# Машинное обучение, ФКН ВШЭ
# Семинар №10

На лекциях были разобраны принципы работы градиентного бустинга, а также его вариация, использующая вторые производные. Ниже мы разберёмся с тем, как устроены популярные библиотеки для бустинга и какие именно трюки в них используются для ускорение обучения или для повышения качества.

## 1 Extreme Gradient Boosting (XGBoost)

### §1.1 Кратко о его возможностях (подробнее - на лекции)

Градиентный бустинг под капотом XGBoost имеет следующие особенности:

- Его базовый алгоритм приближает направление, посчитанное с использованием вторых производных функции потерь.
- Функционал регуляризуется - добавляются штрафы за счет количества листьев и за норму коэффициентов.
- При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига.
- Критерий останова при обучении дерева также зависит от оптимального сдвига.

Как итог, на лекции была получена формула, которую мы пытаемся максимизировать(???) на каждой вершине в нашей структуре дерева. Она довольно страшная, но мы просто запишем её один раз и дальше будем ссылаться.

$$\mathbb{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in R_l} s_i\right)^2}{\sum_{i\in R_l} h_i + \lambda} + \frac{\left(\sum_{i\in R_r} s_i\right)^2}{\sum_{i\in R_r} h_i + \lambda} - \frac{\left(\sum_{i\in R} s_i\right)^2}{\sum_{i\in R} h_i + \lambda}\right] - \gamma$$

Сейчас стоит обратить внимание на то, что XGBoost из себя представляет все также обучение некоторого дерева. Поэтому оно не лишено проблем с поиском оптимального разбиения данных в каждой своей вершине. Эти проблемы могут возникать при очень большом кол-ве данных или при попытке распараллелить обучение на несколько машин. Подходов к её решению есть несколько, о них ниже. [1]

[1] затехано: Анищенко И.И.

## §1.2 Базовый подход - жадный способ разбиения

Этот вариант подходит для поиска лучшего разбиения при запуске обучения на одной машине и при возможности поместить все данные обучения на оперативную память (т.е. проблемы, описанные выше он не решает).

Для нахождения наилучшего разбиения данных мы смотрим все возможные разбиения объектов в вершине на две подгруппы. Чтобы делать это эффективно, сначала придется отсортировать данные в соответствии со значением рассматриваемого признака, и уже после посмотреть все возможные разбиения для получения статистик по первой и второй производным на каждом объекте, чтобы подсчитать $\mathbb{L}_{split}$.

**Алгоритм 1. Точный жадный алгоритм для поиска сплита**

**На вход:** $I$, набор объектов в текущей вершине;
**На вход:** $d$, размерность пр-ва признаков

1. $S \leftarrow \sum_{i \in I} s_i, \;\; H \leftarrow \sum_{i \in I} h_i$
2. `for` $k = 1$ `to` $m$ `do`
3. $\quad S_L \leftarrow 0, \;\; H_L \leftarrow 0$
4. $\quad$ `for` $j \;\; in \;\; sorted(I, by \;\; x_{jk})$ `do`
5. $\quad\quad S_L \leftarrow S_L + s_j, \;\; H_L \leftarrow H_L + h_j$
6. $\quad\quad S_R \leftarrow S - S_L, \;\; H_R \leftarrow H - H_L$
7. $\quad\quad score \leftarrow \max(score, \frac{S_L^2}{H_L + \lambda} + \frac{S_R^2}{H_R + \lambda} - \frac{S^2}{H + \lambda})$
8. $\quad$ `end`
9. `end`
10. `Return: Разбиение с большим значением` $score$

## §1.3 Приближенный алгоритм поиска разбиения

Как уже говорилось выше, простой жадный подход к поиску лучшего сплита не очень эффективен: у нас может не хватить места для всех данных в памяти, и для распределенного подхода к обучению такой вариант также не очень подходит. Поэтому для таких случаев обучения есть приближенный алгоритм. В нем алгоритм сначала предлагает потенциальные точки разделения выборки, допустим они будут получены в соответствии с некоторыми перцентилями распределения признаков (об этом ниже). Далее алгоритм сопоставляет объекты выборки в соответствующие группы, которые выделены этими "кандидатами"на разбиение выборки, агрегирует полученную статистику и находит наилучшее разбиение среди предложенных по предыдущему алгоритму. Сам алгоритм выделения этих групп и кандидатов на разбиение представлен ниже:

**Алгоритм 2. Приближенный алгоритм поиска сплита**

1. for $k = 1$ to $m$ do
2. Предлагаемые разбиения $G_k = \{g_{k1}, g_{k2}, ..., g_{kl}\}$ по перцентилям признака $k$
3. Предлагаемое разбиение может быть рассмотрено по всему дереву(глобальный подход) или по конкретной вершине(локальный подход)
4. end
5. for $k = 1$ to $m$ do
6. $S_{kv} \leftarrow= \sum_{j \in \{j | g_{k,v} \geqslant x_{jk} > g_{k,v-1}\}} s_j$
7. $H_{kv} \leftarrow= \sum_{j \in \{j | g_{k,v} \geqslant x_{jk} > g_{k,v-1}\}} h_j$
8. end
9. По полученным статистикам выполняем предыдущий алгоритм, для поиска лучшего разбиения среди предложенных

Из самого алгоритма также видно, что у нас есть 2 подхода к работе с полученными группами:

- Глобальный - предлагает некоторое кол-во кандидатов на лучшее разбиение и использует их для поиска разбиений на всех вершинах дерева.
- Локальный - рассматривает новых кандидатов разбиения на каждой вершине (т.е. набор кандидатов на лучшее разбиение пересматривается на каждой новой вершине дерева).

Первый подход требует меньше вызовов пересмотра кандидатов (алгоритм 2). Однако для хорошего результата с таким подходом нужно сильно больше таких кандидатов на разбиение (фактически выборку изначально надо дробить на большее кол-во групп), поскольку эти точки не уточняются после каждого разделения. В локальном же подходе идет пересмотр кандидатов на разбиение после каждого сделанного сплита, что хорошо подходит для более глубоких деревьев. Сравнение различных подходов (глобального и локального) на наборе данных по бозонам Хиггса показано на графике ниже:

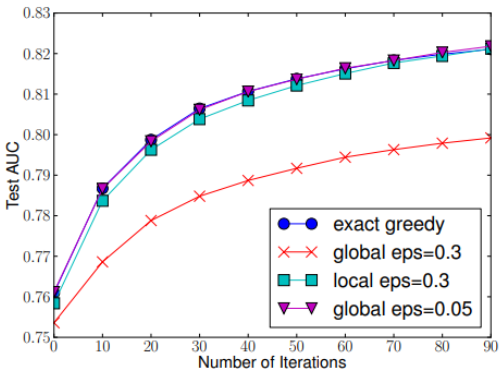


Рис. 1. Сравнение сходимости AUC на тестовых данных бозонов Хиггса размера 10M

Параметр *eps* в рассматриваемых подходах соответствует некоторой дельте в распределении, вдоль которой мы строим одну группу объектов. Фактически мы получаем

$1/eps$ групп объектов в выборке. Также можно заметить, что глобальный подход к выделению групп дает тот же результат, что и локальный подход, но с большим колвом рассматриваемых групп (а значит и с меньшим $eps$).

Большинство существующих приближенных алгоритмов для распределенного обучения деревьев также следуют этой структуре. Кроме того, вместо квантилей можно использовать и другие стратегии разбиения данных. Из графика выше также можно выделить еще один важный момент: квантильная стратегия для поиска лучшего разбиения может дать нам ту же точность, что и алгоритм с простой жадностью при разумном значении $eps$. Как отмечают авторы, под самим капотом XGBoost поддерживается как и простой жадный подход, так и с приближенными методами.

## §1.4 Поиск сплита с учетом разреженности данных

Подумаем чуть больше о данных из жизни. В них очень часто будет присутствовать разреженность в данных, которую мы пока в наших разбиениях никак не учитывали, наличию этой разреженности может быть несколько причин:

- Наличие пропущенных значений в данных
- Частые нулевые записи в статистике
- Артефакты фич инжинирига (к примеру - one-hot кодирование)

В случае работы с деревьям важно, чтобы алгоритм имел какое-то представление о разреженности данных. Для этого авторы XGBooost-а предлагают добавить направление "по умолчанию"в каждую вершину дерева для распределения объектов с пустым значением соответствующего признака (рисунок ниже).

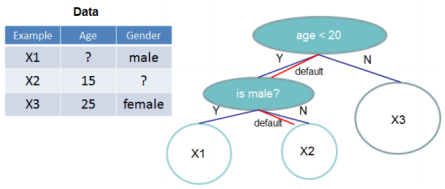


Рис. 2. Пример распределения объектов в дереве по умолчанию

Т.е. если в рассматриваемом признаке у объекта отсутствует значение, то он классифицируется в направлении по умолчанию.

Теперь мы считаем нужные нам статистики для разбиения с учетом того, что все объекты без значения конкретно признака уйдут в правую вершину или левую (выбираем тот вариант, что будет оптимальнее по $\mathbb{L}_{split}$). Также теперь мы сортируем и итеративно рассматриваем для поиска разбиений меньшее число объектов. Представленный ниже алгоритм рассматривает отсутствие признака как пропущенное значение. Также этот алгоритм может быть применем тогда, когда отсутствие признака может соответствовать заданному пользователем значению, ограничивая

перечисление только согласованными решениями.

**Алгоритм 3. Sparsity-aware Split Finding**

**На вход:** $I$, набор объектов в текущей вершине;
**На вход:** $I_k = \{i \in I | x_{ik} \neq missing\ value\}$;
**На вход:** $d$, размерность пр-ва признаков;
Также собираем статистику по объектам в группах с не пропущенными значениями признаков;
$S \leftarrow \sum_{i \in I} s_i;\ H \leftarrow \sum_{i \in I} h_i;$

```
1. for k = 1 to m do
2.     //все объекты с пропуском этого признака помещаем в правую вершину
3.     S_L ← 0,  H_L ← 0
4.     for j in sorted(I_k, ascent order by x_jk) do
5.         S_L ← S_L + s_j,  H_L ← H_L + h_j
6.         S_R ← S − S_L,  H_R ← H_R ← H − H_L
7.         score ← max(score,  S_L^2/(H_L+λ) + S_R^2/(H_R+λ) − S^2/(H+λ))
8.     end
9.     //все объекты с пропуском этого признака помещаем в левую вершину
10.     S_R ← 0,  H_R ← 0
11.     for j in sorted(I_k, descebt order by x_jk) do
12.         S_R ← S_R + s_j,  H_R ← H_R + h_j
13.         S_L ← S − S_R,  H_L ← H − H_R
14.         score ← max(score,  S_L^2/(H_L+λ) + S_R^2/(H_R+λ) − S^2/(H+λ))
15.     end
16. end
17. Return: Лучшее разбиение и полученные направления по умолчанию
```

Также про такой подход работы с разреженными данными стоит добавить, что в XGBoost им обрабатываются все возможные шаблоны разреженности. Под разными шаблонами явно стоит понимать как отсутствие признака в имеющихся данных, так и какое-то его нейтральное значение. Второе будет лучше понятно при рассмотрении работы того же one-hot кодирования: после обработки категориальных признаков таким способом мы получаем очень много нулевых значений в признаках. И эти нули наш алгоритм работы с разреженными данными может вполне интерпретировать как отсутствие признака. Так как параметр "missing"в алгоритме настраивается пользователем, то при работе с данными после OHE мы вполне можем интерпретировать нули в данных как отсутствующие признаки. Тогда алгоритм Sparsity-aware Split Finding более оперативно посплитит наши данные.

Так же стоит добавить, что такой метод обработки дает линейную сложность вычислений по отношению к числу не пропущенных значений во всех данных. На графике ниже показано сравнение наивной реализации и sparsity-aware на датасете Allstate-10K.

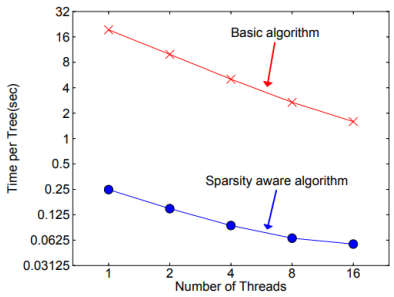


Рис. 3. Данный набор данных разрежен из-за обильного применения One-hot кодирования

В сравнении наивного и более оптимального подхода к разреженным данным можно увидеть, что алгоритм поиска наилучшего сплита с учетом разреженности работает в 50 раз быстрее, чем наивная реализация. Это дает нам дополнительные размышления о том, что учет разреженности данных в градиентном бустинге важен (особенно на этапе разбиения выборки в вершине дерева).

# 2 LightGBM

Данная концепция градиентного бустинга также пытается бороться с необходимостью просмотра всех данных выборки для поиска лучшего разбиения. Они пытаются прийти к меньшему кол-ву используемых объектов выборки и рассмотрению меньшего кол-ва признаков через подходы Gradient based One-Side Sampling (GOSS) и Exclusive Feature Bundling (EFB).

## §2.1 Gradient based One-Side Sampling

Авторы этого подхода обращают внимание на то, что объекты данных из выборки с разными градиентами отклонения играют разные роли в вычислении прироста информативности получающегося дерева. В частности, исходя из формул вычисления информативности, объекты с большими градиентами отклонения (т.е. по которым мы хуже предсказываем результат) будут вносить больший вклад в информационный выигрыш разбиения. Поэтому при попытке уменьшить кол-во рассматриваемых объектов и сохранить первоначальную точность разбиения данных, они предлагают приоритетно сохранять экземпляры с большими градиентами (например, со значениями, что большего какого-то изначально заданного порога), и каким-нибудь случайным образом отбрасывать экземпляры с меньшими значениями. Такой подход позволяет работать с меньшим кол-вом объектов при поиске лучшего сплита, и при этом получить результат, близкий к лучшему (когда смотрим на все объекты выборки).

Метод GOSS хранит все экземпляры с большими градиентами и выдает случайную выборку объектов с малыми градиентами. Чтобы компенсировать влияние на распределение получившегося набора данных, при вычислении информативности мы будем вводить постоянный множитель для объектов данных с малыми градиентами (см. в алгоритме ниже). В частности, в методе сначала идёт сортировка объектов

данных по абсолютному значению их градиентов и выбирается топ $a \times 100\%$ объектов сверху. Затемы выбираем $b \times 100\%$ объектов из остальных. После этого будем домножать градиенты случайно выбранных объектов на коэффициент $\frac{1-a}{b}$ при расчета получившегося критерия информативности. Работая с данными таким образом, метод сильнее учитывает объекты с большими градиентами отклонения, не меняя сильно исходное распределение данных. Операция выделения топа объектов по градиенту отклонений и добавления к ним объектов с малыми значениями градиентов проводится для каждого нового дерева в модели (фактически это можно увидеть в алгоритме ниже)

**Алгоритм 4. Gradient-based One-Side Sampling**

**На вход:** $I$, набор объектов в текущей вершине;
**На вход:**$d$, кол-во итераций;
**На вход:**$a$, коэф-т для отбора больших градиентов;
**На вход:**$b$, коэф-т для отбора малых градиентов;
**На вход:**$loss$, loss function; $L$, weak learner;

```
1.  models ← {}, fact ← (1−a)/b
2.  topN ← a ×  len(I), randN ← b × len(I)
3.  for i = 1 to d do
4.      preds ← models.predict(I)
5.      g ← loss(I, preds),  w ← {1, 1, ...}
6.      sorted ← GetSortedIndices(abs(g))
7.      topSet ← sorted[1:topN]
8.      randSet ← RandomPick(sorted[topN:len(I)],randN)
9.      usedSet ← topSet + randSet
10.     w[randSet] × =fact ▷ Assign weight fact to the small gradient data
11.     newModel ← L(I[usedSet], -g[usedSet],w[usedSet])
12.     models.append(newModel)
```

## §2.2 Exclusive Feature Bundling

В задачах машинного обучения с реальными данными, мы чаще всего будем сталкиваться с разреженными пр-вами признаков. Оказывается, что эту мысль так же можно использовать для возможных оптимизаций обучения решающих деревьев. В частности стоит уточнить, что в разреженном пр-ве признаков многие их значения являются исключительными, т.е. какая-нибудь пара признаков редко будет принимать ненулевые значения одновременно. Достаточно вспомнить результат обработки категориальных признаков через тот же one-hot-encoding, где мы получаем вполне разреженное пр-во признаков (и мало какие пары там одновременно принимают единичное значение). Поэтому авторы этого подхода пытаются отследить все эти "эксклюзивные"комбинации признаков, путём решения задачи раскраски графа, где каждый признак берётся как вершина, а ребра добавляются между ними, если они не взаимоисключающие. Такая дополнительная конструкция позволяет осматривать меньшее кол-во признаков в данных для поиска лучшего прироста информативности.

## §2.3 Немного о правиле роста дерева в LightGBM

Также, говоря про LightGBM, хотелось бы поговорить про отдельный подход к росту этого дерева. В обычном случае параметр ограничение на рост - максимальная глубина дерева. По нему мы просто будем отслеживать глубину каждой вершины от корня, и остановимся делить по ним нашу выборку тогда, когда очередная новая вершина дойдет до указанной ранее макс. глубины в параметре. И что самое важное - порядок появления вершин будет приблизительно такой:

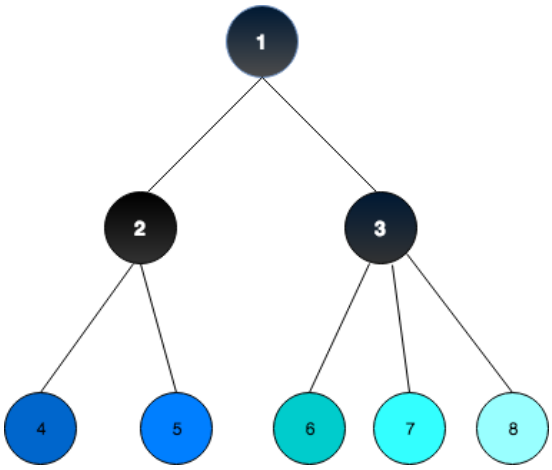


Рис. 4. Каждую вершину от корня мы разбиваем на дочерние, дерево растёт именно в таком порядке

Данный подход можно характеризовать как "level-wise где наше основное правило роста это максимальная глубина.

В LightGBM же новые узлы дерева выбираются на основе максимизации текущей информативности в модели. Т.е. из всех возможных вершин в дереве в приоритете будет создана та, что дает больший прирост в информативности на данном этапе разбиения данных. При таком подходе мы получаем другой порядок появления вершин в дереве, и выглядит он следующим образом:

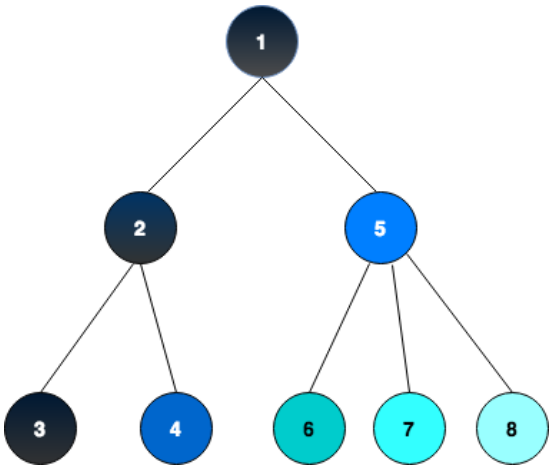


Рис. 5. Каждая новая вершина дает нам лучший прирост в информативности из всех возможных

При таком подходе мы можем получить достаточно несбалансированные деревья. Но

такой подход к их построению логичнее вяжется с нашей начальной целью - минимизации исходной функции потерь.

# 3 CatBoost

Как вы уже наверно знаете (из домашних заданий или лекций) - CatBoost тоже является хорошей базов для обучения решающих деревьев со своими подходами: упорядоченным градиентным бустингом и новый способ обработки категориальных признаков. Но более подробно мы рассмотрим правило построения решающего дерева в этой библиотеке.

## §3.1 Oblivious Decision Trees

Основное изменение построения состоит в следующем: на каждом уровне разбиения данных от корневой вершины мы будем использовать одно и то же правило. Фактически это означает, что на той же глубине дерева 2 мы будем разбивать обе вершины с объектами по пороговому значению одного и того же признака. В результате мы будем получать дерево глубины $H$. Где на каждом уровне $h$ будет одно и тоже правило разбиения $f_h(x)$. И так как каждый уровень дерева будет связан со своим решающим правилом, то фактически, мы получим разбиение пр-ва выборки X на $2^H$ ячеек. И сам классификатор и будет задаваться таблицей решений вида:

$$a(x) = T(f_1(x), \ldots, f_h(x))$$

Также лучше это можно понять на следующем примере:

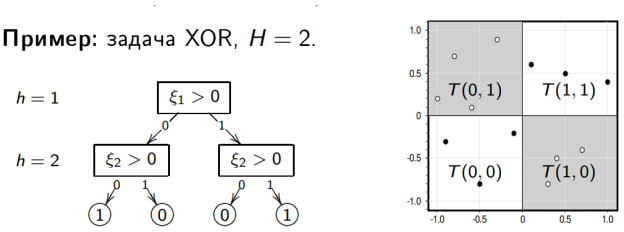


Рис. 6. Пример задачи, с использованием решающей таблицы

Самое полезное, что можно получить из этого подхода - мы получаем сбалансированное дерево решения и четкое разделение пр-ва данных на отдельные сегменты, ячейки таблицы. В таком контексте получение предсказания от имеющейся модели можно понимать как обращение к хэш-таблице: мы передаем значения признаков объекта и нам по ним выдается конкретная область разделенного пр-ва со значением класса в ней (если решаем задачу классификации), или же среднее значение таргета для задачи регрессии.

# Список литературы

[1] Tianqi Chen, Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System"
https://arxiv.org/pdf/1603.02754.pdf

[2] LightGBM: A Highly Efficient Gradient Boosting Decision Tree
https://papers.nips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[3] The Monotonic, Darting, Best-First Boosted Tree
https://medium.com/data-waffles/the-monotonic-darting-best-first-boosted-tree-cd917b

[4] CatBoost: unbiased boosting with categorical features
https://arxiv.org/pdf/1706.09516.pdf

[5] Логические алгоритмы классификации
http://www.machinelearning.ru/wiki/images/9/97/Voron-ML-Logic-slides.pdf