Digital/McKinsey

Can one do better than XGBoost? Presenting 2 new gradient boosting libraries – LightGBM and Catboost

MATEUSZ SUSIK

Pydata Warsaw Talk | 19.10.2017

Introduction

About me

- Currently a data scientist at McKinsey & Company
- Graduated from MIMUW (University of Warsaw, Computer Science)
- In the past I worked in i.a. CERN and Max Planck Institute
- Used to be a software developer. I love statistical/machine/deep learning

Today I will cover

- Extreme gradient boosting theory
- Two new libraries: LightGBM and Catboost
- Benchmarks comparing LightGBM, Catboost and XGBoost

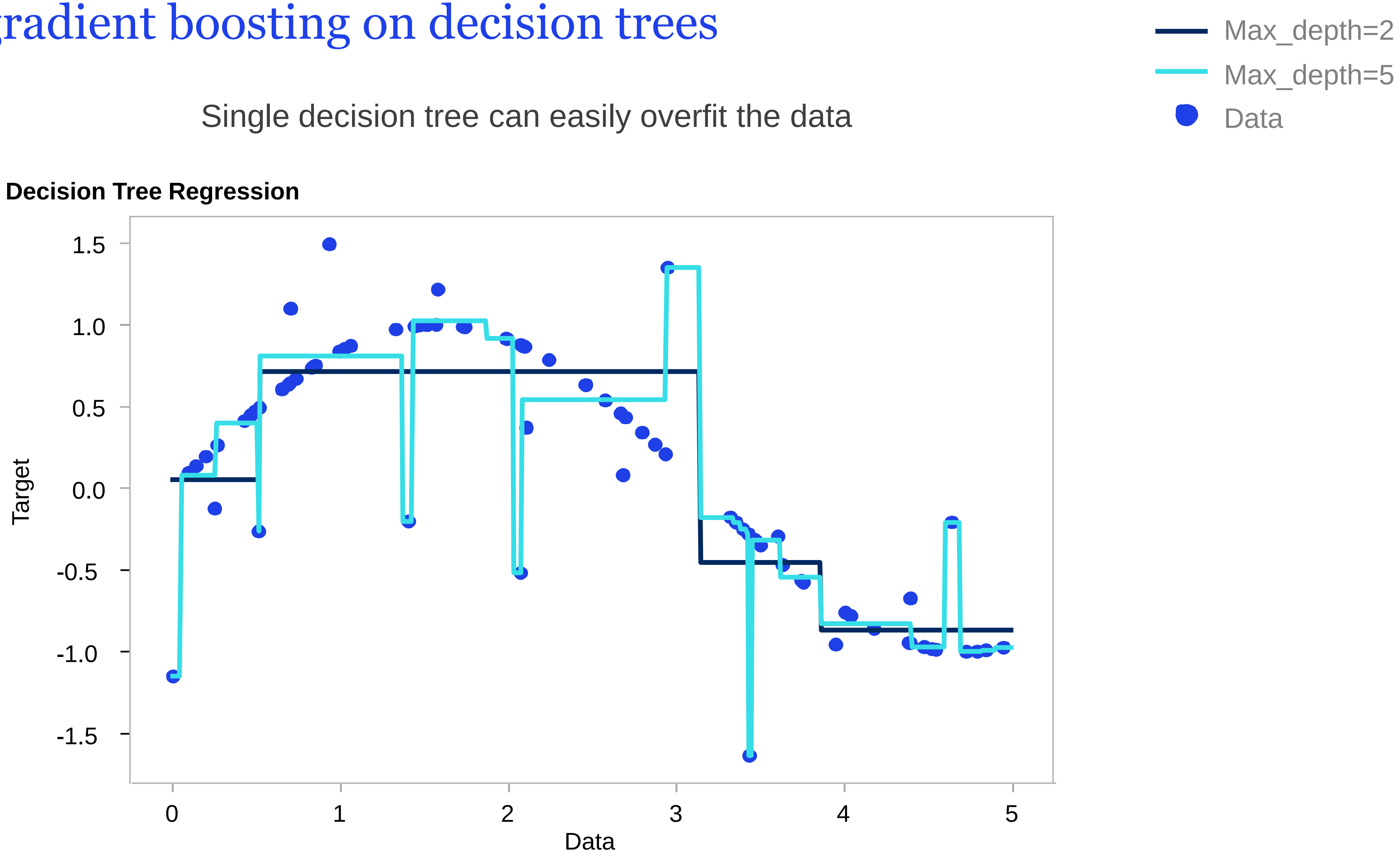
Why I want to cover the new libraries?

- They are successful in Kaggle contest despite being new – solution based on LightGBM won *Sberbank Russian Housing Market,* Catboost was used by 4th and 6th solution in *Instacart Market Basket Analysis*
- I am not an author of any of them – I want to compare them fairly
- I use them in my work

Contents

Motivation for gradient boosting on decision trees

Single decision tree can easily overfit the data



It's better to build ensemble models

Naive gradient boosting

Fit a weak learner

$$f_1(x) = y$$

$$h_1(x) = y - f_1(x)$$

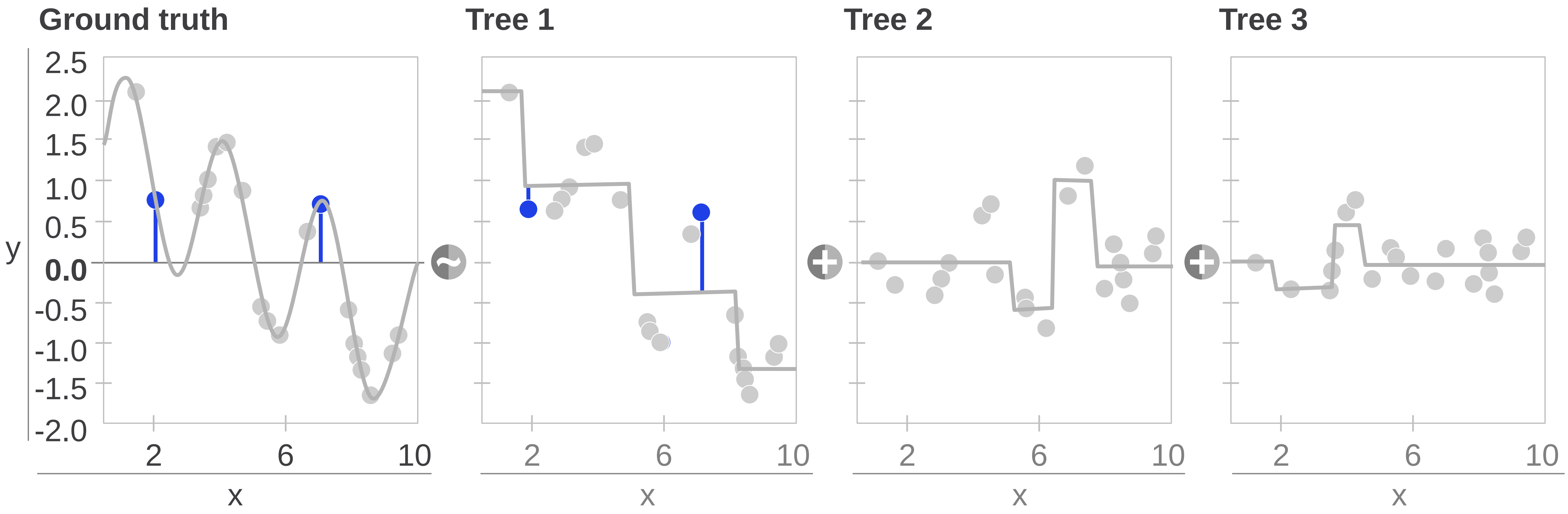
Fit another weak learner to the residuals

$$f_2(x) = f_1(x) + h_1(x)$$

Repeat many times

$$h_m(x) = y - f_m(x)$$

$$f_{m+1}(x) = f_m(x) + h_m(x)$$



SOURCE: “Scikit-Learn Gradient Boosted Regression Trees” – Prettenhofer P., Louppe G.

Gradient boosting on decision trees – correct explanation

I will follow "*Introduction to Boosted Trees*" by Tianqi Chen. Let's define our objective function

$$Obj(\Theta) = \underbrace{L(\Theta)}_{\text{Training loss}} + \underbrace{\Omega(\Theta)}_{\text{Regularization}}$$

In order to simplify, we'll use square loss

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

We will create K regression trees: $\{f_1, \dots, f_k\}$

We can write our objective function as

$$Obj(\Theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \sum_{k=1}^{K} \Omega(f_k)$$

Gradient boosting on decision trees – correct explanation

Let's focus on a single tree. How do we learn it?

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\dots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Let's define

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right) = (MSE)\ 2\left(\hat{y}_i^{(t-1)} - y_i\right)$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right) = (MSE) 2$$

Each time we add a new function (tree) to the constant prediction

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i - \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k)$$
$$= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t)$$
$$= \sum_{i=1}^{n} [2\left(\hat{y}_i^{(t-1)} - y_i\right) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + const$$

After we remove constants

$$\sum_{i=1}^{n} [2\left(\hat{y}_i^{(t-1)} - y_i\right) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t)$$
$$= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{h_i}{2} f_t(x_i)^2] + \Omega(f_t)$$

Gradient boosting on decision trees – regularization

So, what is $\Omega(f_t)$?

$$f_t(x) = w_{q(x)}, \qquad w \in \mathbb{R}^T, q: \mathbb{R}^d \to \{1,2,\dots,Q\}$$

The leaf weight of the tree

The structure of the tree

$$\Omega(f_t) = \frac{1}{2}\lambda\left(\sum_{j=1}^{Q} w_j^2\right) + \alpha\left(\sum_{i=1}^{Q} |w_i|\right) + \gamma Q$$

Xgboost's lambda

L2 penalty

Xgboost's alpha

L1 penalty

Xgboost's gamma

Number of leaves

Gradient boosting on decision trees – correct explanation – cont.

Let's see the whole formula

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{h_i}{2} \ w_{q(x_i)}^2] + \frac{1}{2}\lambda\left(\sum_{j=1}^{Q} w_j^2\right) + +\gamma Q$$

$$= \sum_{j=1}^{Q} [(\sum_{i \in I_j} g_i) \, w_{q(x_i)} + \frac{((\sum_{i \in I_j} h_i) + \lambda)}{2} \ w_{q(x_i)}^2] + \gamma Q$$

Where $I_j = \{i | q(x_i) = k\}$ – set of indices of points assigned to j-th leaf.

In order to simplify let us define

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

Then, from the characteristics of quadratic function, the optimal weight for a fixed tree is

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

Gradient boosting on decision trees – correct explanation – cont.

The algorithm for building a single tree

Grow the tree greedily by computing the gain

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

Left child score

Right child score

Score if we do not split

The complexity cost if we add a new split

In order to find the best split point we simply iterate over sorted attributes and compute the gain

Some of the simple tricks in xgboost

- The tree is grown in breadth first fashion (as opposed to depth first like in the original C4.5 implementation). This provides a possibility of sorting and traversing data only once on each level

- Furthermore, the sorted features can be cached – no need to sort that many times
- Additional regularization parameter called shrinkage. Idea can be represented as

$$y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$$

Where $0 < \epsilon < 1$

What I did not cover

XGBoost offer a GPU implementation – it is often faster for large datasets with large number of trees. LightGBM also has a GPU implementation

There are a lot of optimizations done focused on better parallelization and distribution of computations. They are not covered by this presentation

Contents

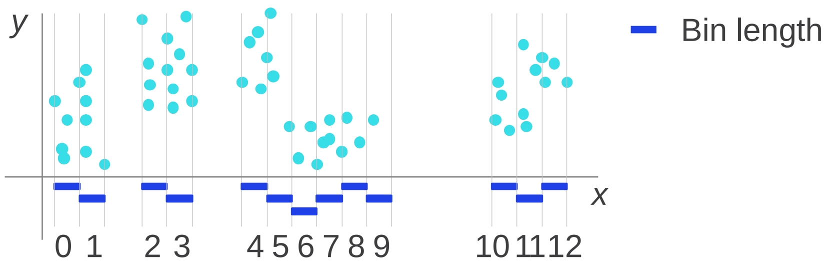
LightGBM

New library, developed by Microsoft, part of Distributed Machine Learning Toolkit.
Main idea: make the training faster

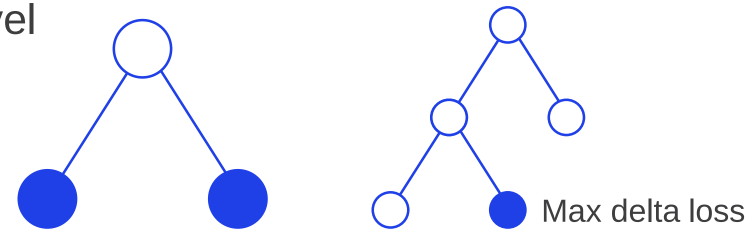
First release: April, 24th 2017

Main improvements

- Histogram based algorithm – each continuous feature is bucketed into discrete bins. Now, in order to compute the best split, we need to iterate over $\#number\ of\ bins$ records instead of $\#number\ of\ points$



- The histogram implementation can be easily optimized for sparse data and most of the datasets we deal with are sparse
- The trees are grown depth first – keeping the presorted state. LightGBM chooses the leaf with maximum delta loss to grow and does not have to grow the whole level



The histogram method has been also implemented in xgboost: *tree_method = 'hist'*

Catboost

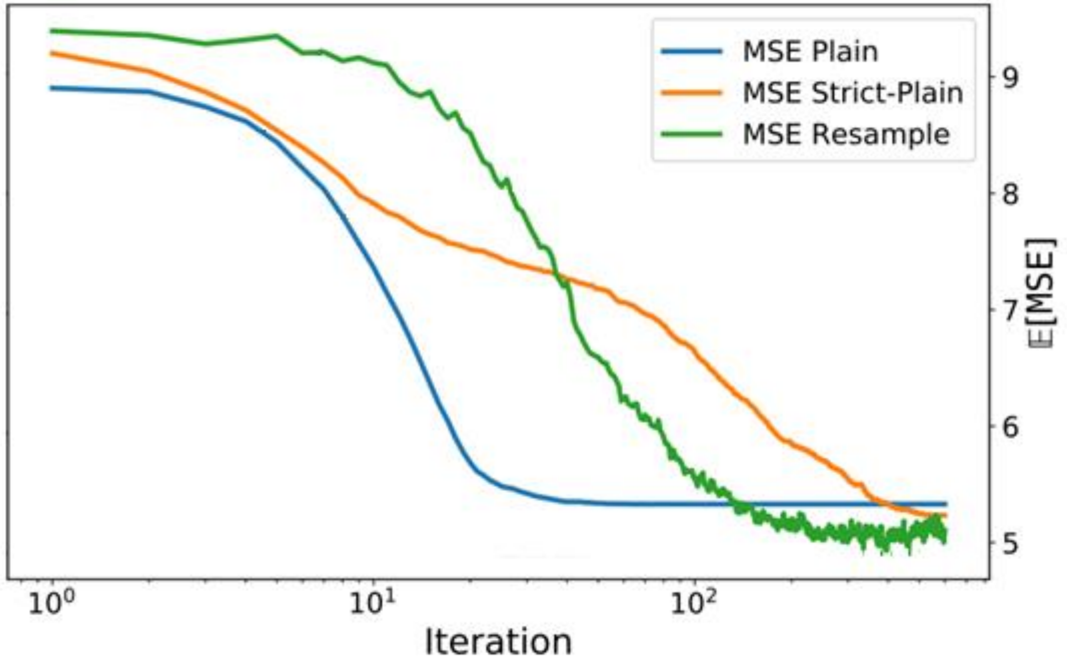
$\hat{B}_t$ Gradient estimation bias
Infinite dataset
XGBoost-like algorithm
Catboost-like algorithm

New library, developed by Yandex
Main idea: prevent overfitting and provide good default parameters

First release: September 14th, 2017

Main improvements

- More sophisticated handling of categorical variables and more choices for bucketing
- Fights the "gradient bias"



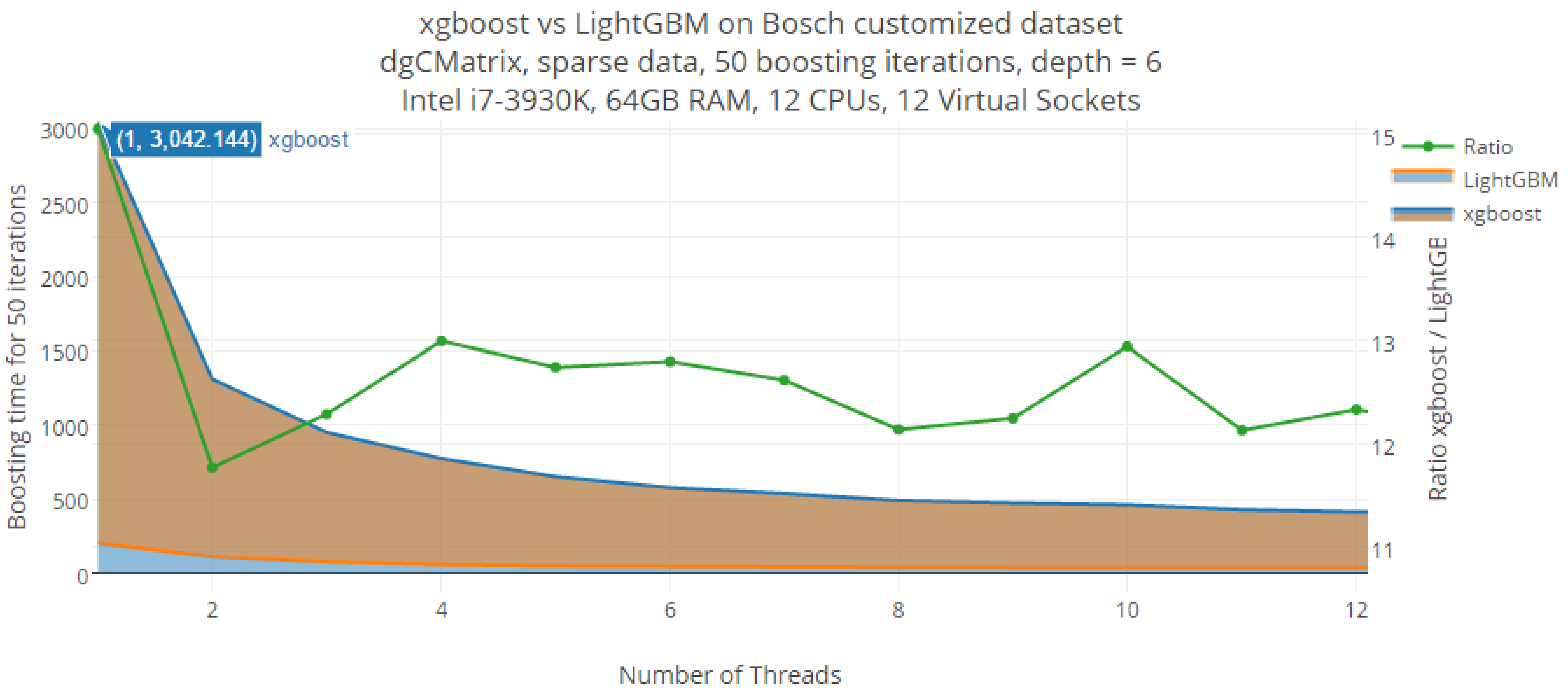
Gradient boosting overfits because of the correlation between the noise in the data and the outputs of the approximations.

Main ideas:
- Use oblivious trees
- For a single tree, use a random order of all the observations. Then, while computing the gradient for an observation, use only the preceding observations. Don't use the current and the following ones.

SOURCE: "Fighting biases with dynamic boosting" (2017) – Dorogush, A.V., Gulin A., Gusev G., Kazeev N., Prokhorenkova L., Vorobev A.

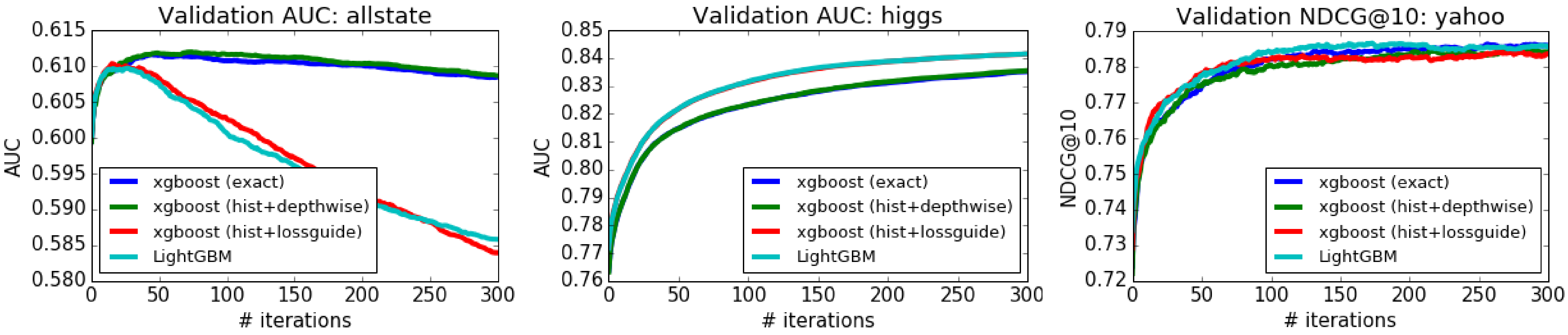
Contents

Benchmarks – Time vs CPU threads



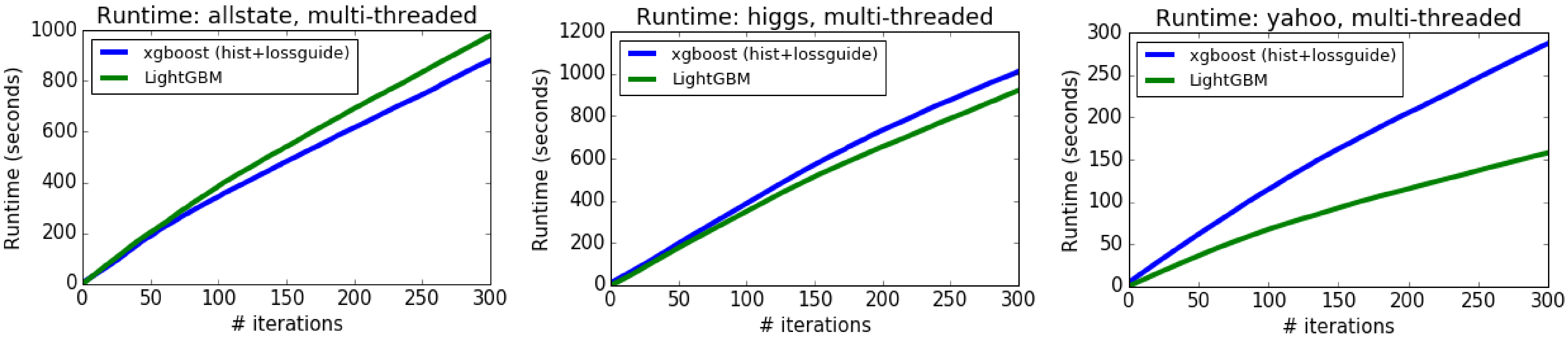
SOURCE: https://medium.com/implodinggradients/benchmarking-lightgbm-how-fast-is-lightgbm-vs-xgboost-15d224568031 - Laurae

Benchmarks – authors’ perspective (xgboost)



SOURCE: https://github.com/dmlc/xgboost/issues/1950

Benchmarks – authors' perspective (xgboost)



SOURCE: https://github.com/dmlc/xgboost/issues/1950

Benchmarks – authors' perspective (Catboost)

	Default CatBoost	Tuned CatBoost	Default LightGBM	Tuned LightGBM	Default XGBoost	Tuned XGBoost	Default H2O	Tuned H2O
Adult	0.272978 (±0.0004) (+1.20%)	**0.269741** (±0.0001)	0.287165 (±0.0000) (+6.46%)	0.276018 (±0.0003) (+2.33%)	0.280087 (±0.0000) (+3.84%)	0.275423 (±0.0002) (+2.11%)	0.276066 (±0.0000) (+2.35%)	0.275104 (±0.0003) (+1.99%)
Amazon	0.138114 (±0.0004) (+0.29%)	**0.137720** (±0.0005)	0.167159 (±0.0000) (+21.38%)	0.163600 (±0.0002) (+18.79%)	0.165365 (±0.0000) (+20.07%)	0.163271 (±0.0001) (+18.55%)	0.169497 (±0.0000) (+23.07%)	0.162641 (±0.0001) (+18.09%)
Appet	**0.071382** (±0.0002) (-0.18%)	0.071511 (±0.0001)	0.074823 (±0.0000) (+4.63%)	0.071795 (±0.0001) (+0.40%)	0.074659 (±0.0000) (+4.40%)	0.071760 (±0.0000) (+0.35%)	0.073554 (±0.0000) (+2.86%)	0.072457 (±0.0002) (+1.32%)
Click	0.391116 (±0.0001) (+0.05%)	**0.390902** (±0.0001)	0.397491 (±0.0000) (+1.69%)	0.396328 (±0.0001) (+1.39%)	0.397638 (±0.0000) (+1.72%)	0.396242 (±0.0000) (+1.37%)	0.397853 (±0.0000) (+1.78%)	0.397595 (±0.0001) (+1.71%)
Internet	0.220206 (±0.0005) (+5.49%)	**0.208748** (±0.0011)	0.236269 (±0.0000) (+13.18%)	0.223154 (±0.0005) (+6.90%)	0.234678 (±0.0000) (+12.42%)	0.225323 (±0.0002) (+7.94%)	0.240228 (±0.0000) (+15.08%)	0.222091 (±0.0005) (+6.39%)
Kdd98	0.194794 (±0.0001) (+0.06%)	**0.194668** (±0.0001)	0.198369 (±0.0000) (+1.90%)	0.195759 (±0.0001) (+0.56%)	0.197949 (±0.0000) (+1.69%)	0.195677 (±0.0000) (+0.52%)	0.196075 (±0.0000) (+0.72%)	0.195395 (±0.0000) (+0.37%)
Kddchurn	0.231935 (±0.0004) (+0.28%)	**0.231289** (±0.0002)	0.235649 (±0.0000) (+1.88%)	0.232049 (±0.0001) (+0.33%)	0.233693 (±0.0000) (+1.04%)	0.233123 (±0.0001) (+0.79%)	0.232874 (±0.0000) (+0.68%)	0.232752 (±0.0000) (+0.63%)
Kick	0.284912 (±0.0003) (+0.04%)	**0.284793** (±0.0002)	0.298774 (±0.0000) (+4.91%)	0.295660 (±0.0000) (+3.82%)	0.298161 (±0.0000) (+4.69%)	0.294647 (±0.0000) (+3.46%)	0.296355 (±0.0000) (+4.06%)	0.294814 (±0.0003) (+3.52%)
Upsel	0.166742 (±0.0002) (+0.37%)	**0.166128** (±0.0002)	0.171071 (±0.0000) (+2.98%)	0.166818 (±0.0000) (+0.42%)	0.168732 (±0.0000) (+1.57%)	0.166322 (±0.0001) (+0.12%)	0.169807 (±0.0000) (+2.21%)	0.168241 (±0.0001) (+1.27%)

Metric: Logloss (lower is better). In the first brackets – std, in the second – the percentage difference from the tuned CatBoost

SOURCE: https://github.com/catboost/benchmarks

My benchmarks

I chose a similar parameter setting to the Catboost experiment. Bayesian optimization performed with scikit-optimize. I used sklearn wrappers for all three libraries

I chose Amazon dataset from Catboost experiment because of the large AUC gain reported

	Default CatBoost	Tuned CatBoost	Default LightGBM	Tuned LightGBM	Default XGBoost	Tuned XGBoost	Default H2O	Tuned H2O
Amazon	0.138114 (±0.0004) (+0.29%)	**0.137720** (±0.0005)	0.167159 (±0.0000) (+21.38%)	0.163600 (±0.0002) (+18.79%)	0.165365 (±0.0000) (+20.07%)	0.163271 (±0.0001) (+18.55%)	0.169497 (±0.0000) (+23.07%)	0.162641 (±0.0001) (+18.09%)

My results (4 cores - Intel Core i7-6700HQ, 8GB RAM DDR4, 2133MHz)
1000 trees, 3fold cross validation, (depth=10 for times), averaged over 5 runs

	Xgboost	**LightGBM**	**Catboost - categorical**	**Catboost**
AUC	0.8496	0.8464	**0.8691**	0.8324
Training time	6.576s	**1.964s**	2m58s	2m29s
Prediction time	266ms	400ms	474ms	**78.4ms**

Contents

Summary

- Try Catboost for small score improvements
- Switch to LightGBM or xgboost's histogram implementation while doing feature selection, parameter hyperoptimization, etc.
- Use Bayesian optimization for parameter tuning (scikit-optimize or hyperopt)

Examples of problems we tackle with GBDT at McKinsey's Advanced Analytics

- Large scale data science transformations of telecommunication clients
- Diagnosing mechanical properties of rolling mill and annealing furnaces for a steelmaking company
- Predicting the default probability for mortgage borrowers
- Predicting hard disks failures from logs for an IT company

References

- Chen T. – “*Introduction to Boosted Trees*”, presentation
- Dorogush A.V., Gulin A., Gusev G., Kazeev N., Prokhorenkova L., Vorobev A. – “*Fighting biases with dynamic boosting*” (2017)
- Friedman J.H. – “*Greedy Function Approximation: A Gradient Boosting Machine”* (1999)
- Mehta M., Agrawal R., Rissanen J. – “*SLIQ: A Fast Scalable Classifier for Data Mining*” (1996)
- Ping L., Wu Q., Burges C.J. – “*Mcrank: Learning to rank using multiple classification and gradient boosting." Advances in neural information processing systems*” (2007)
- Prettenhofer P., Louppe G. – “*Scikit-Learn Gradient Boosted Regression Trees*”, presentation

- Catboost github repo – https://github.com/catboost/catboost
- LightGBM github repo – https://github.com/Microsoft/LightGBM
- XGBoost github repo – https://github.com/dmlc/xgboost
- Blogposts: (https://blogs.technet.microsoft.com/machinelearning/2017/07/25/lessons-learned-benchmarking-fast-machine-learning-algorithms, https://medium.com/implodinggradients/benchmarking-lightgbm-how-fast-is-lightgbm-vs-xgboost-15d224568031)

My presentation is available at github: https://github.com/MSusik/newgradientboosting