

Лекция 9

Бустинг

Всеволод Викулин

Содержание



1. Что такое бустинг?
2. AdaBoost
3. Градиентный бустинг
4. Современные реализации: xgboost, lightgbm

Часть 1

Что такое бустинг?

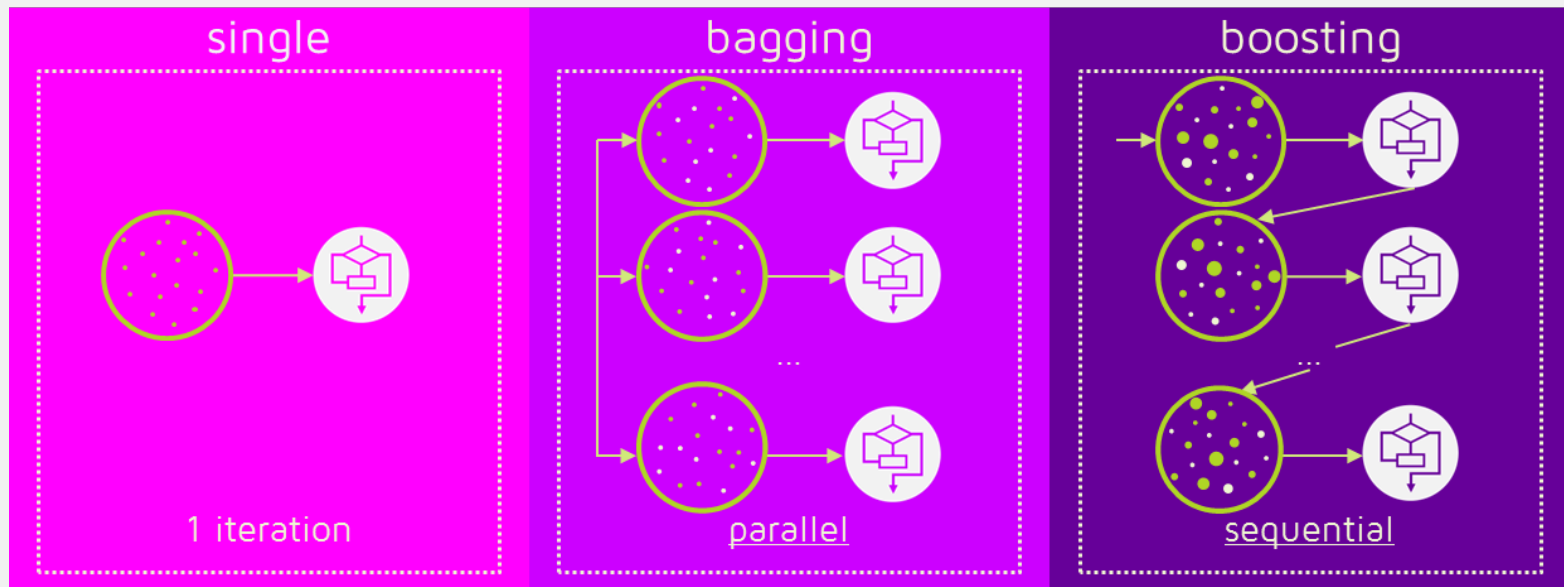


Напоминание



Проходили бэггинг и RandomForest, в них все модели строятся независимо.

Давайте строить модели не независимо, а чтобы следующая модель исправляла ошибки предыдущих.



Источник: kdnuggets.com/2017/11/difference-bagging-boosting.html



$F(x) = f_0(x) + c_1 f_1(x) + \dots c_n f_n(x)$ размера n

$F(x)$ — ансамбль, $f_i(x)$ — базовый алгоритм

Как находить все коэффициенты и базовые алгоритмы?

Будем находить их **итеративно**, сначала $f_0(x)$ потом $c_1, f_1(x)$ и так далее



Хотим минимизировать

$$Q = \sum_{i=1}^N L(F(x_i), y_i) = \sum_{i=1}^N L\left(\sum_{j=1}^n c_j f_j(x_i), y_i\right)$$

1. Взять начальное приближение $f_0(x)$

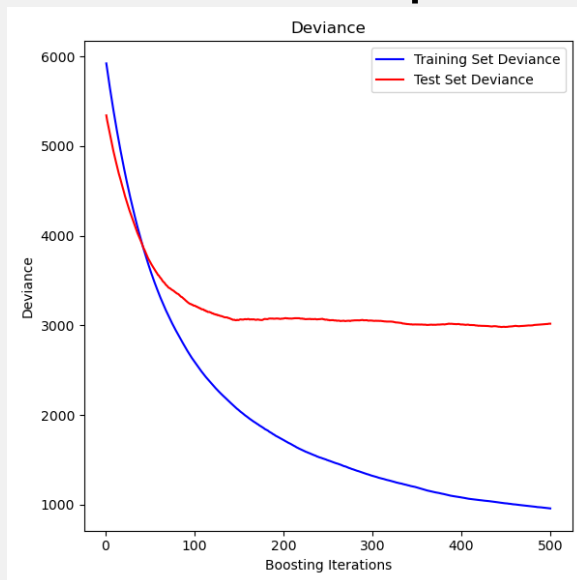
$$2. c_k, f_k = \operatorname{argmin}_{c, f} \sum_{i=1}^N L(F_{k-1}(x_i) + cf(x_i), y_i)$$

$$3. F_k = F_{k-1} + c_k f_k$$

4. Повторять 2) 3) до сходимости



- Для некоторых функций потерь, задача может решаться аналитически, для остальных - градиентный бустинг
- Нельзя использовать сверх переобученные модели, как в случайном лесе
- Число алгоритмов надо тоже подбирать



scikit-learn.org/stable/modules/ensemble.html

Часть 2

AdaBoost





$$c_k, f_k = \underset{c, f}{\operatorname{argmin}} \sum_{i=1}^N L(F_{k-1}(x_i) + cf(x_i), y_i)$$

$$L(x_i, y_i) = \exp(-y_i F(x_i)), y_i \in \{-1, 1\}$$

$$\begin{aligned} L(x_i, y_i) &= \exp(-y_i (F_{k-1} + c_k f_k(x_i))) = \\ &= \exp(-y_i F_{k-1}) * \exp(-y_i c_k f_k(x_i)) = \\ &= w_i * \exp(-y_i c_k f_k(x_i)) \end{aligned}$$

$$Q = \sum_{i=1}^N L(F(x_i), y_i) = \exp(-c_k) \sum_{y_i=f_k(x_i)} w_i + \exp(c_k) \sum_{y_i \neq f_k(x_i)} w_i$$

$$Q = (\exp(c_k) - \exp(-c_k)) \sum_{i=1}^N w_i I[y_i \neq f_k(x_i)] + \exp(-c_k) \sum_{i=1}^N w_i$$

Оптимально учить алгоритм на выборке с весами!

$$f_k = \underset{f}{\operatorname{argmin}} w_i I[y_i \neq f_k(x_i)]$$



Подставляя f_k и оптимизируя по c_k

$$c_k = \frac{1}{2} \log \frac{1 - err_k}{err_k}$$

$$\text{Где } err_k = \frac{\sum_{i=1}^N w_i I[y_i \neq f_k(x_i)]}{\sum_{i=1}^N w_i}$$

1) Инициализировали веса $w_i = \frac{1}{N}$

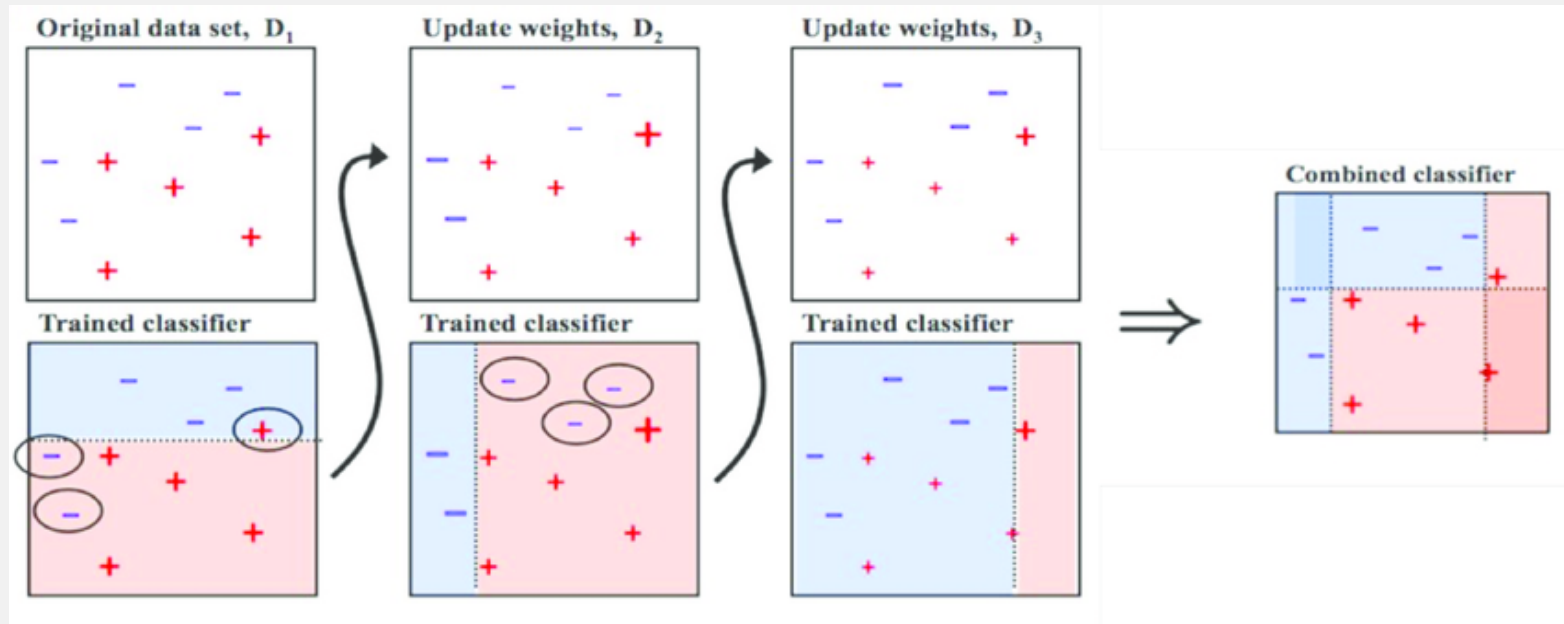
2) обучили $f_k = \operatorname{argmin}_f w_i I[y_i \neq f_k(x_i)]$, нашли $c_k = \frac{1}{2} \log \frac{1 - err_k}{err_k}$

3) Достроили ансамбль $F_k = F_{k-1} + c_k f_k$, обновили $w_i = \exp(-y_i F_k(x_i))$

4) Повторять 2) и 3) M шагов

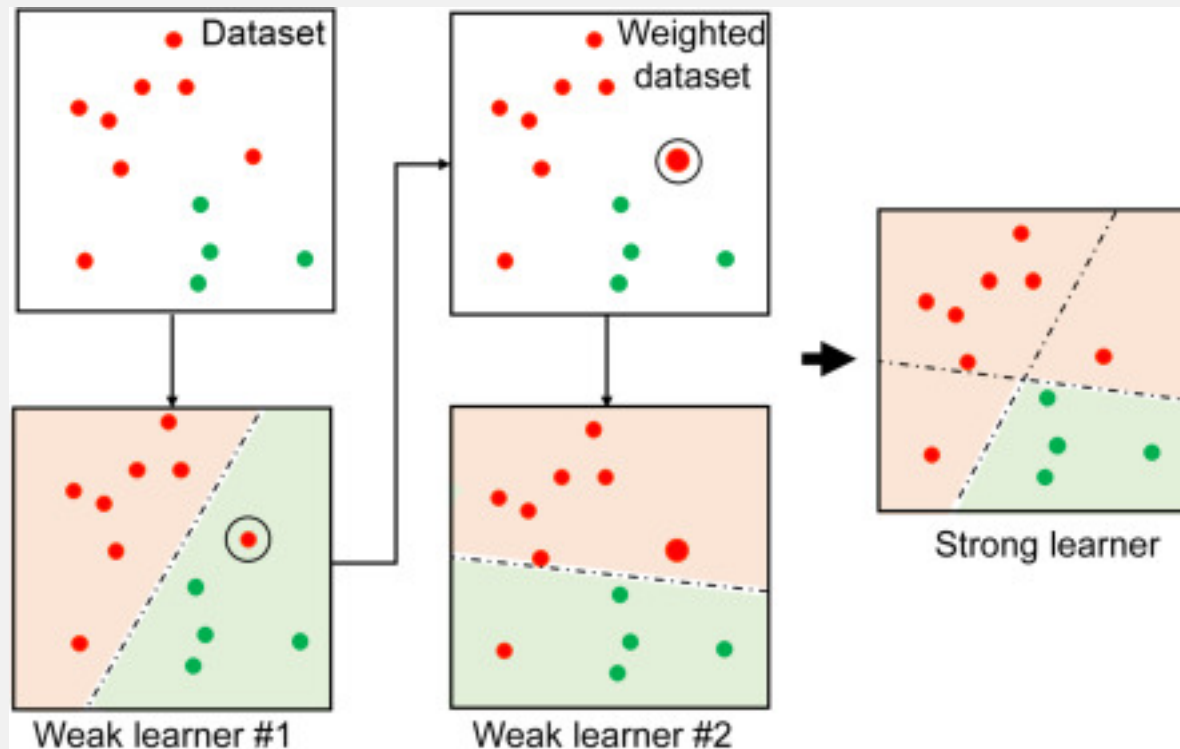
В итоге предсказание $a(x) = \operatorname{sign}(\sum_{j=1}^N c_j f_j(x_i))$

Пример



towardsdatascience.com/tagged/adaboost

Пример

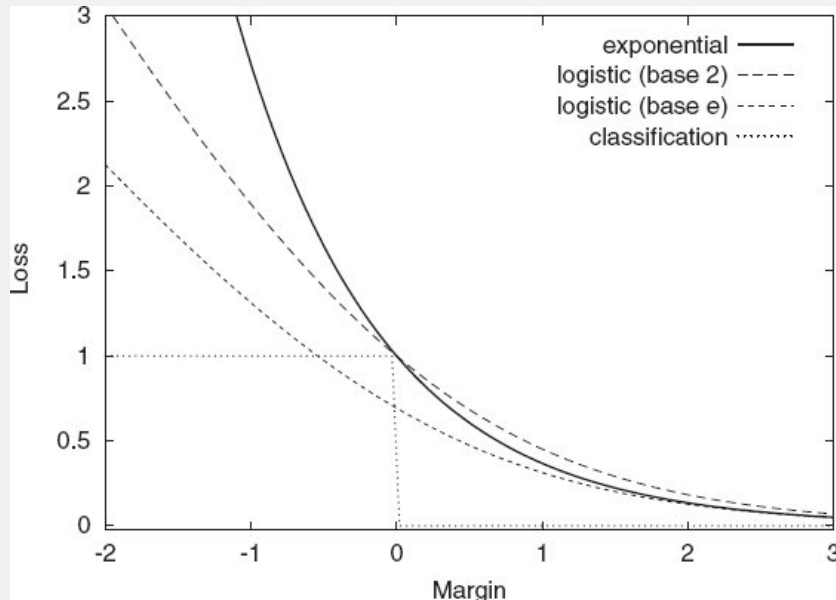


[sciencedirect.com/topics/engineering/adaboost](https://www.sciencedirect.com/topics/engineering/adaboost)

Проблема AdaBoost



Экспоненциальная функция ошибки очень плохо работает с выбросами — задает им очень большие веса. Нужно менять!



mitpress.mit.edu



$$L(x_i, y_i) = \log(1 + \exp(-y_i F(x_i)))$$

LogitBoost (two classes)

1. Start with weights $w_i = 1/N$ $i = 1, 2, \dots, N$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. Repeat for $m = 1, 2, \dots, M$:

(a) Compute the working response and weights

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))},$$

$$w_i = p(x_i)(1 - p(x_i)).$$

(b) Fit the function $f_m(x)$ by a weighted least-squares regression of z_i to x_i using weights w_i .

(c) Update $F(x) \leftarrow F(x) + \frac{1}{2}f_m(x)$ and $p(x) \leftarrow (e^{F(x)})/(e^{F(x)} + e^{-F(x)})$.

3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^M f_m(x)]$.

Еще есть L2Boost для MSE и возможно еще миллион вариантов для разных лоссов. Но сейчас уже их нет смысла разбирать, потому что есть...

Часть 3

Градиентный бустинг





- Хочется уметь делать бустинг для любого лосса (все как в линейной регрессии)

$$c_k, f_k = \operatorname{argmin}_{c, f} \sum_{i=1}^N L(F_{k-1}(x_i) + cf(x_i), y_i)$$

Куда правильнее всего идти? В сторону антиградиента! Давайте f_k приблизим антиградиент и найдем шаг. То есть мы рассмотрим функцию

$Q = \sum_{i=1}^N L(F(x_i), y_i)$ как функцию N переменных и будем оптимизировать ее градиентным спуском



1. Инициализировать f_0

2. Обучить f_k на выборке $(x_i, -\frac{\partial L}{\partial F}(x_i, F_{k-1}(x_i)))_{i=1}^N$,
например, на MSE функции потерь. Посчитали
антиградиент

3. Найти шаг

$$c_k = \operatorname{argmin}_c \sum_{i=1}^N L(F_{k-1}(x_i) + cf_k(x_i), y_i)$$

4. Достроить ансамбль $F_k = F_{k-1} + c_k f_k$

5. Повторять 2) 3) 4) M итераций



Можно честно решить задачу **одномерной** оптимизации: $\operatorname{argmin}_c \sum_{i=1}^N L(F_{k-1}(x_i) + cf_k(x_i), y_i)$

На практике можно сделать перебор.

Используют **shrinkage** $F_k = F_{k-1} + \eta c_k f_k, \eta \in (0, 1]$

Можно вообще шаг сделать константным:

$F_k = F_{k-1} + \eta f_k$, где η - learning rate

Глубина деревьев



В RandomForest строили деревья максимальной глубины, чтобы модели были переобученными.

- Если в бустинге деревья маленькие, то мы очень плохо приближаем аннтиградиент;
- Если в бустинге деревья очень глубокие, то мы за несколько итераций переобучимся.

Глубину надо подбирать на валидации!

Обычно используют глубину 3-6.

? А что с точки зрения Bias-Variance?

Стохастический градиентный бустинг



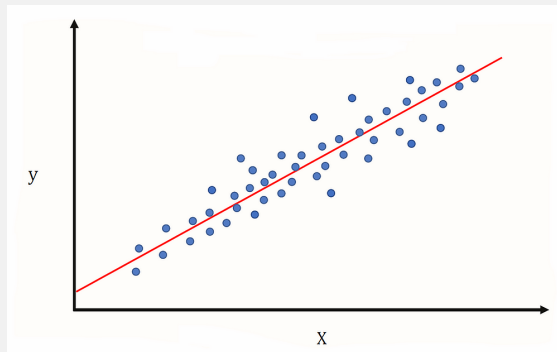
Каждый алгоритм можно учить на случайной подвыборке и случайном множестве признаков.

- + Может уменьшить переобучение
- + Быстрее учится
- + Позволяет считать OOB score

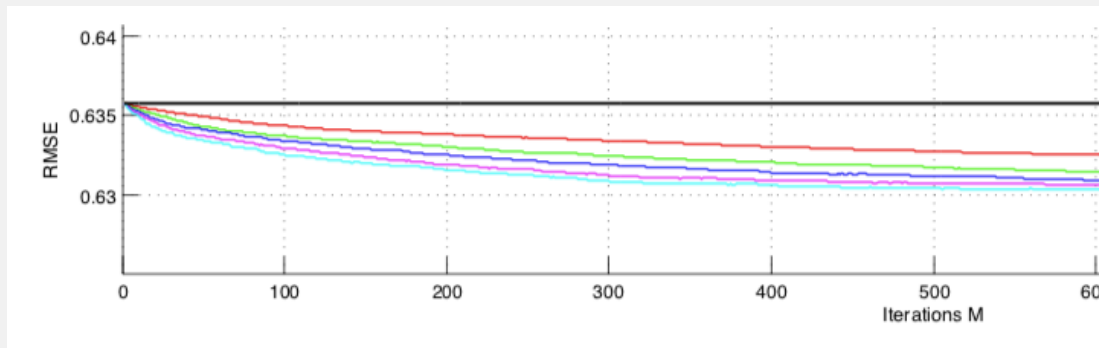
Чем инициализировать



Если есть линейный тренд, можно его отсечь инициализацией



(numpy ninja.com)



proceedings.mlr.press/v14/mohan11a/mohan11a.pdf



1. **Input:** training data D ; $NBag$ and $NBoo$ iterations of bagging and boosting respectively
2. **Output:** Random Forest of $NBag \times NBoo$ trees
3. **for** $i = 1$ **to** $NBag$ **do**
4. $D[i] := SampleData(D)$; # samples both data records and features without replacement
5. $BT[i] := BoostedTree(D[i], NBoo)$; # $NBoo$ iterations of boosting optimization on $D[i]$
6. **endfor**
7. Output additive model $\sum_i BT[i]$;

cache-mskstoredata05.cdn.yandex.net/download.yandex.ru/
company/a_scalable_hybrid_bagging_the_boosting_model.pdf

Часть 4

xgboost, lightgbm

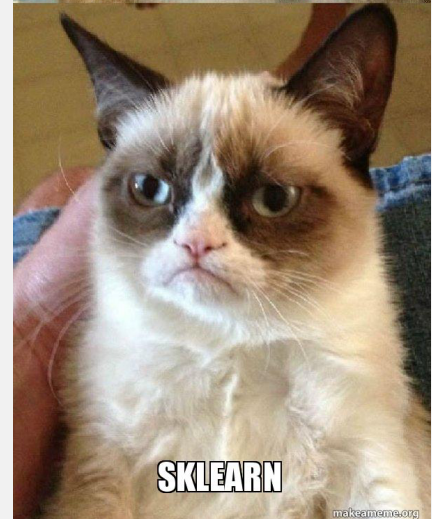
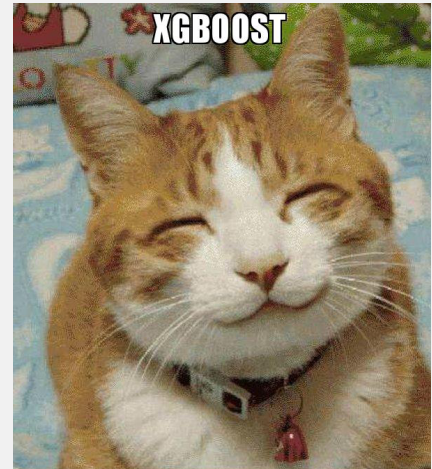




Взорвал Кагл несколько лет назад(породил мемов):
В 2015 году среди 29 соревнований 17 победителей использовали xgboost и 8 использовали только его!

Причины:

1. Быстро учится
2. Может многопоточно или на ГПУ
3. Учится непосредственно под лосс (разберем)
4. Использует хитрую оптимизацию (разберем)
5. Есть регуляризация листьев (разберем)



Почему MSE



$$f_k = \operatorname{argmin}_f \sum_{i=1}^N L(F_{k-1}(x_i) + f(x_i), y_i)$$

$$\sum_{i=1}^N L(F_{k-1}(x_i) + f(x_i), y_i) = \sum_{i=1}^N L(F_{k-1}(x_i), y_i) + g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2$$

А вот с таким лоссом мы учим дерево в бустинге:

$$\frac{1}{2} \sum_{i=1}^N (f(x_i) + g_i)^2 = \frac{1}{2} \sum_{i=1}^N f(x_i)^2 + 2f(x_i)g_i + g_i^2$$

Если гессиан на всех объектах единичен, то эти две формулы совпадают! В xgboost применяется разложение 2 порядка.



Наш функционал мерит качество на трейне, но никак не штрафует за сложность!

Делаем регуляризацию на число листьев и на значения в них

$$\sum_{i=1}^N g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2 + \gamma J + \sum_{j=1}^J |b_j|^2$$

Этот функционал потом берется как критерий информативности и деревья строятся **непосредственного** под него!

В качестве критерия останова смотрится значение этого функционала.

Approximate split finding



Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I , instance set of current node
Input: d , feature dimension
 $gain \leftarrow 0$
 $G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ **to** m **do**
 $G_L \leftarrow 0$, $H_L \leftarrow 0$
 for j **in** $sorted(I, \text{by } x_{jk})$ **do**
 $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
 $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
 $score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$
 end
end
Output: Split with max score

51

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ **to** m **do**
 Propose $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ by percentiles on feature k .
 Proposal can be done per tree (global), or per split (local).
end
for $k = 1$ **to** m **do**
 $G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g_j$
 $H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

51

wiki.math.uwaterloo.ca/statwiki/index.php?
title=summary



1. Можем учить дерево по квантилям неточно
2. Приближаем направление еще с учетом вторых производных
3. Добавляем регуляризацию в функционал
4. Меняется критерий информативности, который непосредственно использует этот функционал
5. Критерий основа тоже меняется под функционал



Статья 2017 года, то есть алгоритм свежее xgboost, взял в себя все лучшее.

Отличия:

1. Используется 1 порядок аппроксимации
2. Gradient Based One Side Sampling (уменьшаем объекты)
3. Exclusive Feature Bundling (уменьшаем признаки)



Какие примеры самые важные для обучения? На которых самый большой градиент!

```
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( $I$ )  
     $g \leftarrow loss(I, \text{preds})$ ,  $w \leftarrow \{1, 1, \dots\}$   
    sorted  $\leftarrow$  GetSortedIndices(abs( $g$ ))  
    topSet  $\leftarrow$  sorted[1:topN]  
    randSet  $\leftarrow$  RandomPick(sorted[topN:len( $I$ )],  
        randN)  
    usedSet  $\leftarrow$  topSet + randSet  
     $w[\text{randSet}] \times = \text{fact}$   $\triangleright$  Assign weight  $fact$  to the  
    small gradient data.  
    newModel  $\leftarrow L(I[\text{usedSet}], -g[\text{usedSet}],$   
         $w[\text{usedSet}])$   
    models.append(newModel)
```

papers.nips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf



1. Сгруппировать фичи в связки
2. Смердживать фичи в связках в одну фичу

Связка строится из фичей, которые **не перекрываются** на разных объектах, например

$[0, 0, 0, 1, 2], [1, 2, 0, 0, 0] \rightarrow [1, 2, 0, 3, 4]$

Идем по всем фичам, если фича перекрывается с текущими связками, создаем новую иначе добавляем в текущую



feature1	feature2	feature_bundle
0	2	6
0	1	5
0	2	6
1	0	1
2	0	2
3	0	3
4	0	4

towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e



**Спасибо за
внимание!**