_i) = G_{jm}$
  - $\sum_{i \in I_j} h_m(x_i) = H_{jm}$
- For a fixed tree structure q(x) the optimal value w_j^* of any leaf j is (why?):

$$w_j^* = -\frac{G_{jm}}{H_{jm} + \lambda}$$

- The optimal value for the loss function is achieved by plugging this back:

$$\tilde{\mathcal{L}}^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_{jm}^2}{H_{jm} + \lambda} + \gamma T \tag{12}$$

Tree splitting criteria

- XGBoost uses Eq.12 to measure tree structure quality and define a gain function for a split.
- Suppose we want to split data into left (L) and right (R) node with corresponding instance sets I_L and I_R. The loss after splitting is:

$$-\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}\right]+2\gamma$$

  On the other hand if we don't split:

$$-\frac{1}{2}\left[\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right]+\gamma$$

- The gain for splitting is then defined as:

$$-\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right]-\gamma$$

Split finding algorithm

Exact algorithm

- Considers all possible split points for a feature by evaluating gain on each of them. Need to sort all of the feature values and then perform a scan.

Algorithm Exact Greedy Algorithm [CG16]

```
Input: I, instance set of current node
Input: d, feature dimension
 1: gain ← 0
 2: $G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
 3: for $k = 1, \ldots, m$ do
 4:     $G_L \leftarrow 0, H_L \leftarrow 0$
 5:     for $j = 1$ in sorted (I by $x_{jk}$) do
 6:         $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
 7:         $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
 8:         score ← $\max\left(\text{score}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$
 9:     end for
10: end for
Output: Split with max score
```

Split finding algorithm

Handling missing values

- While learning decision splits XGBoost also finds a default direction in which missing values should go.
- Direction is calculated based on maximum gain.
- Missing values are send in left and right directions and the best option is selected.
- Gradient statistics for the left direction are maintained by the scan and for right side we use the difference between the sum gradient statistics for this node and the scan value.
    - This means the right direction includes the missing values implicitly.
    - To handle for left direction do a second scan in reverse order.

Split finding algorithm

Approximate algorithm

- Goal – reduce the computational complexity for split point enumeration. For the basic algorithm we have $\mathcal{O}(n_{\text{features}} \times n \log(n))$ complexity, where n is the number of samples in the current node.
- Compute a histogram and bin the input features into integer-valued bins. The candidate split points are the edges of the computed bins.
- Two options for histogram computation:
    - Compute histogram globally, at the start of tree building.
    - Local option recalculates split points after each split.
- Complexity drops to $\mathcal{O}(n_{\text{features}} \times n)$ ($\mathcal{O}(n)$ for histogram building) and we have to consider only max_bins points for split points (max_bins $\ll$ n).

Split finding algorithm

XGBoost tree methods

- exact – The greedy exact approach.
- approx – Runs sketching before building each tree using all the rows (local approach). Hessian is used as weights during sketch. This is the approach described in the original paper[[CG16]]
- hist/gpu_hist – Runs sketching before training using only user provided weights instead of hessian. The subsequent per-node histogram is built upon this global sketch. Used in LightGBM, scikit-learn.

Split finding algorithm

Sampling

- Stochastic Gradient Boosting (SGB) – a subsample of the training data is drawn at random without replacement from the full training dataset.
- Gradient-based One-Side Sampling (GOSS) – the top a × 100% of training instances with the largest gradients are selected, then from the rest of the data a random sample of b × 100% instances is drawn. The samples are scaled by $\frac{1-a}{b}$ to make the gradient statistics unbiased.
- Minimal Variance Sampling (MVS) – The whole dataset is sampled with probability proportional to regularized absolute value of gradients:

$$\hat{g}_i = \sqrt{g_i^2 + \lambda h_i^2},$$

where g_i and h_i are the first and second order gradients, λ can be either a hyperparameter, or estimated from the squared mean of the initial leaf value.

Next time

- Training continuation.
- Handling categorical features.
- Subsampling experiments.

Useful resources

- Original GBM paper – [Fri01]
- Stochastic gradient boosting – `https://jerryfriedman.su.domains/ftp/stobst.pdf` [Fri02]
- Gradient boosting video lecture `https://www.youtube.com/watch?v=wPqtzj5VZus`
- Slides from the lecturer of above lecture – `https://davidrosenberg.github.io/ml2018/#home`
- Paper explaining GBM workings – [NK13]
- Gradient boosting explained, series of 3 blog posts (№3 compares grad descent to grad boosting) – `https://explained.ai/gradient-boosting/`.
- Visually explained Newton method – `https://www.youtube.com/watch?v=W7S94pq5Xuo`
- A summary of [Nie16], explaining XGBoost workings – `https://datasciblog.github.io/2020/02/26/tree-boosting-03/`
- XGBoost extrapolation – `https://github.com/dmlc/xgboost/issues/1581`
- XGBoost tree methods – `https://xgboost.readthedocs.io/en/stable/treemethod.html`, `https://github.com/dmlc/xgboost/issues/5822`

Useful resources

- Scikit-learn – inspired by LightGBM
  - `https://scikit-learn.org/stable/modules/ensemble.html#histogram-based-gradient-boosting`
  - `https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/ensemble/_hist_gradient_boosting/histogram.pyx`
- `https://www.cs.princeton.edu/courses/archive/fall18/cos597G/lecnotes/lecture3.pdf`
- [MF17]
- [Ou20]
- [Ke+]

References I

[CG16] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug. 2016). arXiv: 1603.02754, pp. 785–794. DOI: 10.1145/2939672.2939785. URL: http://arxiv.org/abs/1603.02754 (visited on 11/06/2021).

[Fri01] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[Fri02] Jerome H Friedman. “Stochastic gradient boosting”. In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[Has+09] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[Ke+] Guolin Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. en. In: (). URL: https://papers.nips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

References II

[MF17] Rory Mitchell and Eibe Frank. "Accelerating the XGBoost algorithm using GPU computing". en. In: *PeerJ Computer Science* 3 (July 2017). Publisher: PeerJ Inc., e127. ISSN: 2376-5992. DOI: `10.7717/peerj-cs.127`. URL: `https://peerj.com/articles/cs-127`.

[Nie16] Didrik Nielsen. "Tree boosting with xgboost-why does xgboost win" every" machine learning competition?" MA thesis. NTNU, 2016.

[NK13] Alexey Natekin and Alois Knoll. "Gradient boosting machines, a tutorial". In: *Frontiers in Neurorobotics* 7 (2013). ISSN: 1662-5218. DOI: `10.3389/fnbot.2013.00021`. URL: `https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021`.

[Ou20] Rong Ou. *Out-of-Core GPU Gradient Boosting*. arXiv:2005.09148 [cs, stat]. May 2020. DOI: `10.48550/arXiv.2005.09148`. URL: `http://arxiv.org/abs/2005.09148`.