Tinka Data Meetup

Deep Dive ML

July 2022

TINKA

+ Gradient Boosting

XGBoost

LightGBM

CatBoost

The goal: overview, advantages and comparisons

We need the scientific method. Every product, every feature, every marketing campaign - everything a startup does - is understood to be an experiment designed to achieve validated learning.

The Lean Startup, Eric Ries

Gradient Boosting summary

Leo Breiman originated with boosting as an optimization algorithm

Jerome Friedman developed it

Weak "learners" over the residual

Gradient Boosting in practice

TINKA

1st iteration

Height	Color	Gender	Weight	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

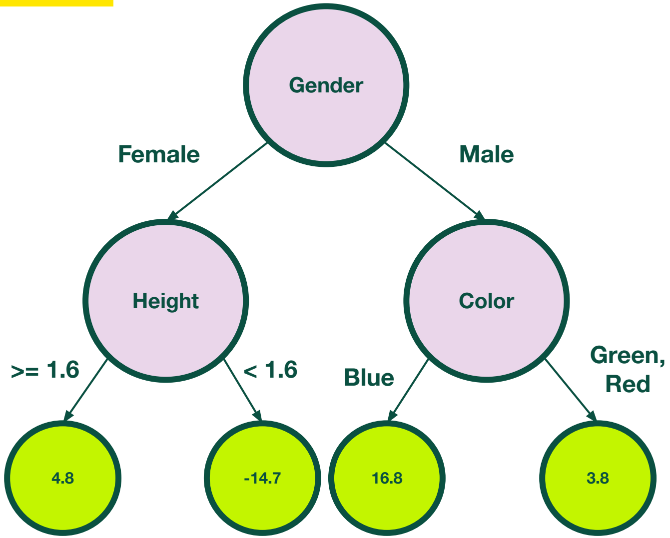
2nd iteration

Height	Color	Gender	Weight	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

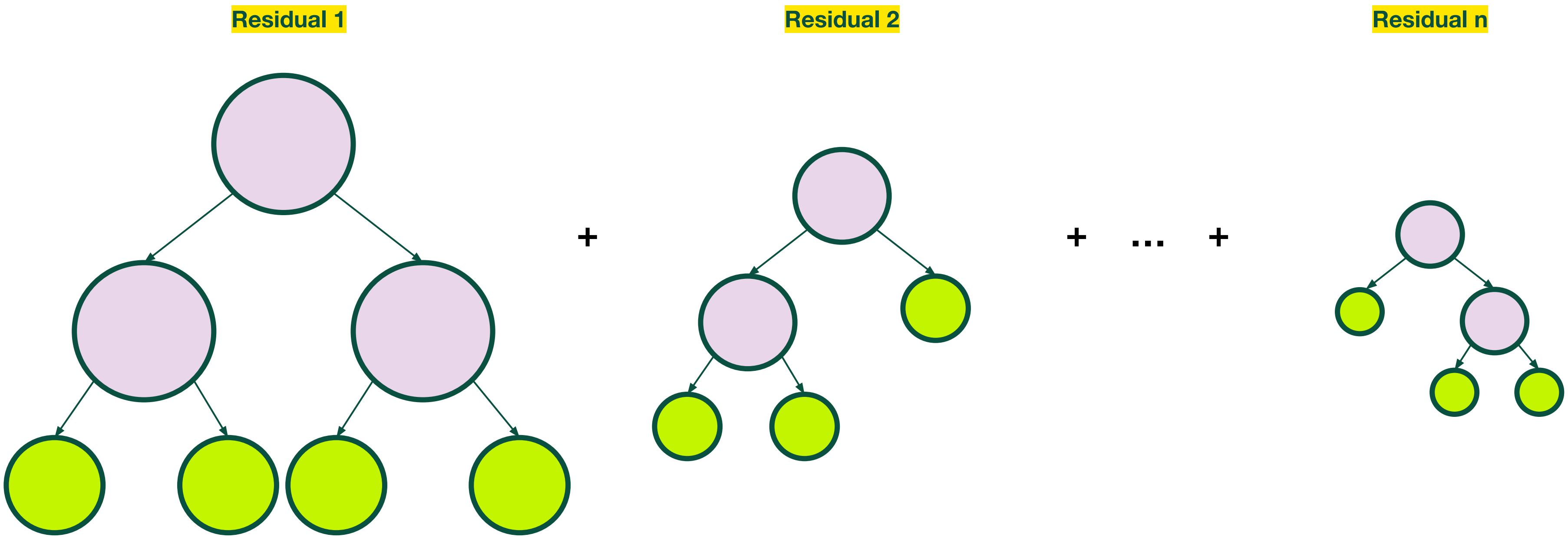
new tree

Average of the target = 71.2

Learning Rate x



Gradient Boosting in practice



Gradient Boosting in theory

1999 paper, **Greedy Function Approximation: A Gradient Boosting Machine** by Jerome Friedman

Algorithm 1: Gradient_Boost

1 $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$

2 For $m = 1$ to M do:

3 $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \ i = 1, N$

4 $\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$

5 $\quad \rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$

6 $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$

7 endFor

end Algorithm

Line 1: "first tree", gets the mean of y

Line 2: iterates, M is the number of trees

Line 3: calculate the residual as the derivative over the value - that derivative is the gradient that gradient boost is named after

Line 4 and 5: finds the ρ that minimizes the loss

Line 6: updates the iteration m with m-1 tree plus weight times new tree

XGBoost summary

TINKA

2016 paper, **XGBoost: A Scalable Tree Boosting System** by Tianqi Chen and Carlos Guestrin

Big winner in Kaggle competitions (17 out of 29 challenges in 2015)

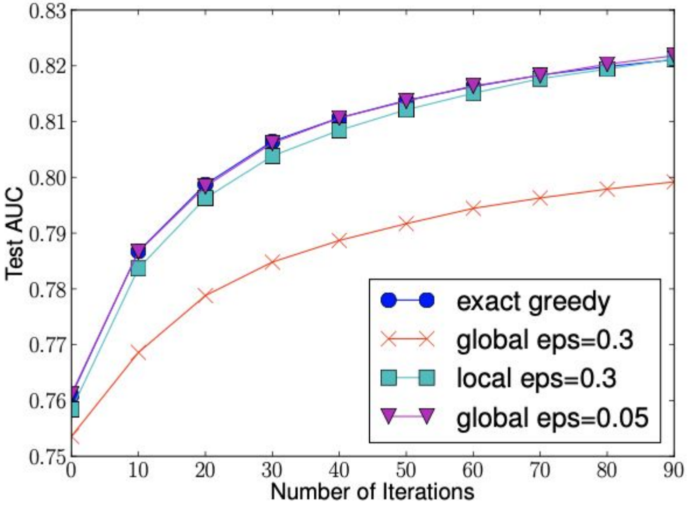
Faster than the current (back then) algorithms

Innovated by: novel technique for sparse data and approximate tree learning

TINKA

Problem: Exact greedy algorithm

Solution: Approximate algorithm



XGBoost in theory

Problem: Sparse x (missing, zeros, OHE)

Solution: Default direction

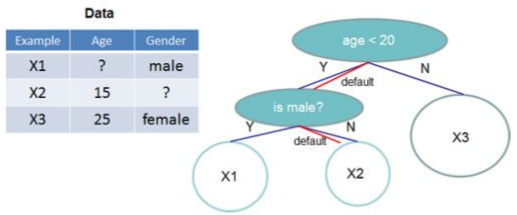


Figure 4: Tree structure with default directions. An example will be classified into the default direction when the feature needed for the split is missing.

TINKA

Problem: Sorting data is time consuming

Solution: Store the data in in-memory units (blocks)

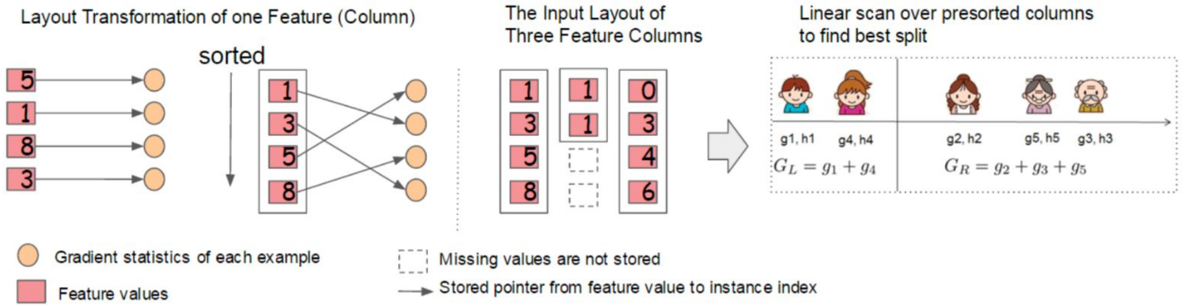


Figure 6: Block structure for parallel learning. Each column in a block is sorted by the corresponding feature value. A linear scan over one column in the block is sufficient to enumerate all the split points.

XGBoost in theory

TINKA

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

Out-of-core computation = when data is too large to fit into memory

Block Compression: block compressed by columns (26%~29% ratio)
Block Sharding: split the data into multiple disks

LightGBM summary

TINKA

2017 paper, **LightGBM: A Highly Efficient Gradient Boosting Decision Tree** by Guolin Ke et al.

It's fast, almost 20 times faster than GBDT

The same or similar performance to XGBoost, e.g.

Innovated by: GOSS and EFB

TINKA

GOSS or Gradient-based One-Side Sampling

Excludes data instances with small gradients when estimating the information gain

Algorithm

1. Firstly sorts the data instances according to the absolute value of their gradients
2. Selects top a × 100% instances and b × 100% instances from the rest
3. Amplifies the sampled data with small gradients by a constant ((1 - a) / b) when calculating the information gain

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models $\leftarrow$ {}, fact $\leftarrow \frac{1-a}{b}$
topN $\leftarrow$ a $\times$ len(I) , randN $\leftarrow$ b $\times$ len(I)
for $i = 1$ ***to*** d **do**
  preds $\leftarrow$ models.predict(I)
  g $\leftarrow$ $loss$(I, preds), w $\leftarrow$ {1,1,...}
  sorted $\leftarrow$ GetSortedIndices(abs(g))
  topSet $\leftarrow$ sorted[1:topN]
  randSet $\leftarrow$ RandomPick(sorted[topN:len(I)], randN)
  usedSet $\leftarrow$ topSet + randSet
  w[randSet] $\times=$ fact ▷ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L(I[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)

TINKA

EFB or Exclusive Feature Bundling

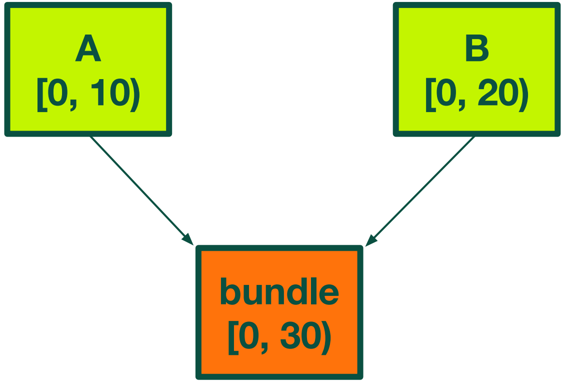
Merges features in the same bundle in order to reduce training complexity

Algorithm

1. Build a graph with weighted edges, the weights correspond to the total **conflicts** between features
2. Sort the features by their degrees in the graph in descending order
3. For each feature in the ordered list, either assign it to an existing bundle with a small conflict or create a new one

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges $\leftarrow \{0\}$, totalBin $\leftarrow 0$
for f ***in*** F **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
  newBin[i] $\leftarrow 0$
  for $j = 1$ ***to*** *len*(F) **do**
    if $F[j].bin[i] \neq 0$ **then**
      newBin[i] $\leftarrow F$[j].bin[i] + binRanges[j]
Output: $newBin$, $binRanges$



LightGBM in theory

xgb_exa = pre-sorted algorithm

xgb_his = histogram-based algorithm

lgb_baseline = LightGBM without GOSS and EFB

SGB = Stochastic Gradient Boosting

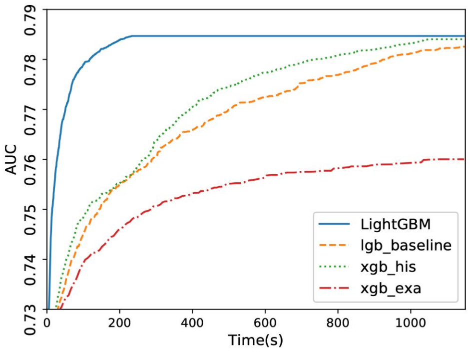


Figure 1: Time-AUC curve on Flight Delay.

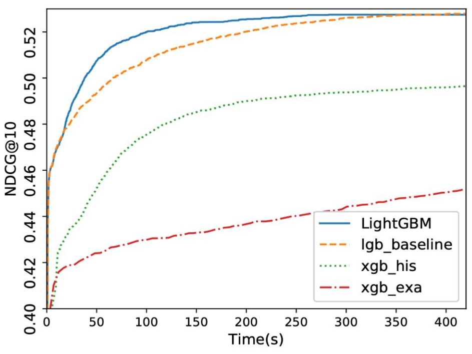


Figure 2: Time-NDCG curve on LETOR.

Table 3: Overall accuracy comparison on test datasets. Use AUC for classification task and NDCG@10 for ranking task. SGB is lgb_baseline with Stochastic Gradient Boosting, and its sampling ratio is the same as LightGBM.

	xgb_exa	xgb_his	lgb_baseline	SGB	**LightGBM**
Allstate	0.6070	0.6089	0.6093	$0.6064 \pm$ 7e-4	**0.6093** $\pm$ **9e-5**
Flight Delay	0.7601	0.7840	0.7847	$0.7780 \pm$ 8e-4	**0.7846** $\pm$ **4e-5**
LETOR	0.4977	0.4982	0.5277	$0.5239 \pm$ 6e-4	**0.5275** $\pm$ **5e-4**
KDD10	0.7796	OOM	0.78735	$0.7759 \pm$ 3e-4	**0.78732** $\pm$ **1e-4**
KDD12	0.7029	OOM	0.7049	$0.6989 \pm$ 8e-4	**0.7051** $\pm$ **5e-5**

CatBoost summary

TINKA

2019 paper, **CatBoost: Unbiased Boosting with Categorical Features** by Liudmila Prokhorenkova et al.

Outperforms the state-of-the-art implementations (XGBoost and LightGBM)

Proposes a new approach for eliminating target leakage during training

No preprocessing for categorical features is needed

CatBoost in theory

Problem

Prediction shift of the learned model

Why?

A model obtained from standard boosting relies on the targets of all training examples. This shifts the distribution of **F(x_k)|x_k for a training example x_k**, from the distribution of **F(x)|x for a test example x**.

Solution

Ordered boosting!

1

2

3

4

M_4

Computes residual at each data point i using the **previous model**

5

$$r(x_5, y_5) = y_5 - M_4(x_5)$$

...

n

CatBoost in theory

TINKA

2001 paper, **A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems** by Daniele Micci-Barreca

Group categories by target statistics (TS) that estimate expected target value in each category. → It's just the **average of the target per category**!

$$\mathbb{E}(y|x^i = x^i_k)$$

Ordered Target Encoding (or target statistics)

$$\hat{x}^i_k = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x^i_j = x^i_k\}} \cdot y_j + a\,p}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x^i_j = x^i_k\}} + a}$$

a > 0 is a parameter
p is usually the avg target value (prior)
D is the full dataset, D_k is a subset excl. x_k

Color	Target	Target Encoding
blue	0	(0+0.05)/(0+1) = 0.05
red	1	(0+0.05)/(0+1) = 0.05
blue	1	(0+0.05)/(1+1) = 0.025
blue	1	(1+0.05)/(2+1) = 0.35
green	0	(0+0.05)/(0+1) = 0.05
red	0	(1+0.05)/(1+1) = 0.025

CatBoost in practice

	XGBoost	LightGBM	CatBoost
Categorical Variables	OHE or target-encoded	categorical_feature parameter	cat_features parameter
Missing Values	Branch direction is learnt	Skip data point during split	Numerical values are set to min

Table 2: Comparison with baselines: logloss / zero-one loss (relative increase for baselines).

	CatBoost	LightGBM	XGBoost
Adult	**0.270 / 0.127**	+2.4% / +1.9%	+2.2% / +1.0%
Amazon	**0.139 / 0.044**	+17% / +21%	+17% / +21%
Click	**0.392 / 0.156**	+1.2% / +1.2%	+1.2% / +1.2%
Epsilon	**0.265 / 0.109**	+1.5% / +4.1%	+11% / +12%
Appetency	**0.072 / 0.018**	+0.4% / +0.2%	+0.4% / +0.7%
Churn	**0.232 / 0.072**	+0.1% / +0.6%	+0.5% / +1.6%
Internet	**0.209 / 0.094**	+6.8% / +8.6%	+7.9% / +8.0%
Upselling	**0.166 / 0.049**	+0.3% / +0.1%	+0.04% / +0.3%
Kick	**0.286 / 0.095**	+3.5% / +4.4%	+3.2% / +4.1%

TINKA

Which model is better? **It depends..**

To consider: amount of data, type of explanatory features, sparsity

Real-time performance

QUESTIONS?