ГРАДИЕНТНЫЙ БУСТИНГ

Предположим, что решается задача машинного обучения (для простоты можно считать, что регрессии) с обучающей выборкой $(x_i, y_i)_{i=1}^m$ и дифференцируемой функцией ошибки $L(y,a)$. Мы построили алгоритм $a(x)$, давайте теперь построим алгоритм $b(x)$ такой, что

$$a(x_i) + b(x_i) = y_i,\ i \in \{1,2,\ldots,m\}, \qquad (0)$$

(с точностью до ошибок алгоритма b). Такой алгоритм осуществляет поправку ответов алгоритма $a(x)$ до верных ответов на обучающей выборке, т.е. на **невязку (residual)** $\varepsilon_i = y_i - a(x_i)$. Заметим, что равенство (0) не будет, скорее всего, точно выполнено, поскольку алгоритмы мы ищем в рамках каких-то моделей.

Главный вопрос здесь – как обучить второй алгоритм. Вообще говоря, нельзя считать, что мы получили новую задачу с обучающей выборкой $(x_i, \varepsilon_i)_{i=1}^m$ и функцией ошибки $L(y,a)$, поскольку наша задача

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

это может отличаться от решения

$$\sum_{i=1}^m L(y_i - a(x_i), b(x_i)) \to \min. \qquad (2)$$

Хотя, справедливости ради, заметим, что для большинства разумных функций ошибок записи (1) и (2) эквивалентны.

Не всегда (1) решается аналитически (из-за достаточно сложных функций ошибок). Перепишем (1) в таком виде

$$F = \sum_{i=1}^m L(y_i, a(x_i) + b_i) \to \min_{(b_1,\ldots,b_m)}, \qquad (3)$$

если рассматривать эту задачу как задачу минимизации функции $F(b_1,\ldots,b_m)$, то полезно вспомнить, что функция многих переменных максимально убывает в направлении своего антиградиента:

$$-(L'(y_1, a(x_1)),\ldots, L'(y_m, a(x_m))), \qquad (4)$$

здесь производная берётся по второму аргументу. Получается, что выгодно считать

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

а это и есть ответы нашего алгоритма b. Получается, что его следует настраивать на обучающей выборке

$$(x_i, -L'(y_i, a(x_i)))_{i=1}^m. \qquad (5)$$

Из выражения (5) понятно будет название – **градиентный бустинг**.

Замечание. Задачу (5) можно решать любыми методами (и с любой функцией ошибки). Часто выбирают простые методы и функции. При этом минимизация исходной функции ошибки (3) как раз гарантируется нашим целевым вектором, который совпадает её антиградиентом.

Итак, мы научились строить алгоритм b, который корректирует ответы алгоритма a. Понятно, что и сумма этих алгоритмов может иметь значительную ошибку, но можно построить третий алгоритм, который корректирует эту сумму. Процесс построения можно продолжить. Получаем классический алгоритм градиентного бустинга

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 $b_t(x)$ на выборке

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

и $a_{t+1}(x) = a_t(x) + b_t(x)$.

Важный частный случай

Рассмотрим отдельно случай

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

Заметим, что тогда

$$L'(y,a)=-(y-a).$$

И алгоритм $b_t(x)$ обучается на выборке

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

т.е. пытается просто реализовать поправки к ответам текущего алгоритма $a_t(x)$[1].

Задача классификации

Рассмотрим задачу бинарной классификации с метками $Y=\{-1,+1\}$, здесь неочевидно, какую функции ошибки выбрать. Стандартные функции, например, процент верных ответов не являются дифференцируемыми. Кроме того, они ориентированы на ответ алгоритма, который сам лежит во множестве Y. Предположим, что алгоритма может выдавать любое вещественное значение, которое мы преобразуем в метку по следующему правилу

$$\begin{cases}+1, & a\geq 0,\\ -1, & a<0.\end{cases}$$

Функция ошибки такого алгоритма

$$L(y,a)=I[ya<0]$$

(предполагаем, что вероятность выполнения равенства $a=0$ нулевая).

[1] **2do: вот тут-то принципиально MSE!!!**

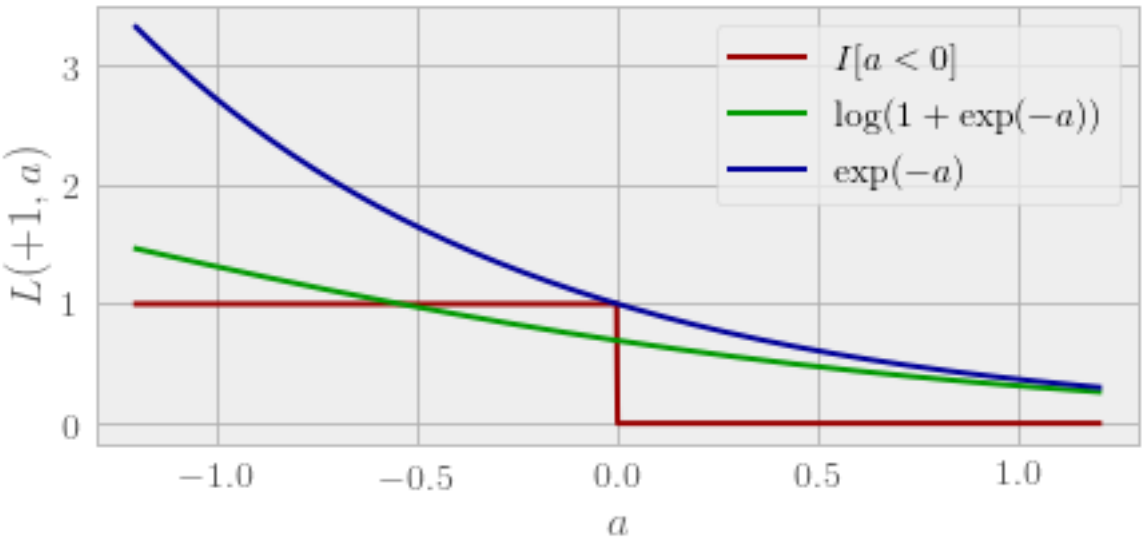

Рис. Разные функции ошибки в задачах бинарной классификации

На рис. показана функция ошибки $L(+1,a)$, она недиффиринцируема, но есть дифференцируемые функции «похожие на неё». Чаще всего используют логистическую ошибку

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

тогда

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

Иногда используют функцию ошибки типа Adaboost:

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

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

Обратите внимание на рис., что ошибка ненулевая и при правильной классификации, но чем увереннее алгоритм в своём правильном ответе, тем ошибка меньше.

Проблемы примитивного варианта бустинга

Заметим, что шагая в сторону антиградиента минимизируемой функции $F(b_1,\ldots,b_m)$, мы всё равно сразу не попадаем в глобальный минимум. Именно этим и оправдана итерационная природа алгоритма.

Также важно отметить, что в итоге (после решения задачи (5)) мы шагнём не в направлении антиградиента, а в близком направлении вектора ответов нашего алгоритма b, т.е. в чистом виде алгоритм градиентного спуска не реализуется.

Наискорейший спуск

Один из способов улучшения примитивной версии градиентного бустинга заключается в том, что после построения $b_t(x)$ можно попробовать решить такую задачу одномерной оптимизации:

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

т.е. мы не просто добавляем к $a_t(x)$ новый алгоритм, который обучили, используя антиградиент функции ошибок, а пытаемся определить, с каким коэффициентом его лучше добавить. На практике это часто можно сделать перебором. Заметим, что в этом случае

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

(т.е. итоговый алгоритм является не суммой базовых, а их линейной комбинацией).

Эвристика сокращения – Shrinkage

Поскольку на элементы антиградиента (4) мы всё равно настраиваемся с какой-то точностью, получается, что при минимизации (3) мы используем не антиградиент, а лишь вектор на него похожий. Поэтому не стоит смещать аргумент в F на весь вектор: умножим его не некоторый коэффициент $\eta \in (0,1]$, т.е.

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

Параметр $\eta \in (0,1]$ называют **скоростью (темпом) обучения** (**learning rate**).

Стохастический градиентный бустинг (Stochastic gradient boosting)

Можно перенести в бустинг идею бэгинга Бреймана: каждый новый алгоритм настраивать на подвыборке обучающей выборки. Объём подвыборки регулируется параметром $\delta \in (0,1]$ и равен

$$\lfloor \delta \cdot m \rfloor.$$

Обучение на небольших подвыборках:

- может улучшить качество ансамбля,
- уменьшает время построения базовых алгоритмов (у каждого уменьшается объём обучения),
- позволяет вычислять out-of-bag-ошибки и ответы алгоритма.

TreeBoost – градиентный бустинг над деревьями

Рассмотрим любой кусочно-постоянный алгоритм, в частности, таким алгоритмом является решающее дерево:

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

т.е. признаковое пространство объектов делится на конечное семейство областей $\{X_j\}$, в области X_j алгоритм выдаёт ответ β_j. В случае, если $b(x)$ решающее дерево, каждому его листу соответствует своя область. Заметим, что при обучении такого алгоритма можно независимо для каждой области выбирать значение β_j минимизируя функцию ошибки. Тогда (1) формально переписывается как

$$\sum_{i=1}^{m} L(y_i, a(x_i) + \sum_j \beta_j I[x \in X_j]) \to \min. \qquad (6)$$

После того как семейство областей $\{X_j\}$ выбрано и зафиксировано, остаётся решить задачу минимизации (6) по переменным β_j. Её можно решать независимо для каждого множества X_j:

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

Замечание. В настоящее время градиентный бустинг применяют чаще как раз над решающими деревьями. Число листьев (терминальных вершин) в дереве определяет, на сколько областей дерево делит наше пространство объектов. Поскольку для каждой области мы подбираем своё значение алгоритма в ней, это число соответствует сложности модели и часто называется **level of interaction between variables**. В некоторых реализациях градиентного бустинга над деревьями нет этого параметра (числа областей), а сложность базовых алгоритмов регулируется глубиной дерева и ограничением на число объектов в

листьях и вершинах дерева, при которых ещё делаются расщепления (см. дальше).

Замечание. Задача минимизации (6) решается не только выбором β_j, но и построением нового дерева $b(x)$, т.е. также выбором областей X_j.

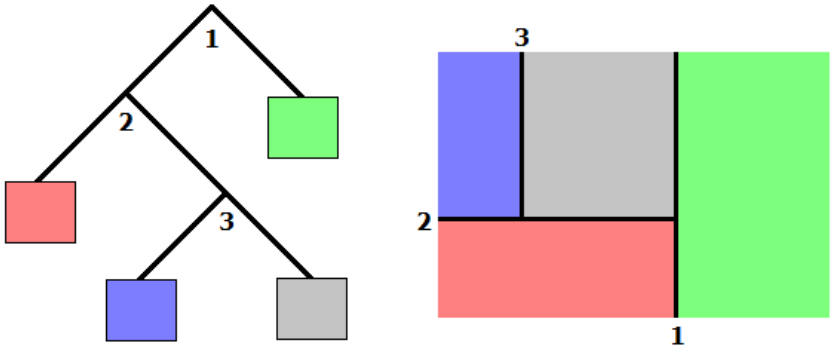

Рис. Пример как решающее дерево разбивает двумерное признаковое пространство на области

Продвинутые методы оптимизации

Рассмотрим нашу центральную задачу (3)

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

заметим, что

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

(здесь везде частные производные по второму аргументу функции ошибки). Поэтому наша задача минимизации сводится к задаче

$$\sum_{i=1}^{m} \left[g_i b_i + \frac{1}{2} h_i b_i^2 \right] \to \min, \qquad (7)$$

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

Сделаем оптимизацию с регуляризацией. Пусть дерево $b(x)$ делит пространство объектов на T областей $X_1,\ldots,X_T$, в каждой области X_j

принимает значение β_j. Добавим к (7) слагаемые, штрафующие за число областей, и квадраты значений

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

Нетрудно видеть, что

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

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

Отсюда в явном виде (приравнивая производную к нулю) получаем

$$\beta_j = -\frac{\sum\limits_{x_i \in X_j} g_i}{\sum\limits_{x_i \in X_j} h_i + \lambda}.$$

Заметим, что эти значения получены для фиксированного дерева (разбиения пространства объектов на области). Поэтому минимальное значение (при фиксированной структуре дерева)

$$\Phi_{\min} = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum\limits_{x_i \in X_j} g_i \right)^2}{\sum\limits_{x_i \in X_j} h_i + \lambda} + \gamma T \,. \qquad (8)$$

Эта формула может использоваться при построении дерева для его оценки. При этом очередное расщеплении в дереве при его итеративном построении можно выбирать минимизируя (8). Именно так делается в библиотеке XGBoost[2].

Замечание. Выше мы описали достаточно много математических формул. Как мы видим, всё это с целью минимизации заданной функции ошибки.

[2] https://github.com/dmlc/xgboost

Параметры градиентного бустинга

Перечислим параметры градиентного бустинга в современных реализациях (мы не будем привязываться к конкретной реализации):

0. Параметры, определяющие задачу:

- **`objective`** – какая задача решается и в каком формате будет ответ,
- **`loss`** – функция ошибки для минимизации,
- **`eval_metric`** – значения какой функции ошибки смотреть на контроле (как правило, задание этого параметра не означает, что эту функцию будем минимизировать при настройке бустинга).

Отметим, что иногда встречается странный на вид параметр **distribution (распределение)**, покажем, что он определяет. Предположим, что есть некоторая модель $h(x)$, которая описывает наши данные, при этом

$$p(x \mid y) = const \cdot e^{-\frac{(y-h(x))^2}{2\sigma^2}}, \qquad (1)$$

т.е. метка объекта x есть реализация случайной величины, распределённой по нормальному закону с матожиданием в $h(x)$. Если применить метод максимального правдоподобия, то получим

$$\prod_i p(x_i \mid y_i) \sim \prod_i e^{-\frac{(y_i-h(x_i))^2}{2\sigma^2}} \to \max,$$

здесь произведение идёт по номерам объектов обучения, волнистая черта – равенство с точностью до константы. Таким образом, необходимо решить такую задачу оптимизации

$$\sum_i (y_i - h(x_i))^2 \to \min,$$

т.е. минимизировать средний квадрат отклонения. Получили, что закон распределения меток относительно значения некоторой «идеальной модели» соответствует задаче минимизации с конкретной функцией ошибки.

Вопрос. Какая функция ошибки соответствует распределению Лапласа?

1. Параметры, определяющие тип бустинга и способы построения деревьев

- **booster** – какой бустинг проводить: над решающими деревьями или линейный,
- **grow_policy** – порядок построения дерева: на следующем шаге расщеплять вершину, ближайшую к корню, или на которой ошибка максимальна,
- **criterion** – критерий выбора расщепления (при построении деревьев),
- **init** – какой алгоритм использовать в качестве первого базисного (именно его ответы будет улучшать бустинг).

2. Основные параметры:

- **eta / learning_rate** – темп (скорость) обучения,
- **num_iterations / n_estimators** – число итераций бустинга,
- **early_stopping_round** – если на отложенном контроле заданная функция ошибки не уменьшается такое число итераций, обучение останавливается.

3. Параметры ограничивающие сложность дерева:

- **max_depth** – максимальная глубина,
- **max_leaves / num_leaves / max_leaf_nodes** – максимальное число вершин в дереве (иногда, если значение больше нуля, то игнорируется ограничение по максимальной глубине),
- **gamma / min_gain_to_split** – порог на уменьшение функции ошибки при расщеплении в дереве,
- **min_data_in_leaf / min_samples_leaf** – минимальное число объектов в листе,
- **min_sum_hessian_in_leaf** – минимальная сумма весов объектов в листе,
- **min_samples_split** – минимальное число объектов, при котором делается расщепление,

- **`min_impurity_split`** – расщепление при построении дерева не будет проведено, если impurity изменяется при расщеплении на величину меньшую этого попрога.

4. Параметры формирования подвыборок

- **`subsample / bagging_fraction`** – какую часть объектов обучения использовать для построения одного дерева,
- **`colsample_bytree / feature_fraction`** – какую часть признаков использовать для построения одного дерева,
- **`colsample_bylevel / max_features`** – какую часть признаков использовать для построения расщепления в дереве,

5. Параметры регуляризации:

- **`lambda / lambda_l2`** (L2),
- **`alpha / lambda_l1`** (L1).

Кроме этого, есть параметры которые помогают обучать бустинг быстрее: число используемых потоков, где проводить вычисления: на CPU или GPU, хранить модель в ОЗУ при обучении или нет, метод поиска расщепления. Про последний стоит заметить, что можно не перебирать все пороги, а, скажем, фиксированное число порогов на каждый признак.

В последнее время появляются возможности перестраивать уже созданную модель по бустинговой схеме: если есть семейство алгоритмов, последовательно проходимся по нему и настраиваем каждый. При этом можно определить, что понимается под настройкой (например, добавление уровней к дереву).

Практика настройки параметров бустинга

В отличие от случайных деревьев, в бустинге увеличение числа деревьев не всегда приводит к улучшению качества решения на тесте. Зависимость, как правило унимодальная, см. рис. Число деревьев, при котором качество максимально, зависит от темпа обучения: чем меньше темп, тем больше деревьев нужно, но зависимость нелинейная.

При оптимизации параметров обычно фиксируют число деревьев (оно должно быть не очень большим, чтобы алгоритм быстро обучался), подбирая под него

темп и значения остальных параметров. При построении итогового алгоритма увеличивают число деревьев и находят соответствующее значение темпа (оставляя остальные параметры неизменными).

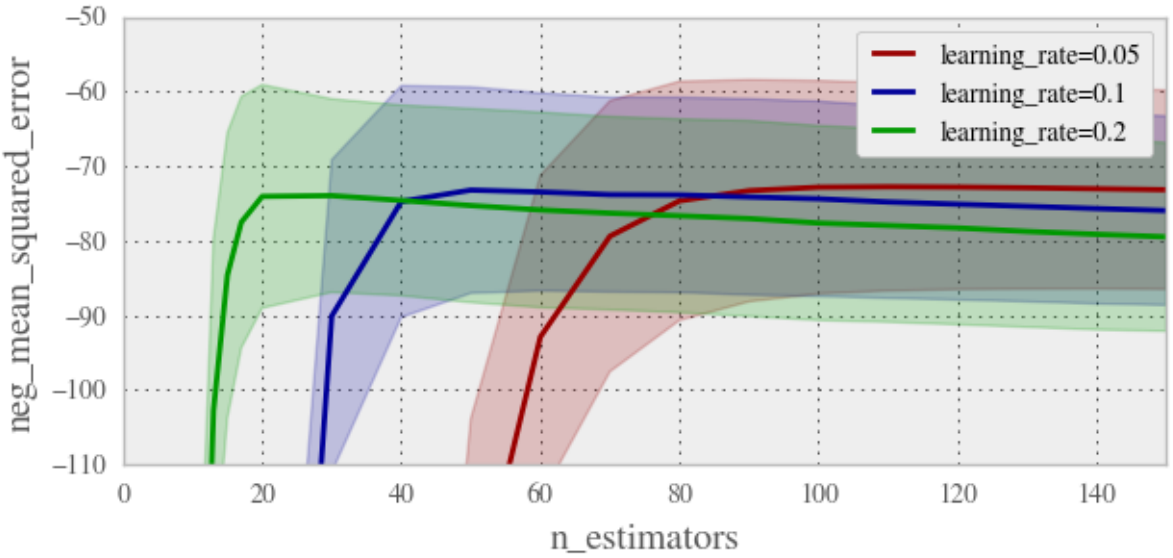

Рис. Зависимость качества решения на тесте от числа деревьев в бустинге

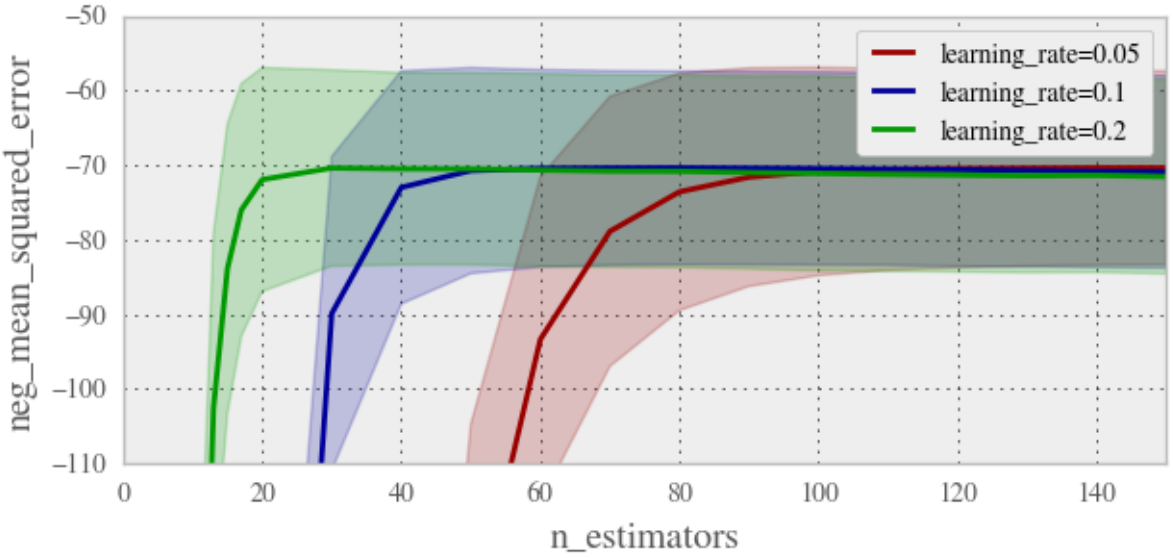

Рис. Зависимость качества решения на тесте от числа деревьев в бустинге при максимальной глубине равной 2.

Замечание. Есть и другая тактика: зафиксировать темп обучения, отложенную контрольную выборку и последовательно добавлять в ансамбль деревья по схеме бустинга. При этом на отложенном контроле проверять качество: если качество не увеличивается заранее заданное число итераций, обучение прекращается.

Также важными являются параметры, ограничивающие сложность базовых алгоритмов. Для деревьев основной такой параметр – глубина дерева (в некоторых реализациях – число листьев). На рис. показаны те же графики, что и на рис, но для максимальной глубины равной двум. Видно, что эффект переобучения для более простой модели меньше: графики практически выходят на асимптоту и качество не сильно падает при увеличении числа деревьев. Как правило, бустинг показывает высокое качество над неглубокими деревьями (глубина от 3 до 6).

Несмотря на то, что бустинг сам является ансамблем, к нему можно применять другие схемы ансамблирования, например, усреднять несколько бустнигов. Даже если мы усредняем бустинги с одинаковыми параметрами, они различаются из-за стохастической природы реализации: выбора случайных подвыборок на каждом шаге и признаков при построении деревьев. Важно понимать, что оптимальные значения параметров для одного бустинга уже могут быть неоптимальными для среднего арифметического нескольких бустингов, см. рис.

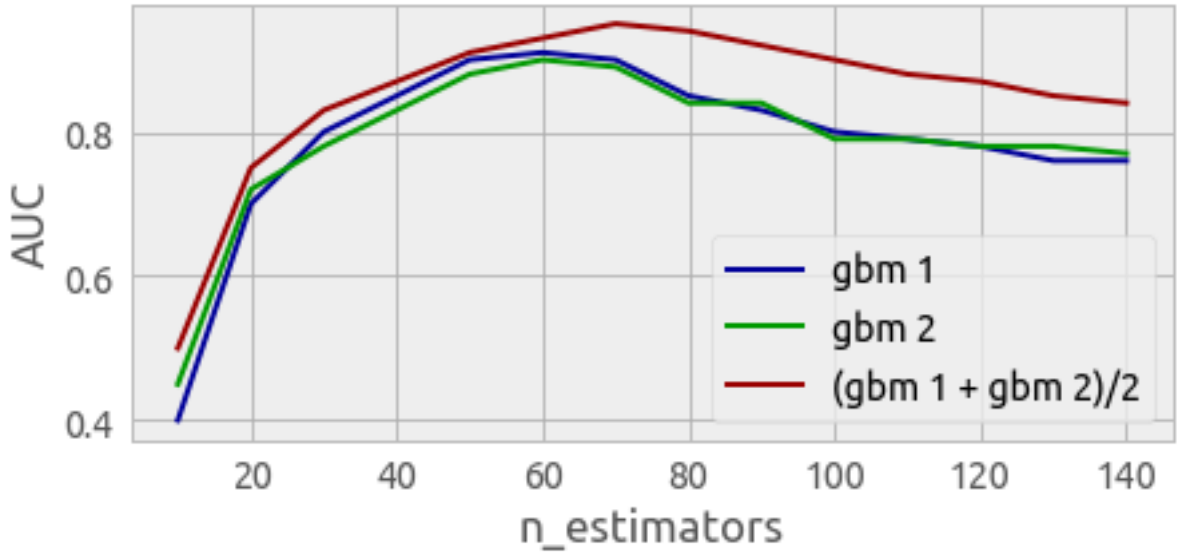

Рис. Качество отдельных бустингов и их усреднения (на тесте).

2do: del subsample

Важные особенности бустинга

Предположим, что мы решаем регрессионную задачу с метками 0 и 1, значения регрессора лежат в отрезке [0, 1] и могут интерпретироваться, как уверенность алгоритма в том, что объект имеет метку 1. Если решать задачу случайным лесом, то ответы на тестовой выборке, действительно, будут лежать в

указанном отрезке, а вот ответы бустинга над деревьями **могут выходить за пределы отрезка**. Особенно часто это может происходить при неоптимальном выборе параметров. На рис. показаны ответы двух бустингов: оптимально настроенного и с число деревьев в два раза больше оптимального (видно, что часть ответов стала больше 1).

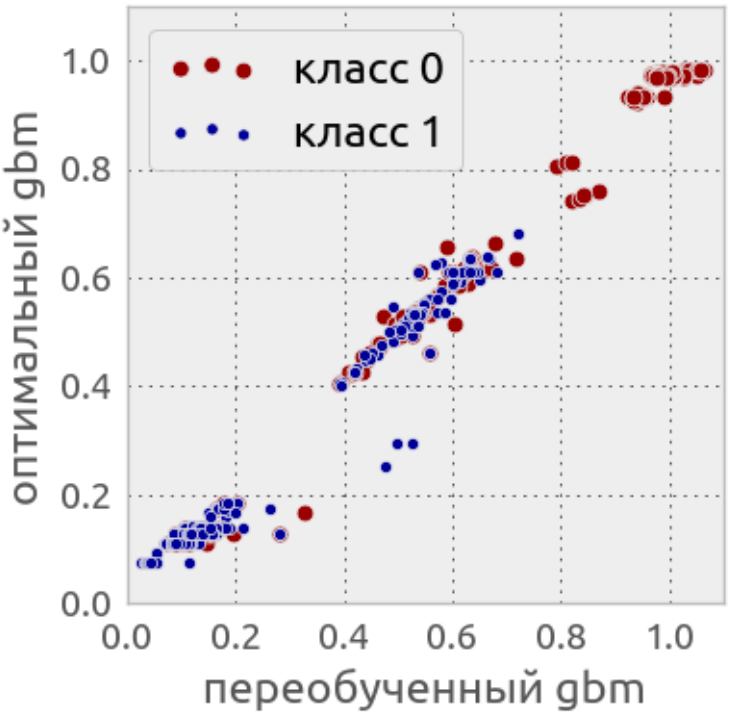

Рис. Ответы бустинга над деревьями.