Занятие 10. Градиентный бустинг. XGBoost, Catboost, LightGBM

Колмагоров Евгений
ml.hse.dpo@yandex.ru

9 декабря 2024

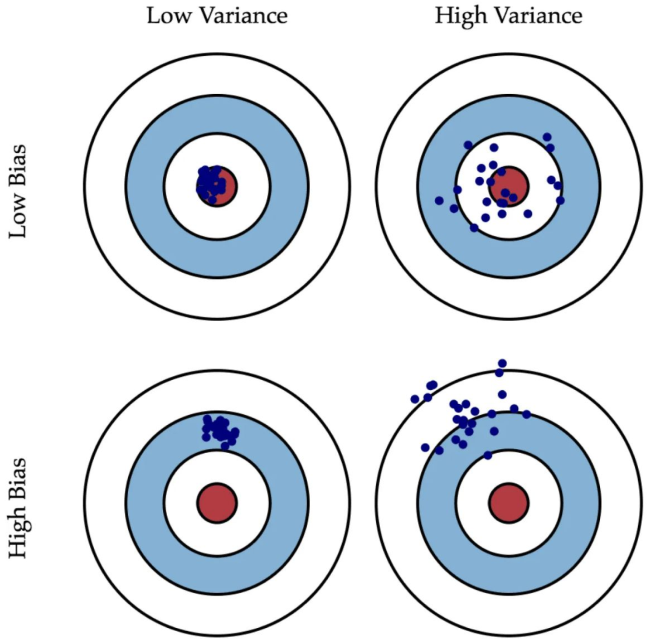
План лекции

1. Идея бустинга в машинном обучении
2. Аппроксимация градиента функции потерь
3. Настройка алгоритмов
4. Современные реализации: XGBoost, CatBoost, LigthGBM

Напоминание. Bias - Variance decomposition

На прошлом занятии была рассмотрена формула ошибка для любой задачи машинного обучения в виде декомпозиции состоящей из трёх частей:

- Смещение от целевой функции
- Дисперсию алгоритма
- Случайного шума в самих данных

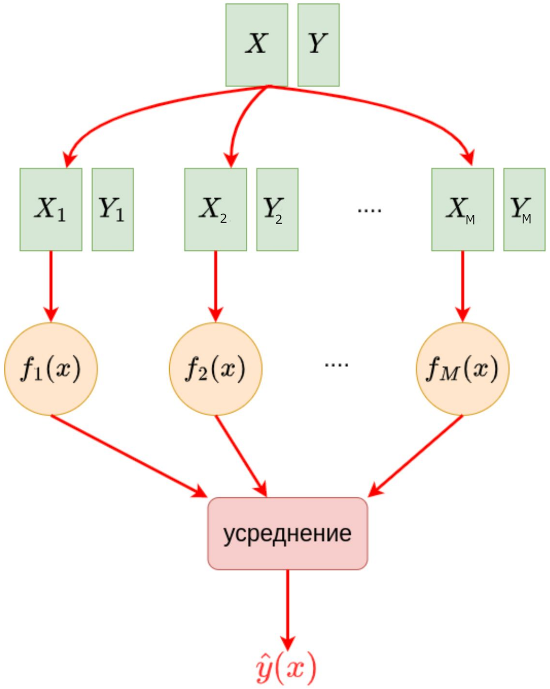


$$\mathbb{E}_X[Err] = bias^2_X(a(x,X)) + Var_X[a(x,X)] + \sigma^2$$

Напоминание. Построение независимых моделей

Для улучшения дисперсии модели предлагалось использовать ассамблирование, при этом обучая как можно более независимые алгоритмы $\{b_1(X_1, Y_1), \ldots, b_m(X_m, Y_m)\}$ и усредняя их ответы

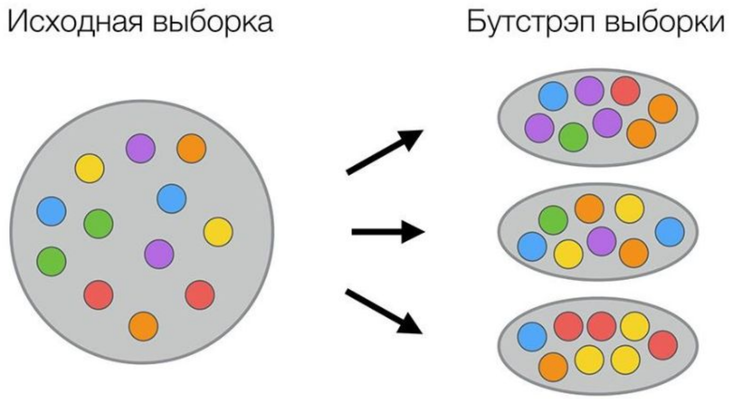
$$a(x) = \frac{1}{K} \sum_{j=1}^{K} b_j(x)$$



Напоминание. Построение независимых моделей

Для построения таких моделей использовался беггинг, который строит одинаковые модели на разных подвыборках исходных данных.

Беггинг улучшает дисперсию модели, но никак не меняет её смещение.



Вопрос: как можно улучшить смещение модели?

Знакомьтесь: бустинг

Бустинг – подход основанный на том, что строится последовательный набор моделей $\{b_0(X, Y), b_1(X, Y_1), ..., b_m(X, Y_m)\}$, в котором каждая последующая модель $b_{i+1}(x)$ учится исправлять ошибки предыдущей $b_i(x)$.

Предсказание строится как линейная комбинация M базовых алгоритмов:

$$a(x) = b_0(x) + c_1 \cdot b_1(x) \ldots + c_m \cdot b_m(x)$$

Наглядный пример

Предположим, что первая модель $b_0(x)$ ошибается на объекте x_l некоторую заданную величину, например, 10 относительно целевой переменной y_l:

- $b_0(x_l) = y_l + 10$

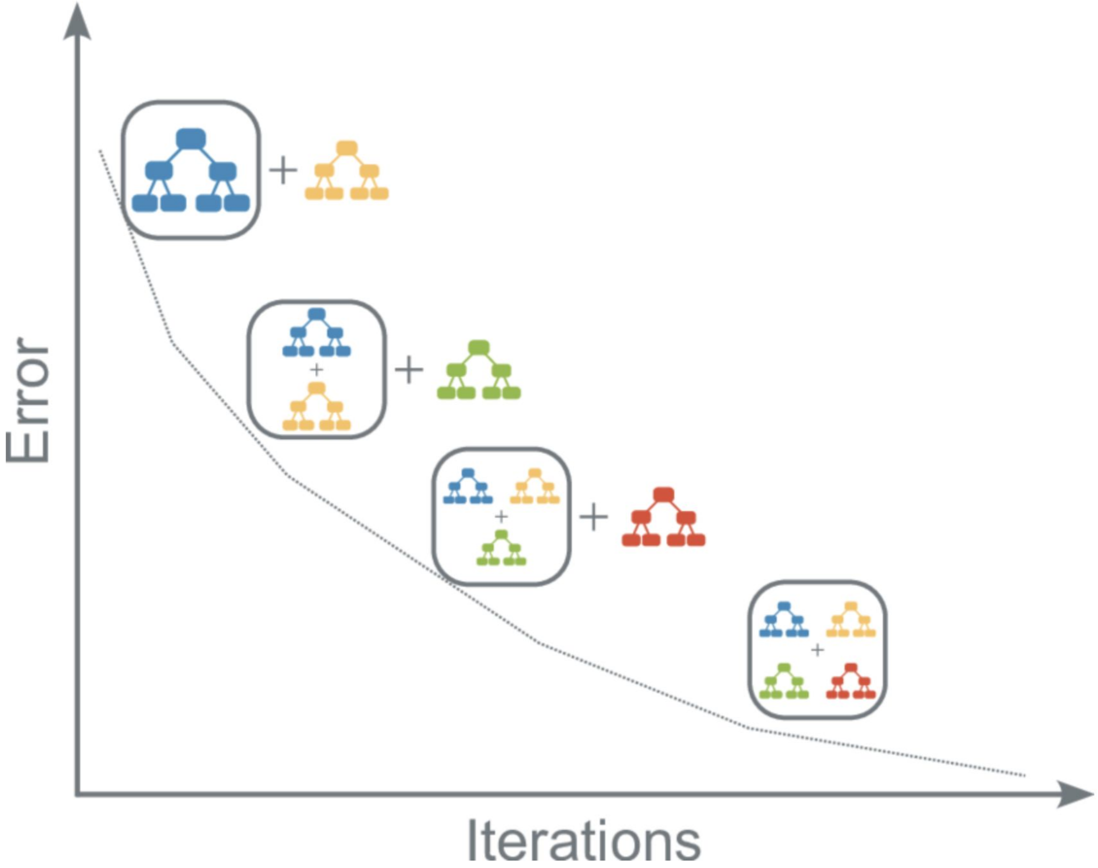
Теперь если обучить новую модель $\mathrm{b}_1(\mathrm{x})$ на то, чтобы она на x_1 объекте предсказывала ошибку $y_l - b_0(x_l) = -10$, то композиция

- $a(x_l) = b_0(x_l) + b_1(x_l) = (y_l + 10) + (-10) = y_l$

Таким образом композиция предсказаний будет давать нулевую ошибку!

Визуализация пример работы

При увеличении числа базовых моделей ошибка на обучении непреклонно уменьшается



Бустинг в задаче регрессии

Решаем задачу регрессии с минимизацией квадратичной ошибки:

$$L(a, y) = \frac{1}{2} \sum_{i=1}^{N} (a(x_i) - y_i)^2 \to min_a$$

Ищем алгоритм a(x) в виде суммы m базовых алгоритмов:

$$a(x) = \sum_{k=1}^{m} b_k(x)$$

Все базовые алгоритмы $b_k(x)$ принадлежат одному семейству A

Бустинг в задаче регрессии

- Шаг 1: Ищем алгоритм $b_0(x)$, минимизирующий ошибку:

$$b_1(x) = argmin_{b \in A} \frac{1}{2} \sum_{i=1}^{N} (b(x) - y_i)^2$$

- Вычислим ошибку на объекте x:

$$s = y - b_1(x)$$

Бустинг в задаче регрессии

- Шаг 1: Ищем алгоритм $b_0(x)$, минимизирующий ошибку:

$$b_1(x) = argmin_{b \in A} \frac{1}{2} \sum_{i=1}^{N} (b(x) - y_i)^2$$

- Вычислим ошибку на объекте x:

$$s^1 = y - b_1(x)$$

Следующий алгоритм будет настраиваться на эту ошибку, т.е. целевая переменная для следующего алгоритма – это вектор ошибок s^1: $s^1 = (s^1_1, s^1_2, \ldots, s^1_N)$

Бустинг в задаче регрессии

- Шаг 2: Ищем алгоритм $b_2(x)$, настраивающийся на ошибки первого алгоритма:

$$b_2(x) = argmin_{b \in A} \frac{1}{2} \sum_{i=1}^{N} (b(x) - s_i^1)^2$$

- Вычисляем ошибку $b_2(x)$:

$$s^2 = s^1 - b_2(x) = \{s^1 = y - b_1(x)\} = y - (b_1(x) + b_2(x))$$

Таким образом ошибка s^2 есть ошибка композиции алгоритмов $b_1(x)$ и $b_2(x)$

Бустинг в задаче регрессии

- Шаг K: Ищем алгоритм $b_k(x)$, настраивающийся на ошибки $b_{k-1}(x)$ алгоритма:

$$b_k(x) = argmin_{b \in A} \frac{1}{2} \sum_{i=1}^{N} (b(x_i) - s_i^{k-1})$$

- Вычисляем ошибку $b_k(x)$:

$$s^k = s^{k-1} - b_k(x) = y - \sum_{j=1}^{k} b_j(x) = y - a_k(x)$$

Об одном свойстве бустинга

Посчитаем производную функции потерь по предсказанию $z = a_k(x)$:

$$\frac{\partial L(z,y_i)}{\partial z}\Big|_{z=a_k(x)} = \frac{\partial}{\partial z}\frac{1}{2}(y - z)^2\Big|_{z=a_k(x)} = y - a_k(x)$$

Можно заметить, что ошибка, на которую обучается k+1 базовый алгоритм, выражается через производную:

$$s^k = y_i - a_k(x) = -\frac{\partial L(z,y_i)}{\partial z}\Big|_{z=a_k(x)}$$

Об одном свойстве бустинга

Посчитаем производную функции потерь по предсказанию $z= a_k(x)$:

$$\frac{\partial L(z,y_i)}{\partial z}\Big|_{z=a_k(x)} = \frac{\partial}{\partial z}\frac{1}{2}(y-z)^2\Big|_{z=a_k(x)} = y - a_k(x)$$

Можно заметить, что ошибка, на которую обучается k+1 базовый алгоритм, выражается через производную:

$$s^k = y_i - a_k(x) = -\frac{\partial L(z,y_i)}{\partial z}\Big|_{z=a_k(x)}$$

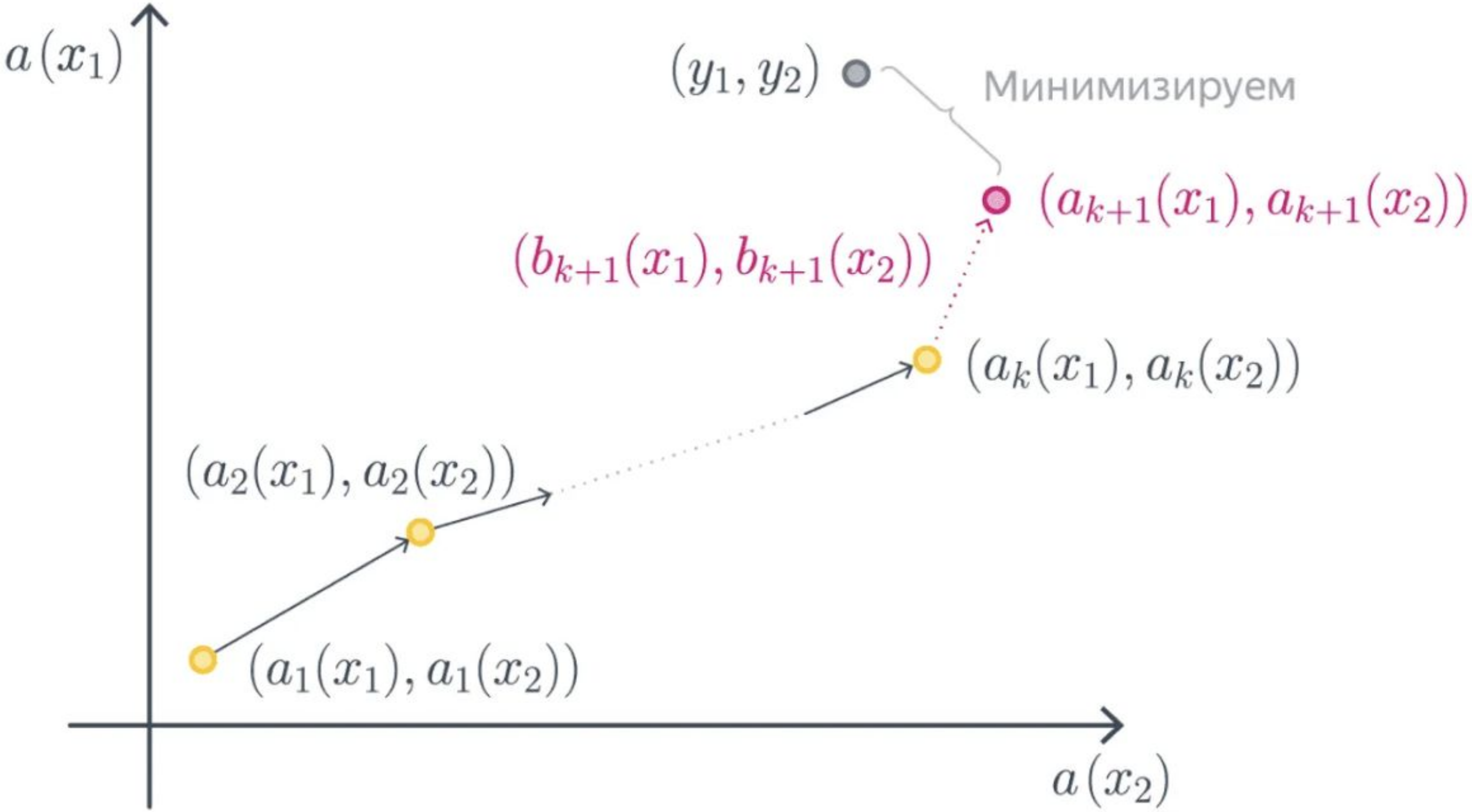
Таким образом очередной алгоритм в бустинге обучается предсказывать антиградиент функции потерь в точке $a_k(x)$, что соответствует текущей композиции базовых моделей. Именно поэтому бустинг называют градиентным

Основная теорема бустинга

Утверждение: Ошибка на K-ом шаге для квадратичной функции потерь – это антиградиент функции потерь по ответу модели, вычисленной в точке ответа уже построенной композиции

$$s_i^k = y_i - a_k(x_i) = -\frac{\partial}{\partial z}\frac{1}{2}(z-y)^2\Big|_{z=a_k(x)}$$

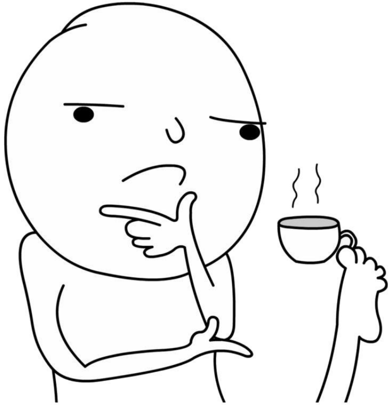
Визуализация работы



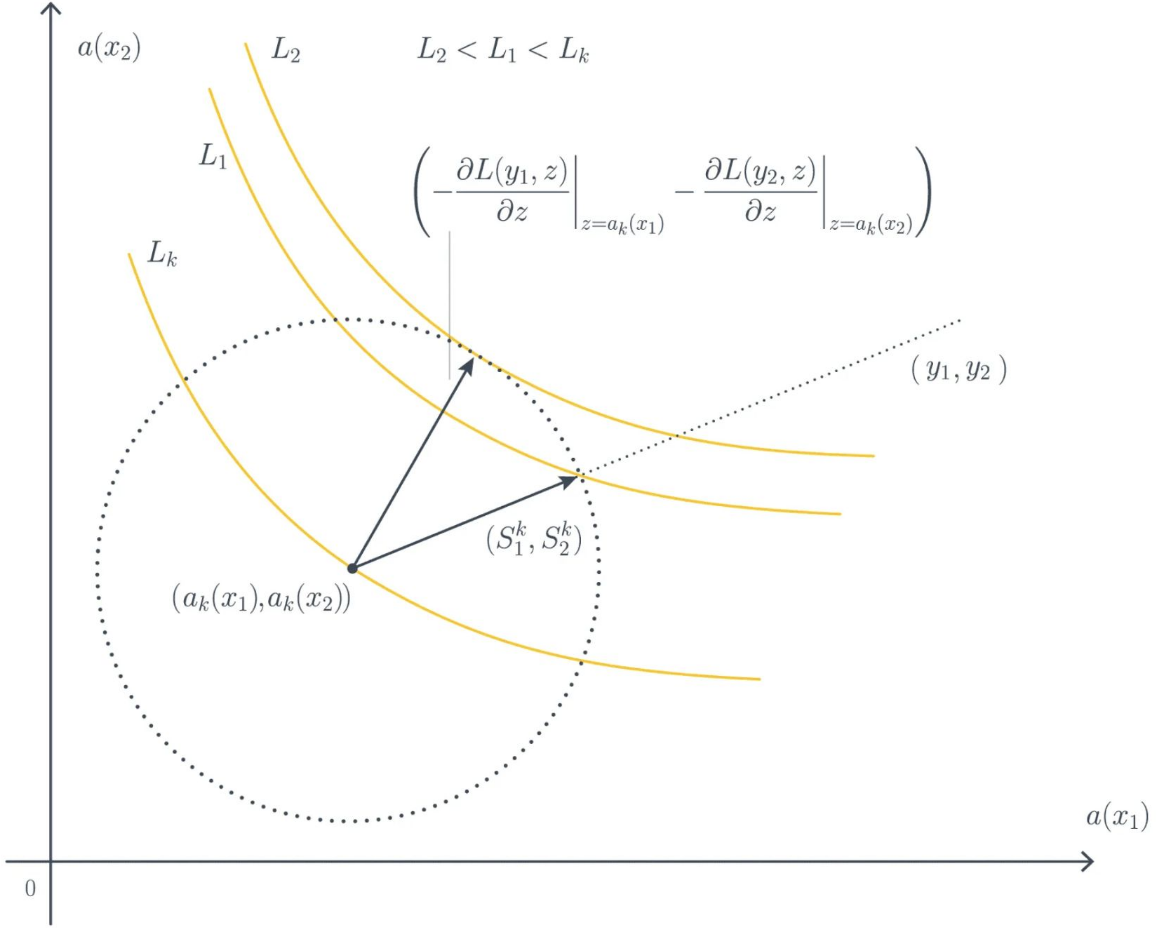
Более общий случай

В предыдущем примере вся математика основывалась на том допущении, что в качестве функции потерь была использована среднеквадратичная ошибка.

В более общем случае такого совпадения ошибки и антиградиента нет, но тем не менее приближать антиградиент имеет смысл и вот почему…



Антиградиент и приближение ошибки



Иной взгляд

Так как композиция алгоритмов строится последовательно, то

$$a_k(x) = a_{k-1}(x) + b_k(x)$$

Обучение $b_k(x)$ производится так, чтобы улучшить ответы текущей композиции:

$$b_k(x) = argmin_{b \in A} \sum_{i=1}^{N} L(y_i, a_{k-1}(x) + b(x_i))$$

Разложим $L(y_i, a_{k-1}(x_i) + b(x_i))$ в ряд Тейлора до первого члена в окрестности точки $(y_i, a_{k-1}(x))$:

$$L(y_i, a_{k-1}(x_i) + b(x_i)) \approx L(y_i, a_{k-1}(x_i)) + b(x_i) \frac{\partial L(y_i, z)}{\partial z}\Big|_{z=a_{k-1}(x_i)} =$$
$$= L(y_i, a_{k-1}(x_i)) + b(x_i) g_i^{k-1}$$

Иной взгляд

Так как член *L(*y_i, a_{k-1}*(*x_i*))* никак не зависит от b(x), то

$$b_k = argmin_{b \in A} \sum_{i=1}^{N} b(x_i) g_i^{k-1}$$

Минимум такого скалярного произведения будет достигаться векторе **b**, который равен антиградиенту:

$$b_k = (-g_1^{k-1}, -g_2^{k-1}, \ldots, -g_N^{k-1})$$

Темп обучения

Чтобы уменьшить возможность переобучения добавляют шаг смещения в сторону градиента γ:

$$a_k(x) = a_{k-1}(x) + \gamma_k \cdot b_k(x)$$

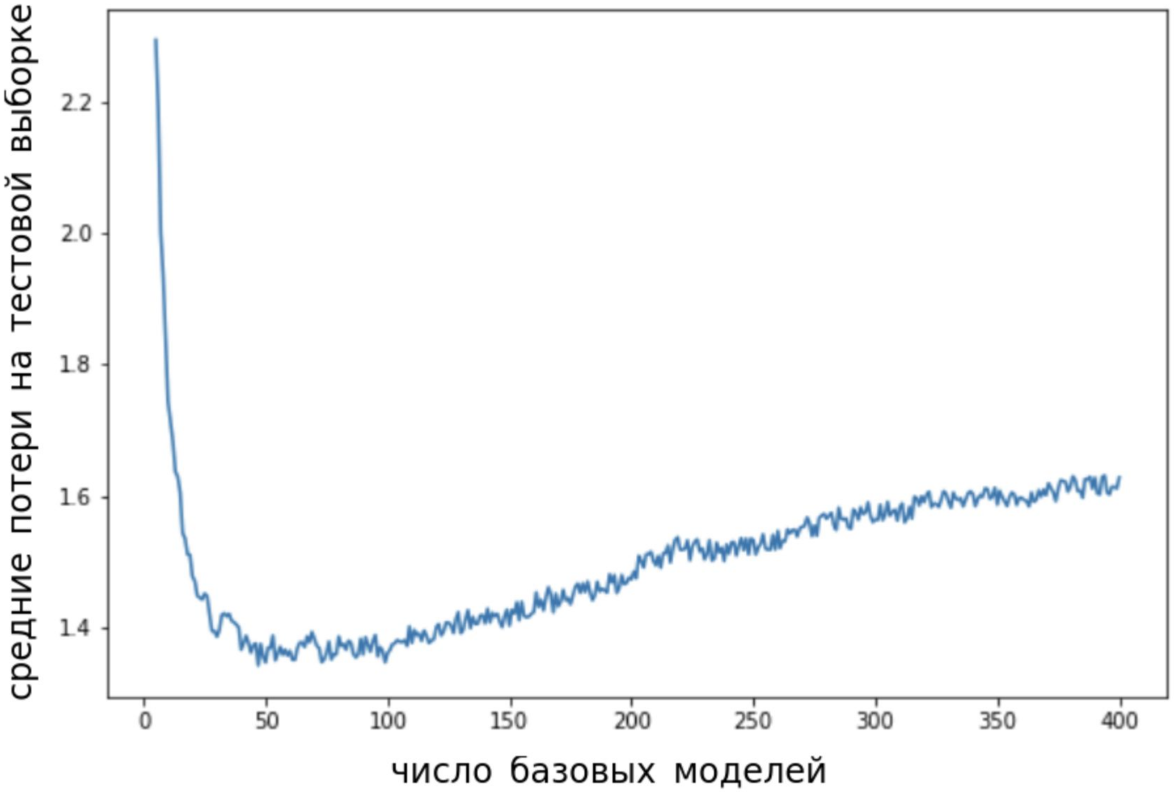
При этом шаг градиента можно подбирать так , чтобы

$$\gamma_k = argmin_{\gamma \in R} \sum_{i=1}^{N} L(y_i, a_{k-1}(x_i) + \gamma b_k(x))$$

И формула итогового предсказания выглядит как

$$a(x) = b_1(x) + \gamma_2 b_2(x) + \ldots + \gamma_K b_K(x)$$

Переобучение бустинга



В отличие от беггинга бустинг может переобучаться!

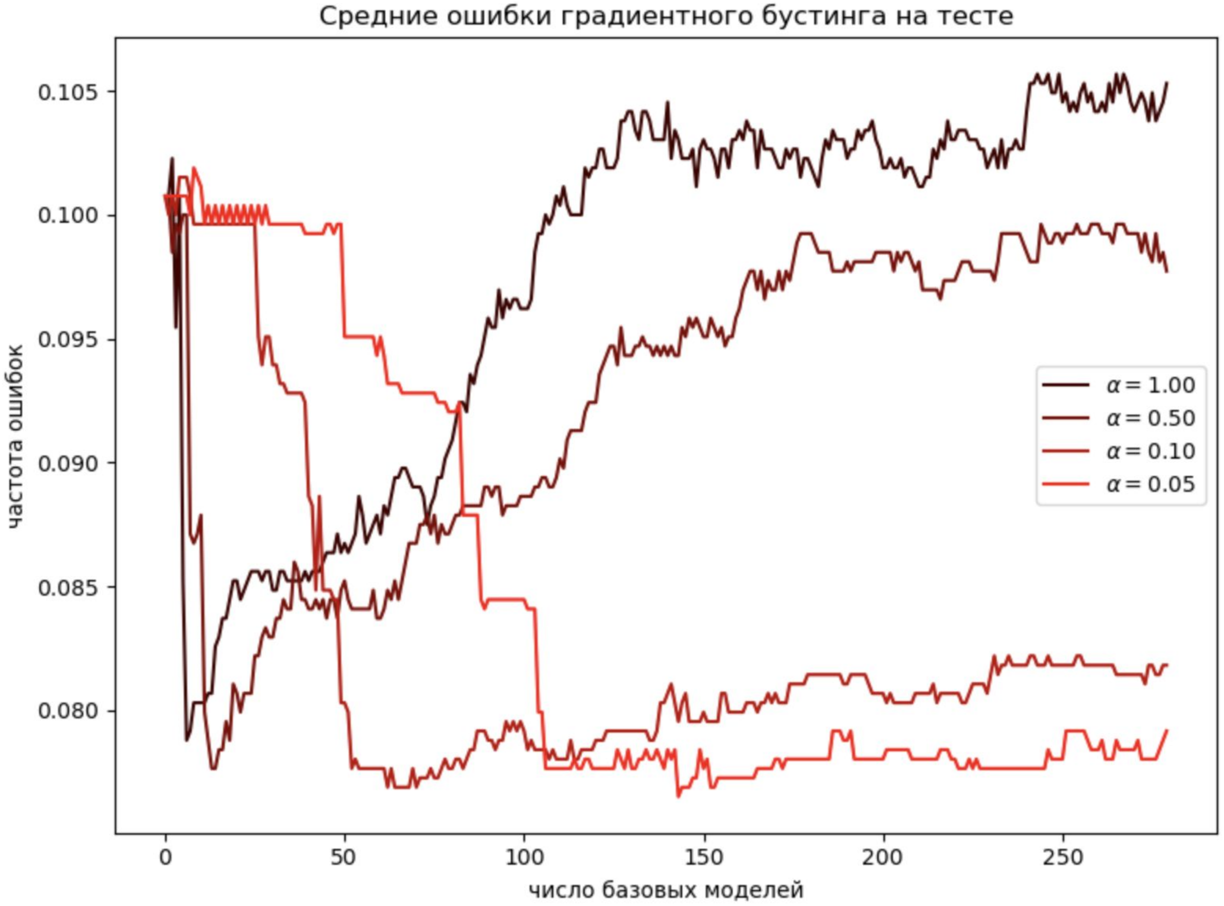
Сжатие (shrinkage)

В качестве дополнительного средства регуляризации добавляют гиперпараметр $\alpha \in (0, 1]$, который несёт смысл уровня доверия новому алгоритму:

$$a_k(x) = a_{k-1}(x) + \alpha \cdot \gamma \cdot b_k(x)$$

чем выше его значение тем больший вклад новый алгоритм вносит в общую композицию

Влияние сжатия на ошибку



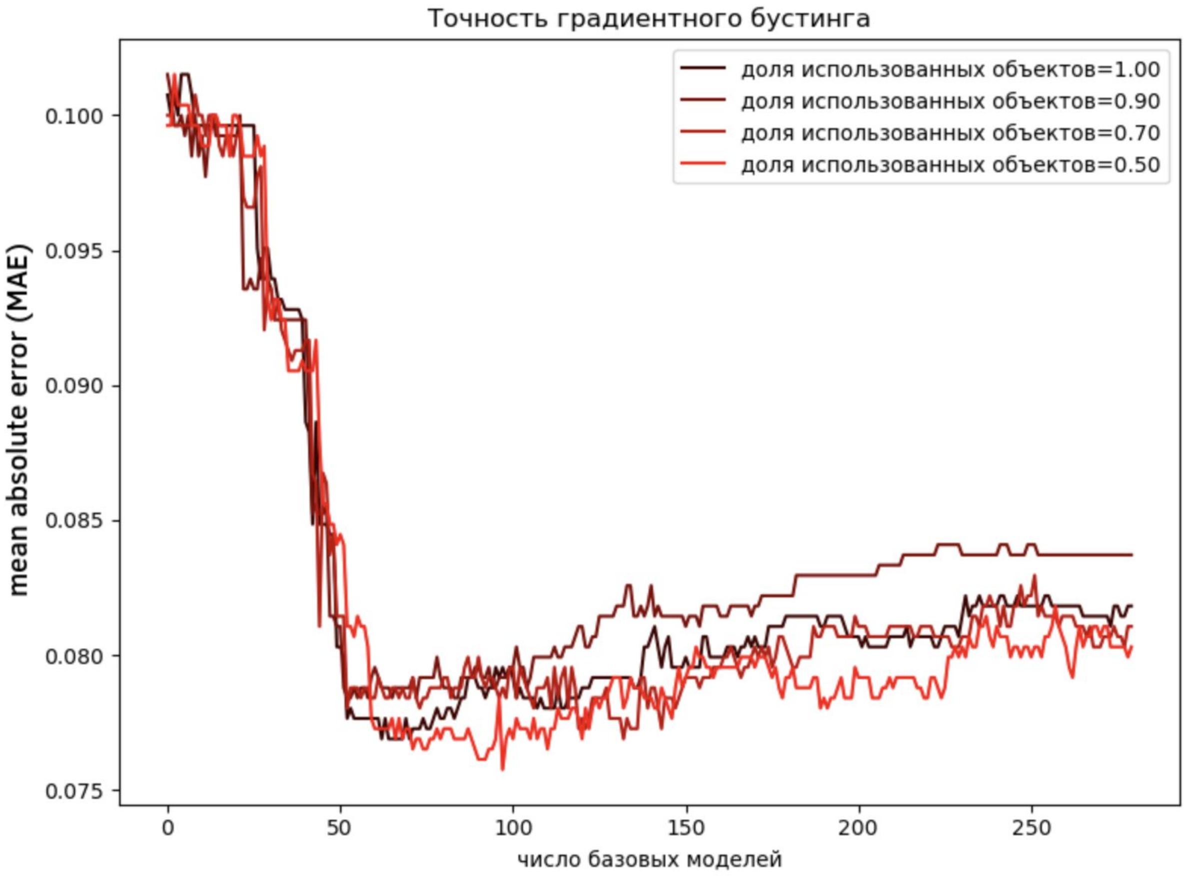
Стохастический градиентный бустинг

Обучаем базовый алгоритм b_K не по всей выборке X, а по случайной подвыборке $X^k \in X$:

- Более быстрое построение базовых моделей
- Базовые модели становятся более разнообразными, что повышает итоговое качество работы
- Можно вычислять Out-of-bag ошибку

Обычно берут $|X^k| = ½ |X|$

Влияние размера семпла на качество обучения



Выбор базовых алгоритмов

При выборе семейства базовых алгоритмов возникают следующие вопросы:

- Что произойдёт с предсказаниями бустинга, если базовые алгоритмы слишком простые?
- Что будет, если базовые алгоритмы наоборот слишком сложные?

Вопрос: Почему в качестве базового алгоритма не стоит брать линейную функцию?

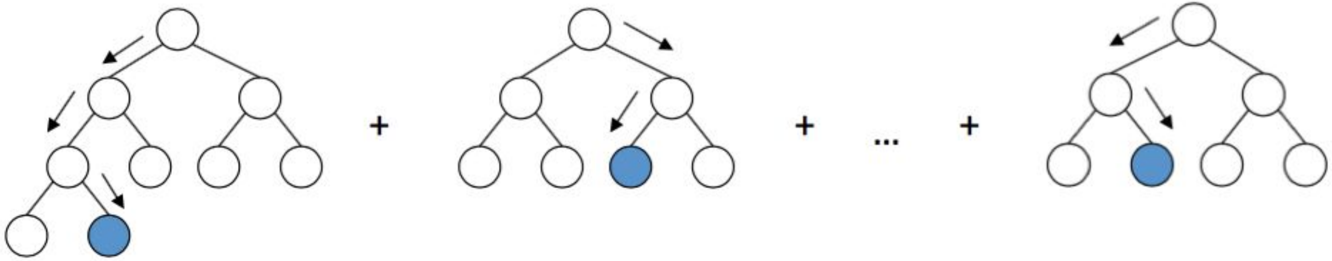
Выбор базовых алгоритмов

- Если базовые алгоритмы очень простые, то они плохо приближают антиградиент антиградиент функции потерь, т.е. градиентный бустинг будет смещаться в слишком отличную от антиградиента сторону
- Если базовые алгоритмы слишком сложные, то за несколько шагов бустинг подгонится под обучающую выборку, и получится переобученный алгоритм

Деревья решений в качестве базовых алгоритмов

Чаще всего в качестве базовых алгоритмов используются решающие деревья небольшой глубины (в среднем глубины 3-6), чтобы иметь достаточную обобщающую способность и быть при этом не переобученными.

Бустинг, использующий деревья решений в качестве базовых алгоритмов, называется **градиентным бустингом над решающими деревьями (Gradient Boosting on Decision Trees, GBDT)**



Оценка градиента деревьями решений

При построении следующего решающего дерева для оценки качества приближения вектора антиградиента g на i-ом объекте используют следующие функции:

$$L_2(g,p) = \sum_{i=1}^{N} (p_i - g_i)^2 \,,$$

$$Cosine(g,p) = -\frac{\sum\limits_{i=1}^{N}(p_i \cdot g_i)}{\sqrt{\sum\limits_{i=1}^{N} p_i^2} \cdot \sqrt{\sum\limits_{i=1}^{N} g_i^2}},$$

, где p_i – предсказание на антиградиента на i-ом объекте, а его g_i истинное значение

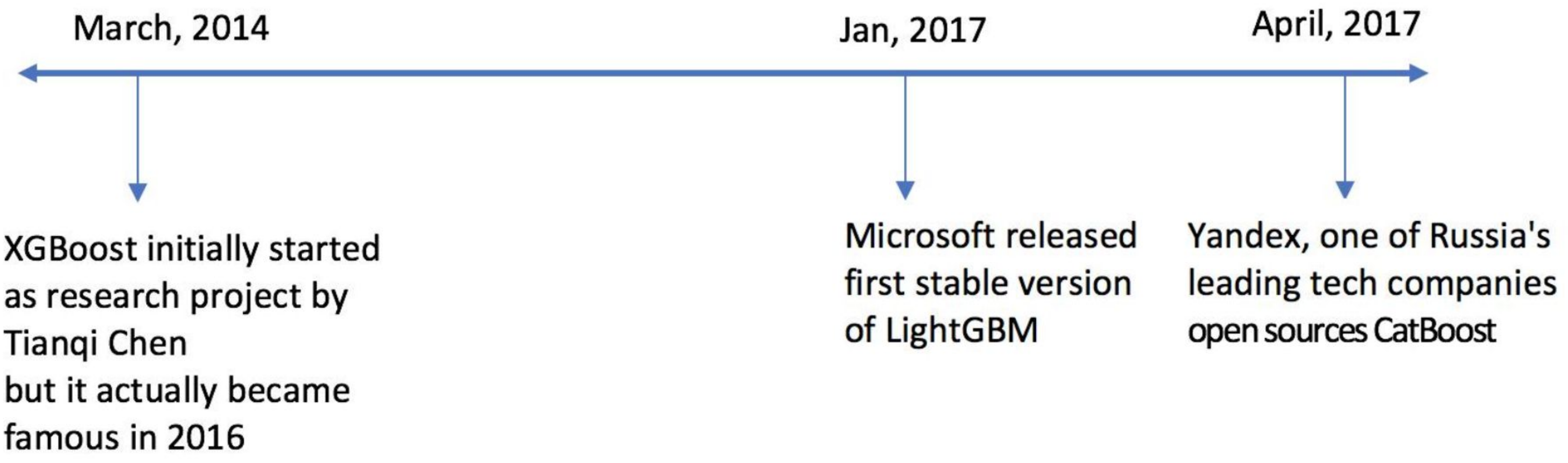
Популярные имплементации

При построении градиентного бустинга над решающими деревьями существует множество способов сделать композиции деревьев.

На сегодняшний день наиболее популярны следующие реализации:

- XGBoost
- CatBoost
- LightGBM

XGBoost, Catboost & LightGBM



XGBoost: https://github.com/dmlc/xgboost
CatBoost: https://github.com/catboost
LightGBM: https://github.com/catboost

XGBoost (eXtreme Gradient Boosting)

В данном подходе функционала ошибки раскладывается в ряд Тейлора до второго члена:

$$L(y_i, a_{k-1}(x_i) + b(x_i)) \approx L(y_i, a_{k-1}(x_i)) + b(x_i) \frac{\partial L(y_i,z)}{\partial z}\Big|_{z=a_{k-1}(x_i)} + \frac{b^2(x_i)}{2} \frac{\partial^2 L(y_i,z)}{\partial z^2}\Big|_{z=a_{k-1}(x_i)}$$

$$g_i^{k-1} = \frac{\partial L(y_i,z)}{\partial z}\Big|_{z=a_{k-1}(x_i)}$$

$$h_i^{k-1} = \frac{\partial^2 L(y_i,z)}{\partial z^2}\Big|_{z=a_{k-1}(x_i)}$$

Оптимальный базовый алгоритм ищется, как

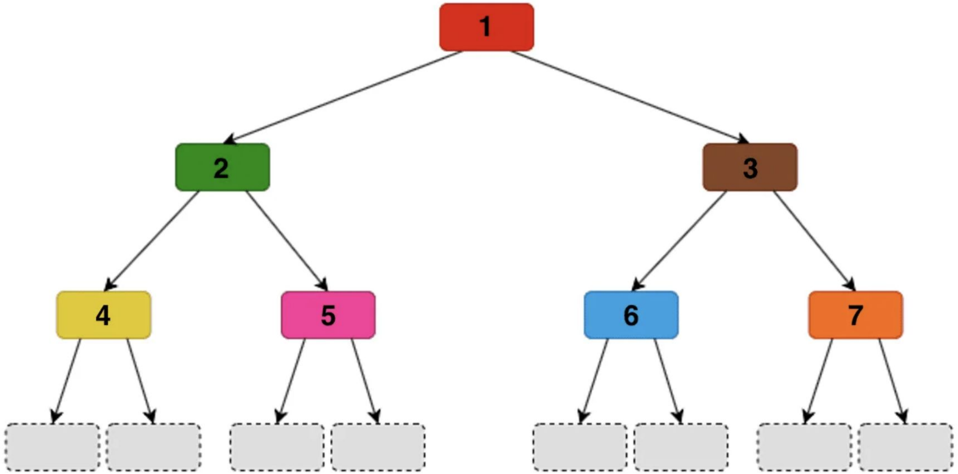
$$b_k(x) = argmin_{b \in A} \sum_{i=1}^{N} [b(x_i) \cdot g_i^{k-1} + \frac{1}{2} b(x_i) \cdot h_i^{k-1}]$$

XGBoost (eXtreme Gradient Boosting)

+ добавляется регуляризационные члены на структуру решающих деревьев

$$b_k(x) = argmin_{b \in A} \sum_{i=1}^{N} [b(x_i) \cdot g_i^{k-1} + \frac{1}{2} b(x_i) \cdot h_i^{k-1}] + \mu T + \frac{\lambda}{2} \sum_{i=1}^{T} b_i^2(x)$$

+ Деревья строятся слой за слоем последовательно до достижения максимальной глубины



XGBoost (eXtreme Gradient Boosting)

$$b_k(x) = argmin_{b \in A} \sum_{i=1}^{N} [b(x_i) \cdot g_i^{k-1} + \frac{1}{2} b(x_i) \cdot h_i^{k-1}] + \mu T + \frac{\lambda}{2} \sum_{i=1}^{T} b_i^2(x)$$

Основные особенности:

- Приближает направление сдвига с учётом второй производной
- Добавляется регуляризация, которая штрафует за число листьев дереве
- Устойчив к переобучению, так как пытается строить дерево минимальной глубины

CatBoost (Categorical Boosting)

Развитие XGBoost подхода разработанное в Яндексе, и в отличии от последнего способен обрабатывать категориальные признаки, и имеет дополнительные встроенные инструменты улучшающие качество обучения

Yandex
CatBoost

Особенности CatBoost

- Используются симметричные деревья с одним и тем же решающим правилом на одном и том же уровне

Особенности CatBoost

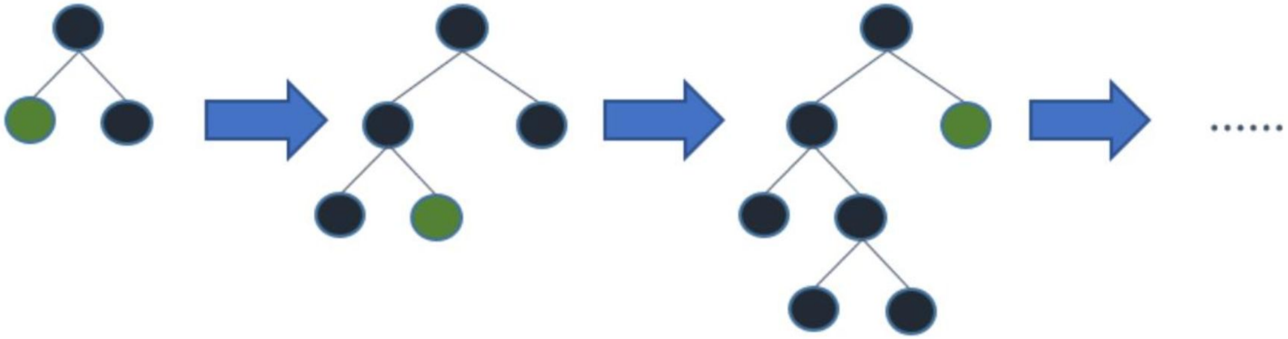
- При кодировании категориальных признаков используются набор методов: one-hot encoding, счётчики, комбинации признаков и тд.

Особенности CatBoost

- Поддержка пропусков в данных
- Обучается быстрее, чем xgboost
- Показывает хороший результат даже без тонкой настройки гиперпараметров
- Есть детекция переобучения, вычисление значений метрик, встроенная кросс-валидация

LightGBM (Light Gradient Boosting Machine)

LightGBM строит деревья, добавляя на каждом шаге один лист, и позволяет добиться более высокой точности работы.



Leaf-wise tree growth

LightGBM (Light Gradient Boosting Machine)

Ключевые особенности LightGBM:

- Как правило деревья решений имеет несимметричную форму
- Хорошо оптимизирован и не требует большого количества вычислительных ресурсов
- Может работать с категориальными признаками, и делает разбиение их на два подмножества достаточно эффективно за O(k*log k) операций

Выводы градиентный бустинг

- Позволяет существенно улучшить смещение базовой модели
- В отличии от беггинга требует более аккуратной настройки, так как может переобучаться
- Из-за последовательного построения базовых алгоритмов сложнее распределённо обучать базовые алгоритмы
- На текущий момент алгоритмы градиентного бустинга показывают наилучшее качество при обработке табличных данных, выигрывая даже у нейросетевых подходов
- Имеет различные эффективно работающие реализации на языке Python

Выводы ансамблирование

	разброс (model's variance)	смещение (model's bias)	функциональная выразимость	основа техники
Bagging «среднее»	уменьшает			bootstrap
Boosting «взвешенное среднее»		уменьшает	(увеличивает)	градиентный спуск (сейчас)
Stacking Мета-алгоритм	(уменьшает)	(уменьшает)	увеличивает	суперпозиция алгоритмов