# Multioutput gradient boosting

## Knowledge sharing session

**Rostislav Stoyanov**

May 28, 2025

## MULTIOUTPUT GRADIENT BOOSTING

- Gradient boosting algorithms has found huge success in various domains.
- Some problems require more than one output:
  - Multiclass classification.
  - Multioutput regression - finance, forecasting.
- Classical boosting algorithms only have a single output.
  - How do we approach this problem?

## Multioutput gradient boosting
### Stategies

- One-versus-all strategy, which relies on building a single output tree for each of the multiple outputs.
- **Single-tree strategy**, in which a single tree is build. The tree leaves, however, are vectors instead of scalars.

## Main presentation topics

- Two main topics
  - Multioutput boosting and XGBoost.
  - Optimizations for single-tree multioutput boosting (PyBoost).

Part I

## Multioutput regression and XGBoost

## Part I: Multioutput regression and XGBoost

**1 Boosting preliminaries . . . 6**

1.1 Model training . . . 7

1.2 Leaf values . . . 10

1.3 Tree structure . . . 11

1.4 Complexity . . . 12

**2 One-versus-all strategy . . . 13**

2.1 Strategy overview . . . 13

2.2 Pros and cons . . . 14

**3 Single-tree strategy . . . 15**

3.1 Strategy overview . . . 15

3.2 Complexity analysis . . . 18

3.3 Pros and cons . . . 19

## Boosting preliminaries

- Let's explore the classical regression task:
    - We are given a training dataset with $n$ examples and $m$ features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ $(|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R})$.
    - Our goal is to create a model, which provides the best predictions on unseen data according to a set of metrics.
- Gradient boosting creates an ensemble $F_T$ of T base learners $f \in \mathcal{F}$, $f : \mathbb{R}^m \rightarrow \mathbb{R}$.
    - Predictions are given by $\hat{y}_i = \sum_{t=1}^{T} f_t\left(\mathbf{x}_i\right),\ f_t \in \mathcal{F}$.
    - In our case the base learners are regression trees.
- The model is trained in a additive and greedy manner using gradient information.

## Boosting preliminaries

### Model training

- Gradient boosting is an iterative process, with each iteration adding new base learner to the ensemble.
- First we need to specify:
  - Convex loss function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$,
  - Regularization term $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w^2$.
- This allows for the creation of a regularized objective that our model should minimize:

$$\mathcal{L}(F_T) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{j=1}^{J} \Omega(f_j).$$

- At each iteration step $t$ we are searching for the function that minimizes this objective:

$$\mathcal{L}_t(f) = \sum_{i=1}^{n} l\left(y_i, F_{t-1}\left(x_i\right) + f\left(x_i\right)\right) + \Omega(f_t).$$

## Boosting preliminaries

### Model training

- This optimization problem is solved by taking a 2nd order Taylor approximation of the objective and the function we are search for at each step is [CG16]:

$$f_t^* \in \underset{f \in \mathcal{F}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left( g_m(x_i f(x_i) + \frac{1}{2} h_m(x_i) f^2(x_i) \right) + \Omega(f) \right\}, \tag{1}$$

where:

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

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

## BOOSTING PRELIMINARIES
### MODEL TRAINING

- Previously written objective(Eq. 1) is valid for any base learner, however, we are interested in decision trees.
- A decision tree can be expressed as

$$f(x) = \sum_{j=1}^{J} w_j \left[x \in I_j\right],$$

where $I_j = \{i|q(x_i) = j\}$ is the instance set of a leaf j and $w_j$ is the prediction for leaf j.
- The problem of learning a tree $f_t$ can be split into two sub-problems:
  - Finding the best tree structure
  - Computing the optimal leaf values for a fixed structure.

It turns out (Eq. 1) can also used for both solving both issues.

## Boosting preliminaries

### Leaf values

- Because of the decision tree structure, we can rewrite (1) as follows:

$$f_t^* \in \operatorname*{argmin}_{f \in \mathcal{F}} \left\{ \sum_{j=1}^{T} [(\sum_{i \in I_j} g_m(x_i))w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)w_j^2] + \gamma T \right\}.$$

- If we have a fixed tree structure, then we have optimal value for each leaf j:

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

where:

- $\sum_{i \in I_j} g_m(x_i) = G_{jm}$
- $\sum_{i \in I_j} h_m(x_i) = H_{jm}$

## BOOSTING PRELIMINARIES

### TREE STRUCTURE

- We can substitute the leaf values back into the objective function:

$$Loss(f_t) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_{jm}^2}{H_{jm}+\lambda} + \gamma T \tag{2}$$

- Tree building is done greedily by iteratively adding branches to the tree, with each step splitting one of the existing leafs.
- Decision for splitting is taken using loss before/after split. The split gain is defined as:

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

where subscript L/R indicates gradient statistics for values in left/right leaf.

## Boosting preliminaries

### Complexity

- Instead of finding the best split point based on all of the possible feature values, most frameworks use histogram-based algorithms[CG16; Ke+; Pro+19].
- These algorithms bucked feature values into $h$ ($h \ll n$) discrete bins and splits finding decisions are made based on these bins.
- The time complexities of the two main previously mentioned operations are:
  - For histogram building for a leaf with $n_j$ samples – $\mathcal{O}(mn_j)$, where $m$ is the number of input features.
  - For split finding the complexity is $\mathcal{O}(hm)$.
- When building a tree using depth-first-search algoirthm the complexity for a complete tree building is $\mathcal{O}(Dnm + 2^D hm)$ [IV22].

## One-versus-all strategy

### Strategy overview

- Build a single model for each of the output targets. For example, if forecasting for $h$ days ahead, will have $h$ models, one for the each day we are forecasting.
- Scikit-learn provides meta-estimators (e.g. MultiOutputRegressor), which can be used to provide fit a single model for each target.
- XGBoost also has experimental support for multi-output regression and multi-label classification with Python package since version 1.6.
  - Internally, XGBoost builds one model for each target similar to sklearn meta estimators, with the added benefit of reusing data and other integrated features like SHAP.

## One-versus-all strategy

### Pros and cons

- Positive sides of one-versus-all strategy:
  - Ease of implementation, steps on already optimized algorithms for scalar prediction.
- Negative sides of one-versus-all strategy:
  - Fits one regressor per target – cannot take advantage of correlations between targets.

## Single-tree strategy

### Strategy overview

- A single multivariate decision tree is built. All output are being handled together.
- Currently, this approach is being partly supported in:
  - CatBoost – limited support for target functions, CPU only training.
  - XGBoost – under development, poor optimization, CPU only training.
  - PyBoost.
- Multioutput regression leads to changes in derivative calculation.

## Single-tree strategy

### Strategy overview

- Recall that at each step $t$ we are searching for the tree $f_t$ that best fits the objective:

$$f_t^* \in \underset{f \in \mathcal{F}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left( g_m(x_i f(x_i) + \frac{1}{2} h_m(x_i) f^2(x_i) \right) + \Omega(f) \right\},$$

where:

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

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

- In the single output case, $y_i, f(x_i)$ and $L(y_i, f(x_i))$ were scalars, and consequently the first and second-order derivatives were scalars.

## Single-tree strategy

### Strategy overview

- In the multioutput case, $y_i, f(x_i)$ are vectors of length $d$ and $L(y_i, f(x_i)) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a scalar. $g_m(x_i)$ and $h_m(x_i)$ now look differently:
  - $g_m(x_i)$ is now a matrix $J \in \mathbb{R}^{d \times d}$ (Jacobian), where each element is described as:

$$J_{j,k} = \frac{\partial (L(y_i, f(x_i))_j}{\partial f(x_i)_k}$$

  - $h_m(x_i)$ is now a tensor (or a vector of matrices) of partial derivatives.

## Single-tree strategy

### Complexity analysis

- Compared to the single output scenario, the time complexity increases by d times:
  - The complexity for histogram building of leaf with $n_j$ samples is $\mathcal{O}(mn_jd)$.
  - The complexity for split finding is $\mathcal{O}(hmd)$.
- Overall complexity is $\mathcal{O}(mnd + 2^D hmd)$.

## Single-tree strategy

### Pros and cons

- Pros:
  - Can model dependencies between targets.
- Cons:
  - Extremely slow and has memory issues on both XGBoost and Catboost.
  - Loss of flexibility.

## Part II

## Accelerating multioutput gradient boosting

## Part II: Accelerating multioutput gradient boosting

**1 Optimization Techniques** . . . . . . . . . . **22**

**2 Sketched split scoring** . . . . . . . . . . **23**

2.1 Top outputs . . . . . . . . . . 25

2.2 Random sampling . . . . . . . . . . 26

2.3 Random projections . . . . . . . . . . 27

**3 Conclusion** . . . . . . . . . . **29**

## Optimization Techniques

- Some of the optimization techniques that are most commonly used, rely on reducing the calculations for second-order derivatives:
  - Second order derivatives are ignored during split search.
  - Only the main diagonal of the second-order derivatives is used when calculating the leaf values.
- There are transformation methods, which rely on reducing the number of target before training a model using compression and decompression techniques.
  - Information loss due to compression, which impacts performance.
  - Not part of this presentation.
- [IV22] proposes three different methods for gradient boosting speed up for multioutput tasks. These methods rely on approximating the scoring function for finding the best tree structure. Focus for rest of this part.

## Sketched split scoring

- Key idea is to reduce the number of gradient values needed for split searching, in such a way that the scoring function result doesn't change much.
- We can rewrite the scoring function from earlier (2) to:

$$S_G(R) = \frac{\left\|G^\top v_R\right\|^2}{|R| + \lambda}, \quad \text{where } G = \begin{pmatrix} g_1^1 & g_1^2 & \cdots & g_1^d \\ \vdots & \vdots & \ddots & \vdots \\ g_n^1 & g_n^2 & \cdots & g_n^d \end{pmatrix} \quad \text{and } v_R = \begin{pmatrix} [x_1 \in R] \\ \vdots \\ [x_n \in R] \end{pmatrix},$$

where:

- $G \in \mathbb{R}^{n \times d}$ is the gradient matrix.
- $S_G(R)$ is the scoring funciton for leaf R, with subscript to $S$ indicating its dependence on the gradient matrix $G$.
- $v_R$ is the indicator vector of the leaf $R$ (its $i$-th coordinate is equal to 1 if $x_i \in R$ and 0 otherwise).
- $g_i^j$ is the j-th column of the Jacobian for the i-th input data point.
- $|R|$ is the number of points in the leaf R.

## Sketched split scoring

- To reduce the complexity of computing $S_G$ in $d$, it is approximated with $S_{G_k}$ for some other matrix $G_k \in \mathbb{R}^{n \times k}$ with $k \ll d$.
- $G_k$ is called the sketch matrix and to $k$ as the reduced dimension or sketching dimension.
- Several methods for sketch matrix calculation.
- Looking for approximation that is universal and uniformly accurate for all splits we will possibly iterate over. The approximation error is given by:

$$\text{Error}\left(S_G, S_{G_k}\right) = \sup_R \left|S_G(R) - S_{G_k}(R)\right|$$

- Instance of Integer Programming and is NP-complete so the upper bounds are not optimal.

## Sketched split scoring

### Top outputs

- This approach chooses the $k$ columns of G with the largest Euclidian norm.
- Denote the columns of $G$ by $g_1, \ldots, g_d$. Let also $i_1, \ldots, i_d$ be the indexes which sort the columns of $G$ in descending order by their norm, that is, $\|g_{i_1}\| \geq \|g_{i_2}\| \geq \ldots \geq \|g_{i_d}\|$. The gradient matrix and the sketch can be written as:

$$G = \begin{pmatrix} | & | & & | \\ g_1 & g_2 & \ldots & g_d \\ | & | & & | \end{pmatrix} \quad \text{and} \quad G_k = \begin{pmatrix} | & | & & | \\ g_{i_1} & g_{i_2} & \ldots & g_{i_k} \\ | & | & & | \end{pmatrix}.$$

- A drawback is that this method chooses top k output dimensions which may not vary much from step to step. For instance, if several columns have large norms and others have medium norms, Top Outputs may completely ignore the latter columns during the split search.

## Sketched split scoring

### Random sampling

- The key idea of Random Sampling is to randomly sample the columns of $G$ with probabilities proportional to their norms. The sampling probabilities for each column are defined by:

$$p_i = \|g_i\|^2 / \sum_{j=1}^{d} \|g_j\|^2, \quad i = 1, \dots, d.$$

- Let $i_1, \dots, i_k$ be independent and identically distributed random variables taking values $j$ with probabilities $p_j$, $j = 1, \dots, k$. These random variables represent indexes of the chosen columns of $G$. The sketch is the given by:

$$G_k = \begin{pmatrix} | & | & & | \\ \bar{g}_{i_1} & \bar{g}_{i_2} & \dots & \bar{g}_{i_k} \\ | & | & & | \end{pmatrix}, \quad \text{where} \quad \bar{g}_i = \frac{1}{\sqrt{kp_i}} g_i.$$

  - The additional column normalization by $1/\sqrt{kp_i}$ is needed for unbiasedness of the resulting estimate.

## Sketched split scoring

### Random projections

- An alternative view for the previous approaches is that the sketch $G_k$ was constructed by sampling columns from $G$ according to some probability distribution. This process can be viewed as multiplication of $G$ by a matrix $\Pi$,:

$$G_k = G\Pi,$$

where $\Pi \in \mathbb{R}^{d\times k}$ has independent columns, and each column is all zero except for a 1 in a random location.

## Sketched split scoring

### Random projections

- In Random Projections, matrices $\Pi$ are sampled, every entry of which is an independently sampled random variable.
- More specifically, the following sketch:

$$G_k = G\Pi$$

  is used, where $\Pi \in \mathbb{R}^{d \times k}$ is a random matrix filled with independently sampled $\mathcal{N}\left(0, k^{-1}\right)$ entries.
- Random Projections has the same merits as Random Sampling since it is also a random approach. Besides that, the sketch matrix $G_k$ here uses gradient information from all outputs since each column of $G_k$ is a linear combination of columns of $G$.

## CONCLUSION

- One-vs-all strategy gives good results, is worth exploring more.
- Single tree strategy is very computationally expensive. Unusable without approximation techniques.
  - Currently results are rather poor, possibly due to hyperparamter tuning.
  - Run a sweep using PyBoost implementation.
  - Keep track of progress and revisit at a later stage.

## Useful resources

- Gradient boosting – `https://www.youtube.com/watch?v=wPqtzj5VZus`
- XGBoost in detail – `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2433761`
- Presentation for SketchBoost – `https://cs.hse.ru/mirror/pubs/share/690635329`.
- SketchBoost talk (in Russian have fun) – `https://www.youtube.com/watch?v=Y9VmmSI8HN8`
- XGBoost multioutput issues – `https://github.com/dmlc/xgboost/issues/2087`, `https://github.com/dmlc/xgboost/issues/9043`

## 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).

[IV22] Leonid Iosipoi and Anton Vakhrushev. *SketchBoost: Fast Gradient Boosted Decision Tree for Multioutput Problems*. arXiv:2211.12858 [cs, stat]. Nov. 2022. URL: http://arxiv.org/abs/2211.12858 (visited on 01/05/2023).

[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.

[Pro+19] Liudmila Prokhorenkova et al. *CatBoost: unbiased boosting with categorical features*. arXiv:1706.09516 [cs]. Jan. 2019. DOI: 10.48550/arXiv.1706.09516. URL: http://arxiv.org/abs/1706.09516 (visited on 03/05/2023).