

курс «Машинное обучение»

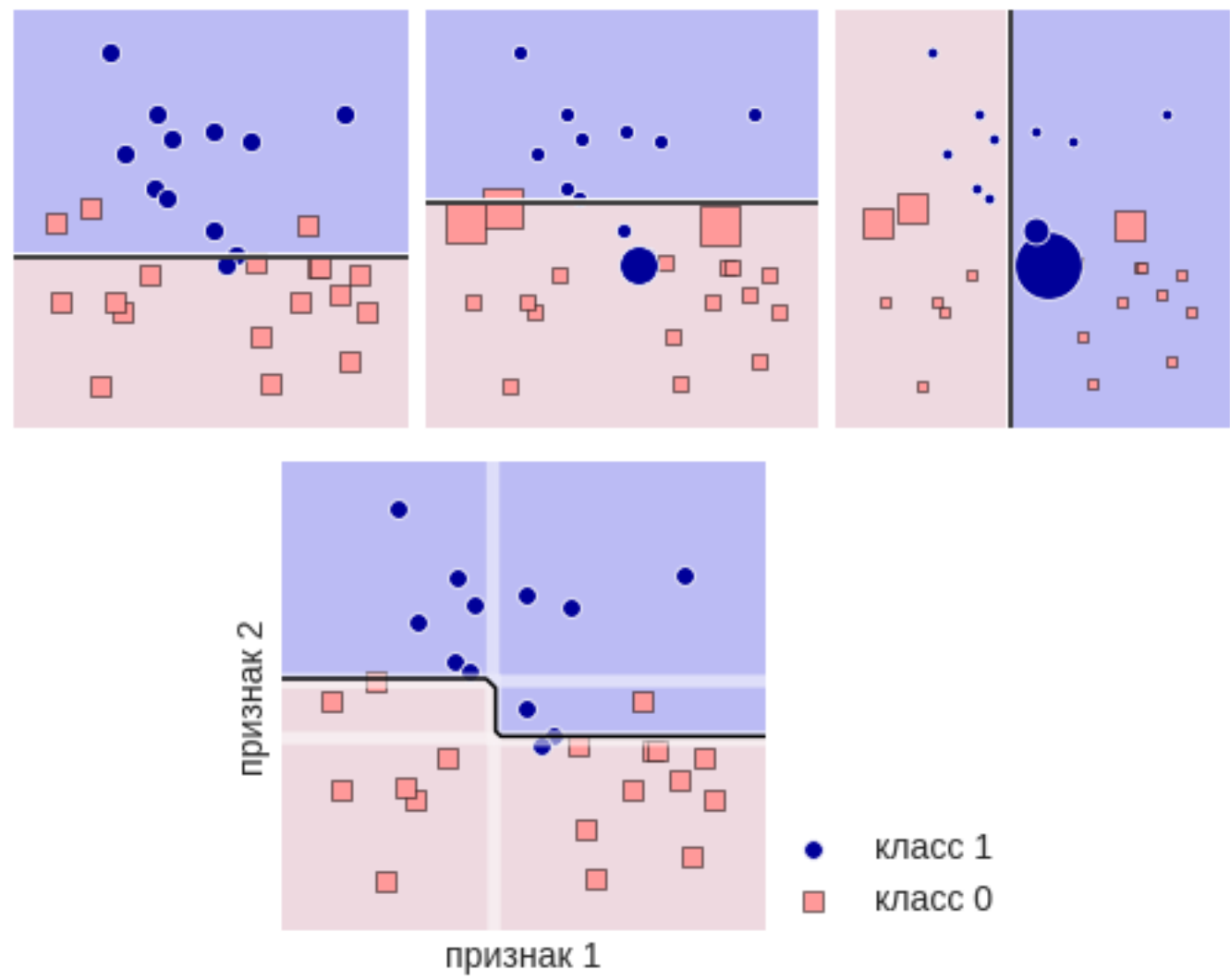
Градиентный бустинг

Александр Дьяконов



Градиентный бустинг над деревьями

Вспоминаем идею...



Идея градиентного бустинга

FSAM + минимизация в случае дифференцируемой ф-ии ошибки

**Задача регрессии с выборкой $(x_i, y_i)_{i=1}^m$,
дифференцируемая функция ошибки $L(y, a)$,
уже есть алгоритм $a(x)$ – строим $b(x)$:**

$$a(x_i) + b(x_i) = y_i, i \in \{1, 2, \dots, m\}.$$

т.е. настраиваемся на невязку

$$b(x_i) \approx y_i - a(x_i)$$

формально надо:

$$\sum_{i=1}^m L(y_i, a(x_i) + b(x_i)) \rightarrow \min$$

а не

$$\sum_{i=1}^m L(y_i - a(x_i), b(x_i)) \rightarrow \min$$

ХОТЯ ЧАСТО ОНИ ЭКВИВАЛЕНТНЫ

Проблема

Задача

$$\sum_{i=1}^m L(y_i, a(x_i) + b(x_i)) \rightarrow \min$$

может не решаться аналитически

$$F(b_1, \dots, b_m) = \sum_{i=1}^m L(y_i, a(x_i) + b_i) \rightarrow \min_{(b_1, \dots, b_m)}$$

**Функция $F(b_1, \dots, b_m)$ убывает в направлении антиградиента,
поэтому выгодно считать**

$$b_i = -L'(y_i, a(x_i)), \quad i \in \{1, 2, \dots, m\}.$$

новая задача для настройки второго алгоритма:

$$(x_i, -L'(y_i, a(x_i)))_{i=1}^m.$$

Алгоритм градиентного бустинга (примитивный вариант)

- Строим алгоритм в виде

$$a_n(x) = \sum_{t=1}^n b_t(x),$$

для удобства можно даже считать, что $a_0(x) \equiv 0$.

- Пусть построен $a_t(x)$, тогда обучаем алгоритм $b_{t+1}(x)$ на выборке

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^m$$

- $a_{t+1}(x) = a_t(x) + b_{t+1}(x)$.

Итерационно получаем сумму алгоритмов...

Вот почему называется **градиентный бустинг**

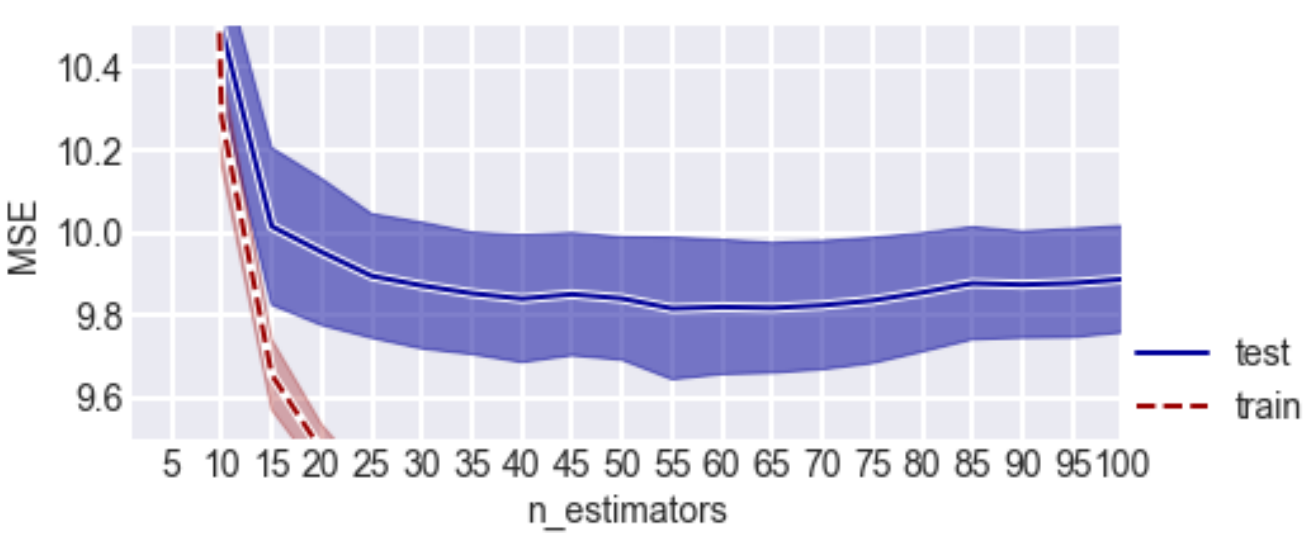
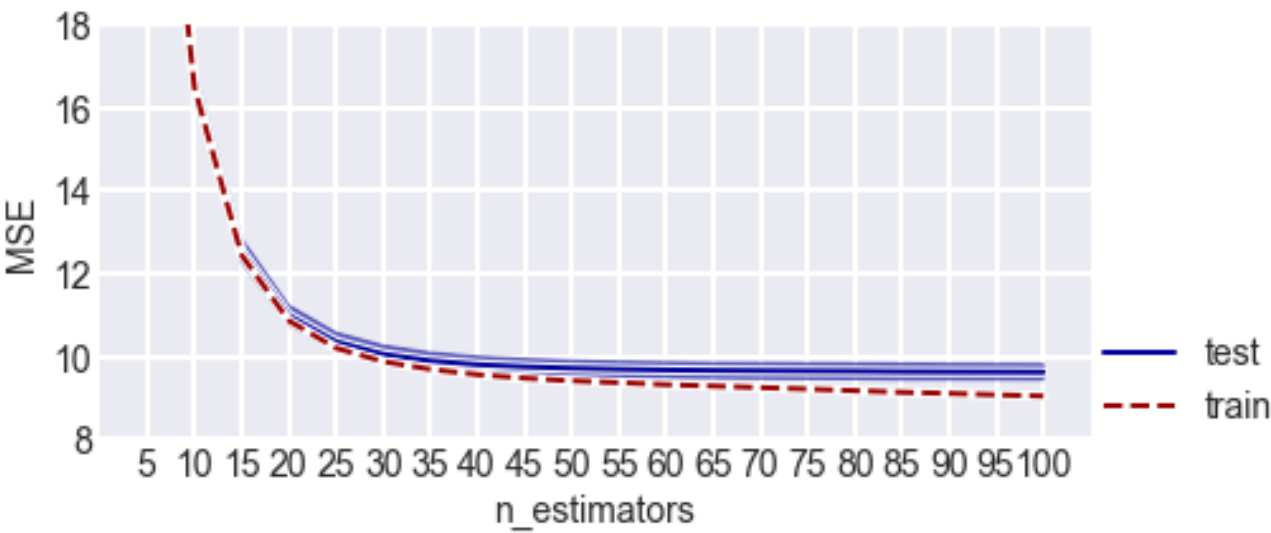
Частный случай: регрессия с СКО

$$L(y,a) = \frac{1}{2}(y - a)^2, \quad L'(y,a) = -(y - a)$$

Задача для настройки следующего алгоритма

$$(x_i, y_i - a_t(x_i))_{i=1}^m$$

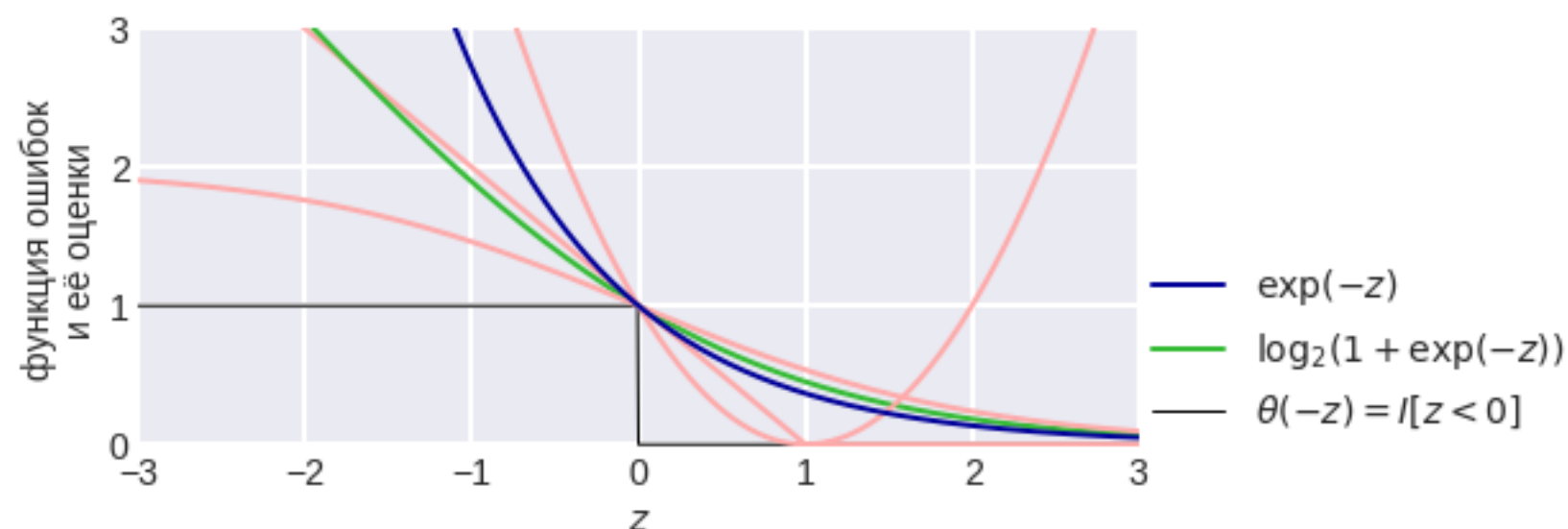
т.е. очень логично: настраиваемся на невязку!



Частный случай: классификация на два класса

нужна дифференцируемая функция ошибки...

- предполагаем, что алгоритм выдаёт вещественные значения
 - нам подходят суррогатные функции ошибки



Частный случай: классификация на два класса

BinomialBoost – логистическая функция ошибки:

$$L(y, a) = \log(1 + e^{-y \cdot a}), \quad a \in (-\infty, +\infty), \quad y \in \{-1, +1\},$$

$$L'(y, a) = -\frac{y}{1 + e^{-y \cdot a}} = -y\sigma(ya).$$

Функция ошибки типа Adaboost:

$$L(y, a) = e^{-y \cdot a}, \quad a \in (-\infty, +\infty), \quad y \in \{-1, +1\},$$

$$L'(y, a) = -ye^{-y \cdot a}.$$

здесь что-то выводится явно...

Итерация градиентного бустинга

Как решать задачу с выборкой

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^m?$$

Любым простым методом!

Мы уже настраиваемся на нужную функцию ошибки.

Проблемы

Шаг в сторону антиградиента

- не приводит в локальный минимум (сразу) \Rightarrow итерации
- мы всё равно не можем сделать такой шаг, а лишь шаг по ответам какого-то алгоритма модели \Rightarrow не нужно стремиться шагать именно туда

Дальше решение проблем...

Наискорейший спуск

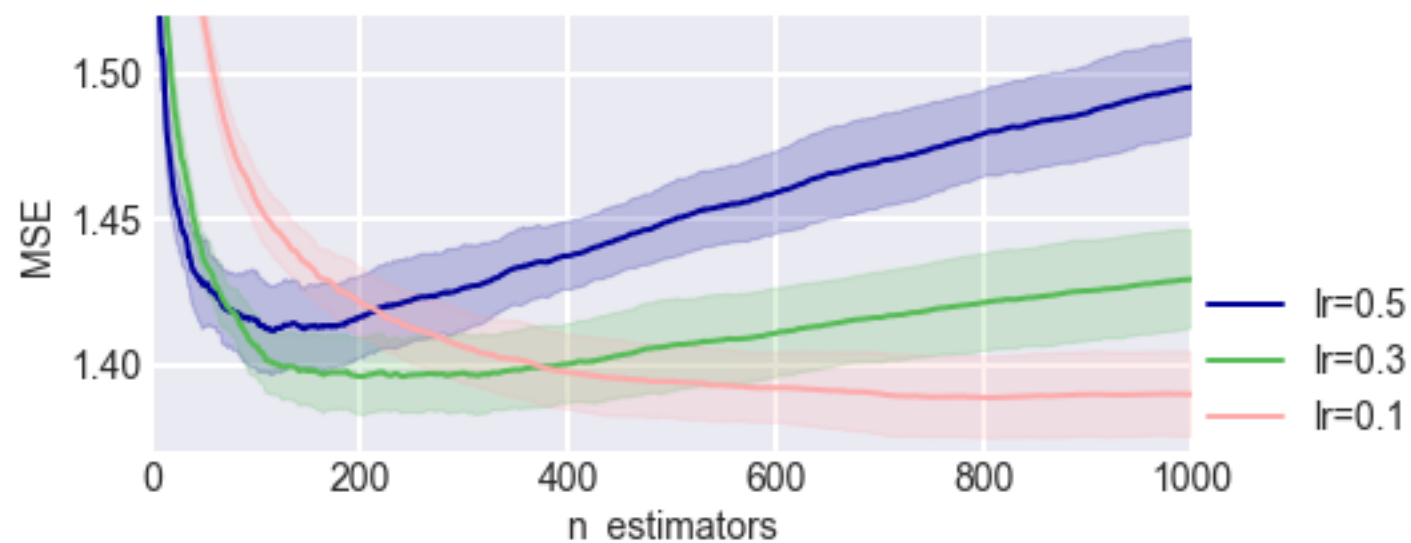
$$\sum_{i=1}^m L(y_i, a_t(x_i) + \eta \cdot b_t(x_i)) \rightarrow \min_{\eta},$$

$$a_{t+1}(x) = a_t(x) + \eta_t \cdot b_t(x) = \eta_1 \cdot b_1(x) + \dots + \eta_t \cdot b_t(x)$$

Эвристика сокращения – Shrinkage

$$a_{t+1}(x) = a_t(x) + \eta \cdot b_t(x),$$

$\eta \in (0, 1]$ – скорость (темп) обучения (learning rate)

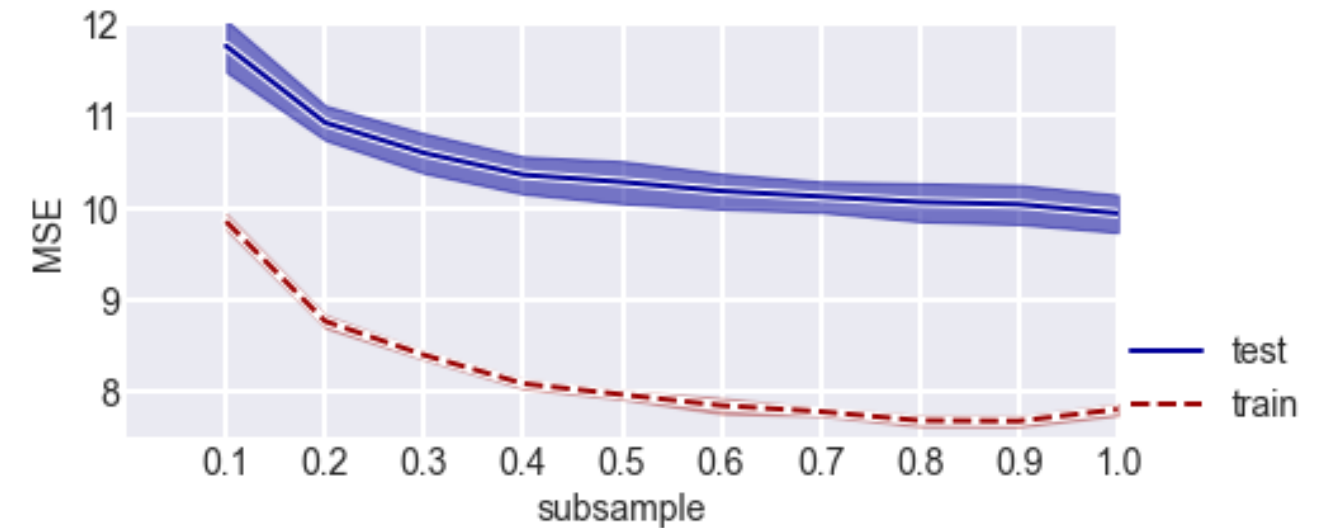
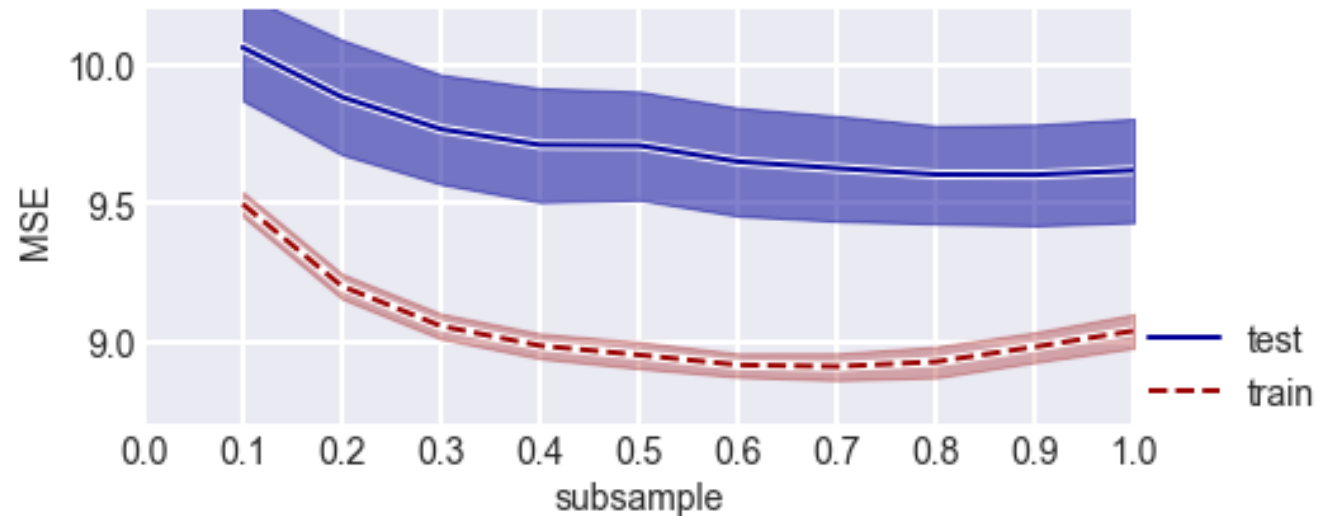


Видно, что число слагаемых (базовых алгоритмов) – шагов бустинга – надо контролировать (при увеличении можем переобучиться)

Чем меньше скорость, тем больше итераций надо

Стохастический градиентный бустинг (Stochastic gradient boosting)

**Идея бэгинга Бреймана: bag fraction ~ берём часть всей выборки
бутстрепа обычно нет...**

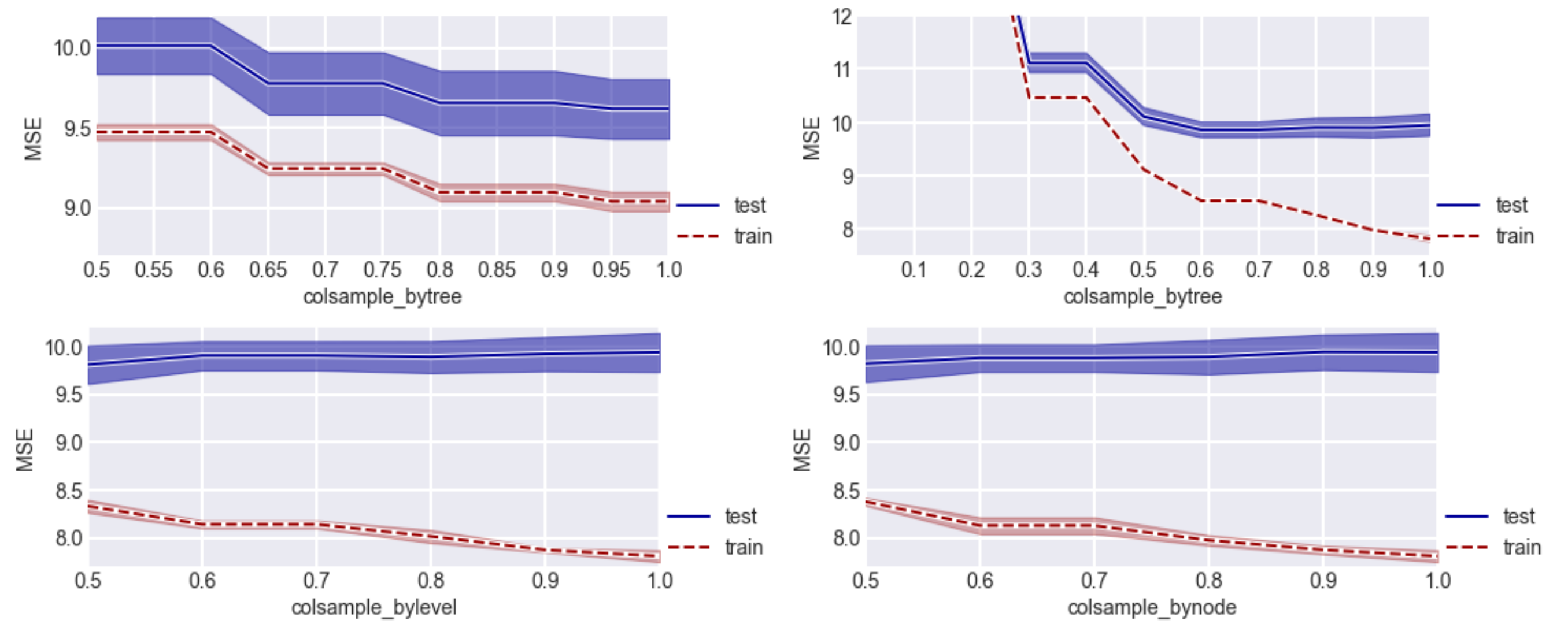


- м.б. лучше качество («регуляризация»)
- быстрее
- аналог обучения по минибатчам
- можно вычислить OOB-ошибки

J. Friedman «Stochastic Gradient Boost» // 1999 <http://statweb.stanford.edu/~jhf/ftp/stobst.pdf>

Column / Feature Subsampling for Regularization

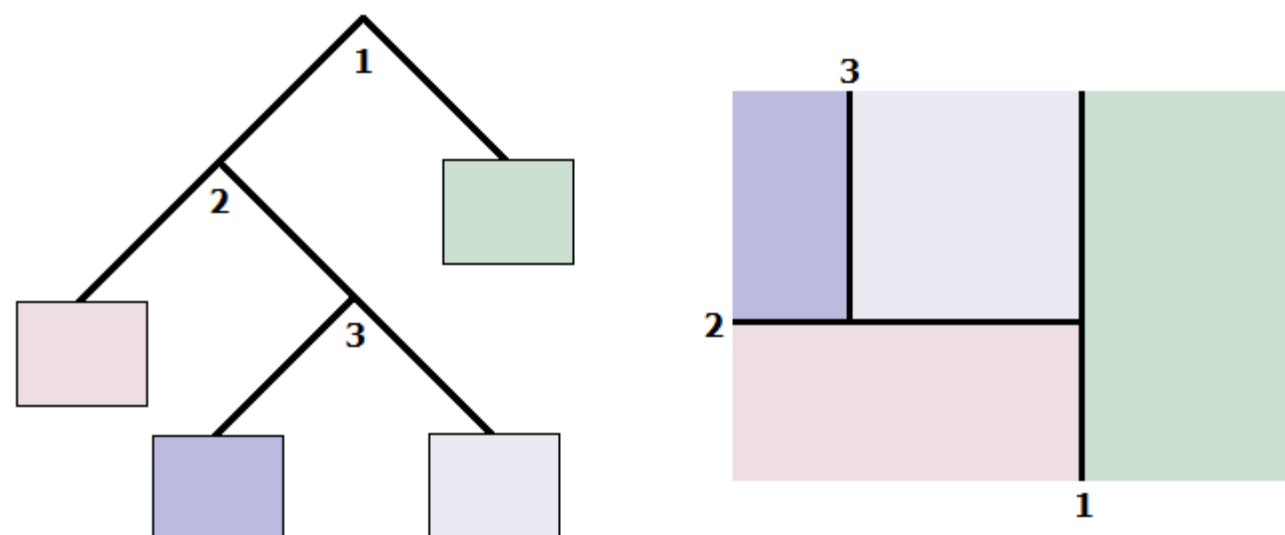
аналогичная идея с признаками



TreeBoost – градиентный бустинг над деревьями

Решающее дерево:

$$b(x) = \sum_j \beta_j I[x \in X_j]$$



TreeBoost – градиентный бустинг над деревьями

Наша основная задача

$$\sum_{i=1}^m L(y_i, a(x_i)) + \sum_j \beta_j I[x_i \in X_j] \rightarrow \min$$

Разбиваем по областям:

$$\sum_{x_i \in X_j} L(y_i, a(x_i) + \beta_j) \rightarrow \min_{\beta_j}.$$

если разбиение выбрано и зафиксировано,
то в каждой области осталось выбрать оптимальную константу

Продвинутые методы оптимизации

Наша основная задача

$$F(b_1, \dots, b_m) = \sum_{i=1}^m L(y_i, a(x_i) + b_i) \rightarrow \min_{(b_1, \dots, b_m)},$$

заметим, что

$$F = \sum_{i=1}^m L(y_i, a(x_i) + b_i) \approx \sum_{i=1}^m \left[L(y_i, a(x_i)) + L'(y_i, a(x_i)) \cdot b_i + \frac{1}{2} L''(y_i, a(x_i)) \cdot b_i^2 \right]$$

(частные производные по второму аргументу функции ошибки)

Продвинутые методы оптимизации

$$\sum_{i=1}^m \left[g_i b_i + \frac{1}{2} h_i b_i^2 \right] \rightarrow \min,$$

$$g_i = L'(y_i, a(x_i)),$$

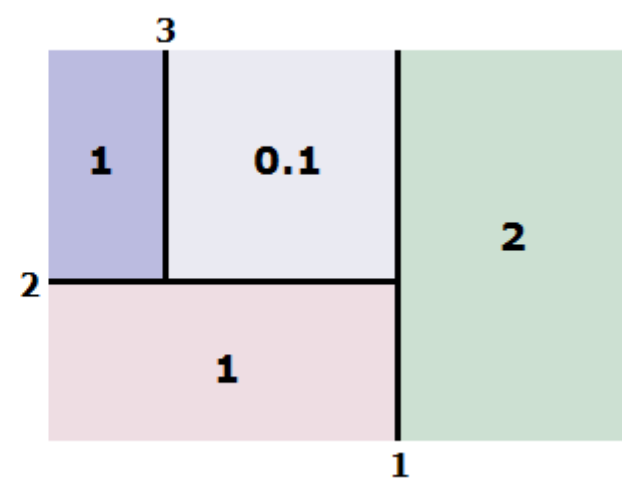
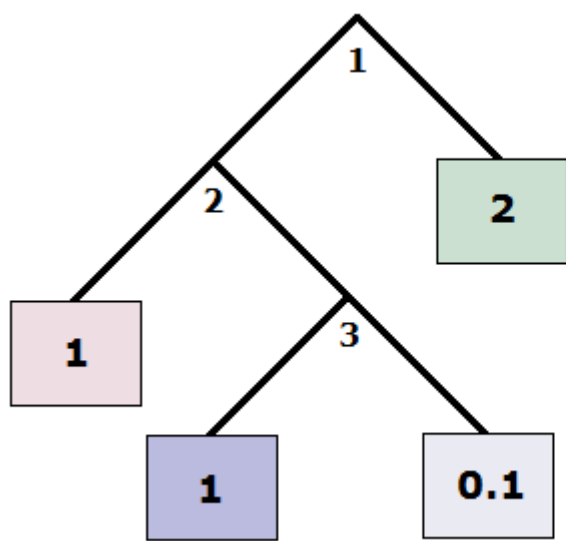
$$h_i = L''(y_i, a(x_i)).$$

Сделаем оптимизацию с регуляризацией

Продвинутые методы оптимизации

Пусть дерево $b(x)$ делит пространство объектов на T областей X_1, \dots, X_T ,
в каждой области X_j принимает значение β_j .

$$\Phi = \sum_{i=1}^m \left[g_i b_i + \frac{1}{2} h_i b_i^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T \beta_j^2 \rightarrow \min$$



$$\Phi = \dots + \gamma 4 + \lambda \frac{1}{2} (1 + 1 + 0.01 + 4)$$

Продвинутые методы оптимизации

$$\begin{aligned}\Phi &= \sum_{j=1}^T \left[\sum_{x_i \in X_j} \left[g_i \beta_j + \frac{1}{2} h_i \beta_j^2 \right] + \lambda \frac{1}{2} \beta_j^2 \right] + \gamma T = \\ &= \sum_{j=1}^T \left[\beta_j \sum_{x_i \in X_j} g_i + \frac{1}{2} \beta_j^2 \left(\sum_{x_i \in X_j} h_i + \lambda \right) \right] + \gamma T\end{aligned}$$

Приравнивая производную к нулю:

$$\beta_j = - \frac{\sum_{x_i \in X_j} g_i}{\sum_{x_i \in X_j} h_i + \lambda}$$

Продвинутые методы оптимизации

Минимальное значение (при фиксированной структуре дерева)

$$\Phi_{\min} = -\frac{1}{2} \sum_{j=1}^T \frac{\left(\sum_{x_i \in X_j} g_i \right)^2}{\sum_{x_i \in X_j} h_i + \lambda} + \gamma T$$

Можно использовать при построении дерева для его оценки:

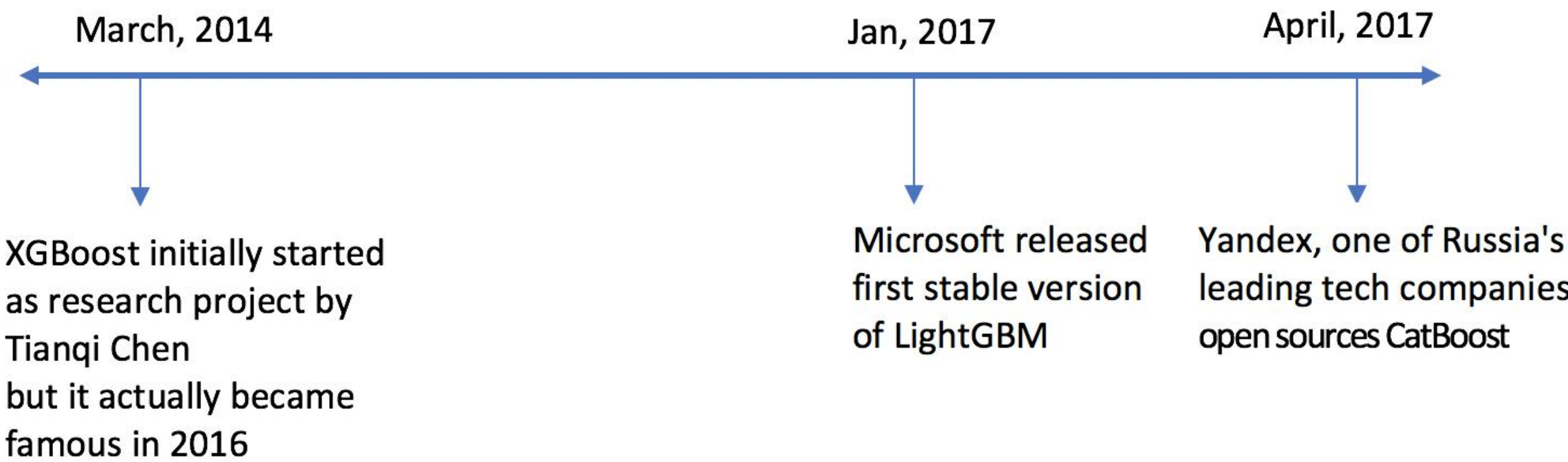
$$\text{Gain} = \frac{1}{2} \left(\frac{\left(\sum_{x_i \in X_{\text{left}}} g_i \right)^2}{\sum_{x_i \in X_{\text{left}}} h_i + \lambda} + \frac{\left(\sum_{x_i \in X_{\text{right}}} g_i \right)^2}{\sum_{x_i \in X_{\text{right}}} h_i + \lambda} - \frac{\left(\sum_{x_i \in X_{\text{left}}} g_i + \sum_{x_i \in X_{\text{right}}} g_i \right)^2}{\sum_{x_i \in X_{\text{left}}} h_i + \sum_{x_i \in X_{\text{right}}} h_i + \lambda} \right) - \gamma$$

Продвинутые методы оптимизации

**можно использовать прунинг –
строим дерево, рекурсивно обрезаем, если $\text{Gain} < 0$**

**Не используем какой-то традиционный критерий расщепления.
Исходим из функции ошибки!**

История продвинутых методов / современные реализации



sklearn.ensemble.	GradientBoostingRegressor GradientBoostingClassifier
XGBoost (eXtreme Gradient Boosting)	https://github.com/dmlc/xgboost
LightGBM, Light Gradient Boosting Machine	https://github.com/Microsoft/LightGBM
CatBoost	https://github.com/catboost/catboost

<https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db>

Особенности реализаций продвинутых методов

	XGBoost	LightGBM	CatBoost
Построение деревьев	По уровням (Level-wise) потом добавили по листьям, но для гистограмм	По листьям (Leaf-wise) best-first	По уровням однородно (oblivious trees)
Поиск расщеплений	Exact greedy algorithm (полный перебор) + добавили потом гистограммный подход tree_method='hist'	Гистограммный подход (использование бинов) +	Предварительный биннинг
Фишки		Exclusive Feature Bundling Связываем разреженные признаки, которые одновременно не нули Random forest mode	Динамический бустинг Overfitting Detector Ранний останов od_type='Iter' use_best_model=True eval_metric=...

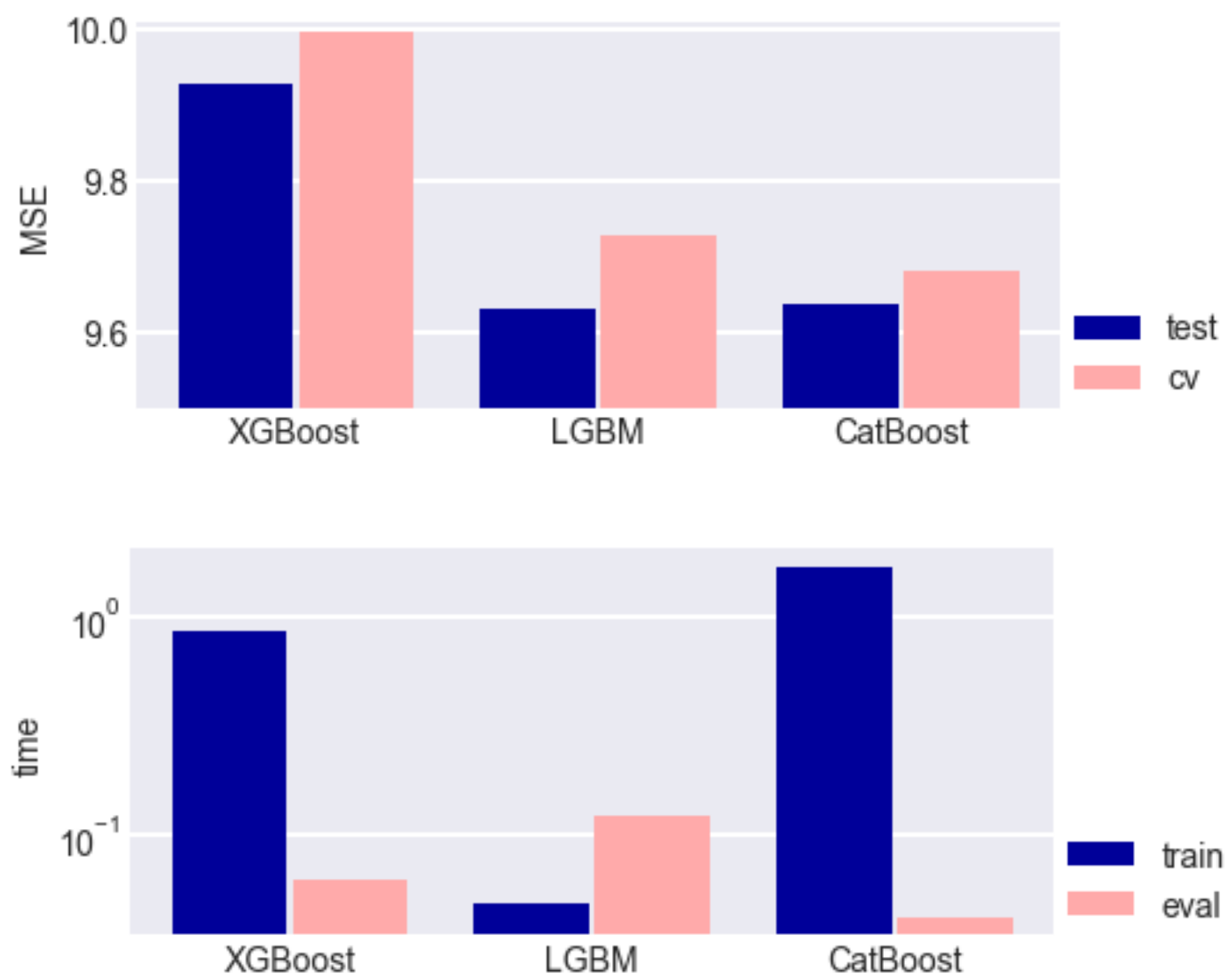
Особенности реализаций продвинутых методов

	XGBoost	LightGBM	CatBoost
Сэмплирование / градиенты / сплиты	– Сплиты медленнее, чем у конкурентов pre-sorted algorithm & Histogram-based	Gradient-based One-Side Sampling (GOSS) Среди малых градиентов сэмплируем, но с большим весом не выбран ли по умолчанию	Бернулли или байесовская подвыборка
Важности признаков	Gain / Frequency или Weight / Coverage	Gain / split	Prediction Values Change / Loss Function Change
Нули обрабатываются как NaN	+ см. ниже Sparsity-aware Split Finding	+ По умолчанию use_missing=True	
Неизвестные значения	На оптимальную сторону сплита	На оптимальную сторону сплита	Min / Max

Особенности реализаций продвинутых методов

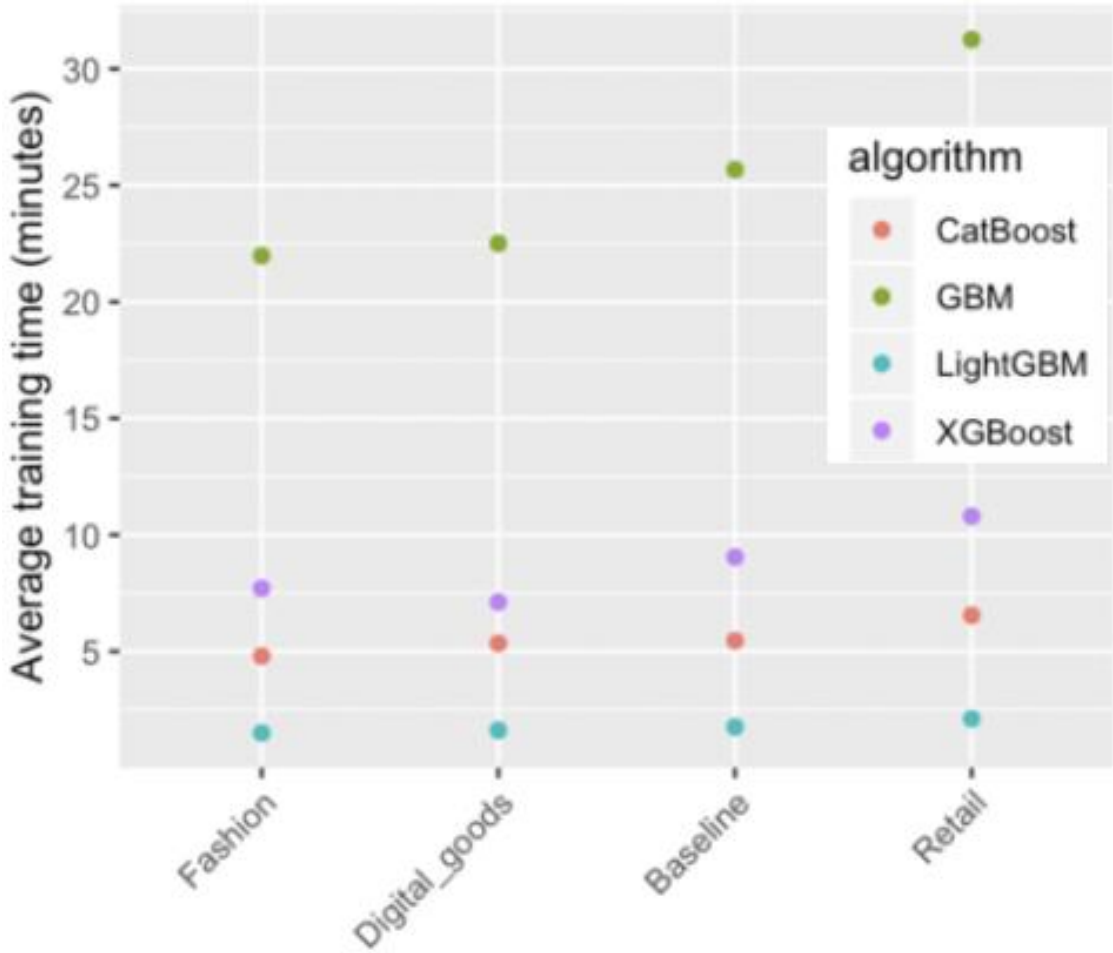
	XGBoost	LightGBM	CatBoost
Категориальные признаки	–	+	+
	вводится в версии 1.5.1	На две части с учётом mean target	Ordered Target Statistics Smoothed target encoding ONE с числом категорий < one_hot_max_size Жадные комбинации категориальных признаков

Сравнение на задаче «Google Brain - Ventilator Pressure Prediction»



тут сравнение с параметрами по умолчанию (что плохо)

Сравнение



	GBM	CatBoost	LightGBM	XGBoost
Baseline	0.9455	0.9534	0.9453	0.9438
Fashion	0.9826	0.9833	0.9815	0.9814
Retail	0.9614	0.9617	0.96	0.96
Digital goods	0.8739	0.8767	0.8732	0.8751

<https://medium.com/riskified-technology/xgboost-lightgbm-or-catboost-which-boosting-algorithm-should-i-use-e7fda7bb36bc>

Сравнение

TABLE II TIME AND AUC USING XGBOOST			TABLE III TIME AND AUC USING LIGHTGBM			TABLE IV TIME AND AUC USING CATBOOST		
#Rows	AUC	Time	#Rows	AUC	Time	#Rows	AUC	Time
307507	0.788320	4306	307507	0.789996	786	307507	0.787629	1803
250000	0.784516	3550	250000	0.788589	638	250000	0.784402	1257
200000	0.781219	2892	200000	0.786344	512	200000	0.782895	851
150000	0.773347	2098	150000	0.786215	393	150000	0.780762	567
100000	0.772771	1219	100000	0.782477	263	100000	0.776168	442
50000	0.768899	9487	50000	0.777649	121	50000	0.770666	286

Essam al Daoud «Comparison between XGBoost, LightGBM and CatBoost Using a Home Credit Dataset»

ниже рис. из <https://arxiv.org/pdf/1809.04559.pdf>
m_catboost – т.к. не было многоклассовой версии на GPU

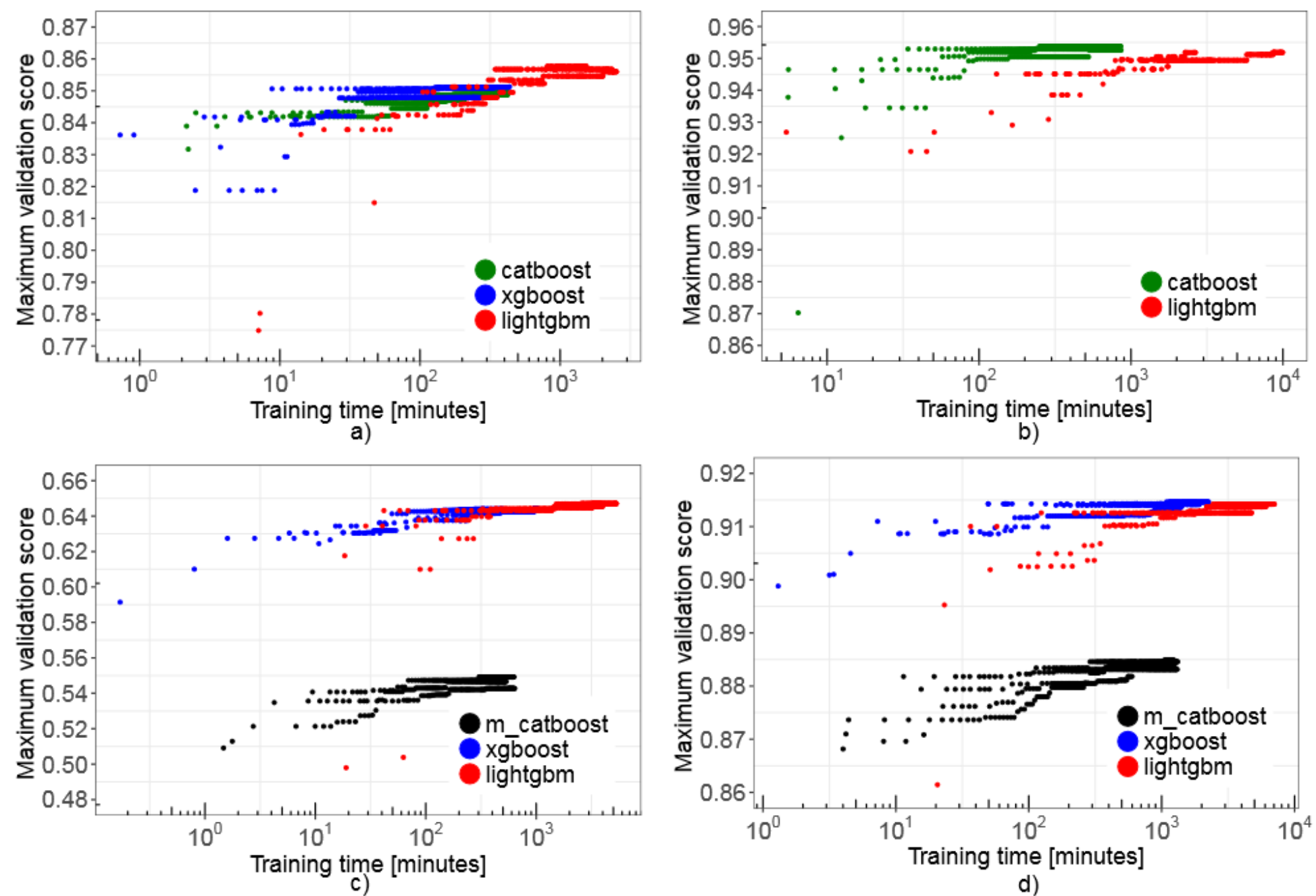
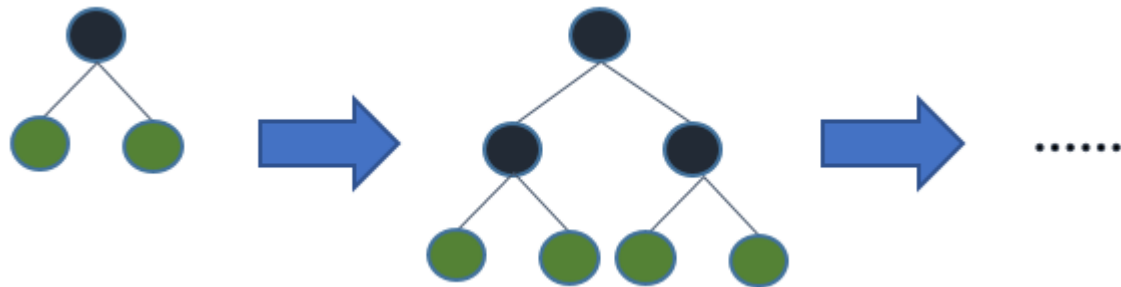


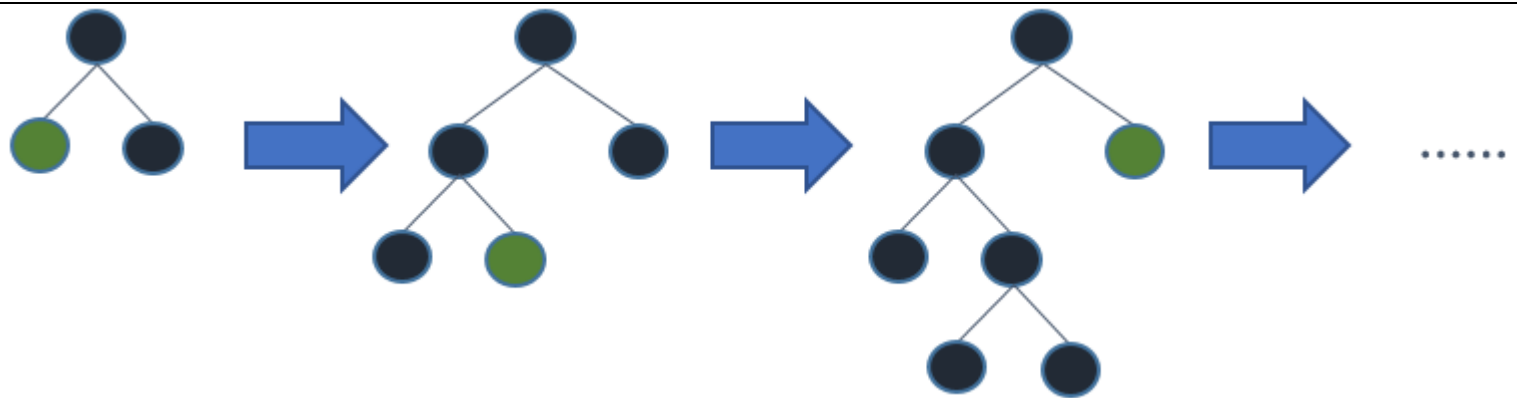
Figure 2: Max. validation score vs. total HPO runtime (a) Higgs (b) Epsilon (c) Microsoft (d) Yahoo.

Построение деревьев

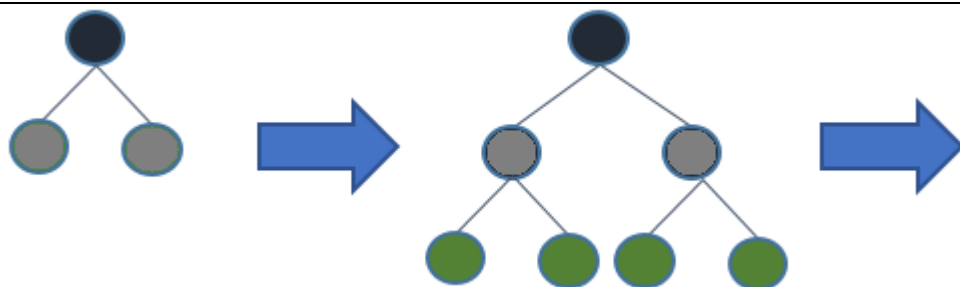
По уровням
(Level-wise)



По листьям
(Leaf-wise)

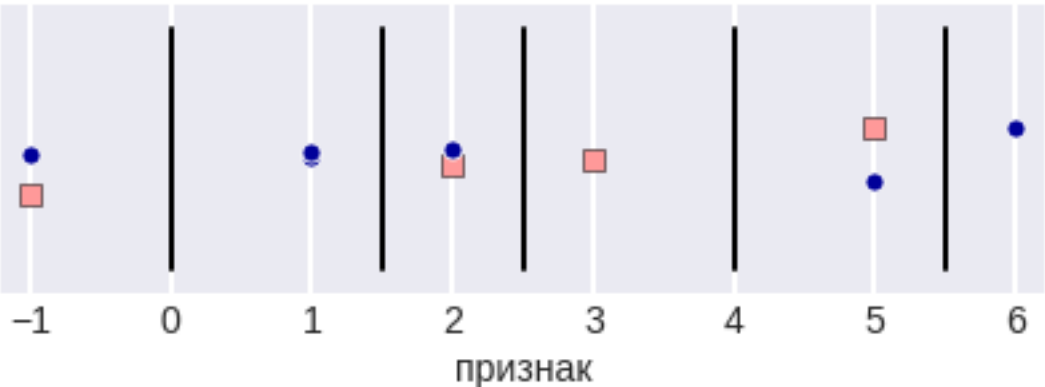


По уровням однородно
(oblivious trees)



<https://github.com/Microsoft/LightGBM/blob/master/docs/Features.rst#references>

Игнорирование нулей / NaN

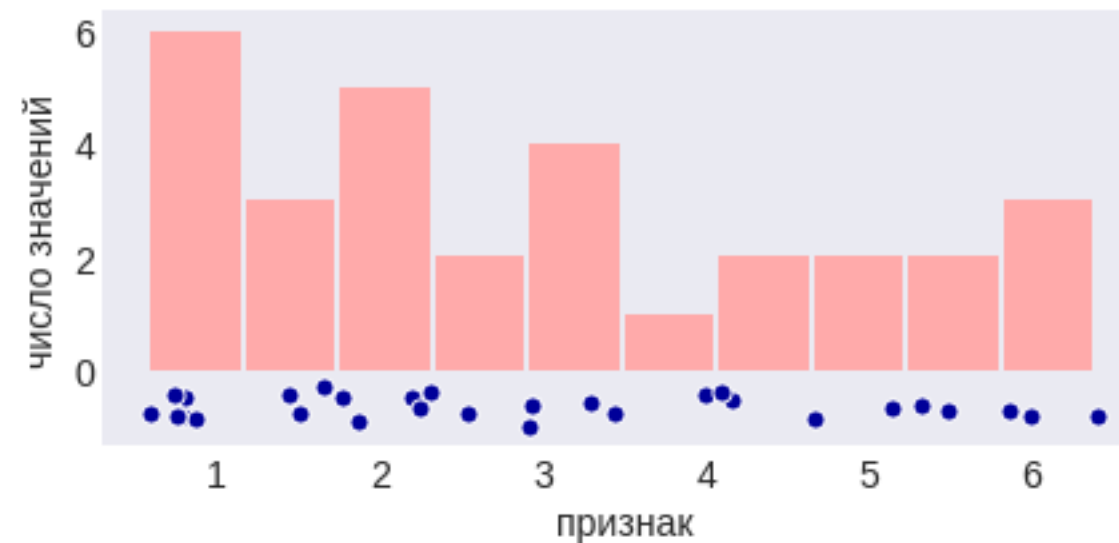


**убираем нули,
выбираем сплит,
нули добавляем в «выгодное поддерево»**

<https://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/>

Гистограммный подход (Histogram based algorithm)

**каждый вещественный признак дискретизуется –
разбивается на бины**



теперь число порогов, которые надо посмотреть ~ число бинов

Exclusive Feature Bundling (EFB)

объединение признаков с большим числом нулей (жадный алгоритм)
поиск оптимального решения – NP-полная задача

	признак 1	признак 2	bundle
0	1	0	1
1	0	1	3
2	1	0	1
3	2	0	2
4	0	2	4
5	0	3	5
6	1	0	1
7	0	2	4
8	2	0	2

строим граф признаков,
веса рёбер = число конфликтов между признаками,
сортируем признаки по степени

идём по признакам, по возможности включаем в
существующие бандлы
(если мало конфликтов)

<https://towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e>

Gradient-based One-Side Sampling (GOSS)

считаем градиенты

$$g(x_1) = 1, g(x_2) = 2, g(x_3) = 0.1, g(x_4) = 0.5, g(x_5) = 2.5, g(x_6) = 0$$

выбираем top-2

$$g(x_1) = 1, g(x_2) = 2, g(x_3) = 0.1, g(x_4) = 0.5, g(x_5) = 2.5, g(x_6) = 0$$

из остальных сэмплируем

$$2 \times g(x_1) = 1, g(x_2) = 2, 2 \times g(x_4) = 0.5, g(x_5) = 2.5$$

но берём с весом

**из объектов с маленькими градиентами сэмплируем (используем не все),
но учитываем с большим весом**

CatBoost = Category + Boosting: проблема смещения

Smoothed target encoding – для категориальных признаков

**При построении дерева
значение в листе = сумма антиградиентов
получается утечка
мы оцениваем значение в точке, зная метку на ней!**

Динамический бустинг (Ordered Boosting)

- случайная перестановка обучения
- оценивания значения, используя информацию до рассматриваемой точки в таблице

oblivious trees – один предикат на каждом уровне

**работает «из коробки»
с параметрами по умолчанию**

CatBoost: Ordered Boosting

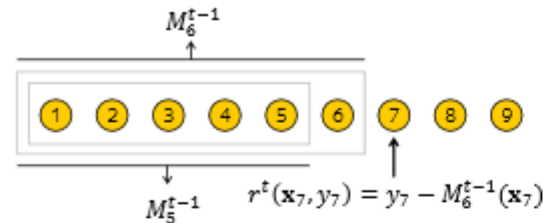


Figure 1: Ordered boosting principle, examples are ordered according to σ .

Algorithm 1: Ordered boosting

input : $\{(\mathbf{x}_k, y_k)\}_{k=1}^n, I$;
 $\sigma \leftarrow$ random permutation of $[1, n]$;
 $M_i \leftarrow 0$ for $i = 1..n$;
for $t \leftarrow 1$ **to** I **do**
 for $i \leftarrow 1$ **to** n **do**
 $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$;
 for $j \leftarrow 1$ **to** n **do**
 $\Delta M \leftarrow$
 $\text{LearnModel}((\mathbf{x}_j, r_j) :$
 $\sigma(j) \leq i)$;
 $M_i \leftarrow M_i + \Delta M$;
return M_n

Algorithm 2: Building a tree in CatBoost

input : $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, \text{Mode}$
 $\text{grad} \leftarrow \text{CalcGradient}(L, M, y)$;
 $r \leftarrow \text{random}(1, s)$;
if $\text{Mode} = \text{Plain}$ **then**
 $G \leftarrow (\text{grad}_r(i) \text{ for } i = 1..n)$;
if $\text{Mode} = \text{Ordered}$ **then**
 $G \leftarrow (\text{grad}_{r, \sigma_r(i)-1}(i) \text{ for } i = 1..n)$;
 $T \leftarrow$ empty tree;
foreach *step of top-down procedure* **do**
 foreach *candidate split* c **do**
 $T_c \leftarrow$ add split c to T ;
 if $\text{Mode} = \text{Plain}$ **then**
 $\Delta(i) \leftarrow \text{avg}(\text{grad}_r(p) \text{ for } p : \text{leaf}_r(p) = \text{leaf}_r(i))$ for $i = 1..n$;
 if $\text{Mode} = \text{Ordered}$ **then**
 $\Delta(i) \leftarrow \text{avg}(\text{grad}_{r, \sigma_r(i)-1}(p) \text{ for } p : \text{leaf}_r(p) = \text{leaf}_r(i), \sigma_r(p) < \sigma_r(i))$ for $i = 1..n$;
 $\text{loss}(T_c) \leftarrow \text{cos}(\Delta, G)$
 $T \leftarrow \arg \min_{T_c} (\text{loss}(T_c))$
if $\text{Mode} = \text{Plain}$ **then**
 $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha \text{avg}(\text{grad}_{r'}(p) \text{ for } p : \text{leaf}_{r'}(p) = \text{leaf}_{r'}(i))$ for $r' = 1..s, i = 1..n$;
if $\text{Mode} = \text{Ordered}$ **then**
 $M_{r', j}(i) \leftarrow M_{r', j}(i) - \alpha \text{avg}(\text{grad}_{r', j}(p) \text{ for } p : \text{leaf}_{r'}(p) = \text{leaf}_{r'}(i), \sigma_{r'}(p) \leq j)$ for $r' = 1..s, i = 1..n, j \geq \sigma_{r'}(i) - 1$;
return T, M

Параметры градиентного бустинга: определяющие тип бустинга

<code>objective</code> <code>/ loss_function</code>	– какая задача решается, какая целевая функция и в каком формате будет ответ
<code>booster</code>	– какой бустинг проводить: над решающими деревьями, линейный или <code>dart</code>
<code>boosting_type</code>	– <code>lgb</code>: <code>gbdt</code> / <code>dart</code> (Dropouts meet Multiple Additive Regression Trees) / <code>goss</code> / <code>rf</code>
<code>tree_method</code>	– как строить деревья (<code>grow_policy</code> – порядок построения дерева: на следующем шаге расщеплять вершину, ближайшую к корню, или на которой ошибка максимальна)
<code>base_score</code>	– начальный ответ на всех объектах (<code>bias</code>)
<code>eval_metric</code>	– значения какой функции ошибки смотреть на контроле (как правило, задание этого параметра не означает, что эту функцию будем минимизировать при настройке бустинга)

DART: Dropouts meet Multiple Additive Regression Trees

Вместо

$$a_n(x) = \sum_{t=1}^n b_t(x)$$

берём подмножество построенных деревьев $Q = \text{randsubset}(\{1, 2, \dots, n\})$

пытаемся дополнить их

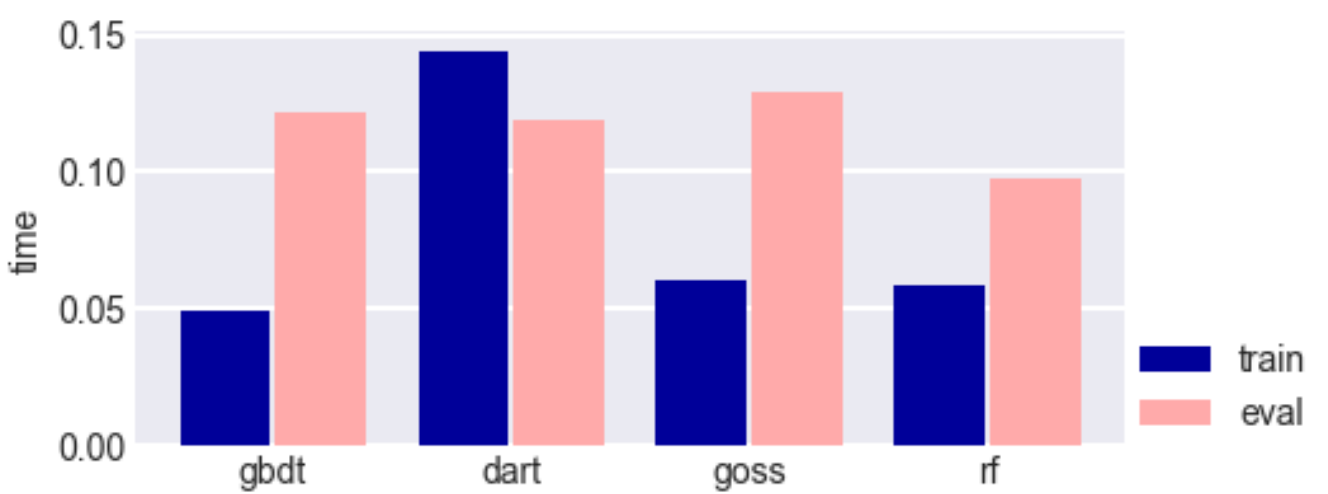
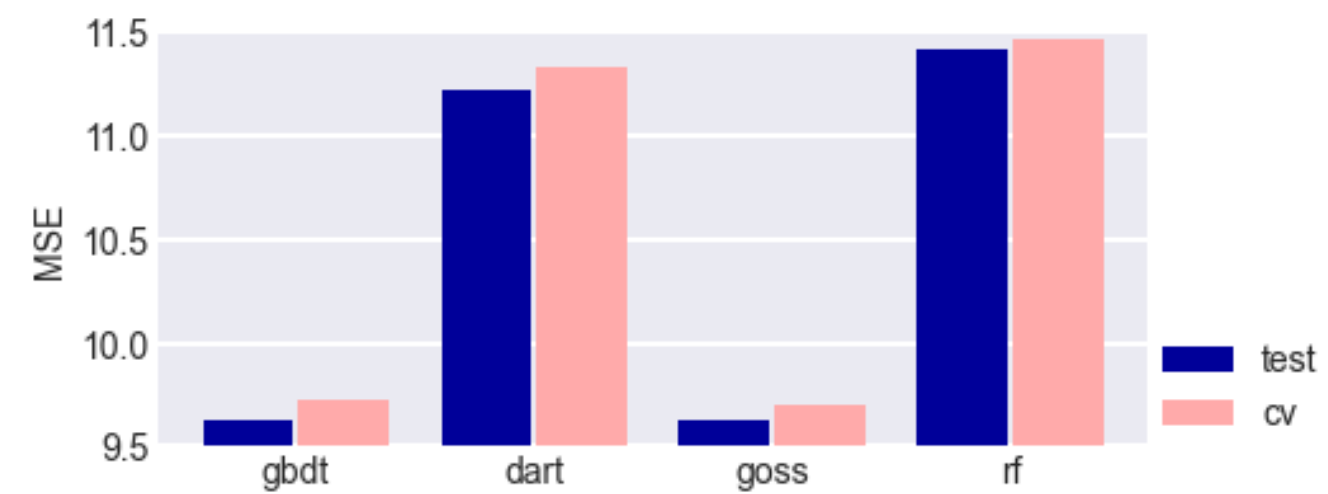
$$b_{t+1} = \arg \min_b \sum_{i=1}^m L(y_i, \sum_{t \in Q} b_t(x_i) + b(x_i))$$

потом нужна поправка, что смещали ответ не всего ансамбля (подробно об это не будем)

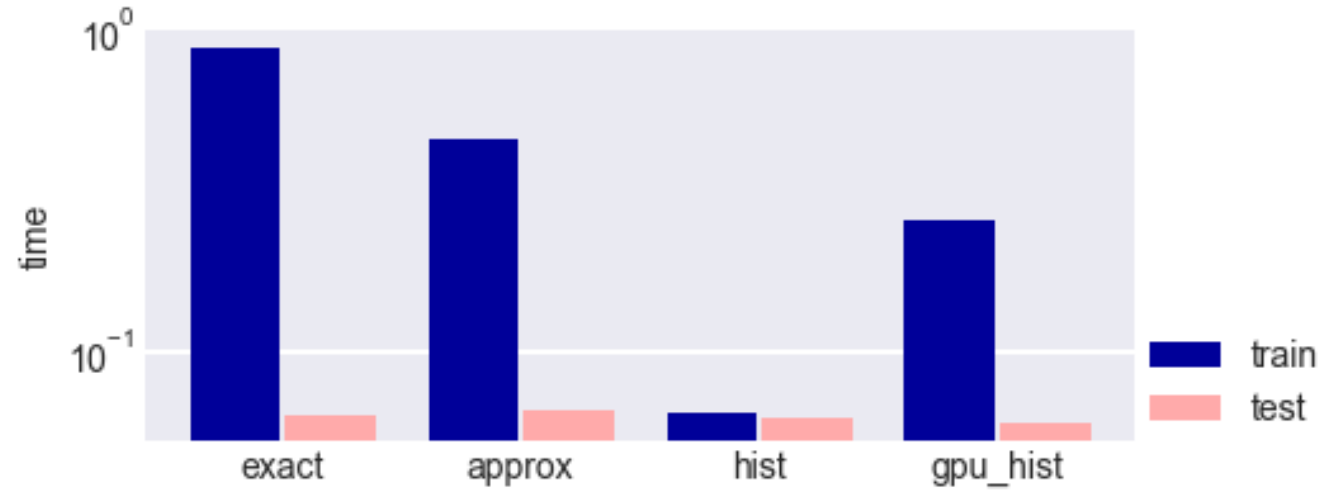
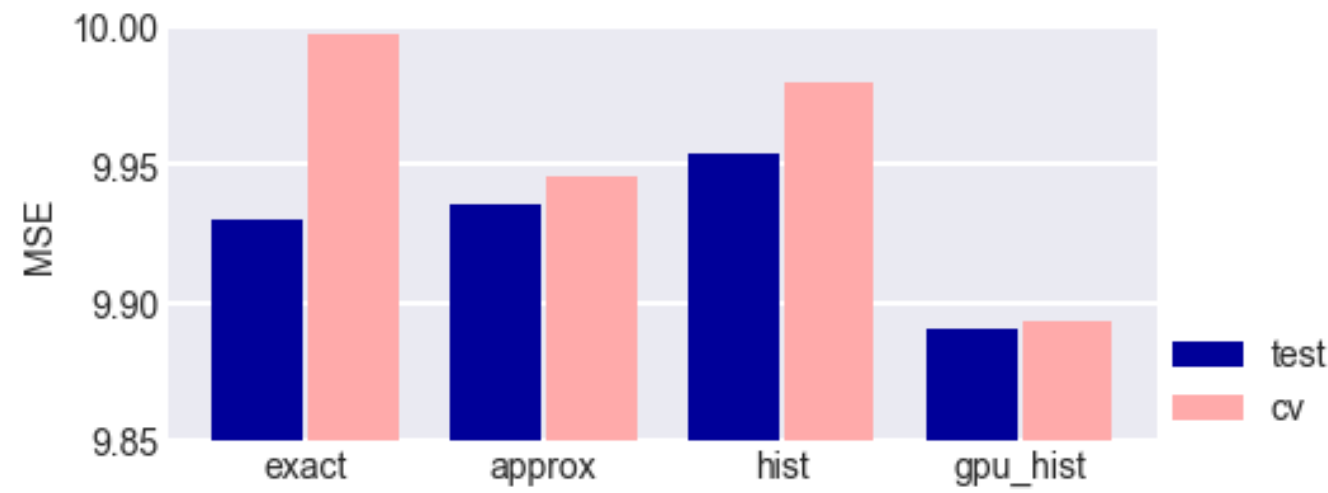
$$a_n(x) = \sum_{t=1}^n b_t(x) + \eta b_{t+1}(x)$$

<http://proceedings.mlr.press/v38/korlakaivinayak15.pdf>

LGBM: разные варианты



XGBoost: разные варианты



Минутка кода

```
from lightgbm import LGBMRegressor
model = LGBMRegressor()
model = LGBMRegressor(boosting_type='dart')
model = LGBMRegressor(boosting_type='goss')
model = LGBMRegressor(boosting_type='rf', subsample_freq=1, subsample=0.75)
```

Без subsample_freq не берутся подвыборки!

```
model = XGBRegressor()
model = XGBRegressor(tree_method='approx')
model = XGBRegressor(tree_method='hist')
model = XGBRegressor(tree_method='gpu_hist')
```

Как правильно строить RF

```
params = {'colsample_bynode': 0.8, 'learning_rate': 1,
          'max_depth': 5, # глубину м.б. надо сделать больше
          'num_parallel_tree': 100, # именно это число деревьев в RF
          'objective': 'binary:logistic',
          'subsample': 0.8, 'tree_method': 'gpu_hist'}

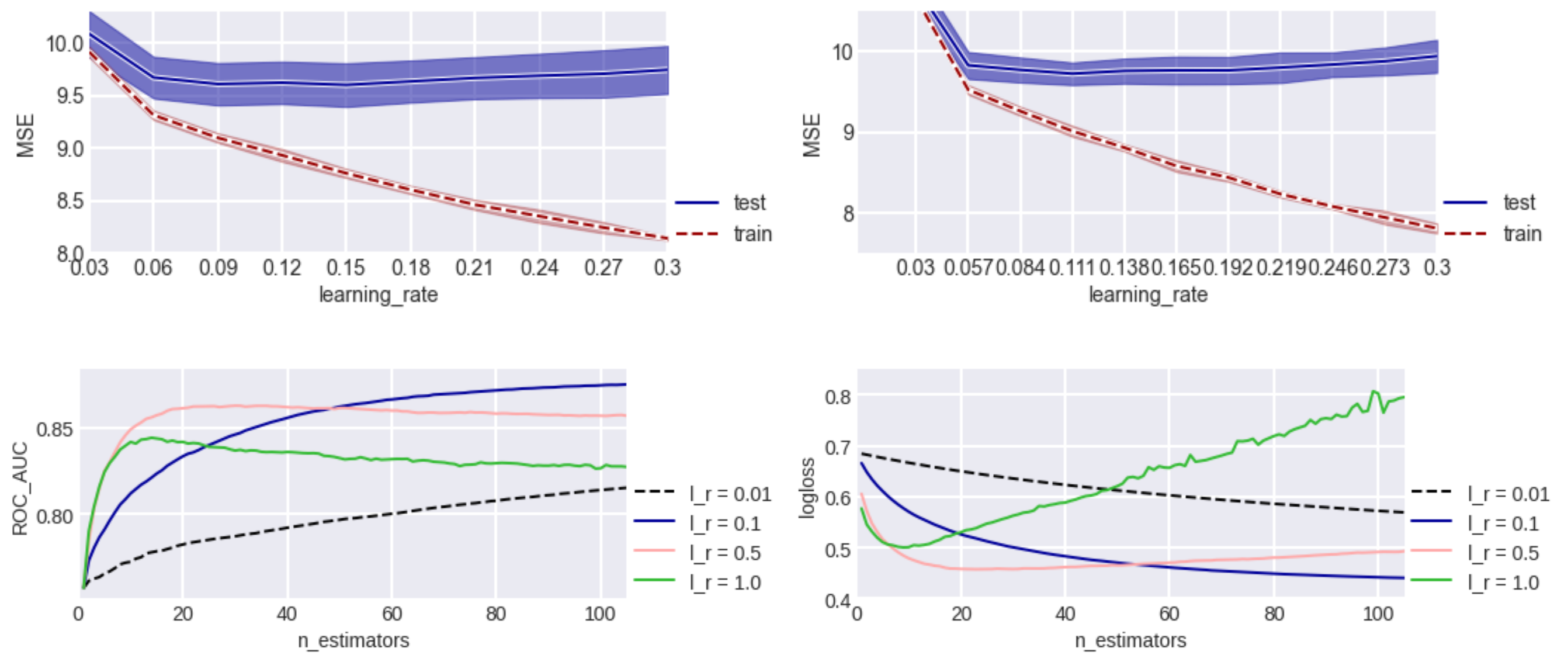
bst = train(params, dmatrix, num_boost_round=1) # а вот бустить не надо!
```


Параметры градиентного бустинга: основные

learning_rate / eta	– темп (скорость) обучения
n_estimators / num_iterations / iterations	– число итераций бустинга (базовых алгоритмов)
num_parallel_tree	– для режима RF

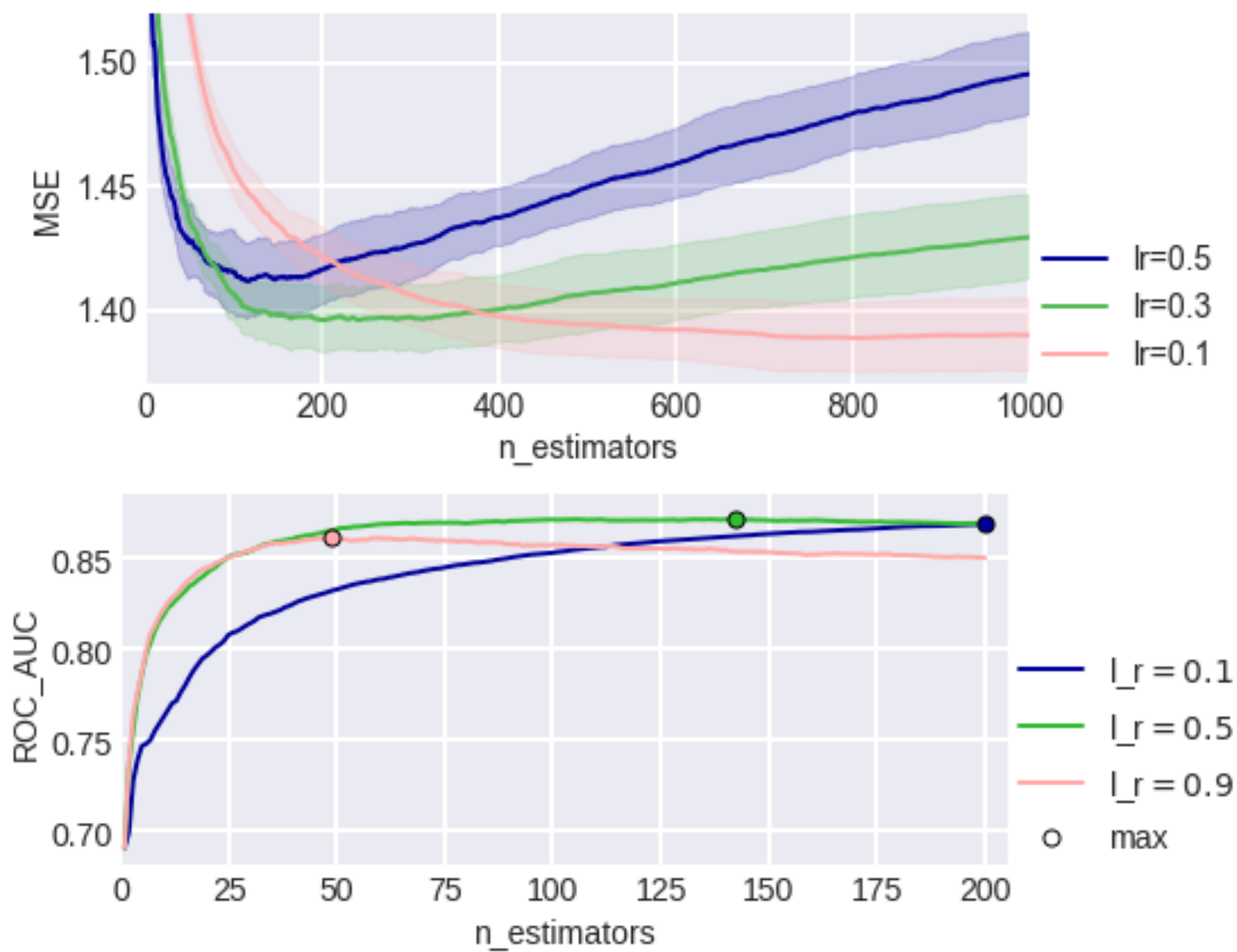
categorical_feature	– какие признаки категориальные
---------------------	---------------------------------

Темп обучения



пример малого, среднего и большого темпов

Темп обучения и число базовых алгоритмов



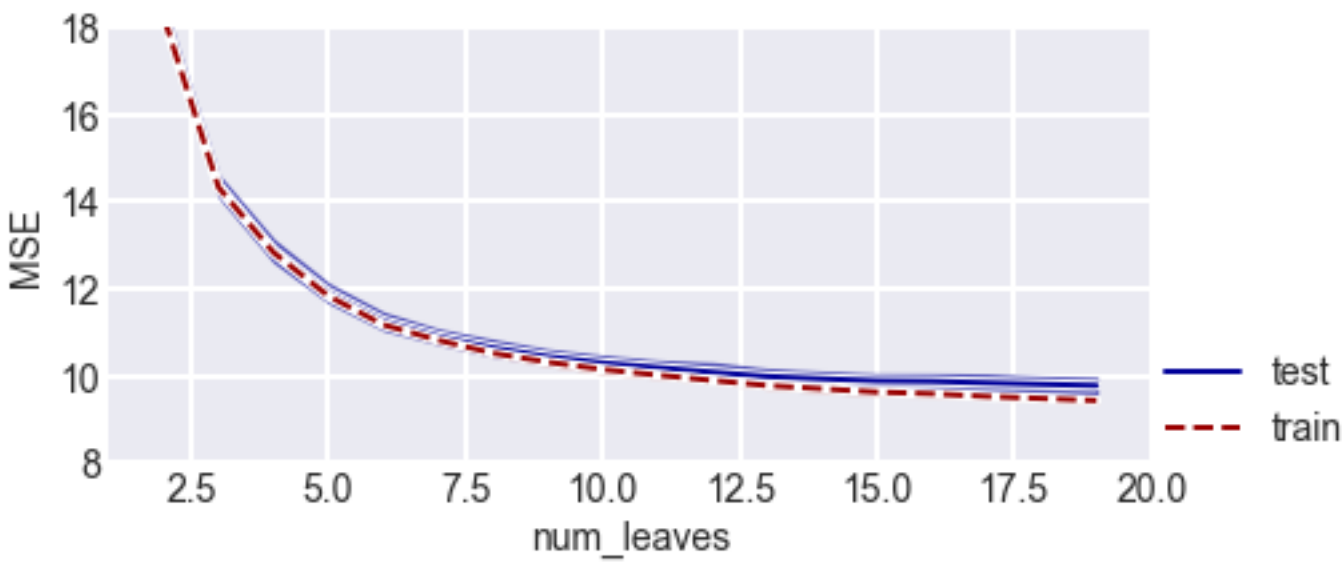
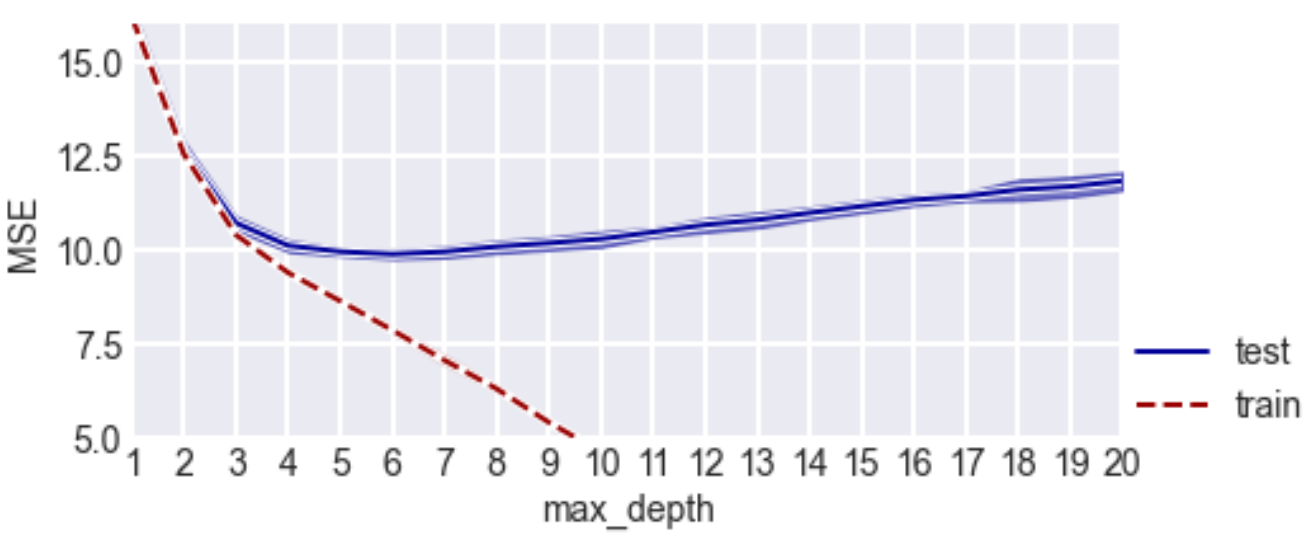
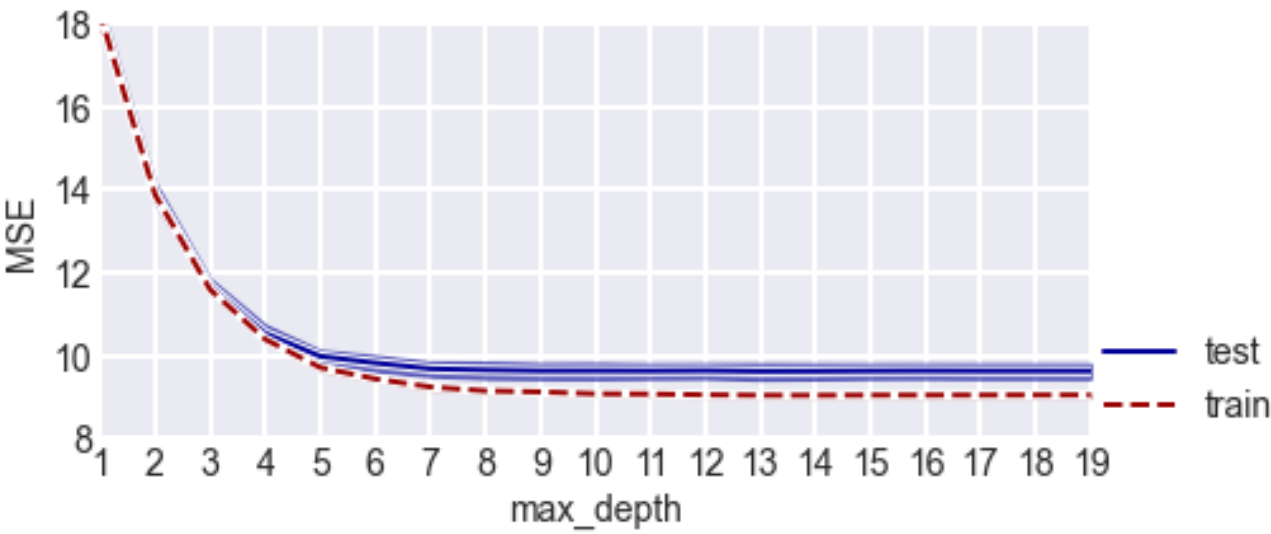
Нет логики «чем больше деревьев, тем лучше»

Нет логики «уменьшили темп в 2 раза – число деревьев надо увеличить в 2 раза»!

Параметры градиентного бустинга: сложность

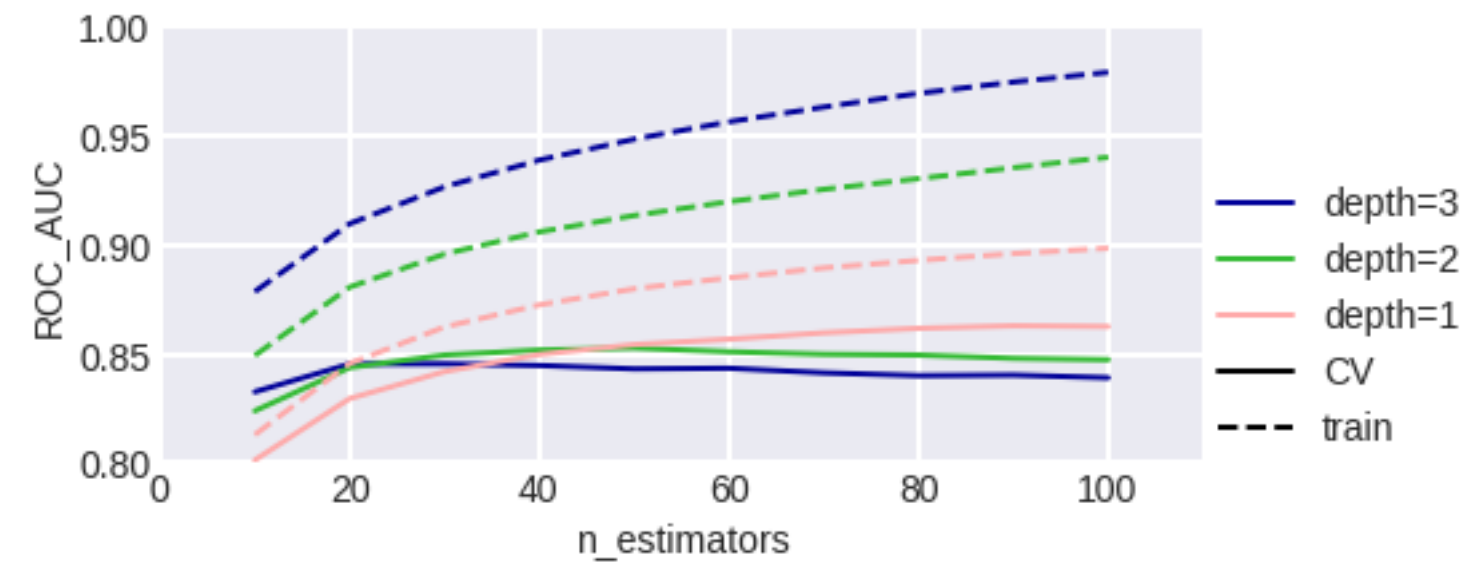
max_depth	– максимальная глубина
gamma	– порог на уменьшение функции ошибки при расщеплении в дереве
min_child_weight	– минимальная сумма весов объектов в потомках
max_delta_step	– порог на изменение весов
max_leaves / num_leaves	– максимальное число вершин в дереве
min_split_gain / min_gain_to_split	– порог на изменение loss-функции
min_child_samples / min_data_in_leaf	– минимальное число объектов в листьях
/ min_sum_hessian_in_leaf	– минимальная сумма весов объектов в листе, минимальное число объектов, при котором делается расщепление

Сложность деревьев

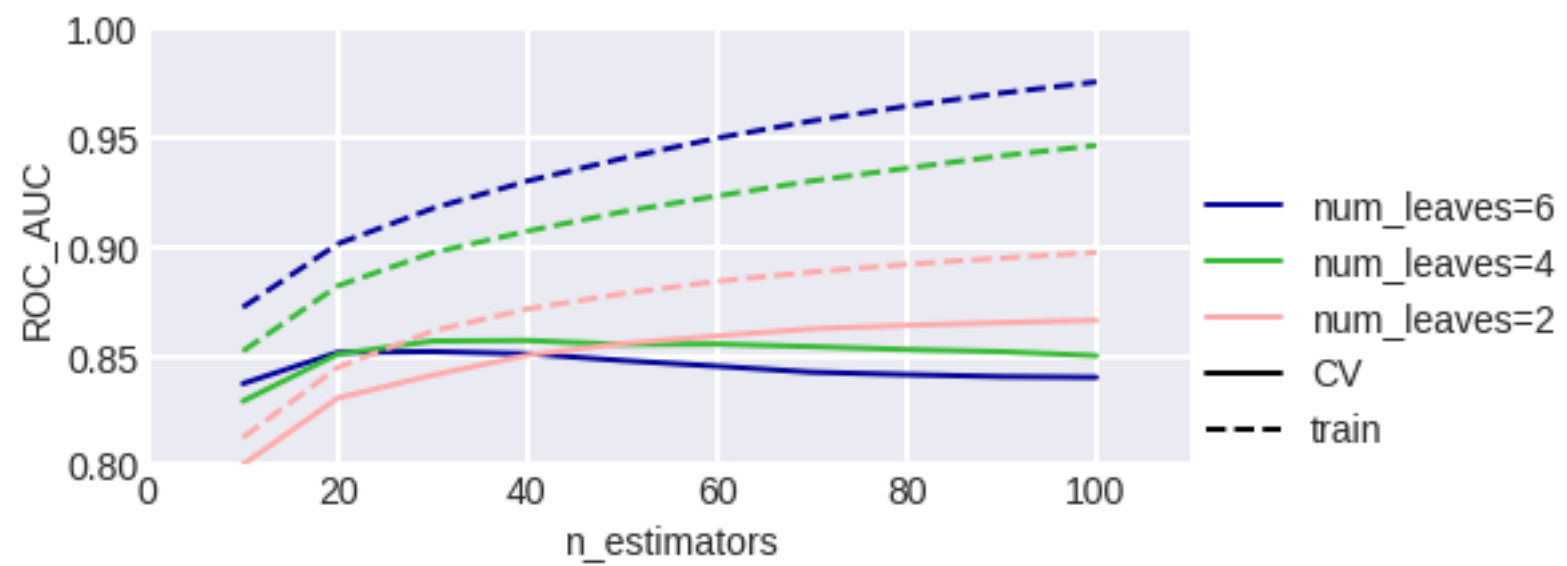


Сложность деревьев

scikit-learn

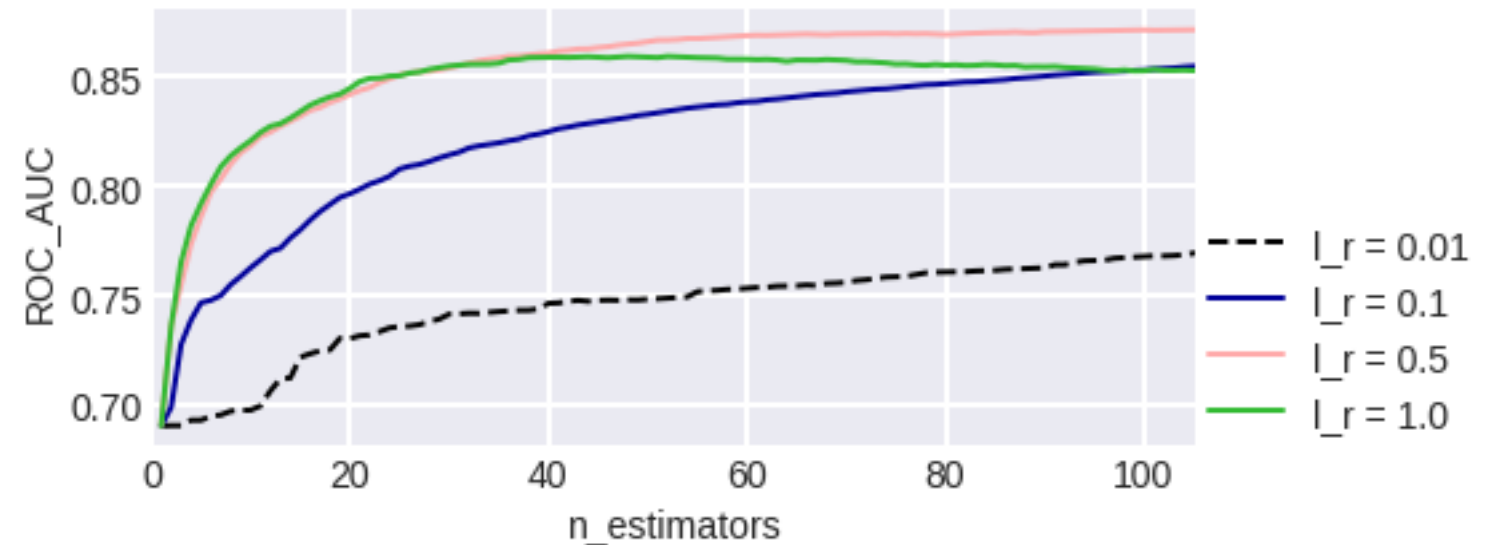
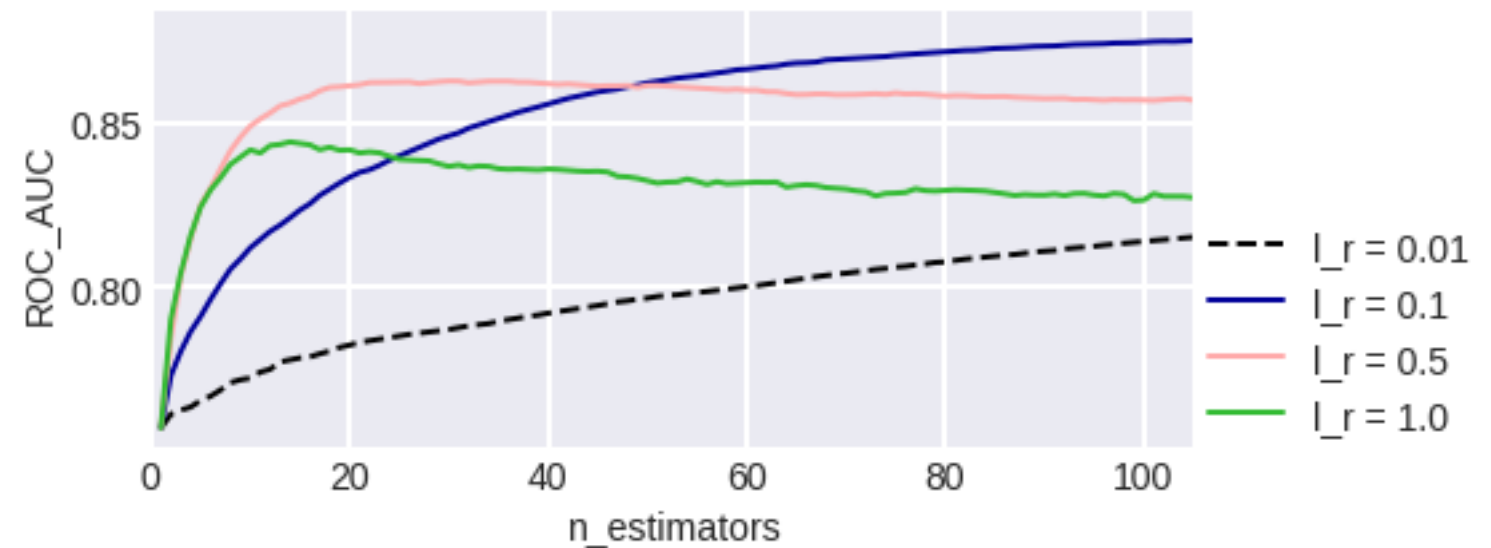


lightgbm



Для разной глубины – разное оптимальное число деревьев

Сложность деревьев

num_leaves = 3**num_leaves = 10****верно ли, что для малых деревьев нет большого темпа?**

Ограничение на расщепления / листья

**Здесь могут быть большие оптимальные значения (10 – 50 – 1000),
но параметры менее значимые, чем другие...**

Что такое веса

min_child_weight – «minimum sum of instance weight (**hessian**) needed in a child»

max_delta_step – «Maximum delta step we allow each tree's weight estimation to be»

Пример:

$$L_i = \frac{1}{2}(y_i - a_i)^2$$

$$h_i = L_i'' = 1$$

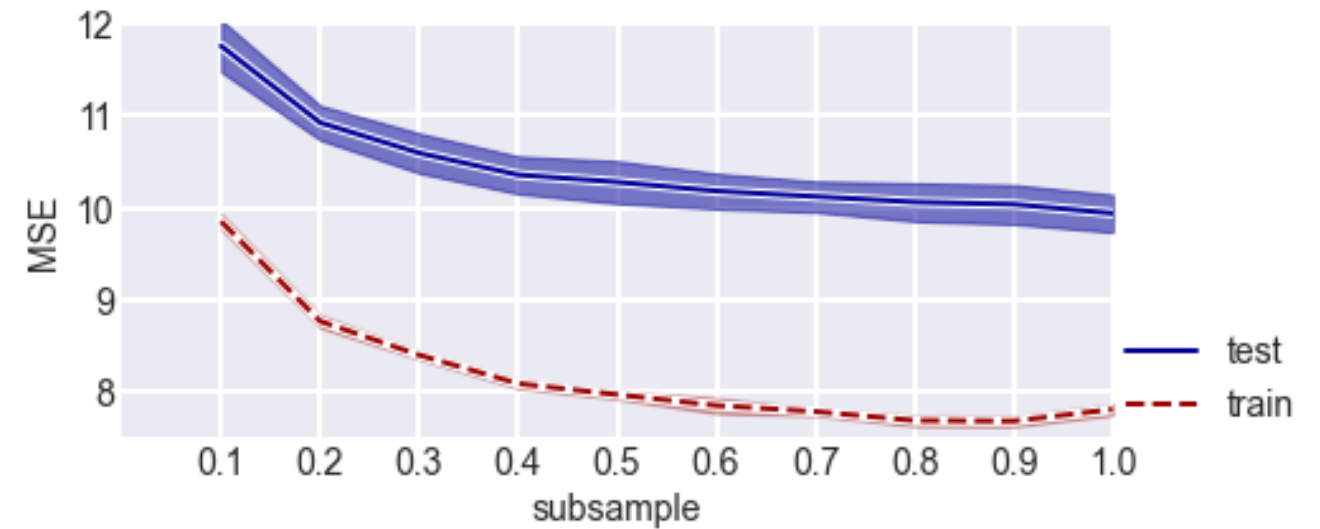
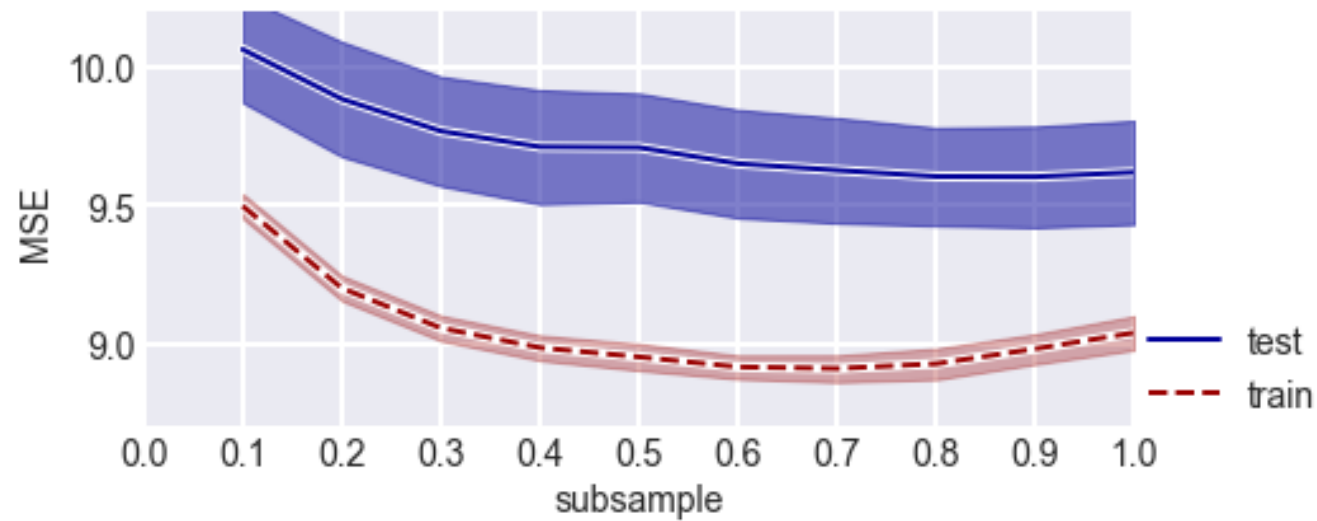
– это как бы вес одного объекта

Параметры градиентного бустинга: формирование подвыборок

<code>subsample / bagging_fraction</code>	– какую часть объектов обучения использовать для построения одного дерева
<code>colsample_bytree/ feature_fraction / rsm</code>	– какую часть признаков использовать для построения одного дерева
<code>colsample_bylevel</code>	– какую часть признаков использовать для построения расщепления в уровне
<code>colsample_bynode</code>	– какую часть признаков использовать для построения расщепления в вершине
<code>scale_pos_weight</code>	– для сбалансирования позитивных и негативных весов
<code>/ class_weight</code>	– веса классов
<code>// bootstrap_type</code>	– тип бутстрепа (для Bayesian bootstrap есть <code>bagging_temperature</code>)

`/ subsample_freq (int, optional (default=0))` – частота взятия подвыборок
`// sampling_frequency`

Подвыборки

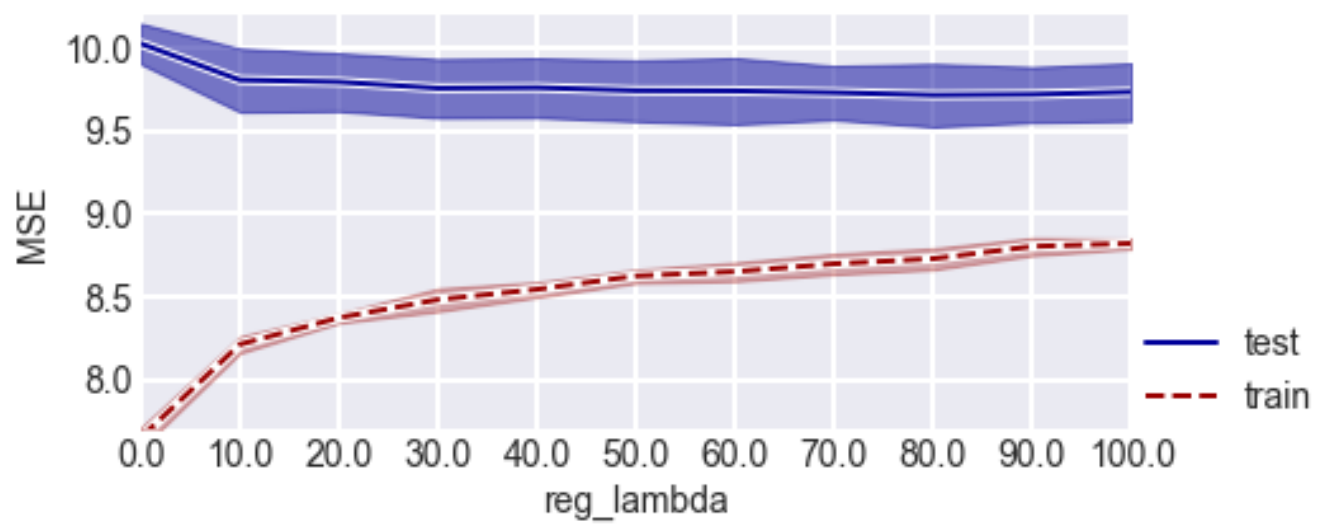
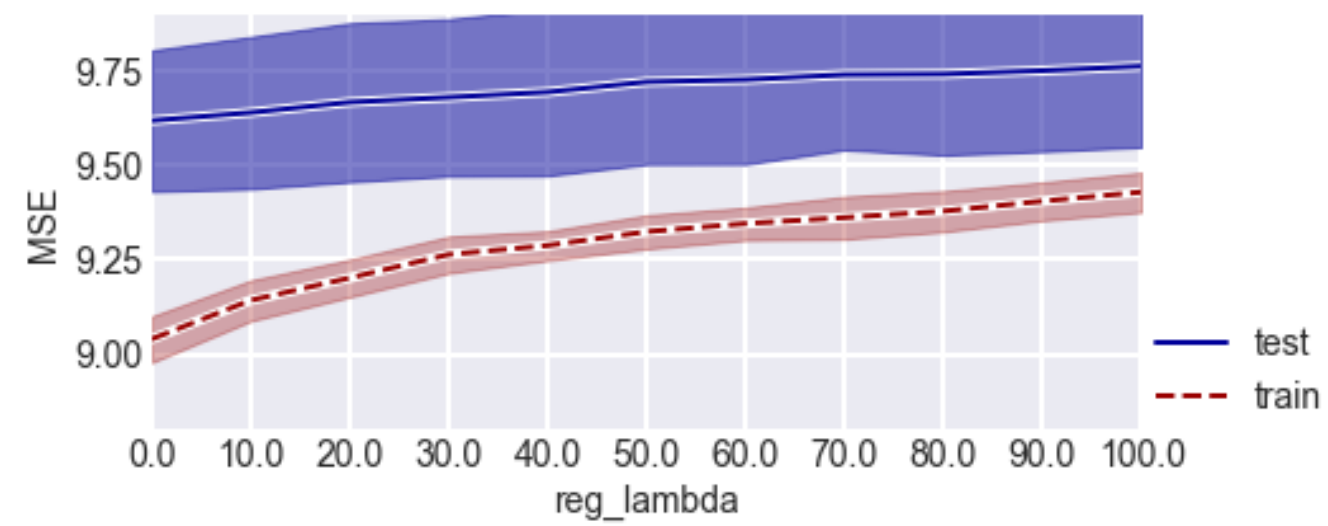
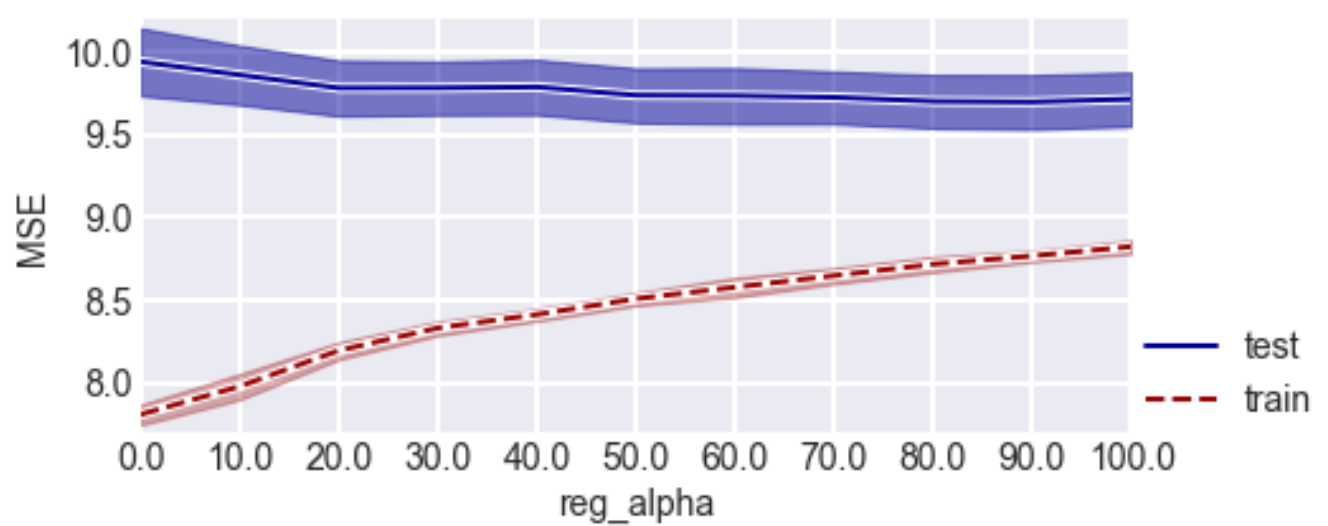
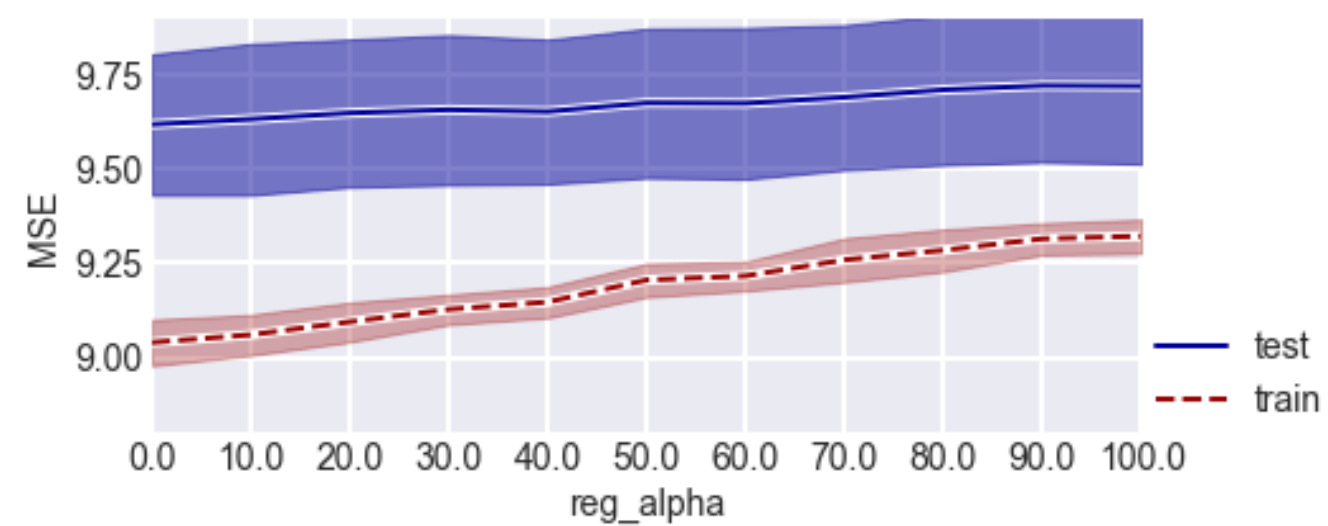


**Обычно больше – лучше (в XGBoost это не всегда так)
Если берём подвыборки бустинги «хуже суммируются»**

Параметры градиентного бустинга: регуляризация

reg_alpha / alpha	– коэффициент L1-регуляризации
reg_lambda / lambda // l2_leaf_reg	– коэффициент L2-регуляризации
random_strength	– шум при оценки сплитов (CatBoost)
border_count	– число сплитов для вещественных признаков (CatBoost)
ctr_border_count	– число сплитов для категориальных признаков (CatBoost)

Регуляризация



Параметры градиентного бустинга: остальные

<code>verbosity / silent</code>	– вывод информации при обучении
<code>n_jobs / nthread / num_threads</code>	– число используемых потоков
<code>random_state</code>	– инициализация генератора псевдослучайных чисел
<code>missing</code>	– что обозначает пропуски
<code>importance_type</code>	– как вычислять важность «weight» (как часто признак выбирался), «total_gain» (по той функции \uparrow), «gain» ($\text{total_gain} / \text{weight}$), «total_cover» (за разделение скольких объектов отвечает), «cover» ($\text{total_cover} / \text{weight}$).
<code>/ subsample_for_bin</code>	– число объектов для бинов

Параметры градиентного бустинга: fit

early_stopping_round (fit)	– если на отложенном контроле заданная функция ошибки не уменьшается такое число итераций, обучение останавливается
sample_weight	– веса объектов
eval_metric	– метрика качества
callbacks	– какие функции вызывать после каждой итерации

- CPU / GPU
- хранить модель в ОЗУ

Early stopping (через fit)

```
params = {'objective': 'binary', 'reg_lambda': 0.0001, 'reg_alpha': 0.0001,  
          'num_leaves': 7, 'learning_rate': 0.1, 'colsample_bytree': 0.75,  
          }  
  
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.33, random_state=42)  
  
model = lgb.LGBMClassifier(n_jobs=-1, n_estimators=1000, **params, metric='auc') # использовать auc  
  
model.fit(X_train, y_train,  
          eval_set=[(X_test, y_test)],  
          early_stopping_rounds=200,  
          verbose=50)
```

Training until validation scores don't improve for 200 rounds

```
[50]    valid_0's auc: 0.685867  
[100]   valid_0's auc: 0.687435  
[150]   valid_0's auc: 0.687931  
[200]   valid_0's auc: 0.688472  
[250]   valid_0's auc: 0.686139  
[300]   valid_0's auc: 0.685785  
[350]   valid_0's auc: 0.684596
```

Early stopping, best iteration is:

```
[198]   valid_0's auc: 0.688558
```


Early stopping (через train)

```
def accuracy(preds, train_data):
    labels = train_data.get_label()
    preds = 1. / (1. + np.exp(-preds))
    return 'accuracy', np.mean(labels == (preds > 0.5)), True

def rmsle(y_true, train_data):
    y_pred = train_data.get_label()
    return 'RMSLE', np.sqrt(np.mean(np.power(np.log1p(y_pred) - np.log1p(y_true), 2))), False

bst = lgb.train(params,
                train_set=lgb.Dataset(X_train, y_train),
                num_boost_round=1000,
                valid_sets=[lgb.Dataset(X_test, y_test), lgb.Dataset(X_train, y_train)],
                valid_names=['A', 'B'], # имена датасетов для удобства
                init_model = tmp, # тут м.б. booster для продолжения обучения МОЖНО задать текстовый файл - см. ниже
                early_stopping_rounds=50,
                feval=lambda a, b: [accuracy(a,b), rmsle(a,b)], # несколько самописных функций
                verbose_eval=25)

Training until validation scores don't improve for 50 rounds
[125]   A's binary_logloss: 0.106202 A's accuracy: 0.0227515 A's RMSLE: 0.10447
        B's binary_logloss: 0.0928391 B's accuracy: 0.0218412 B's RMSLE: 0.0983035
[150]   A's binary_logloss: 0.106231 A's accuracy: 0.0227515 A's RMSLE: 0.104485
        B's binary_logloss: 0.0921399 B's accuracy: 0.0218412 B's RMSLE: 0.0980522
Early stopping, best iteration is:
[111]   A's binary_logloss: 0.106148 A's accuracy: 0.0227515 A's RMSLE: 0.104426
        B's binary_logloss: 0.0932007 B's accuracy: 0.0218412 B's RMSLE: 0.0984656

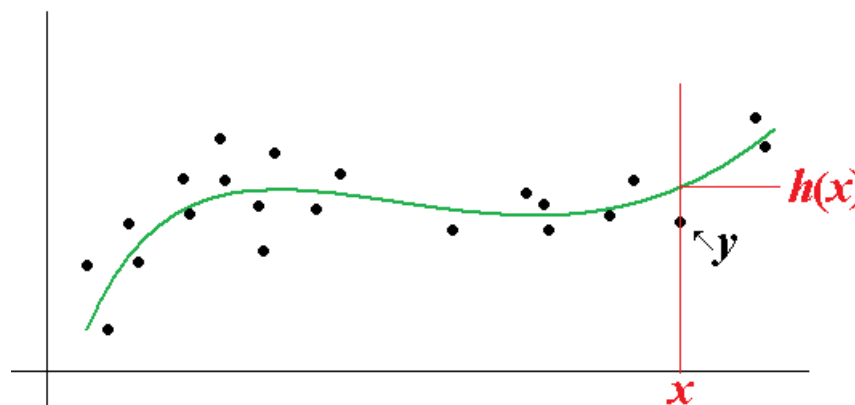
bst.save_model('model.txt') # см. в файле прописаны деревья
```

GBM: Концепция чёрного ящика

```
model <- gbm(is_client_cancel~. , # название целевой переменной
T, # as.data.frame
distribution="gaussian", # распределение... лучше всего gaussian
n.trees=ntrees, # число деревьев (лучше больше, а потом выбрать)
shrinkage=0.07, # скорость сходимости
verbose=TRUE, # вывод сообщений
interaction.depth=12 # сложность модели
)
```

```
class sklearn.ensemble.GradientBoostingClassifier
(loss='deviance', # в классификации - логистическая регрессия или AdaBoost
learning_rate=0.1, # скорость сходимости
n_estimators=100, # число деревьев
subsample=1.0,
min_samples_split=2,
min_samples_leaf=1,
min_weight_fraction_leaf=0.0,
max_depth=3, # глубина
max_features=None) # сколько признаков смотреть для расщепления
```

Что означает «распределение»



пусть ошибки распределены по нормальному закону

$$p(y | x) = \text{const} \cdot e^{-\frac{(y-h(x))^2}{2\sigma^2}}$$

метод максимального правдоподобия

$$\prod_i p(y_i | x_i) \sim \prod_i e^{-\frac{(y_i-h(x_i))^2}{2\sigma^2}} \rightarrow \max$$
$$\text{const} \cdot \sum_i (y_i - h(x_i))^2 \rightarrow \min$$

Что означает «распределение»

пусть ошибки ~ распределение Лапласа

$$p(y | x) = \text{const} \cdot e^{-\alpha|y-h(x)|}$$

метод максимального правдоподобия

$$\prod_i p(y_i | x_i) \sim \prod_i e^{-\alpha|y_i-h(x_i)|} \rightarrow \max$$

это эквивалентно минимизации такой ошибки

$$\text{const} \cdot \sum_i |y_i - h(x_i)| \rightarrow \min$$

Встроенные способы контроля

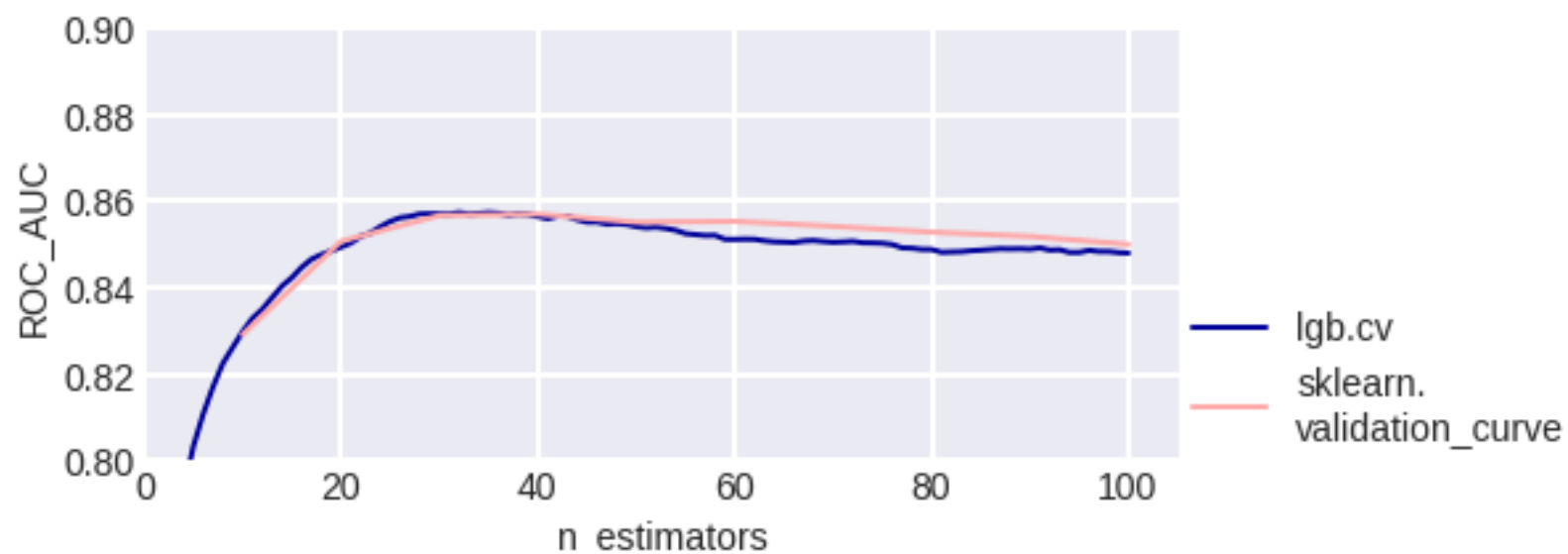
встроенный

```
params = {'objective': 'binary', 'learning_rate': 0.9,  
          'num_leaves': 4, 'n_estimators': 100}  
tmp = lgb.cv(params,  
             train_set=lgb.Dataset(data, y_data),  
             metrics=['auc', 'binary_logloss', 'rmse'])
```

универсальный

```
cv = KFold(n_splits=n_splits, shuffle=shuffle, random_state=random_state)  
train_errors, test_errors = validation_curve(estimator,  
                                             data, y_data,  
                                             param_name=param_name,  
                                             param_range=pars,  
                                             cv=cv,  
                                             scoring=scoring,  
                                             n_jobs=-1)
```

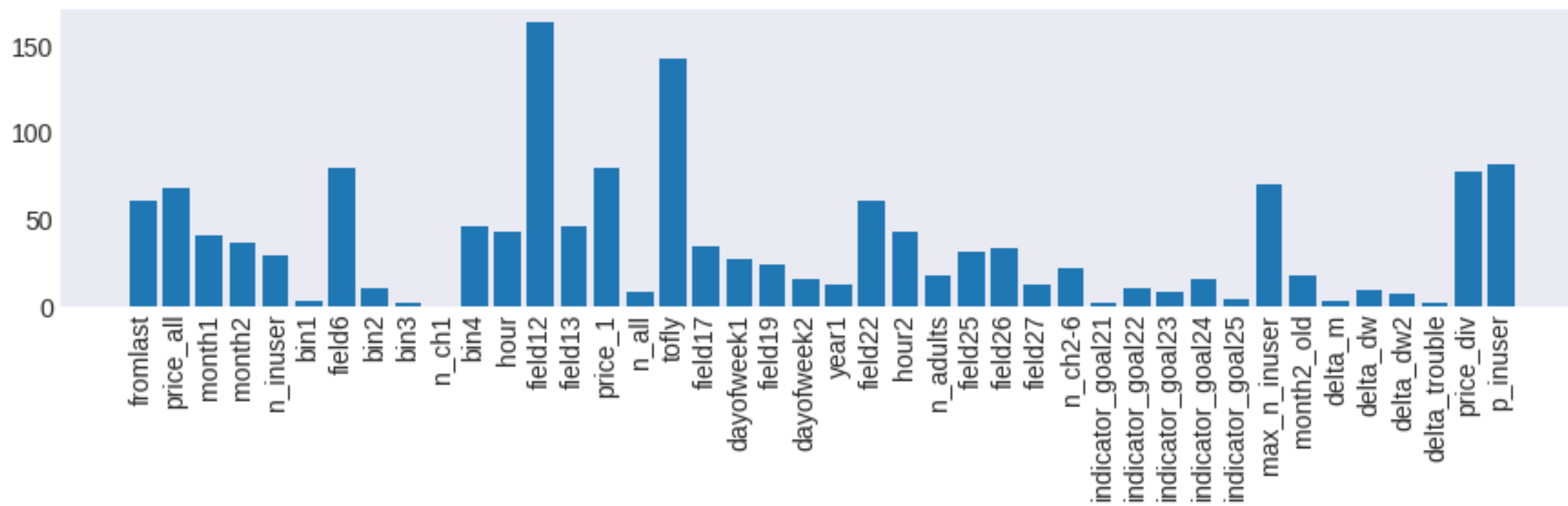
Встроенные способы контроля



существенно быстрее 2 мин – 2 сек

- не перестраивают ансамбль с начала
- нечестный контроль – упрощённый способ выбора порогов (хитрость!)
 - сразу получаем с шагом 1
 - результаты похожи

Важности признаков – Задача OneTwoTrip (сейчас об этом не будем)



```
model = lgb.LGBMClassifier(learning_rate=0.05,  
                             num_leaves=16,  
                             subsample=1.0,  
                             n_estimators=100,  
                             random_state=1,  
                             reg_alpha=0.01,  
                             reg_lambda=0.0001)
```

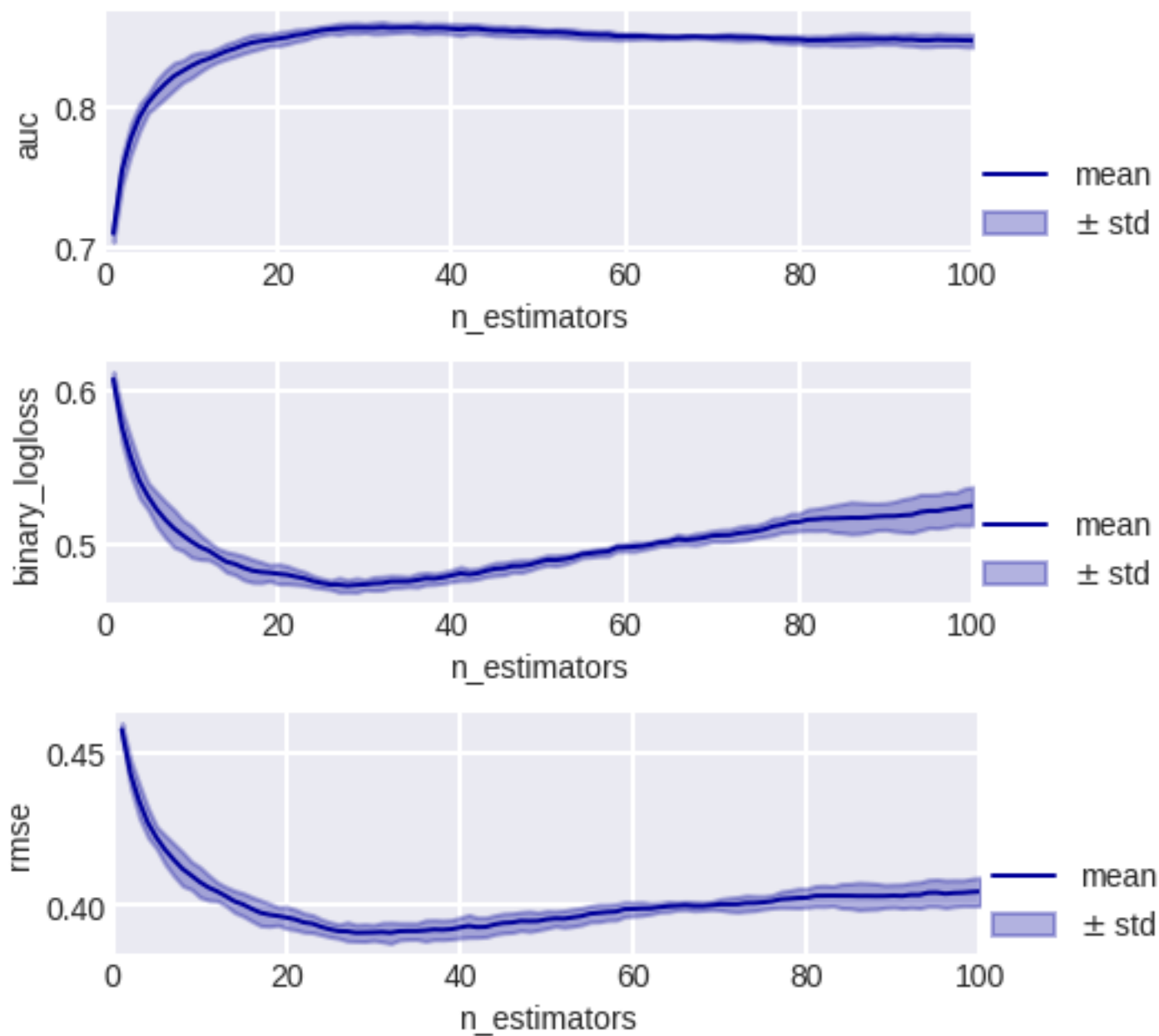
```
model.fit(data.values, y)  
model.feature_importances_
```

Перебор параметров

```
params = {'learning_rate': [0.05, 0.1, 0.2],  
          'subsample': [0.5, 0.75, 1.0],  
          'max_depth': [1, 2, 3, 4],  
          'reg_alpha': [0, 0.0001, 0.01],  
          'reg_lambda': [0, 0.0001, 0.01],  
          }  
  
model = xgb.XGBRFClassifier()  
  
rs = RandomizedSearchCV(model, params, n_iter=100,  
                        scoring='roc_auc',  
                        n_jobs=-1, cv=gss)  
  
rs.fit(data, y, groups=groups)
```

тут есть грубая ошибка... какая?

Советы по обучению: мониторьте разные метрики качества



Советы по обучению: темп обучения `learning_rate`

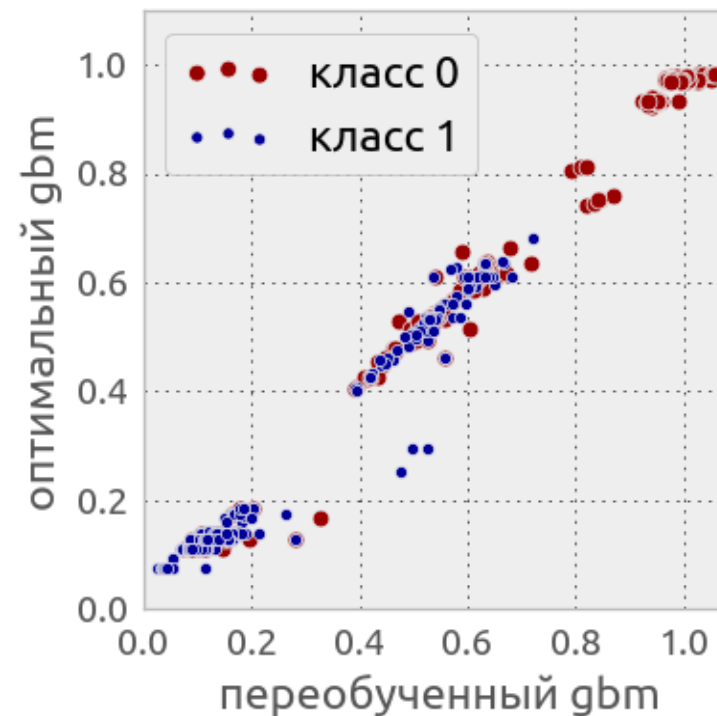
для разной сложности (ex: глубина)

обычно глубина 3–6 – смещённые (high bias), разброс ниже, чем в глубоких

- смещение как раз устраняется бустингом
- модель не должна переобучаться \Rightarrow простая
 - быстрее строить
- зафиксируйте (достаточно большое) число деревьев в ансамбле
- настройте `learning_rate` (для этого числа деревьев)
 - выберите оптимальную глубину
 - настраивайте другие параметры

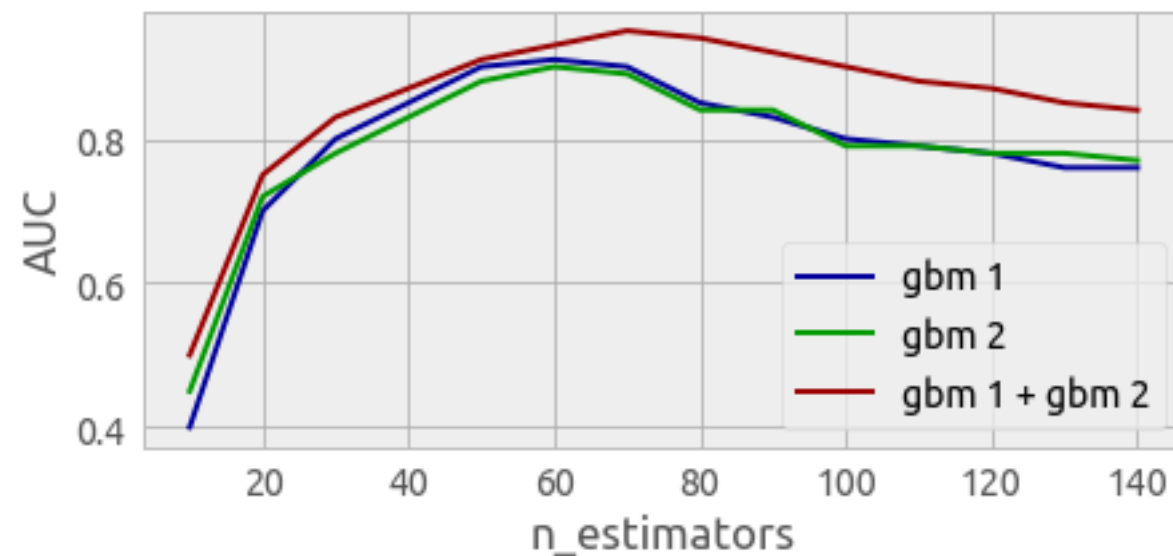
Есть стратегия – сделать очень маленький темп и очень много деревьев
(но для настройки других параметров не годится)

Совет: постобработка ответов



Значения gbm могут выходить за пределы отрезка!
за пределы какого-то компактного множества в задаче регрессии

**Вообще говоря, не важно,
как их вернуть обратно...**

Совет: сумма бустингов

**Качество может улучшиться,
но оптимальные параметры меняются!**

Приложение GBM: задача скоринга (TKS)

tcs_customer_id	bureau_cd	bki_request_date	inf_confirm_date	type	status	open_date	final_pmt_date	fact_close_date	credit_limit	currency	outstanding	next_pmt	curr_balar
1	2	12Aug2011	20Jul2011	99	00	13May2011	11May2012		28967	RUB	24606,00000	2743,00000	
1	1	12Aug2011	18Feb2009	99	13	27Feb2008	26Feb2009	26Feb2009	30000	RUB	0,00000		
1	1	12Aug2011	21Apr2009	99	13	28Jun2007	30Jun2008	20Apr2009	19421	RUB	0,00000		
1	1	12Aug2011	18Aug2009	9	13	15Jul2008	17Aug2009	17Aug2009	11858	RUB	0,00000		
1	1	12Aug2011	06Sep2010	99	13	09May2009	10May2010	08Sep2010	19691	RUB	0,00000		
1	1	12Aug2011	28Jul2011	7	52	07Sep2010	07Sep2040		10000	RUB			
1	1	12Aug2011	01Aug2011	9	00	31Aug2010	31Aug2015		169000	RUB			
1	1	12Aug2011	03Aug2011	9	00	04Mar2009	03Mar2014		300000	RUB			
1	3	12Aug2011	09Jul2008	9	00	28Jun2007	30Jun2008		19421	RUB	1761,00000		198
1	3	12Aug2011	19Sep2008	9	00	27Feb2008	26Feb2009		30000	RUB	15517,00000		163
1	3	12Aug2011	14Sep2010	9	13	09May2009	10May2010	06Sep2010	19691	RUB	0,00000		
1	3	12Aug2011	11Jul2011	9	00	31Aug2010	31Aug2015		169000	RUB		0,00000	433

Решение = GBM + RF + Линейная регрессия

Приложение GBM: задача скоринга (TKS)

Name	Description	Type
TCS_CUSTOMER_ID	Идентификатор клиента	ID
BUREAU_CD	Код бюро, из которого получен счет	numeric
BKI_REQUEST_DATE	Дата, в которую был сделан запрос в бюро	date
CURRENCY	Валюта договора (ISO буквенный код валюты)	string
RELATIONSHIP	Тип отношения к договору	string
	1 - Физическое лицо	
	2 - Дополнительная карта/Авторизованный пользователь	
	4 - Совместный	
	5 - Поручитель	
	9 - Юридическое лицо	
OPEN_DATE	Дата открытия договора	date
FINAL_PMT_DATE	Дата финального платежа (плановая)	date
TYPE	Код типа договора	string
	1 – Кредит на автомобиль	
	4 – Лизинг. Срочные платежи за наем/пользование транспортным средством, предприятием или оборудованием и т.п.	
	6 – Ипотека – ссудные счета, имеющие отношение к домам, квартирам и прочей недвижимости. Ссуда выплачивается циклично согласно договоренности до тех пор, пока она не будет полностью выплачена или возобновлена.	
	7 – Кредитная карта	
	9 – Потребительский кредит	
	10 – Кредит на развитие бизнеса	
	11 – Кредит на пополнение оборотных средств	
	12 – Кредит на покупку оборудования	
	13 – Кредит на строительство недвижимости	
	14 – Кредит на покупку акций (например, маржинальное кредитование)	
	99 – Другой	
PMT_STRING_84M	Дисциплина (своевременность) платежей. Строка составляется из кодов состояний счета на моменты передачи банком данных по счету в бюро, первый символ - состояние на дату PMT_STRING_START, далее последовательно в порядке убывания дат.	string
	0 – Новый, оценка невозможна	
	X – Нет информации	
	1 – Оплата без просрочек	
	A – Просрочка от 1 до 29 дней	

Приложение GBM: предсказание правильности ответов студентов на вопросы тестов

Разработать алгоритм,
который предсказывает правильность ответа
на вопросы теста.

Зачем?

для рекомендательной системы (алгоритм решает за студента тест и сообщает ему
«потенциально неприятные для него» вопросы).

GMAT, SAT, ACT

Победитель – LibFM

#	Team Name	\$5,000 • 241 teams	Score	Entries
1	Steffen Rendle *		0.24598	16
2	Alexander D'yakonov *		0.24729	38
3	ekla *		0.24745	87
4	PlanetThanet & Birutas		0.24772	51

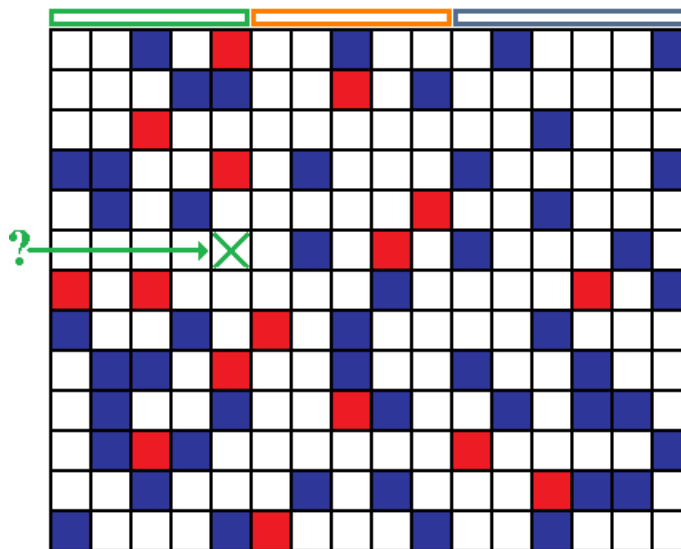
Приложение GBM: идея использование GBM для предсказания правильности ответов

Классическое решение

- контекстная рекомендация
- коллаборативная фильтрация

Наше решение

**свести задачу о рекомендациях
к задаче классификации (регрессии)**



пара «студент–вопрос» ~ признаковое описание
генерация кучи признаков

Приложение GBM: примеры признаков

Пусть ответы студента:

correct, incorrect, correct, correct, correct, incorrect

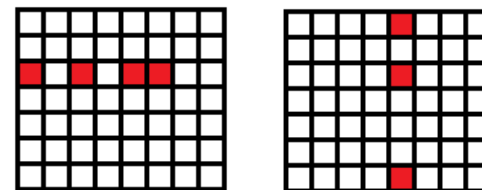
$$\mathbf{IQ} \sim \frac{+1-1+1+1+1-1}{6}$$

$$\mathbf{weighted\ IQ} \sim \frac{+1w_1 - 1w_2 + 1w_3 + 1w_4 + 1w_5 - 1w_6}{w_1 + w_2 + w_3 + w_4 + w_5 + w_6}$$

ниже – как вводились веса

Аналогично, признаки вопросов... «сложность вопроса»

(здесь усредняются ответы на данный вопрос)



Приложение GBM: веса признаков

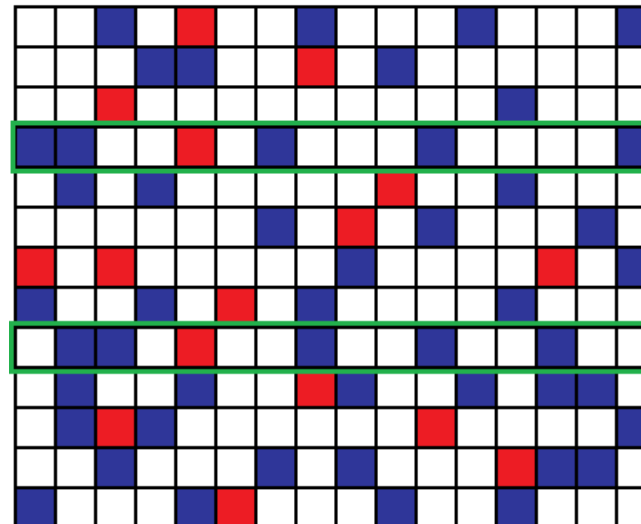
1) веса измеряют «похожесть времени ответов для вопросов»

$$w_j = \frac{2}{1 + |t - t_j|^{0.3}} - 1 \text{ или } w_j = 1 - \sqrt{|t - t_j|}$$

t_j – время ответа на j-й вопрос,

t – время ответа на этот вопрос.

2) веса – корреляция столбцов матрицы «студент–ответ»

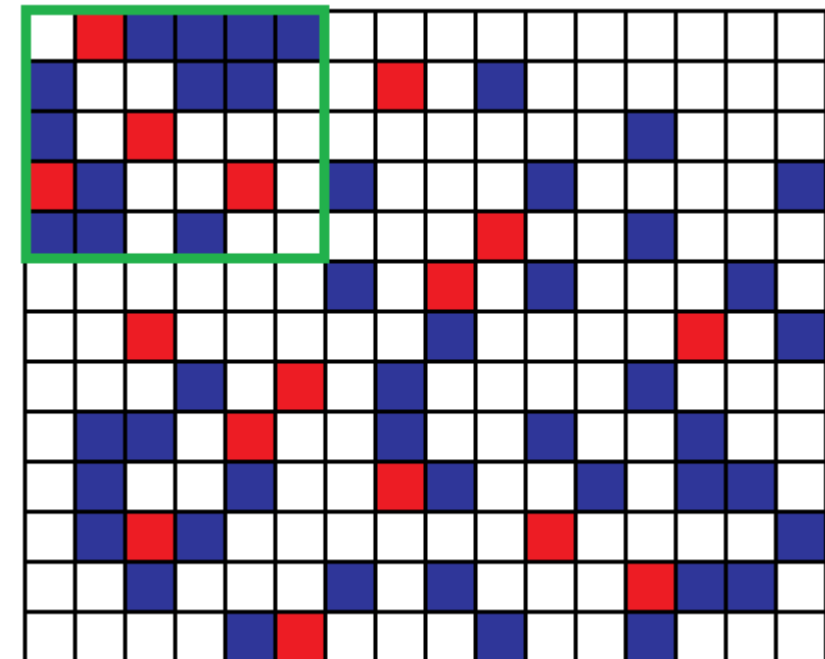
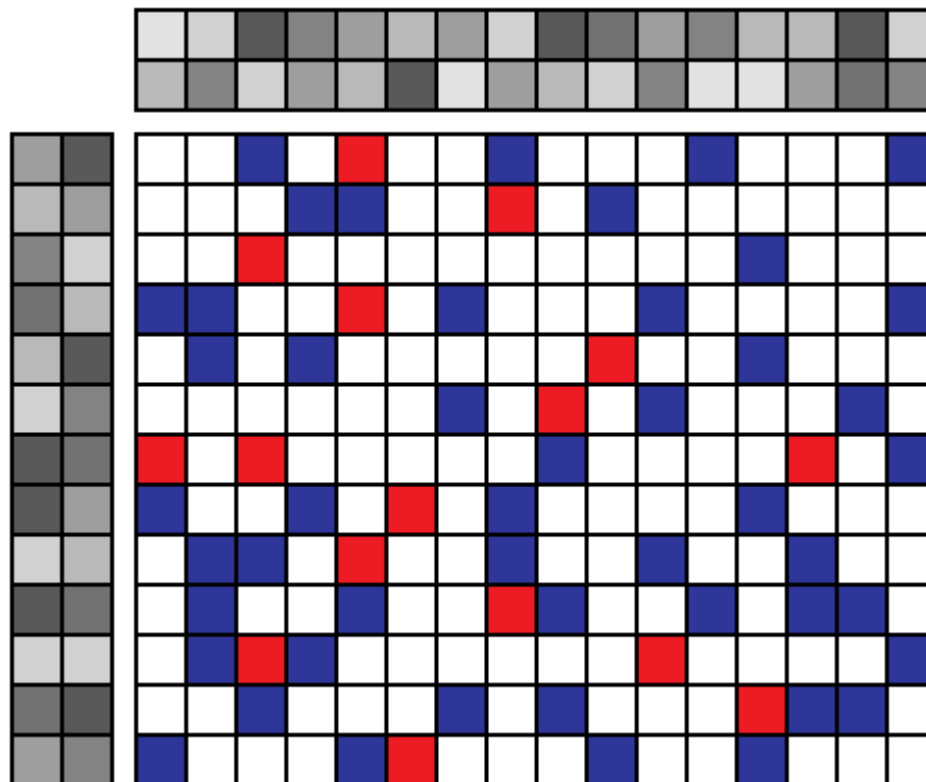


Приложение GBM: веса признаков

3) Простые признаки:

- время ответа
- $1/(\text{число ответов всего})$

4) SVD-признаки (восстановление элементов матрицы с помощью SVD-преобразования) (даже по подматрице)



Приложение GBM: итоговое решение

gbm (бустинг) + glm (линейная модель) + NN (нейросеть, CLOP)

Опять: хорошо «смешиваются» разные алгоритмы...

Как настраивать – чуть позже...

Итог

- **выбрать вид бустинга / критерий расщепления / функцию ошибки «по задаче»**
 - **три самых важных параметра:**
сложность, темп, число деревьев
при разных сложностях (глубина / число листьев)
настроить два остальных **связных параметра**
для настройки можно немного деревьев
- **в продакшене: увеличить число деревьев, взять маленький темп обучения**
 - **использовать сумму нескольких gbm**
проверить, помогает ли это
проверить, как нужно менять параметры для суммы

Литература

A. Natekin, A. Knoll Gradient boosting machines, a tutorial // Front Neurorobot. 2013; 7: 21.
<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/>

все статьи по XGBoost, LightGBM, CatBoost

Сравнения

<https://www.kaggle.com/nholloway/catboost-v-xgboost-v-lightgbm>
<https://medium.com/riskified-technology/xgboost-lightgbm-or-catboost-which-boosting-algorithm-should-i-use-e7fda7bb36bc>

Про параметры

<https://neptune.ai/blog/lightgbm-parameters-guide>