XGBoost vs LightGBM vs CatBoost

Цель

- Параллельное агрегирование данных в Random Forest- позволяет нам уменьшить дисперсию с небольшим компромиссом в смещении.
- Было бы неплохо, если бы мы могли как-то сделать наоборот: агрегировать модели, чтобы уменьшить смещение, не вводя слишком большой разброс.
- Boosting - это инструмент, который позволяет нам делать именно это!

математическое ожидание квадрата отклонения ответа алгоритма от истинного значения:

$$\mathrm{E}(y-a)^2 = \mathrm{E}(y^2 + a^2 - 2ya) =$$
$$= \mathrm{E}\,y^2 - (\mathrm{E}\,y)^2 + (\mathrm{E}\,y)^2 + \mathrm{E}\,a^2 - (\mathrm{E}\,a)^2 + (\mathrm{E}\,a)^2 - 2f\,\mathrm{E}\,a =$$
$$= \mathrm{D}\,y + \mathrm{D}\,a + (\mathrm{E}\,y)^2 + (\mathrm{E}\,a)^2 - 2\mathrm{E}\,ya =$$
$$= \mathrm{D}\,y + \mathrm{D}\,a + f^2 + (\mathrm{E}\,a)^2 - 2f\,\mathrm{E}\,a =$$
$$= \mathrm{D}\,y + \mathrm{D}\,a + (\mathrm{E}(f-a))^2 \equiv \sigma^2 + \mathrm{variance}(a) + \mathrm{bias}^2(f,a)$$

В чем идея?

- Давайте ненадолго забудем о деревьях и начнем с чистого листа.
- Подумайте о задаче регрессии, когда нам передают набор данных $\{(\vec{x}_i, y_i)\}_{i=1}^{N}$, и мы хотим создать модель $f(\vec{x})$, которая минимизирует среднеквадратичную ошибку:

$$\mathcal{L}(f) = \frac{1}{2N}\sum_i \big(y_i - f(\vec{x}_i)\big)^2$$

- Предположим, нам дана модель f_0 , которая пытается решить эту проблему, что нам делать?

В чем идея?

- Что мы можем сделать, так это посмотреть на ошибки, которые допускает эта модель

$$\epsilon_i = y_i - f_0(\vec{x}_i)$$

- Имея эти ошибки на месте, мы можем попытаться изучить новую модель $g_0(\vec{x}_i)$, чтобы предсказать эти ошибки.

$$\mathcal{L}(g_0) = \frac{1}{2N}\sum_i \left(\epsilon_i - g_0(\vec{x}_i)\right)^2$$

Или более интуитивно :

$$g_0(\vec{x}_i) \approx \epsilon_i$$

- Затем мы создаем новую модель $f_1(\vec{x}) = f_0(\vec{x}) + g_0(\vec{x})$

В чем идея?

- Теперь мы можем повторить эту процедуру :
  - Возьмите нашу модель f_k
  - Вычислить остатки $\epsilon_i = y_i - f_k(\vec{x}_i)$
  - Обучить модель $g_k(\vec{x}_i) \approx \epsilon_i$
  - Определите новую модель $f_{k+1}(\vec{x}) = f_k(\vec{x}) + g_k(\vec{x})$
- Теоретически мы могли бы начать с любой модели, скажем, среднего значения:

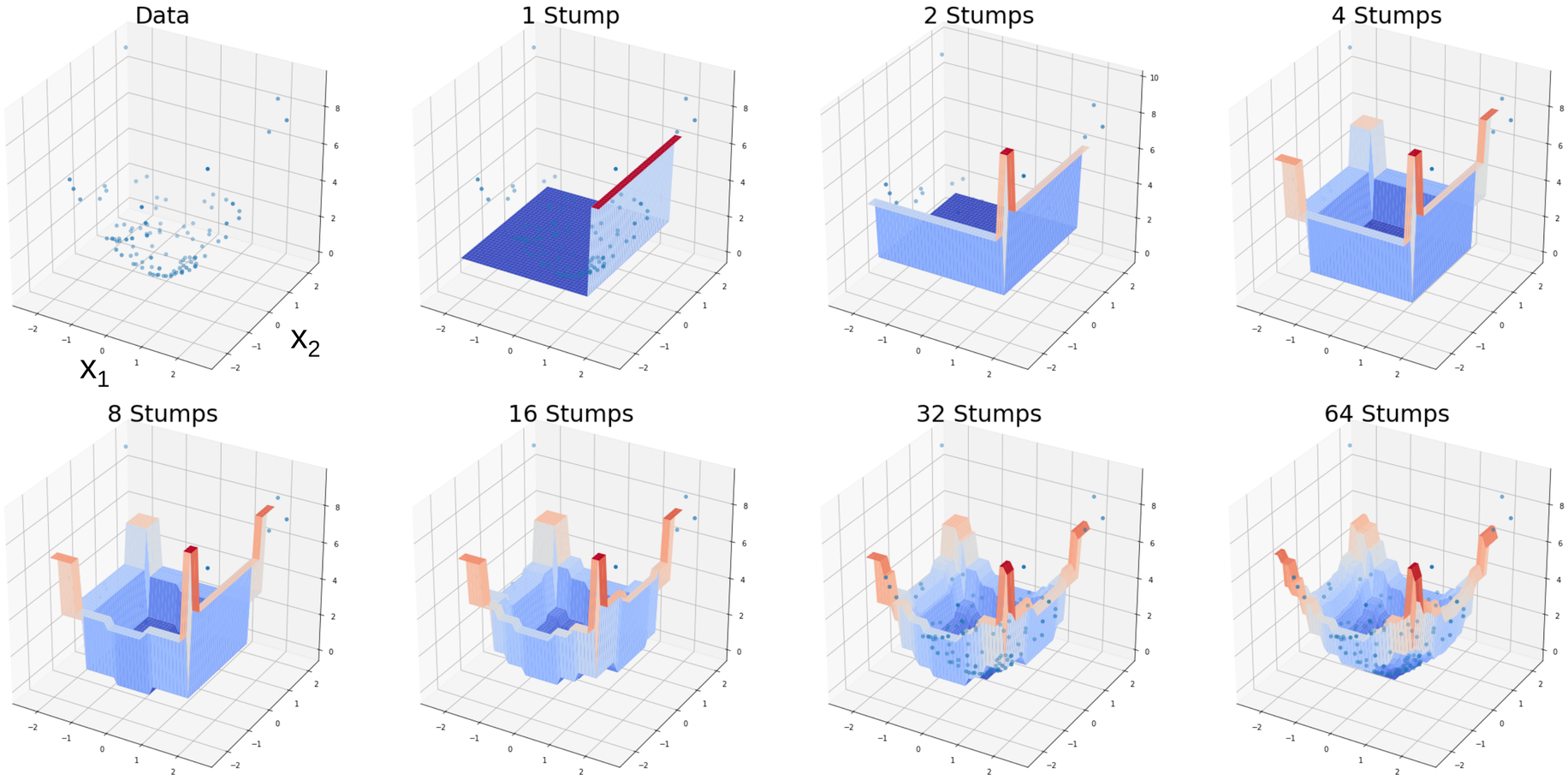
$$f_0(\vec{x}) = \frac{1}{N}\sum_i y_i$$

- Пока наши модели g_k могут работать лучше, чем угадывать среднее значение для любого набора данных, мы будем продолжать улучшать общие потери (обучение)!

Boosting Example

$x_1, x_2 \sim N(0, 1)$

$y = x_1^2 + x_2^2$



Обобщая

- В предыдущем разделе мы повысили эффективность путем подбора моделей по остаткам. Очевидно, это имеет смысл только для регрессии, так что же мы можем сделать в целом?
- Обратите внимание на следующее :

$$\mathcal{L}(f) = \frac{1}{2N}\sum_i \left(y_i - f(\vec{x}_i)\right)^2 = \frac{1}{N}\sum_i l_i$$

где

$$l_i = \frac{1}{2}\left(y_i - f(\vec{x}_i)\right)^2$$

Обобщая

- Наши остатки (Loss) - это сумма остатков на точку данных::

$$l_i = \frac{1}{2}\left(y_i - f(\vec{x}_i)\right)^2$$

- Если мы возьмем производную от loss и изменим условия, остатки будут :

$$\epsilon_i = y_i - f(\vec{x}_i) = -\frac{\partial l_i}{\partial f(\vec{x}_i)}$$

- Остаток - это не что иное, как отрицательный градиент функции потерь относительно нашего прогноза!

Обобщая

- Теперь мы можем повторить эту процедуру :
  - Возьмите нашу модель f_k
  - Вычислить ~~невязки~~ (~~the residuals~~) ~~$\epsilon_i = y_i - f_k(\vec{x}_i)$~~ псевдо-остатки $\epsilon_i = -\frac{\partial l_i}{\partial f(\vec{x}_i)}$
  - Обучить модель $g_k(\vec{x}_i) \approx \epsilon_i$
  - Определите новую модель $f_{k+1}(\vec{x}) = f_k(\vec{x}) + g_k(\vec{x})$
- Теперь мы видим, что то, что мы делаем, является не чем иным, как градиентным спуском функции потерь, где мы аппроксимируем градиент, используя слабого ученика.

Несколько комментариев

- Действительно, слабость обучаемого является ключевой, поскольку она обеспечивает форму регуляризации, при которой обновления градиента вынуждены быть простыми. Если бы слабые ученики были сильными (скажем, очень глубокими деревьями решений), они бы мгновенно переобучились за один шаг обучения.

- Оптимальный размер шага не обязательно должен быть равен единице, поэтому многие реализации выполняют поиск по строке, чтобы найти оптимальный параметр γ_k, поэтому обновление выглядит следующим образом:

$$f_{k+1}(\vec{x}) = f_k(\vec{x}) + \gamma_k \, g_k(\vec{x})$$

- Мы также можем использовать **скорость обучения** ν, в этом случае часто называемую **сжатием**, чтобы этап обновления был следующим:

$$f_{k+1}(\vec{x}) = f_k(\vec{x}) + \nu \, \gamma_k \, g_k(\vec{x})$$

Типичный выбор функций потерь

- Regression
  - Mean Squared Error

$$\mathcal{L}(f) = \frac{1}{2N}\sum_i \left(y_i - f(\vec{x}_i)\right)^2$$

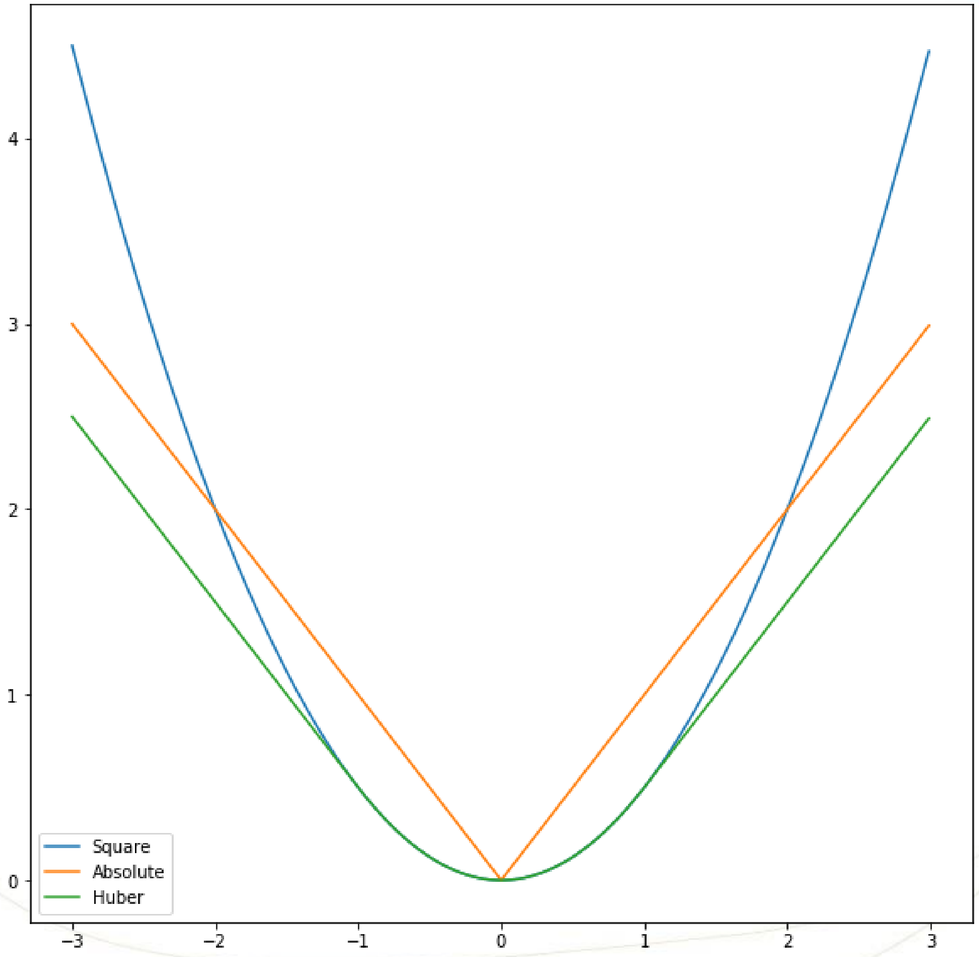
  - Mean Absolute Error

$$\mathcal{L}(f) = \frac{1}{N}\sum_i |y_i - f(\vec{x}_i)|$$

  - Huber Loss (mixture of the two)

$$\mathcal{L}(f) = \frac{1}{N}\sum_i \begin{cases} \frac{1}{2}\left(y_i - f(\vec{x}_i)\right)^2, & |y_i - f(\vec{x}_i)| < 1 \\ |y_i - f(\vec{x}_i)| - \frac{1}{2}, & |y_i - f(\vec{x}_i)| \geq 1 \end{cases}$$

Типичный выбор функций потерь



Типичный выбор функций потерь

- Classification
  - Отклонение (Logistic Regression Loss)

$$\mathcal{L}(f) = \frac{1}{N}\sum_i \log(1 + e^{-2y_i \cdot f(\vec{x}_i)})$$

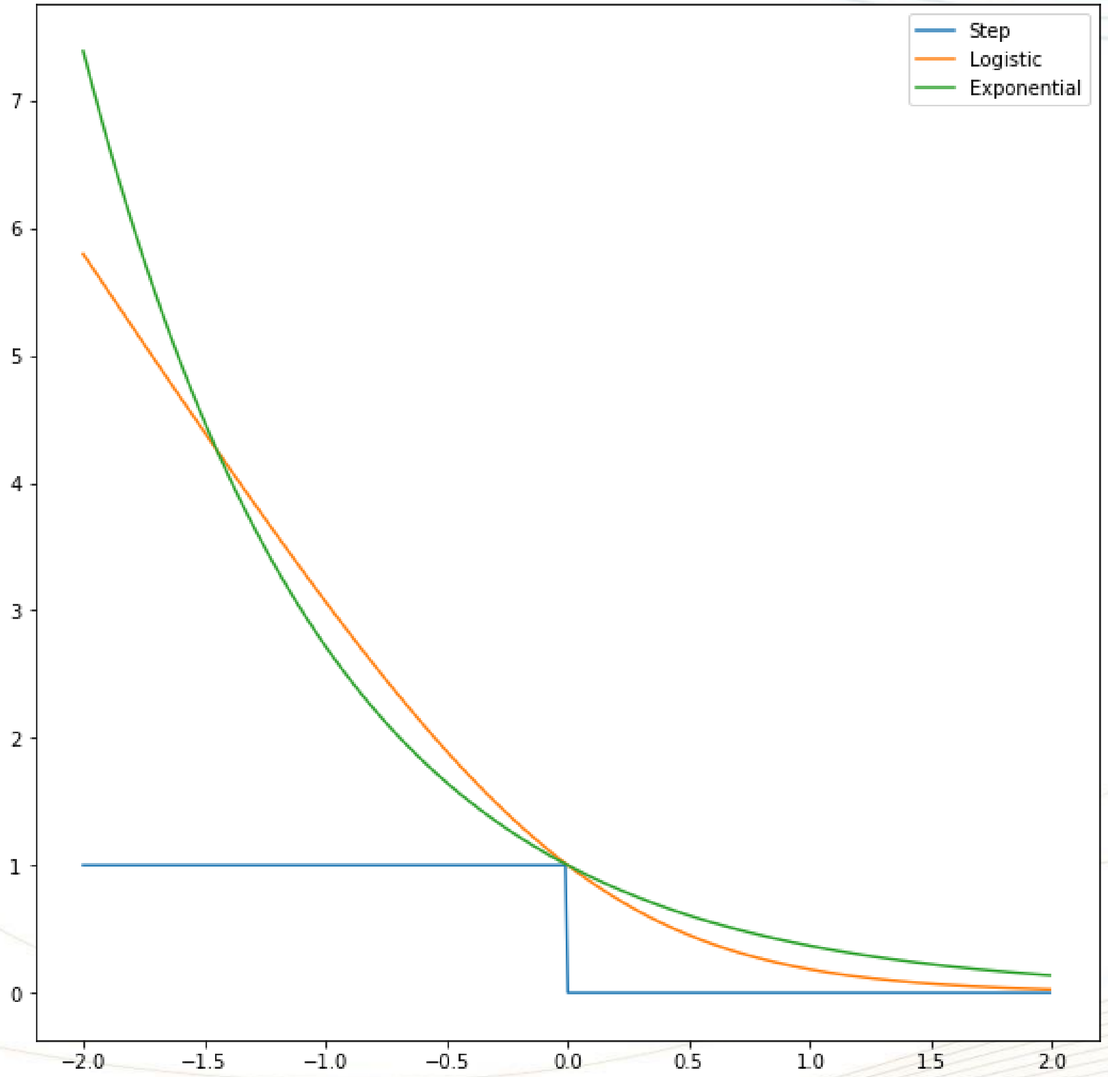
  - Exponential loss (Adaboost)

$$\mathcal{L}(f) = \frac{1}{N}\sum_i e^{-y_i \cdot f(\vec{x}_i)}$$

  - Hinge Loss (Perceptrons and SVMs)

$$\mathcal{L}(f) = \frac{1}{N}\sum_i \max\left(0, 1 - y_i \cdot f(\vec{x}_i)\right)$$

Типичный выбор функций потерь



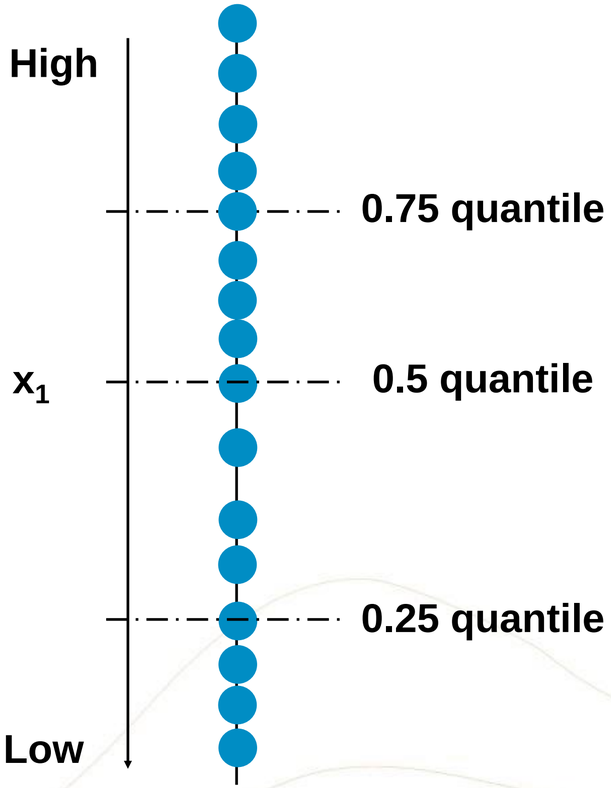
XGBoost

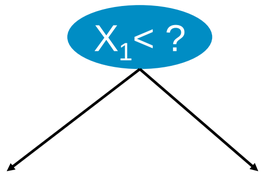
- XGBoost означает **“Extreme Gradient Boosting”**
- Подробности алгоритма описаны в 2016 paper* here.
- Некоторые дополнения этой модели (их гораздо больше):
  - Аппроксимационный жадный алгоритм (использующийся для жадного поиска приближённого решения оптимизационной задачи.)
  - Переформулируйте условия разделения и приросты с учетом весов в листьях.
  - Методы повышения производительности: параллельные блоки, доступ с учетом кеширования и внепрограммные вычисления.

* Chen, T. & Guestrin, C. Xgboost: A scalable tree boosting system. In Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 785–794 (ACM, 2016).

Аппроксимационный жадный алгоритм

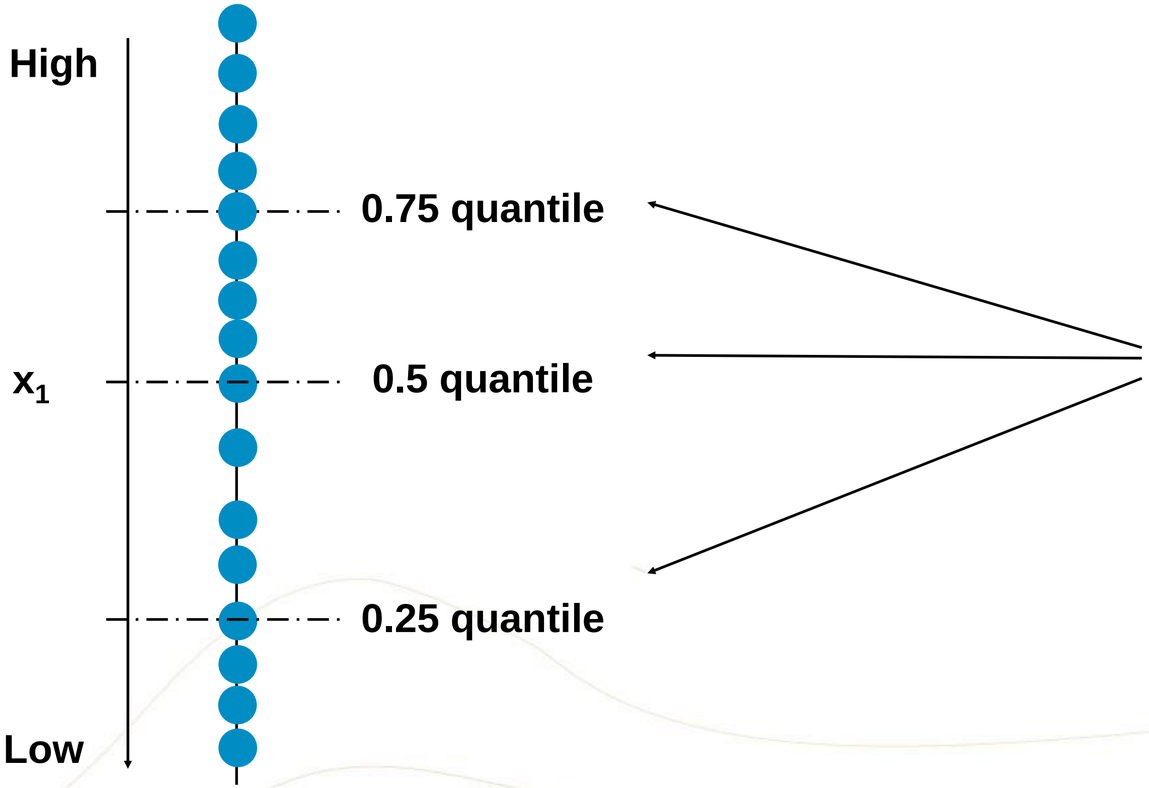
- Испытание всех возможных пороговых значений разделения может занять много времени для больших наборов данных.
- XGBoost использует квантили в качестве кандидатов на порог. Предположим, что объект x1 упорядочен следующим образом:

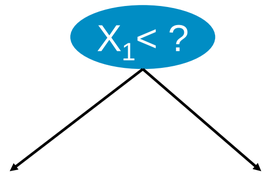




Приближенный жадный алгоритм

- Испытание всех возможных пороговых значений разделения может занять много времени для больших наборов данных.
- XGBoost использует квантили в качестве кандидатов на порог. Предположим, что объект x1 упорядочен следующим образом:





- Мы можем использовать эти три квантиля в качестве разбиения.
- Обычно существует более трех квантилей, и они применяют некоторые дополнительные оптимизации, чтобы делать это параллельно.

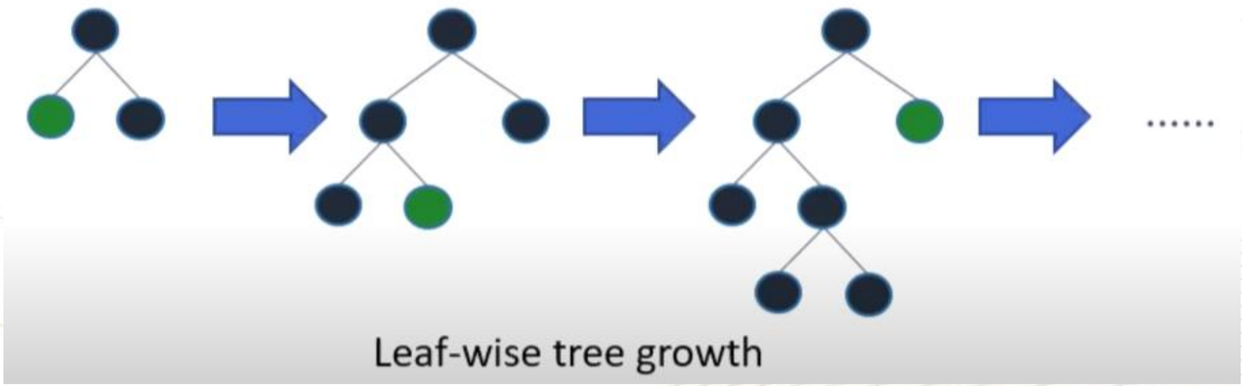
XGBoost Interface

Интерфейс XGBoostClassifier с некоторыми параметрами по умолчанию (полный интерфейс гораздо больше):

```
XGBClassifier(max_depth=3,
              learning_rate=0.1,
              n_estimators=100,
              objective='binary:logistic’,
              booster='gbtree’,
              n_jobs=1,
              …)
```

LightGBM

- Высокоэффективный метод повышения градиента и библиотека, выпущенные в 2017 году. Оригинал статьи here.
- Основные важные особенности:
  - **Односторонняя выборка на основе градиента (Gradient-based One-Side Sampling - GOSS):** Puts more focus on under-trained data points.
  - **Exclusive Feature Bundling (EFB):** эффективное представление разреженных функций, таких как функции (признаков) полученных с помощью one-hot-encoded features (для преобразования категориальных признаков в числовые).



Односторонняя выборка на основе градиента (Gradient-based One-Side Sampling - GOSS):

- **Идея:** при построении деревьев мы можем уделять больше внимания (важности) недостаточно обученным точкам данных
- Мы можем использовать градиенты как некую меру важности.
  - **Небольшой градиент**: означает небольшую ошибку, точка данных хорошо изучена**. (не важный)**
  - **Большой градиент:** означает большую ошибку, точка данных плохо изучена**. (важный)**
- Алгоритм (с параметрами a и b):
  - Отсортируйте данные по абсолютному значению градиента.
  - Сохраняйте верхние $\boldsymbol{a} \times \mathbf{100}\%$ выборок данных (большие градиенты)
  - Произвольно выберите $\boldsymbol{b} \times \mathbf{100}\%$ от остальных данных (небольшие градиенты)
  - Усильте небольшие градиенты путем умножения $\frac{1-a}{b}$ при вычислении информационного выигрыша.

Пакет эксклюзивных функций (Exclusive Feature Bundling - EFB)

Хороший способ уменьшить количество функций за счет разреженности больших наборов данных.

Главное наблюдение заключается в том, что многие функции **никогда не могут быть ненулевыми одновременно**. Следовательно, мы можем **объединить (связать)** их и сэкономить место, сделав это.

Пример:

Feature 1	Feature 2	Feature 3
0	1	2
0	0	0
1	3	0
0	0	3
5	0	0
0	0	7

Пакет эксклюзивных функций (Exclusive Feature Bundling - EFB)

Хороший способ уменьшить количество функций за счет разреженности больших наборов данных.

Главное наблюдение заключается в том, что многие функции **никогда не могут быть ненулевыми одновременно**. Следовательно, мы можем **объединить (связать)** их и сэкономить место, сделав это.

Пример:

Feature 1	Feature 2	Feature 3
0	1	2
0	0	1
1	3	0
0	0	3
5	0	0
0	0	7

- **Мы не можем объединить функцию 1 и функцию 2. Есть частичное совпадение**

Пакет эксклюзивных функций (Exclusive Feature Bundling - EFB)

Хороший способ уменьшить количество функций за счет разреженности больших наборов данных.

Главное наблюдение заключается в том, что многие функции **никогда не могут быть ненулевыми одновременно**. Следовательно, мы можем **объединить (связать)** их и сэкономить место, сделав это.

Пример 1:

Feature 1	Feature 2	Feature 3
0	1	2
0	0	1
1	3	0
0	0	3
2	0	0
0	0	7

- **Нет совпадения между Feature 1 и Feature 3. Мы можем объединить их!**

Пример 2:

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

Пакет эксклюзивных функций (Exclusive Feature Bundling - EFB)

Хороший способ уменьшить количество функций за счет разреженности больших наборов данных.

Главное наблюдение заключается в том, что многие функции **никогда не могут быть ненулевыми одновременно**. Следовательно, мы можем **объединить (связать)** их и сэкономить место, сделав это.

Пример 1:

Feature 1	Feature 2	**Feature 3**
0	1	2+1
0	0	1
1	3	0
0	0	3+1
2	0	2
0	0	7+1

- **Мы добавим +1 к ненулевым строкам Feature 3, чтобы у Feature 3 и Feature 1 были разные диапазоны.**

Пакет эксклюзивных функций (Exclusive Feature Bundling - EFB)

Хороший способ уменьшить количество функций за счет разреженности больших наборов данных.

Главное наблюдение заключается в том, что многие функции **никогда не могут быть ненулевыми одновременно**. Следовательно, мы можем **объединить (связать)** их и сэкономить место, сделав это.

Пример:

Feature 1 + **Feature 3**	Feature 2
3	1
1	0
0	3
4	0
2	0
8	0

Bundle

Если значение больше 2: это относится к функции 3.

Catboost

- Метод бустинга, который фокусируется на обработке категориальных признаков и бустинге деревьев с некоторым «принципом упорядочения».
- Подробности алгоритма описаны в paper.
- Главный вывод - применить принцип **упорядочения**:
  - Целевая кодировка категориальных функций
  - Boosting trees

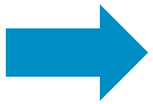
Целевое кодирование (Target encoding)

- Эффективный способ работы с категориальными переменными - заменить их числовыми значениями (обычно это некоторая целевая статистика).
- Среднее целевое кодирование: замените категории средним целевым значением для них.

Пример:

color	target
blue	0
red	1
blue	1
blue	1
green	0
red	0

color	mean_target
blue	(0+1+1)/3=0.67
red	(0+1)/2=0.5
green	0/1=0



color	encoded_color	target
blue	0.67	0
red	0.5	1
blue	0.67	1
blue	0.67	1
green	0	0
red	0.5	0

Целевое кодирование со сглаживанием (Target encoding with smoothing)

Обычно мы применяем некоторое сглаживание в расчетах с предшествующим условием.

$$avg\_target = \frac{count\_inclass + prior}{total\_count + 1}$$

where

count_inclass: сколько раз значение метки было равно «1» для объектов для категориальной функции.

Prior: предварительное значение числителя. Это определяется стартовыми параметрами.

total_count: общее количество объектов со значением категориальной характеристики.

Пример: Допустим prior=0.05

color	target
blue	0
red	1
blue	1
blue	1
green	0
red	0

color	target_encoding
blue	(2+0.05)/4=0.51
red	(1+0.05)/3=0.35
green	(0+0.05)/2=0.025

color	encoded_color	target
blue	0.51	0
red	0.35	1
blue	0.51	1
blue	0.51	1
green	0.025	0
red	0.35	0

Упорядоченная целевая кодировка (Ordered target encoding)

- Зачем использовать **«упорядоченное»** кодирование? Это помогает предотвратить переобучение из-за «**канала утечки цели**» (“**target leakage**”).
- Основанная на методах онлайн-обучения, целевая статистика опирается только на историю наблюдений (observed history).

$$avg\_target = \frac{count\_inclass + prior}{total\_count + 1}$$

count_inclass: сколько раз значение метки было равно «1» для объектов для категориального признака (до текущего, не включая его).

Prior: предварительное значение числителя. Это определяется стартовыми параметрами.

total_count: общее количество объектов (до текущего, не включая его), у которых значение категориальной характеристики совпадает с текущим.

Example: Assume prior=0.05

color	target
blue	0
red	1
blue	1
blue	1
green	0
red	0

color	encoded_target
blue	(0+0.05)/(0+1)=0.05

Упорядоченная целевая кодировка (Ordered target encoding)

- Зачем использовать **«упорядоченное»** кодирование? Это помогает предотвратить переобучение из-за «**канала утечки цели**» (“**target leakage**”).
- Основанная на методах онлайн-обучения, целевая статистика опирается только на историю наблюдений (observed history).

$$avg\_target = \frac{count\_inclass + prior}{total\_count + 1}$$

where

count_inclass: сколько раз значение метки было равно «1» для объектов для категориального признака (до текущего, не включая его).

Prior: предварительное значение числителя. Это определяется стартовыми параметрами.

total_count: общее количество объектов (до текущего, не включая его), у которых значение категориальной характеристики совпадает с текущим.

Example: Assume prior=0.05

color	target
blue	0
red	1
blue	1
blue	1
green	0
red	0

color	encoded_target
blue	(0+0.05)/(0+1)=0.05
red	(**0**+0.05)/(**0**+1)=0.05

Still no red = 1 before

Still no red before

Упорядоченная целевая кодировка (Ordered target encoding)

- Зачем использовать **«упорядоченное»** кодирование? Это помогает предотвратить переобучение из-за «**канала утечки цели**» (“**target leakage**”).
- Основанная на методах онлайн-обучения, целевая статистика опирается только на историю наблюдений (observed history).

$$avg\_target = \frac{count\_inclass + prior}{total\_count + 1}$$

where

count_inclass: сколько раз значение метки было равно «1» для объектов для категориального признака (до текущего, не включая его).

Prior: предварительное значение числителя. Это определяется стартовыми параметрами.

total_count: общее количество объектов (до текущего, не включая его), у которых значение категориальной характеристики совпадает с текущим.

Example: Assume prior=0.05

color	target
blue	0
red	1
blue	1
blue	1
green	0
red	0

color	encoded_target
blue	(0+0.05)/(0+1)=0.05
red	(0+0.05)/(0+1)=0.05
blue	(**0**+0.05)/(**1**+1)=0.025

Still no blue = 1 before

Now one blue before

Упорядоченная целевая кодировка (Ordered target encoding)

- Зачем использовать **«упорядоченное»** кодирование? Это помогает предотвратить переобучение из-за «**канала утечки цели**» (“**target leakage**”).
- Основанная на методах онлайн-обучения, целевая статистика опирается только на историю наблюдений (observed history).

$$avg\_target = \frac{count\_inclass + prior}{total\_count + 1}$$

where

count_inclass: сколько раз значение метки было равно «1» для объектов для категориального признака (до текущего, не включая его).

Prior: предварительное значение числителя. Это определяется стартовыми параметрами.

total_count: общее количество объектов (до текущего, не включая его), у которых значение категориальной характеристики совпадает с текущим.

Example: Assume prior=0.05

color	target
blue	0
red	1
blue	1
blue	1
green	0
red	0

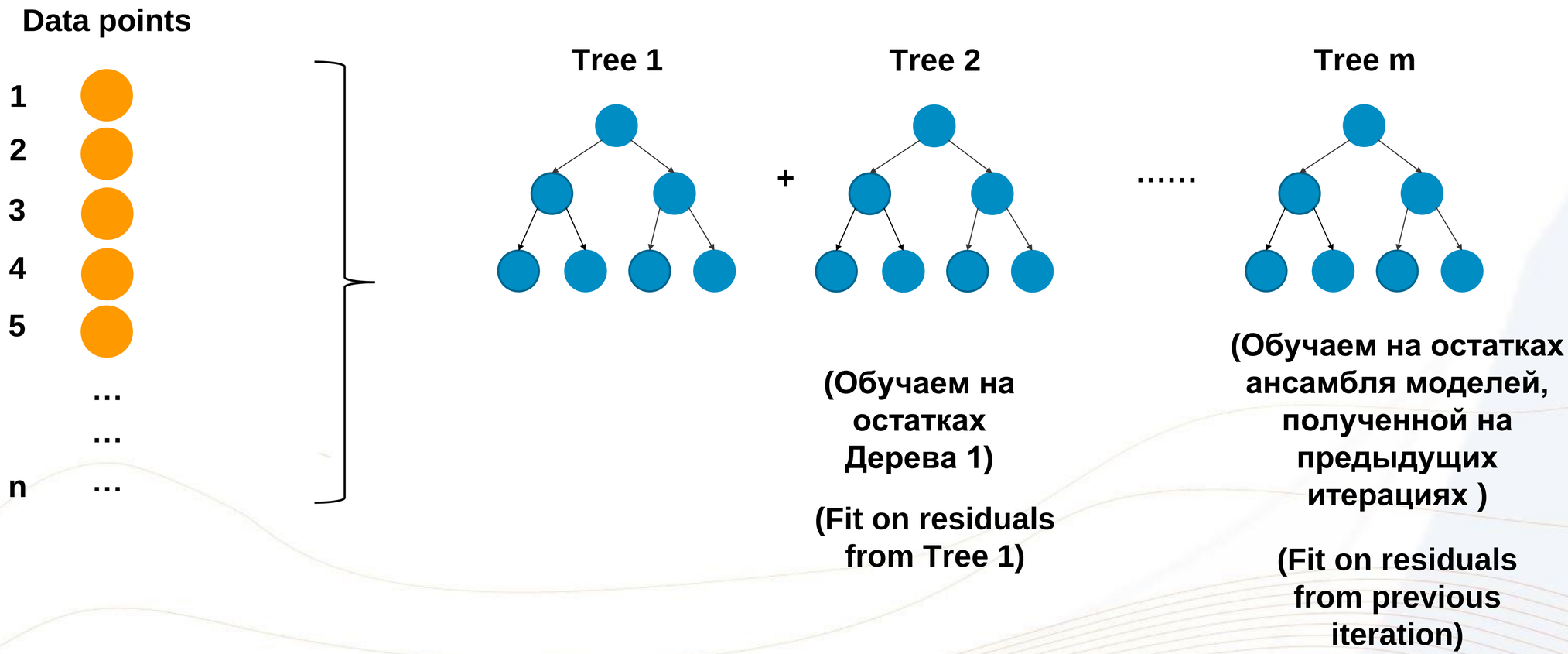
color	encoded_target
blue	(0+0.05)/(0+1)=0.05
red	(0+0.05)/(0+1)=0.05
blue	(0+0.05)/(1+1)=0.025
blue	(1+0.05)/(2+1)=0.35
green	(0+0.05)/(0+1)=0.05
red	(**1**+0.05)/(**1**+1)=0.025

Now one red = 1 before

Now one red before

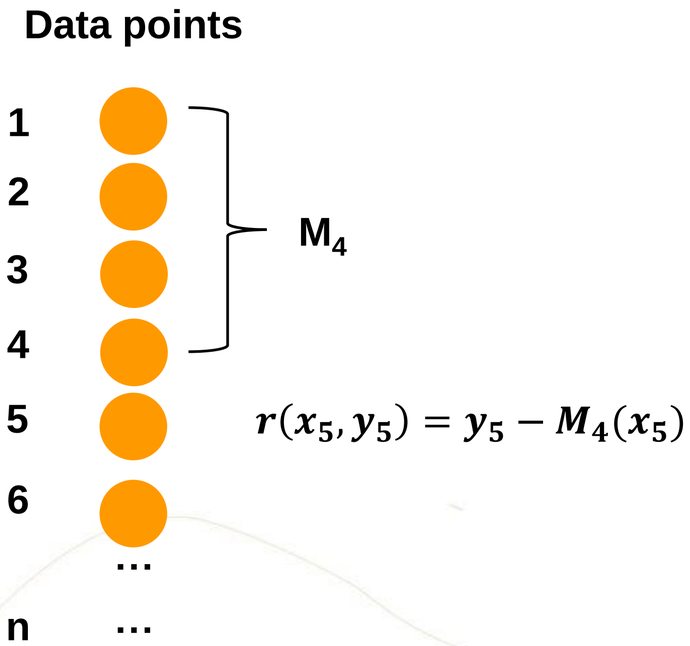
Classical Boosting

При классическом повышении мы подгоняем несколько деревьев, используя весь набор данных ($\mathbf{x}_n$). Это может привести к **переобучению**.



Упорядоченный бустинг (Ordered boosting)

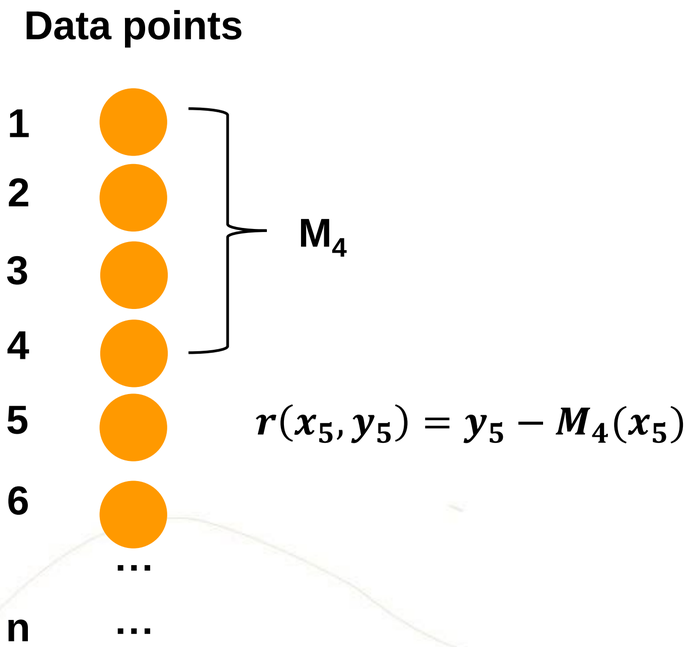
- Предположим, что модель M_i была обучена на первых i точках данных.
- Мы вычисляем остатки в каждой точке данных i, используя модель M_{i-1} (используйте дерево, которое раньше не видело эту точку данных)



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

Упорядоченный бустинг (Ordered boosting)

- Предположим, что модель M_i была обучена на первых i точках данных.
- Мы вычисляем остатки в каждой точке данных i, используя модель M_{i-1} (используйте дерево, которое раньше не видело эту точку данных)



- Нам нужно обучать **n** отдельных деревьев, (это невозможно).
- На практике мы работаем только с деревьями в точках 2^j, где j = 1, 2,…, $\log_2(n)$.

XGBoost vs LightGBM vs CatBoost

	XGBoost	LightGBM	CatBoost
Категориальные переменные (Categorical variables)	Должен быть предварительно закодирован **one-hot-encoded** или **target-encoded**	• Установите тип столбца как “category” во фрейме данных или • Используйте параметр **category_feature**	• Установите тип столбца как “category” во фрейме данных или • Используйте параметр **cat_features**
Отсутствующие значения (Missing values)	Направление ветвления изучено для пропущенных значений	Пропускает точку данных во время разделения, распределяет позже листьям	Для числовых пропущенных значений устанавливается минимальное значение для этой функции (default)

Final Project – Predict Pet Adoption Time

- You will working with **pet adoption** data from **Austin Animal Center**.
- We joined two datasets that cover **intake** and **outcome** of animals. Intake data is available from here and outcome is from here.
- We want you to predict **whether a pet is adopted within the 30 days** stay time in the animal center.
- We give you a starter notebook: **DTE-FINAL-PROJECT.ipynb**

Dataset schema:

Pet ID - Unique ID of pet

Outcome Type - State of pet at the time of recording the outcome

Sex upon Outcome - Sex of pet at outcome

Name - Name of pet

Found Location - Found location of pet before entered the center

Intake Type - Circumstances bringing the pet to the center

Intake Condition - Health condition of pet when entered the center

Pet Type - Type of pet

Sex upon Intake - Sex of pet when entered the center

Breed - Breed of pet

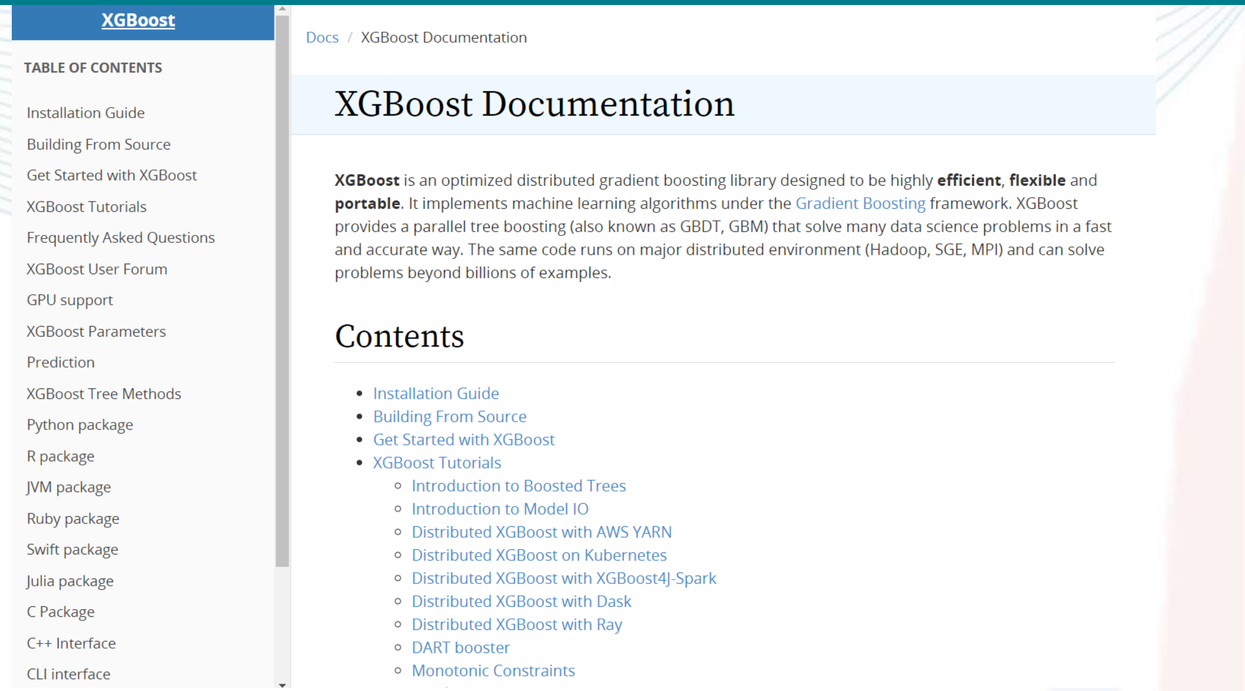
Color - Color of pet

Age upon Intake Days - Age of pet when entered the center (days)

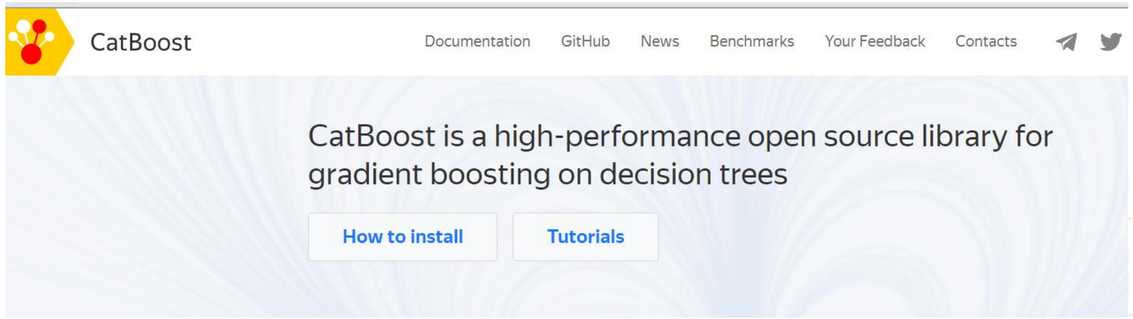
Time at Center - Time at center (0 = less than 30 days; 1 = more than 30 days). This is the value to predict.

Libraries-tools and licenses

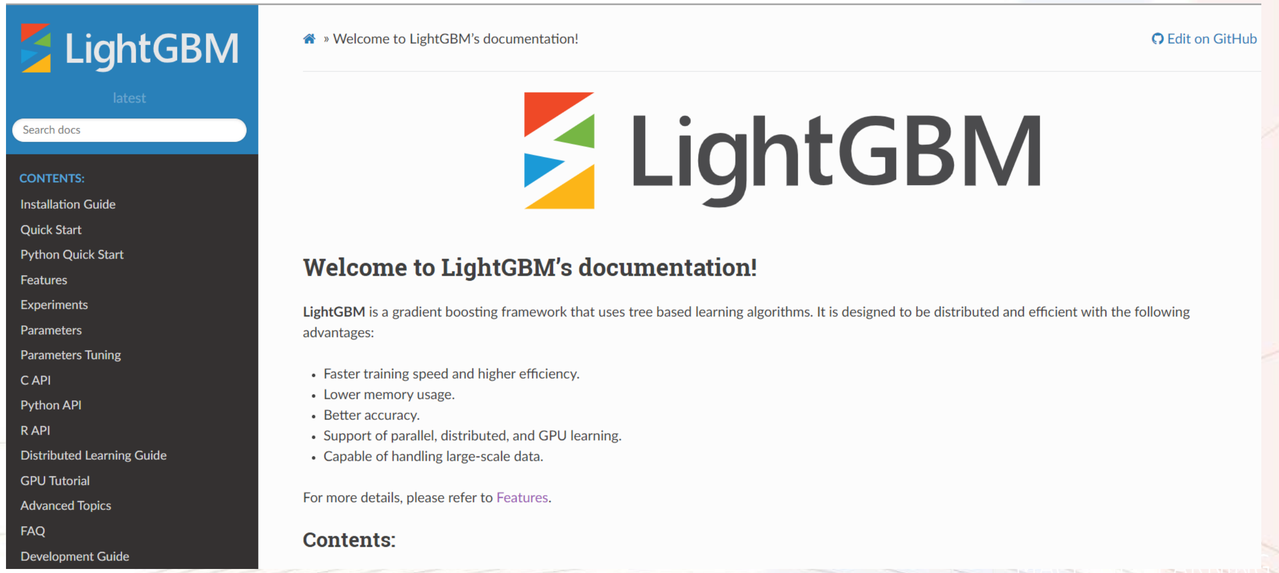
- Numpy: BSD
- Pandas: BSD
- Sagemaker: Apache license 2.0
- Seaborn: BSD
- Sklearn: BSD
- Matplotlib: BSD
- CatBoost: Apache license 2.0
- LightGBM: MIT
- XGBoost: Apache license 2.0
- MXNet: Apache license 2.0



https://xgboost.readthedocs.io/en/latest/index.html#



https://catboost.ai/



https://lightgbm.readthedocs.io/en/latest/Features.html