



# ТЕХНОСФЕРА

## Лекция 4 LightGBM & CatBoost

Владимир Гулин

21 сентября 2019 г.

# План лекции

Напоминание

Blending & stacking

LightGBM

Catboost

# Задача обучения с учителем

## Постановка задачи

Пусть дан набор объектов

$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ ,  $\mathbf{x}_i \in \mathcal{X}$ ,  $y_i \in \mathcal{Y}$ ,  $i \in 1, \dots, N$ , полученный из неизвестной закономерности  $y = f(\mathbf{x})$ . Необходимо построить такую  $h(\mathbf{x})$ , которая наиболее точно аппроксимирует  $f(\mathbf{x})$ .

Будем искать неизвестную

$$h(\mathbf{x}) = C(a_1(\mathbf{x}), \dots, a_T(\mathbf{x}))$$

$a_i(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{R}$ ,  $\forall i \in \{1, \dots, T\}$  - базовые модели

$C : \mathcal{R} \rightarrow \mathcal{Y}$  - решающее правило

# Методы построения алгоритмических композиций

Стохастические методы

Бустинг

Мета алгоритмы. Stacking & Blending

# Алгоритмические композиции

## Simple Voting

$$h(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T a_i(\mathbf{x})$$

## Weighted Voting

$$h(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T b_i a_i(\mathbf{x}), \quad b_i \in \mathcal{R}$$

## Mixture of Experts

$$h(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T b_i(\mathbf{x}) a_i(\mathbf{x}), \quad b_i(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{R}$$

# Gradient boosting algorithm

1. Инициализировать  $h_0(\mathbf{x}) = \operatorname{argmin}_{\gamma} \sum_{j=1}^N L(y_j, \gamma)$
2. Для всех  $i$  от 1 до  $T$ :
  - (a) Для всех  $j = 1, 2, \dots, N$  вычислить

$$g_{i,j} = - \left[ \frac{\partial L(y_j, h(\mathbf{x}_j))}{\partial h(\mathbf{x}_j)} \right]_{h=h_{i-1}}$$

- (b) Построить базовую модель  $a_i$  на ответах  $g_{i,j}$

$$a_i = \operatorname{argmin}_a \sum_{j=1}^N (g_{i,j} - a(\mathbf{x}_j))^2$$

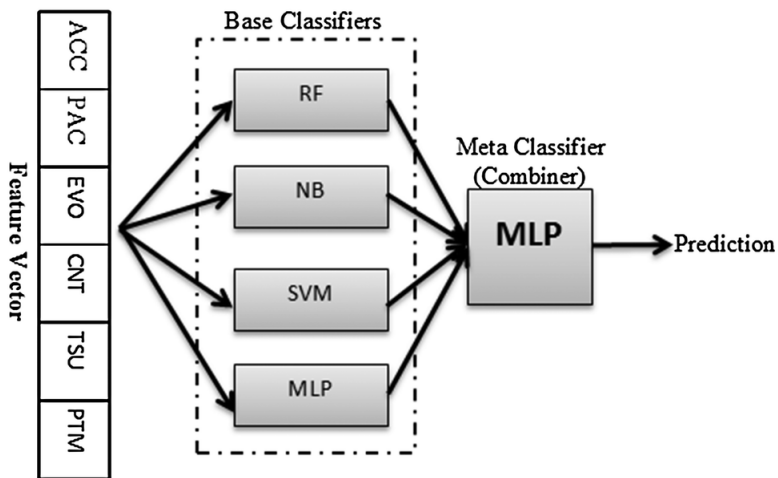
- (c) Определить вес  $b_i$

$$b_i = \operatorname{argmin}_b \sum_{j=1}^N L(y_j, h_{i-1}(\mathbf{x}) + b \cdot a_i(\mathbf{x}_j))$$

- (d) Присвоить  $h_i(\mathbf{x}) = h_{i-1}(\mathbf{x}) + b_i \cdot a_i(\mathbf{x})$
3. Вернуть  $h(\mathbf{x}) = h_T(\mathbf{x})$

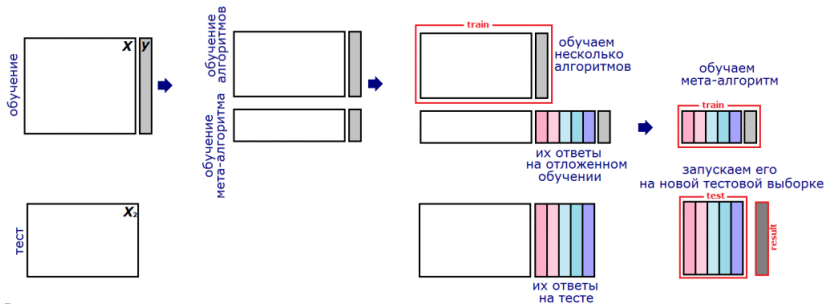
# Stacking

## Мета-алгоритм



# Blending

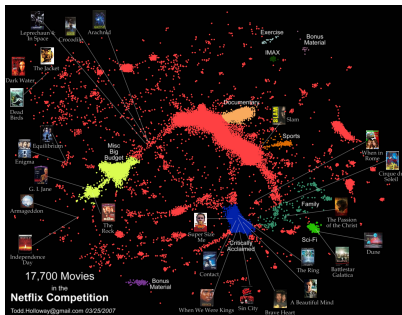
## Классическая схема



### ► Недостатки?



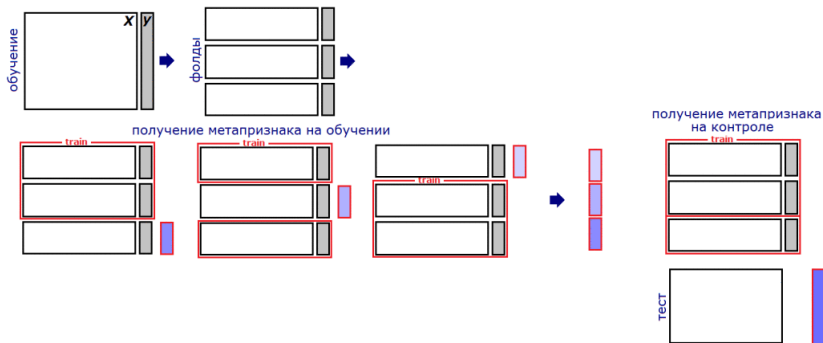
# Netflix Challenge



- ▶ Задача предсказания оценки фильму
- ▶ ПФ - 1000000\$
- ▶ Победил Stacking на “зоопарке” алгоритмов

# Stacking

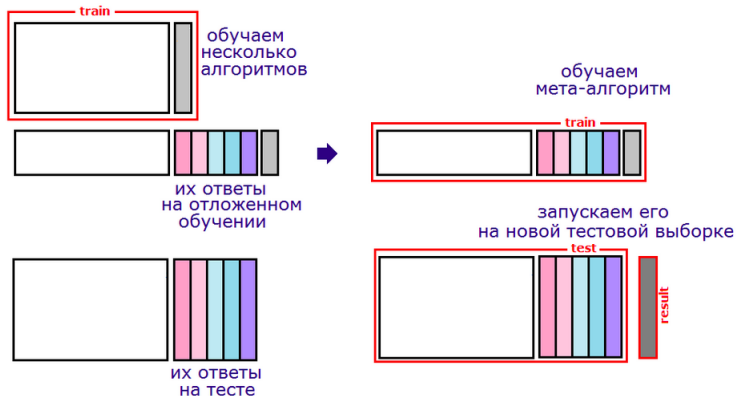
## Классическая схема



► Недостатки?

# Stacking

## Признаки вместе с метапризнаками



► Недостатки?

# LightGBM (2017)

## A Highly Efficient Gradient Boosting Decision Tree

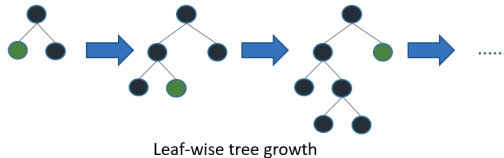
- ▶ Microsoft
- ▶ Позиционируется для больших данных
- ▶ Показывает высокое качество
- ▶ Gradient-based One-Side Sampling
- ▶ Exclusive Feature Bundling

# LightGBM

Level-wise Splits:



Leaf-wise Splits:



# Best-first decision trees

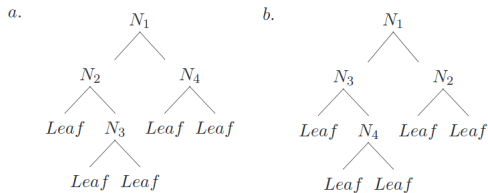


Figure 1.1: Decision trees: (a) a hypothetical depth-first decision tree, (b) a hypothetical best-first decision tree.

# LightGBM

## Gradient-based One-Side Sampling

Row id	gradients
4	-5
3	3
2	0.5
6	0.2
5	0.1
1	0

select top 2  
and randomly  
sample 2  
from the rest

Row id	gradients	weights
4	-5	1
3	3	1
6	0.2	2
5	0.1	2

---

**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

$models \leftarrow \{\}$ ,  $fact \leftarrow \frac{1-a}{b}$

$topN \leftarrow a \times \text{len}(I)$ ,  $randN \leftarrow b \times \text{len}(I)$

**for**  $i = 1$  **to**  $d$  **do**

$preds \leftarrow models.predict(I)$

$g \leftarrow loss(I, preds)$ ,  $w \leftarrow \{1, 1, \dots\}$

$sorted \leftarrow \text{GetSortedIndices}(\text{abs}(g))$

$topSet \leftarrow sorted[1:topN]$

$randSet \leftarrow \text{RandomPick}(sorted[topN:\text{len}(I)],$   
         $randN)$

$usedSet \leftarrow topSet + randSet$

$w[randSet] \times = fact \triangleright$  Assign weight  $fact$  to the  
    small gradient data.

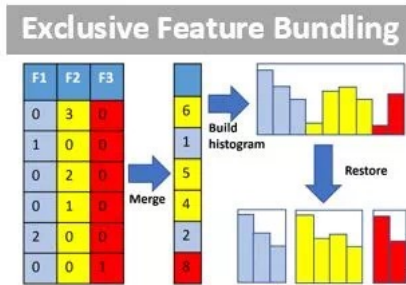
$newModel \leftarrow L(I[usedSet], -g[usedSet],$   
         $w[usedSet])$

$models.append(newModel)$

---



# LightGBM



# LightGBM

---

## Algorithm 3: Greedy Bundling

---

**Input:**  $F$ : features,  $K$ : max conflict count

Construct graph  $G$

$searchOrder \leftarrow G.sortByDegree()$

$bundles \leftarrow \{\}, bundlesConflict \leftarrow \{\}$

**for**  $i$  **in**  $searchOrder$  **do**

$needNew \leftarrow True$

**for**  $j = 1$  **to**  $len(bundles)$  **do**

$cnt \leftarrow ConflictCnt(bundles[j], F[i])$

**if**  $cnt + bundlesConflict[i] \leq K$  **then**

$bundles[j].add(F[i])$ ,  $needNew \leftarrow False$   
            **break**

**if**  $needNew$  **then**

        Add  $F[i]$  as a new bundle to  $bundles$

**Output:**  $bundles$

---

---

## 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

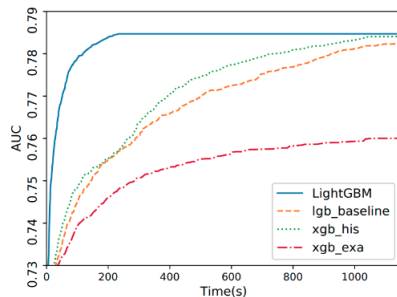


Figure 1: Time-AUC curve on Flight Delay.

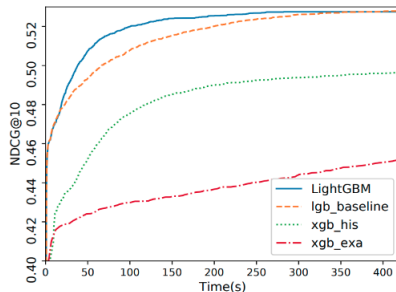
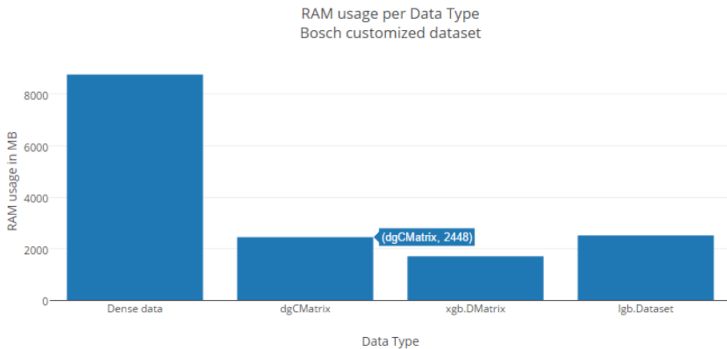


Figure 2: Time-NDCG curve on LETOR.

# LightGBM



RAM efficiency of data types

# Catboost (2017)

## Gradient boosting with categorical features support

- ▶ Yandex
- ▶ Наследник MatrixNet
- ▶ Позиционируется как лучшая по качеству
- ▶ Умеет “умно” работать с категориальными фичами
- ▶ Oblivious trees → fast scoring

---

**Algorithm 1:** Exponential dynamic boosting

---

**input** :  $\{(\mathbf{X}_k, Y_k)\}_{k=1}^n, I;$ 

```

1  $M_S^0 \leftarrow 0$  for all  $S \subset [1, n], |S| \leq I;$ 
2 for  $iter \leftarrow 1$  to  $I$  do
3   foreach  $S$  s.t.  $|S| \leq I - iter$  do
4     foreach  $i \in [1, n] \setminus S$  do
5        $r_i \leftarrow Y_i - M_{S \cup i}^{iter-1}(i);$ 
6        $M \leftarrow LearnModel((\mathbf{X}_i, r_i)$  for
          $i \in [1, n] \setminus S;$ 
7        $M_S^{iter} \leftarrow M_S^{iter-1} + M;$ 
8 return  $M_\emptyset^I$ 
```

---

# Ordered dynamic boosting

---

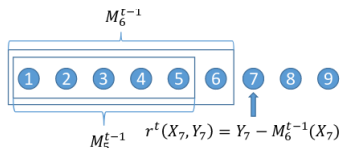
**Algorithm 1:** Updating the models and calculating model values for gradient estimation

---

**input** :  $\{(\mathbf{X}_k, Y_k)\}_{k=1}^n$  ordered according to  $\sigma$ , the number of trees  $I$ ;

```
1  $M_i \leftarrow 0$  for  $i = 1..n$ ;  
2 for  $iter \leftarrow 1$  to  $I$  do  
3   for  $i \leftarrow 1$  to  $n$  do  
4     for  $j \leftarrow 1$  to  $i - 1$  do  
5        $g_j \leftarrow \frac{d}{da} Loss(y_j, a)|_{a=M_i(\mathbf{X}_j)}$ ;  
6        $M \leftarrow LearnOneTree((\mathbf{X}_j, g_j) \text{ for } j = 1..i - 1)$ ;  
7        $M_i \leftarrow M_i + M$ ;  
8 return  $M_1 \dots M_n$ ;  $M_1(\mathbf{X}_1), M_2(\mathbf{X}_2) \dots M_n(\mathbf{X}_n)$ 
```

---



# Скор сплита

$$\text{score}(\text{split}) = \frac{\sum_{doc} \text{leafValue}(doc) * \text{gradient}(doc) * w(doc)}{\sqrt{\sum_{doc} w(doc) * \text{leafValue}(doc)^2}}$$







$$\text{leafValue}(doc) = \frac{\text{sumWeightedDer}}{\text{sumWeights}}$$



# Вычисление значений в листьях

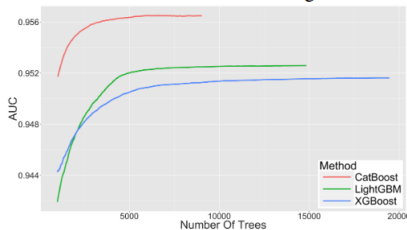
1. Метод Ньютона или шаг по градиенту
2. Несколько шагов внутри одного дерева

# Benchmarks

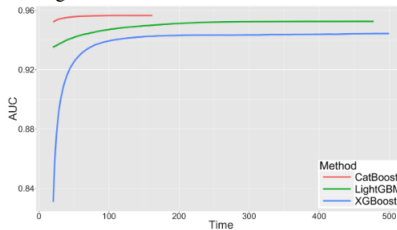
	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
 Adult	<b>0.26974</b>	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
 Amazon	<b>0.13772</b>	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +20.07%	0.16264 +18.10%	0.16950 +23.08%
 Click prediction	<b>0.39090</b>	0.39112 +0.06%	0.39633 +1.39%	0.39749 +1.69%	0.39624 +1.37%	0.39764 +1.73%	0.39759 +1.72%	0.39785 +1.78%
 KDD appetency	0.07151	<b>0.07138</b> -0.19%	0.07179 +0.40%	0.07482 +4.63%	0.07176 +0.35%	0.07466 +4.41%	0.07246 +1.33%	0.07355 +2.86%
 KDD churn	<b>0.23129</b>	0.23193 +0.28%	0.23205 +0.33%	0.23565 +1.89%	0.23312 +0.80%	0.23369 +1.04%	0.23275 +0.64%	0.23287 +0.69%
 KDD internet	<b>0.20875</b>	0.22021 +5.49%	0.22315 +6.90%	0.23627 +13.19%	0.22532 +7.94%	0.23468 +12.43%	0.22209 +6.40%	0.24023 +15.09%

# GPU learning curves

Figure 1: GPU learning curves



(a) AUC vs Number of trees



(b) AUC vs Time

## Scorer comparison

	1 thread	32 threads
CatBoost	2.4s	231ms
XGBoost	78s (x32.5)	4.5s (x19.5)
LightGBM	122s (x50.8)	17.1s (x74)

# Perfomance

Но есть один большой минус — это скорость работы. По моим предварительным наблюдениям, сразу после выхода Кэтбуст отставал от своих аналогов по этому параметру в десятки раз.



**Szilard** @DataScienceLA · 19 июл.

It does not seem to be very fast, preliminary results:



**Benchmark for catboost (from yandex) · Issue #4 · s...**

New boosting lib from yandex:

<https://github.com/catboost/catboost>

github.com



1



3



**Zygmunt Zając** @zygmuntzajac · 20 июл.

More like "very slow"



# Задача

**Дано:** Имеется набор данных из системы поискового антиспама.

**Требуется:** Требуется сравнить ранее рассмотренные классификаторы с lightGBM и catboost.

Пошаговая инструкция

1. Скачать данные и запустить шаблон кода на python  
`goo.gl/CCM2Yo`

```
$ python compos.py -h  
$ python compos.py -tr spam.train.txt -te spam.test.txt
```

2. Построить графики качества классификации в зависимости от параметров алгоритмов
3. Построить графики скорости обучения в зависимости от числа базовых моделей

Вопросы

