

Лекция 6. Бустинг. Adaboost. Градиентный бустинг. XGBoost. LightGBM. Catboost.

Глинский А.В.

Цель занятия: В этом модуле мы познакомимся с такой темой, как бустинг. После прохождения модуля ученики смогут понимать основные принципы работы моделей на основе бустинга, таких как Adaboost, XGBoost, LightGBM и Catboost, и применять XGBoost, LightGBM и Catboost для решения задач машинного обучения.

Содержание

0.1	Бустинг	4
0.1.1	Adaboost	5
0.1.2	Градиентный бустинг	8
0.1.3	Модификации градиентного бустинга	12
0.1.4	Вопросы для самопроверки	13
0.1.5	Резюме по разделу	14
0.2	XGBoost	16
0.2.1	Визуальная демонстрация алгоритма	16
0.2.2	Подготовка данных для алгоритма	17
0.2.3	Процесс обучения	17
0.2.4	Оценка качества алгоритма	19
0.2.5	Интерпретация признаков с помощью алгоритма	19
0.2.6	Применение алгоритма	20
0.2.7	Плюсы и минусы алгоритма	20
0.2.8	Реализация алгоритма в Python	21
0.2.9	Вопросы для самопроверки	21
0.2.10	Резюме по разделу	22
0.3	LightGBM	23
0.3.1	Визуальная демонстрация алгоритма	23
0.3.2	Подготовка данных для алгоритма	24
0.3.3	Процесс обучения	24
0.3.4	Оценка качества алгоритма	26
0.3.5	Интерпретация признаков с помощью алгоритма	26
0.3.6	Применение алгоритма	26
0.3.7	Плюсы и минусы алгоритма	27
0.3.8	Реализация алгоритма в Python	27
0.3.9	Вопросы для самопроверки	28
0.3.10	Резюме по разделу	29
0.4	Catboost	30
0.4.1	Визуальная демонстрация алгоритма	30
0.4.2	Подготовка данных для алгоритма	31
0.4.3	Процесс обучения	31
0.4.4	Оценка качества алгоритма	32
0.4.5	Интерпретация признаков с помощью алгоритма	33
0.4.6	Применение алгоритма	33

0.4.7	Плюсы и минусы алгоритма	34
0.4.8	Реализация алгоритма в Python	34
0.4.9	Вопросы для самопроверки	35
0.4.10	Резюме по разделу	36
0.5	Резюме по модулю	37

0.1 Бустинг

Мы уже рассмотрели бэггинг как способ построения композиции моделей на деревьях.

Бэггинг (bootstrap aggregating) представляет собой метод ансамблирования, при котором строится несколько независимых моделей на разных подвыборках обучающего набора данных, а затем их результаты усредняются. Бэггинг помогает бороться с разбросом (variance), то есть с тем, что результаты модели могут сильно варьироваться в зависимости от выбора обучающей выборки.

Бустинг (boosting) также представляет собой метод ансамблирования, но он построен на итеративном улучшении одной базовой модели путем добавления новых моделей, каждая из которых исправляет ошибки предыдущих (Рисунок 1). Бустинг помогает бороться со смеще-

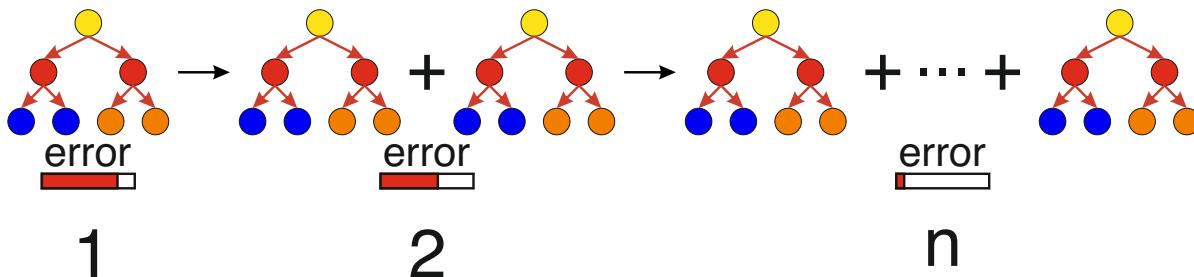


Рисунок 1: Визуализация принципа бустинга

нием (bias), то есть с тем, что модель может недообучаться и давать неточные предсказания на новых данных.

Таким образом, бэггинг и бустинг являются двумя различными подходами к решению проблемы ансамблирования моделей, и каждый из них борется с определенным типом ошибки модели.

Основные отличия между бэггингом и бустингом:

- Бэггинг использует ансамбль независимых моделей, каждая из которых обучается на случайной подвыборке данных. Бустинг использует последовательный ансамбль моделей, каждая из которых учитывает ошибки предыдущей модели.
- Бэггинг может использоваться с любым алгоритмом обучения, тогда как бустинг обычно используется с деревьями решений.
- Бэггинг может использоваться для уменьшения дисперсии модели, тогда как бустинг может использоваться для уменьшения как дисперсии, так и смещения модели.

Можно отметить следующие преимущества бустинга:

- Бустинг использует последовательное обучение, что позволяет каждой новой модели учитывать ошибки предыдущих моделей, что может привести к более точным прогнозам.
- Бустинг может обнаруживать сложные зависимости в данных, тогда как бэггинг склонен к созданию более простых моделей.
- Бустинг может использоваться для решения различных задач, включая классификацию, регрессию и ранжирование, тогда как бэггинг чаще используется для решения задач классификации.

В целом, бустинг может работать лучше, когда требуется более точная модель, способная обнаруживать сложные зависимости в данных, и когда задача может быть решена последовательно. Однако, бэггинг также может быть эффективным, особенно если требуется уменьшить дисперсию модели и улучшить ее устойчивость к шуму в данных.

Две наиболее популярные разновидности бустинга — это AdaBoost и градиентный бустинг (Gradient Boosting).

Основными различиями между AdaBoost и градиентным бустингом являются следующие:

- Обучение: AdaBoost обучает модель путем последовательного добавления новых моделей, которые перевзвешивают объекты, на которых допущены ошибки на предыдущих итерациях, в то время как градиентный бустинг обучает модель, используя градиентный спуск, минимизируя функцию потерь.
- Сложность моделей: AdaBoost использует простые модели, такие как деревья решений с одним разделением (decision stumps), в то время как градиентный бустинг может использовать более сложные модели, такие как деревья решений с несколькими уровнями.
- Регуляризация: градиентный бустинг поддерживает регуляризацию для уменьшения переобучения, в то время как AdaBoost не поддерживает регуляризацию.
- Шум: AdaBoost чувствителен к шуму в данных, потому что объекты с ошибками классификации получают более высокие веса, что может привести к переобучению, в то время как градиентный бустинг более устойчив к шуму.

В целом, градиентный бустинг обычно дает более высокое качество модели, чем AdaBoost, за счет использования более сложных моделей и регуляризации. Однако AdaBoost может быть быстрее и более простым в использовании, особенно в задачах с небольшим объемом данных.

Давайте разберем эти модели более подробно.

0.1.1 Adaboost

AdaBoost (англ. Adaptive Boosting) — это алгоритм машинного обучения, используемый для задач классификации и регрессии. Он был предложен Йоавом Фрейдом и Робертом Шапиро в 1996 году.

AdaBoost работает путем последовательного добавления «слабых» классификаторов в композицию. Каждый классификатор обучается на выборке, на которой предыдущие классификаторы допустили ошибки. При этом объекты, которые были неправильно классифицированы, получают более высокие веса, чтобы следующий классификатор мог сосредоточиться на них и исправить ошибки.

Каждый слабый классификатор представляет собой простую модель, такую как дерево решений с одним разделением (decision stump), которая предсказывает значение целевой переменной на основе одного признака. Каждый классификатор получает вес, который зависит от его точности, при этом более точные классификаторы получают больший вес. Соответственно, AdaBoost обучает набор слабых моделей на последовательно изменяющихся весах данных и комбинирует их в сильную модель.

Когда все слабые классификаторы обучены, они объединяются в одну композицию с помощью взвешенного голосования, где вес каждого классификатора зависит от его точности. Эта композиция является итоговой моделью.

Основным преимуществом AdaBoost является его способность уменьшать ошибку на обучающей выборке с каждой новой итерацией, что приводит к более высокой точности на тестовых данных. Однако он может быть чувствителен к шуму в данных и может приводить к переобучению, если слабые классификаторы слишком сложны.

Давайте рассмотрим пример с двумя классами: синими и оранжевыми метками. Для обучения AdaBoost мы будем использовать базовый алгоритм классификации - решающие деревья глубины 1, называемые пнями решений. Иллюстрация демонстрирует, как каждый новый классификатор (пень решений) настраивает веса объектов в обучающей выборке (Рисунок 2).

Первоначально все объекты имеют одинаковый вес. Затем мы обучаем первый классификатор и вычисляем его ошибку. Ошибка равна количеству объектов, которые были неправильно классифицированы, поделенному на общее количество объектов. Как только мы вычислили ошибку, мы можем рассчитать вес этого классификатора. Чем меньше ошибка, тем выше вес.

Затем мы пересчитываем веса объектов в обучающей выборке. Если объект был неправильно классифицирован, то его вес увеличивается. Если объект был правильно классифицирован, то его вес уменьшается. Затем мы повторяем процедуру с новым классификатором, вычисляя ошибку, вес и пересчитывая веса объектов. На каждой итерации мы добавляем новый классификатор к существующим и пересчитываем веса объектов.

После нескольких итераций AdaBoost начинает фокусироваться на объектах, которые ему трудно классифицировать. Это означает, что такие объекты получают больший вес. В конечном итоге мы получаем ансамбль классификаторов, каждый из которых специализируется на определенной области пространства признаков.

Визуализация работы AdaBoost

Ниже приведена визуализация работы AdaBoost на наборе с двумя признаками (Рисунок 2).

Формальное определение AdaBoost

Формальное определение AdaBoost выглядит так:

Исходно имеется обучающая выборка $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, где \mathbf{x}_i - вектор признаков i -го объекта, а y_i - его метка класса.

На каждой итерации m AdaBoost строит классификатор $h_m(\mathbf{x})$ с помощью базового алгоритма обучения.

На первой итерации все объекты имеют равные веса: $w_{1,i} = 1/n$, где $i = 1, \dots, n$.

Далее, на каждой итерации $m = 1, \dots, M$, AdaBoost делает следующее:

1. Обучает классификатор $h_m(\mathbf{x})$ на выборке $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ с весами $w_{m,i}$.
2. Вычисляет ошибку классификатора на обучающей выборке:

$$err_m = \frac{\sum_{i=1}^n w_{m,i} I(y_i \neq h_m(\mathbf{x}_i))}{\sum_{i=1}^n w_{m,i}},$$

где $I(\cdot)$ - индикаторная функция (функция, которая принимает значение 1 при выполнении определенного условия и значение 0 в противном случае).

3. Вычисляет вес классификатора:

$$\alpha_m = \log \frac{1 - err_m}{err_m}.$$

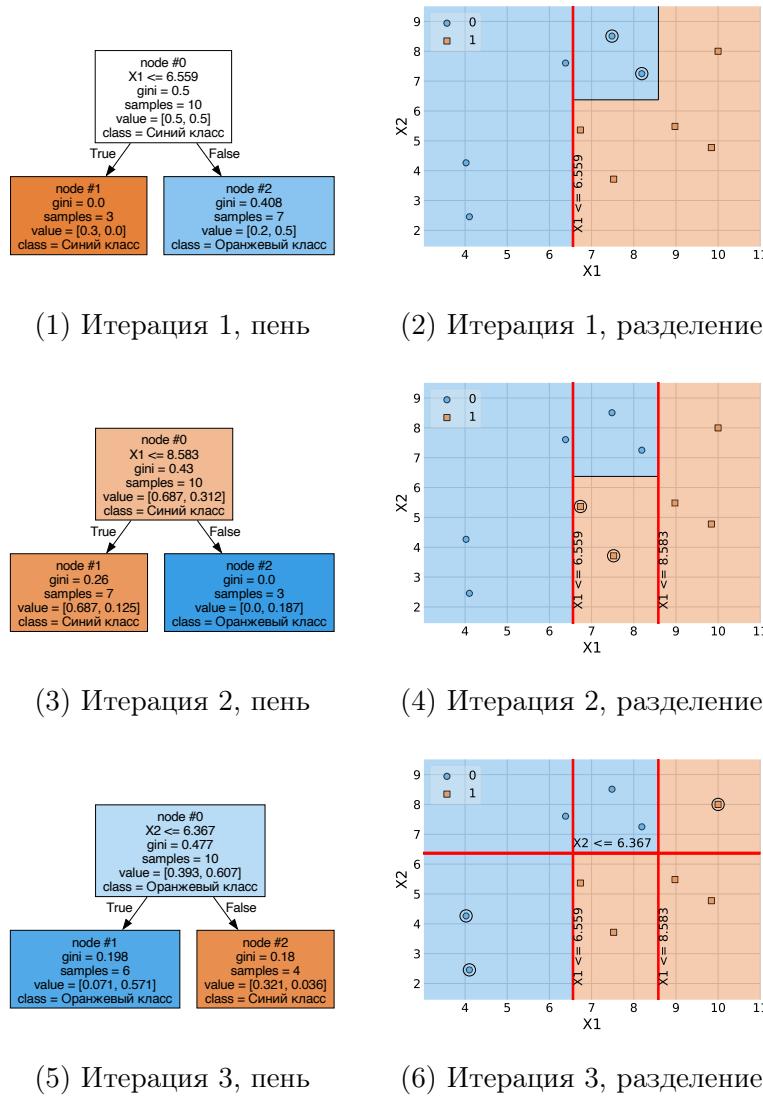


Рисунок 2: Визуализация работы AdaBoost

4. Обновляет веса объектов:

$$w_{m+1,i} = w_{m,i} \exp(\alpha_m I(y_i \neq h_m(\mathbf{x}_i))),$$

где $i = 1, \dots, n$.

После M итераций AdaBoost объединяет классификаторы $h_1(\mathbf{x}), \dots, h_M(\mathbf{x})$ с помощью взвешенного голосования:

$$H(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(\mathbf{x}) \right).$$

Здесь функция $\text{sign}(\cdot)$ возвращает знак своего аргумента (т.е. +1 или -1), что соответствует бинарной классификации.

AdaBoost был одним из популярных алгоритмов бустинга, однако на сегодняшний момент большей популярностью пользуется градиентный бустинг.

0.1.2 Градиентный бустинг

Градиентный бустинг - это алгоритм машинного обучения, использующий последовательное обучение слабых моделей (например, деревьев решений), каждая из которых нацелена на исправление ошибок предыдущей модели. Ошибки модели измеряются при помощи градиента функции потерь, и каждая последующая модель обучается на остатках (разности между предсказаниями текущей модели и правильными ответами) предыдущих моделей. После обучения всех моделей, их предсказания комбинируются в итоговый ансамбль с помощью взвешенного голосования или суммирования. Градиентный бустинг является одним из наиболее мощных алгоритмов машинного обучения и широко используется в различных областях, таких как рекомендательные системы, естественный язык и другие.

Рассмотрим формальное определение задачи для градиентного бустинга. Пусть у нас есть обучающая выборка (x_i, y_i) , где x_i — входные данные, а y_i — их соответствующие метки класса или значения целевой переменной. Для решения задачи регрессии будем считать, что y_i — действительные числа, а для задачи классификации — целые числа, каждое из которых соответствует классу.

В градиентном бустинге используется модель ансамбля деревьев решений, которая представляет собой сумму T деревьев с параметрами θ_j :

$$F(x) = \sum_{j=1}^T f_j(x; \theta_j)$$

Каждое дерево $f_j(x; \theta_j)$ является функцией, которая принимает входные данные x и возвращает соответствующий вклад в ответ ансамбля. Цель градиентного бустинга — найти параметры θ_j каждого дерева таким образом, чтобы ансамбль $F(x)$ минимизировал функционал ошибки¹ $L(y, F(x))$ на обучающей выборке. Для этого на каждом шаге добавляется новое дерево, которое минимизирует остаточную ошибку, вычисленную как отрицательный градиент функционала ошибки по отношению к предсказаниям текущей модели:

$$r_{ij} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

$$f_j = \operatorname{argmin}_f \sum_{i=1}^n (f(x_i; \theta_j) - r_{ij})^2$$

Здесь r_{ij} — это остаточное значение для i -го объекта на j -м шаге, а f_j — это новое дерево решений, которое добавляется к ансамблю на j -м шаге.

Алгоритм продолжает добавлять новые деревья, пока не будет достигнуто заданное количество итераций или пока ошибка на валидационной выборке не перестанет уменьшаться.

Градиентный бустинг получил такое название от того, что он использует градиент функционала ошибки по предсказаниям модели для построения последовательности деревьев, которые пытаются уменьшить эту ошибку.

Когда мы строим модель, мы обычно оптимизируем функционал ошибки, который зависит от параметров модели. Для нахождения оптимальных параметров мы должны вычислить градиент этого функционала ошибки по параметрам и выполнить шаг градиентного спуска, чтобы переместиться в направлении уменьшения ошибки.

¹усредненное значение функции потерь для набора данных или выборки

В градиентном бустинге мы также хотим уменьшить функционал ошибки, но мы делаем это не напрямую, а путем построения последовательности деревьев, каждое из которых старается уменьшить остаточную ошибку на обучающей выборке. В результате, градиентный бустинг может быть рассмотрен как метод градиентного спуска в пространстве ответов на каждой итерации.

Градиентный бустинг для регрессии

Для задачи регрессии градиентный бустинг можно формально определить следующим образом:

Дано множество объектов X и соответствующие им значения целевой переменной y , где $y_i \in \mathbb{R}$ для каждого $i = 1, \dots, n$. Требуется построить модель регрессии в виде суммы функций $F(x) = \sum_{j=1}^M f_j(x)$, где каждое слагаемое $f_j(x)$ является деревом решений небольшой глубины.

Для построения модели градиентного бустинга используется функция потерь $L(y, F)$, которая зависит от истинных значений целевой переменной y и предсказанных значений $F(x)$ на каждом объекте x . Обычно используется квадратичная функция потерь:

$$L(y, F) = \frac{1}{2}(y - F(x))^2$$

Градиент функции потерь на каждом объекте x_i определяется как производная функции потерь по предсказанию на этом объекте:

$$\nabla L(y_i, F) = -(y_i - F(x_i))$$

На каждой итерации $m = 1, \dots, M$ мы добавляем новое дерево f_m в модель, который приближает градиент функции потерь $\nabla L(y_i, F)$ на каждом объекте. Таким образом, новая модель выглядит как:

$$F_m(x) = F_{m-1}(x) + \gamma_m f_m(x)$$

где γ_m - предсказания на m -ой итерации, которые вычисляются как:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma f_m(x_i))$$

То есть, мы ищем предсказания γ_m , которые минимизируют функционал ошибки на текущей итерации. На каждой итерации мы строим дерево f_m с помощью градиентного спуска по квадратичной функции ошибки:

$$f_m = \arg \min_f \sum_{i=1}^n (r_i - f(x_i))^2$$

где $r_i = -\nabla L(y_i, F)$ - остатки модели на i -ом объекте на текущей итерации.

Таким образом, на каждой итерации мы добавляем новое дерево, которое приближает остатки модели на текущей итерации и уменьшает функционал ошибки на обучающей выборке, умножаем его прогноз на скорость обучения (learning rate) и добавляем в общую композицию. Обычно градиентный бустинг продолжает до тех пор, пока не достигнута заданная максимальная глубина деревьев или пока не будет достигнуто заданное количество деревьев.

После того, как была построена модель градиентного бустинга, мы можем использовать ее для предсказания значений целевой переменной на новых объектах. Для этого на вход модели подается объект x , и вычисляется значение предсказания $F(x)$. Как и в случае обучения, вычисление предсказаний также осуществляется путем суммирования предсказаний всех деревьев.

Метод градиентного бустинга позволяет получить высокую точность предсказаний на различных задачах регрессии и классификации. Однако он может быть довольно требователен к вычислительным ресурсам и может иметь тенденцию к переобучению на небольших выборках. Для уменьшения риска переобучения можно использовать регуляризацию, например, ограничения на глубину деревьев или использование случайного выбора подмножества объектов и признаков на каждой итерации.

Градиентный бустинг для классификации

Формальное определение градиентного бустинга для бинарной классификации можно записать следующим образом:

Пусть имеется обучающая выборка $X = \{(x_1, y_1), \dots, (x_n, y_n)\}$, где $x_i \in \mathbb{R}^d$ - вектор признаков i -го объекта, $y_i \in \{0, 1\}$ - метка класса объекта. Требуется построить функцию $F(x)$, которая будет предсказывать метку класса на новых объектах.

Аналогично регрессии, начинаем с построения начальной модели - константы $F_0(x)$. Затем на каждой следующей итерации m строится новое дерево решений $h_m(x)$, которое «исправляет» ошибки предыдущих деревьев. Функцию, которую мы пытаемся оптимизировать, можно определить как:

$$L(F) = \sum_{i=1}^n l(y_i, F(x_i))$$

где $l(y, F(x))$ - функция потерь для объекта (x, y) , которая показывает, насколько сильно отличается предсказание $F(x)$ от реальной метки y .

На каждой итерации мы хотим найти дерево $h_m(x)$, которое будет минимизировать функцию потерь $L(F_{m-1} + h_m)$:

$$h_m = \arg \min_h \sum_{i=1}^n l(y_i, F_{m-1}(x_i) + h(x_i))$$

Чтобы найти дерево $h_m(x)$, используется метод градиентного спуска, причем спуск производится не напрямую по функции $L(F)$, а с учетом ответов на текущей итерации. Градиент функции потерь для бинарной классификации выражается следующим образом:

$$\nabla L(y_i, F) = - \left(y_i - \frac{1}{1 + e^{-F(x_i)}} \right)$$

где $F(x_i)$ - значение предсказания на i -м объекте на текущей итерации. Градиент функции потерь выражает, насколько сильно нужно изменить значение предсказания на данном объекте, чтобы уменьшить значение функции потерь.

После того, как найдено дерево $h_m(x)$, предсказания γ_m определяются путем минимизации функции потерь по γ :

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n l(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

Итоговая функция предсказания $F(x)$ на новом объекте вычисляется как сумма всех построенных деревьев с учетом их коэффициентов:

$$F(x) = \sum_{m=1}^M \gamma_m h_m(x)$$

где M - количество построенных деревьев.

На каждой итерации градиентного бустинга строится дерево, которое минимизирует функцию потерь, учитывая значения предсказания на предыдущих итерациях. Далее, коэффициенты для каждого дерева находятся путем решения задачи оптимизации. Итоговая функция предсказания получается как линейная комбинация всех деревьев с учетом их коэффициентов.

Для многоклассовой классификации градиентный бустинг обычно используется в сочетании с методом один против всех (one-vs-all), где для каждого класса строится своя модель градиентного бустинга, которая отличает этот класс от всех остальных.

Подбор параметров для градиентного бустинга

Подбор оптимальных параметров для модели градиентного бустинга является важным этапом, который может существенно повлиять на качество предсказания.

Основными параметрами, которые нужно настроить, являются:

- Learning rate (шаг обучения) - коэффициент, на который умножается значение градиента на каждой итерации. Маленькие значения learning rate могут привести к более точным предсказаниям, но требуют большего числа итераций для обучения. Большие значения learning rate могут привести к более быстрой сходимости, но могут также привести к переобучению модели.
- Количество деревьев - определяет, сколько деревьев будет использоваться в модели. Большое количество деревьев может улучшить качество предсказания, но может также привести к переобучению модели.
- Максимальная глубина деревьев - определяет, сколько раз дерево будет делить данные. Большая глубина деревьев может привести к переобучению модели, тогда как слишком маленькая глубина может привести к недообучению.
- Минимальное число объектов в листе - определяет минимальное количество объектов, которые должны оказаться в листе дерева. Большее значение этого параметра может привести к сокращению переобучения модели.
- Размер выборки для построения деревьев - определяет, какое количество объектов будет выбрано для построения каждого дерева. Это может привести к уменьшению шума и улучшению качества предсказания.

Чтобы подобрать оптимальные параметры для модели градиентного бустинга, можно использовать перекрестную проверку (cross-validation), которая позволяет оценить качество модели на независимых выборках данных. Можно также использовать методы оптимизации, такие как Grid Search или Random Search, чтобы автоматически подобрать оптимальные значения параметров.

Регуляризация для градиентного бустинга

Основные методы регуляризации для градиентного бустинга включают в себя:

- Ограничение глубины деревьев (`max_depth`) - установка максимальной глубины деревьев может предотвратить переобучение и ускорить обучение.
- Ограничение на число листьев (`max_leaf_nodes`) - установка максимального количества листьев может также предотвратить переобучение и упростить модель.
- Ограничение на минимальное количество выборок в листе (`min_samples_leaf`) - установка минимального количества выборок в листе может предотвратить переобучение, особенно если выборка маленькая.
- Ограничение на минимальное количество выборок в узле (`min_samples_split`) - установка минимального количества выборок в узле может предотвратить создание слишком мелких узлов, что может привести к переобучению.
- Ограничение на максимальное количество признаков при поиске наилучшего разделения (`max_features`) - установка максимального количества признаков при поиске наилучшего разделения может предотвратить переобучение и ускорить обучение.
- L1-регуляризация (`alpha`) добавляет штраф к абсолютному значению весов признаков. Это заставляет модель делать выбор между использованием всех признаков с низкой точностью или только нескольких признаков с высокой точностью.
- L2-регуляризация (`lambda`) добавляет штраф к квадрату весов признаков. Это заставляет модель предпочитать использование всех признаков с умеренной точностью, чем только нескольких признаков с высокой точностью.

Комбинация различных методов регуляризации может помочь создать более устойчивую модель, уменьшить переобучение и улучшить ее обобщающую способность.

0.1.3 Модификации градиентного бустинга

Существует множество модификаций градиентного бустинга, которые могут помочь улучшить его производительность и точность. Некоторые из них включают в себя:

- XGBoost - еще один быстрый и эффективный алгоритм градиентного бустинга, который использует решающие деревья и оптимизирует функцию потерь, добавляя регуляризацию и использование градиента второго порядка.
- LightGBM - быстрый и эффективный алгоритм градиентного бустинга, который использует специальные техники, такие как гистограммы признаков, для ускорения процесса обучения.
- CatBoost - алгоритм градиентного бустинга, который использует категориальные признаки, включая их в процесс обучения без необходимости предварительной обработки данных.

Основные отличия между XGBoost, LightGBM и CatBoost можно выделить следующим образом (Таблица 2):

Здесь:

- Алгоритм указывает, что все три библиотеки реализуют градиентный бустинг.
- Поддержка GPU, распределенное обучение и поддержка пропущенных значений имеются во всех трех библиотеках.

Таблица 1: Сравнение основных свойств XGBoost, LightGBM и CatBoost

Свойство	XGBoost	LightGBM	CatBoost
Алгоритм	Градиентный бустинг	Градиентный бустинг	Градиентный бустинг
Поддержка GPU	Да	Да	Да
Распределенный обучение	Да	Да	Да
Поддержка категориальных признаков	Нет	Да	Да
Поддержка пропущенных значений	Да	Да	Да
Скорость обучения	Средняя	Высокая	Средняя
Размер модели	Средний	Маленький	Средний
Качество предсказаний	Высокое	Высокое	Высокое

Таблица 2: Сравнение основных свойств XGBoost, LightGBM и CatBoost

- LightGBM и CatBoost поддерживают категориальные признаки, в то время как XGBoost этого не делает.
- Скорость обучения в LightGBM наиболее высокая, в XGBoost средняя, а в CatBoost - средняя.
- Размер модели в LightGBM наименьший, в XGBoost средний, а в CatBoost - средний.
- Качество предсказаний очень высокое во всех трех библиотеках, но в LightGBM и CatBoost оно наиболее высокое.

Более подробно о XGBoost, LightGBM и CatBoost мы поговорим далее.

0.1.4 Вопросы для самопроверки

Какова сущность метода бустинга в машинном обучении?

1. Метод обучения без учителя
2. Метод генерации случайного леса
3. Метод последовательного построения композиции моделей
4. Метод независимого построения композиции моделей
5. Метод классификации объектов

Правильные ответы: 3

В чем суть алгоритма Adaboost (два ответа)?

1. AdaBoost обучает набор слабых моделей с помощью последовательно изменяющихся весов для объектов и комбинирует их в сильную модель.
2. AdaBoost для классификации выбирает на каждом шаге новую модель, которая за счет весов в большей степени сфокусирована на классификации неправильно классифицированных объектов в предыдущей модели.

3. AdaBoost использует градиентный спуск для обновления весов данных на каждом шаге обучения модели.
4. AdaBoost использует алгоритм случайного леса для комбинирования слабых моделей в сильную модель.

Правильные ответы: 1, 2

В чем суть алгоритма градиентного бустинга (два ответа)?

- Градиентный бустинг обучает набор деревьев решений на последовательно изменяющихся остатках и комбинирует их в сильную модель.
- Градиентный бустинг создает n независимых деревьев, на каждом из которых минимизирует функцию потерь в соответствии с градиентом.
- Градиентный бустинг использует градиентный спуск на остатках для оптимизации параметров моделей.

Правильные ответы: 1, 3

0.1.5 Резюме по разделу

Бустинг - это метод машинного обучения, который используется для улучшения качества прогнозов, создаваемых слабыми моделями машинного обучения. При бустинге каждая следующая модель (называемая базовой моделью) фокусируется на тех примерах данных, которые были неправильно классифицированы предыдущими моделями. В результате, каждая базовая модель «улучшает» работу предыдущих моделей, и в конечном итоге, бустинг представляет собой комбинацию всех базовых моделей, которые работают вместе, чтобы дать лучший результат, чем любая из них по отдельности.

Существует несколько подходов для реализации бустинга:

1. Adaboost
2. Градиентный бустинг

Adaboost - это алгоритм бустинга, который используется для улучшения качества классификации или регрессии в машинном обучении. Он работает путем создания последовательности слабых моделей машинного обучения и комбинирования их в единую сильную модель.

На каждой итерации Adaboost выбирает новую слабую модель, которая фокусируется на тех примерах данных, которые были классифицированы неправильно на предыдущих итерациях. Веса этих неправильно классифицированных примеров увеличиваются, чтобы следующая слабая модель сфокусировалась на них больше. Затем Adaboost комбинирует все слабые модели, чтобы получить сильную модель, которая дает более точные прогнозы.

Градиентный бустинг - это метод машинного обучения, который используется для создания сильной модели путем последовательного добавления слабых моделей, которые минимизируют ошибку предыдущих моделей.

В градиентном бустинге на каждой итерации создается новая слабая модель машинного обучения, которая пытается исправить ошибки предыдущей модели. Для этого градиентный бустинг использует градиент функции потерь, чтобы определить, какие примеры данных предыдущая модель классифицировала неправильно, и насколько сильно эти примеры влияют на функцию потерь.

Затем градиентный бустинг обучает новую слабую модель на этих неправильно классифицированных примерах и добавляет ее в сильную модель в виде слагаемого. Повторяя этот процесс много раз, градиентный бустинг создает последовательность слабых моделей, которые совместно дают более точные прогнозы.

0.2 XGBoost

Цель занятия: ученик может применить алгоритм XGBoost для решения задач классификации и регрессии на подготовленных и неподготовленных данных.

План занятия:

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

0.2.1 Визуальная демонстрация алгоритма

XGBoost (eXtreme Gradient Boosting) - это алгоритм машинного обучения, основанный на градиентном бустинге деревьев решений. Он был разработан в 2014 году и является одним из наиболее эффективных алгоритмов для задач классификации, регрессии и ранжирования.

XGBoost обучает ансамбль деревьев решений последовательно, приближаясь к оптимальному прогнозу с каждой новой итерацией. На каждой итерации алгоритм добавляет новое дерево, которое предсказывает остатки предыдущих деревьев. Таким образом, каждое новое дерево корректирует ошибки предыдущих деревьев.

По умолчанию XGBoost строит деревья level-wise, то есть строит все узлы на одном уровне перед переходом к следующему уровню (Рисунок 3). Однако существует также возможность

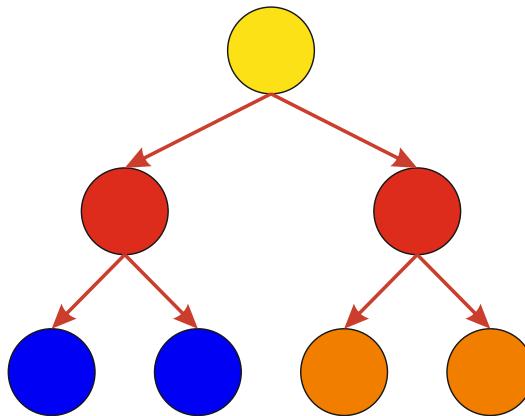


Рисунок 3: Level-wise построение дерева в XGBoost

строить деревья по листьям (leaf-wise), когда каждый узел строится таким образом, чтобы максимизировать уменьшение функции потерь. Это может привести к более быстрому обучению и более точным моделям, но может также вызывать переобучение в некоторых случаях. При выборе стратегии построения деревьев необходимо учитывать особенности конкретной задачи и набора данных.

Основные компоненты алгоритма XGBoost включают:

- Функцию потерь: Определяет, какой функционал оптимизируется в процессе обучения. В XGBoost используется функция потерь, которая состоит из двух частей: функции потерь для задачи (например, MSE для регрессии или логистическая функция потерь для классификации) и штрафа за сложность модели (регуляризация).
- Деревья решений: В качестве базовых алгоритмов используются решающие деревья. Каждое дерево строится на основе взвешенной версии обучающего набора данных, которая зависит от остатков предыдущих деревьев.
- Регуляризация: XGBoost поддерживает несколько методов регуляризации, включая L1 и L2 регуляризацию, ограничение глубины деревьев, ограничение на число листьев, ограничение на минимальное количество выборок в листе и ограничение на максимальное количество признаков при поиске наилучшего разделения. Регуляризация помогает предотвратить переобучение модели и повысить ее обобщающую способность.
- Градиентный спуск: Для нахождения оптимальных весов модели XGBoost использует градиентный спуск. На каждой итерации градиентного спуска алгоритм вычисляет градиент функции потерь по отношению к весам модели и изменяет их с определенным шагом в направлении уменьшения функции ошибки.

0.2.2 Подготовка данных для алгоритма

Шаги по предобработке наборов данных, полезные для использования с алгоритмом XGBoost. Некоторые из них включают:

- Обработка выбросов и пропущенных значений: XGBoost может работать с данными, содержащими пропущенные значения и выбросы, но для достижения лучшей производительности рекомендуется предварительно обработать эти аномалии.
- Обработка категориальных признаков: XGBoost поддерживает категориальные признаки, но они должны быть закодированы в числовом формате. Существуют различные методы кодирования категориальных признаков, такие как One-Hot Encoding, Label Encoding, и выбор метода зависит от конкретной задачи.
- Сбалансированность классов: Если ваш набор данных имеет несбалансированные классы, то может потребоваться принять меры для сбалансирования данных. Например, можно использовать взвешивание классов или сэмплирование данных, чтобы уравнять количество примеров в каждом классе.
- Отбор признаков: XGBoost может работать с большим количеством признаков, но для предотвращения переобучения модели может быть полезно отбирать только наиболее важные признаки. Это можно сделать с помощью методов отбора признаков, таких как Recursive Feature Elimination (RFE) или SelectKBest.

Это не полный список обработок данных, которые можно применить для использования с XGBoost, но это некоторые из наиболее распространенных методов. В целом, подготовка данных для использования с XGBoost зависит от конкретного набора данных и задачи машинного обучения.

0.2.3 Процесс обучения

Основная идея алгоритма XGBoost состоит в том, чтобы строить деревья решений последовательно, учитывая ошибки предыдущих деревьев, и объединять их в итоговую модель. Каждое новое дерево обучается на ошибках предыдущих деревьев, чтобы исправить

эти ошибки и улучшить качество модели. Этот процесс продолжается до тех пор, пока не будет достигнута определенная степень точности или не будут исчерпаны все ресурсы.

Основные шаги обучения алгоритма XGBoost:

1. Инициализация модели: Начальное значение целевой переменной устанавливается как среднее значение обучающей выборки.
2. Вычисление градиента и гессиана: Для каждого объекта в обучающей выборке вычисляются градиент и гессиан целевой функции. Градиент представляет собой первую производную функции ошибки, а гессиан - вторую производную (Градиент ошибки - это вектор первых частных производных функции ошибки по каждому параметру модели. Гессиан - это матрица вторых частных производных функции ошибки по каждой паре параметров модели). Градиент и гессиан используются для обучения каждого нового дерева решений.
3. Обучение дерева: Каждое новое дерево обучается на остатках (разнице между целевой переменной и предсказанным значением) предыдущих деревьев. Для этого используется метод градиентного бустинга, где дерево решений на каждой итерации обучается минимизировать целевую функцию, которая может быть выбрана из различных вариантов в зависимости от задачи.
4. Вычисление весов деревьев: Деревья решений имеют различные веса, которые определяют их важность в модели. Веса деревьев вычисляются на основе ошибок, которые они исправляют, и регуляризации, которая учитывает сложность модели.
5. Обновление целевой переменной: Целевая переменная обновляется путем вычитания предсказанного значения из текущего значения, что позволяет модели сфокусироваться на ошибках, которые еще не исправлены.
6. Повторение шагов: Шаги 2-6 повторяются до тех пор, пока не будет достигнута заданная точность или не будет исчерпано максимальное количество итераций.

Особенности обучения XGBoost:

- Обработка отсутствующих значений: XGBoost поддерживает обработку отсутствующих значений в данных, но лучше делать предобработку данных.
- Регуляризация: XGBoost использует регуляризацию для контроля сложности модели и избежания переобучения.
- Подбор гиперпараметров: XGBoost позволяет настраивать различные гиперпараметры, такие как глубина дерева, скорость обучения и количество деревьев. Это позволяет найти оптимальные настройки модели для каждой конкретной задачи.
- Автоматический выбор признаков: XGBoost может автоматически выбирать наиболее важные признаки для моделирования. Это происходит путем оценки важности признаков на основе их вклада в улучшение целевой функции.

Важные параметры при обучении XGBoost:

- learning_rate: скорость обучения, которая определяет, насколько сильно корректируются веса при обновлении модели на каждой итерации. Выбор этого параметра влияет на скорость сходимости модели и на ее способность к обобщению.
- max_depth: максимальная глубина дерева, которая определяет количество уровней дерева решений. Выбор этого параметра влияет на способность модели к обобщению и на скорость обучения.

- `min_child_weight`: минимальный вес дочернего узла, который определяет, какую минимальную величину должен иметь вес дочернего узла, чтобы продолжать делить узел. Выбор этого параметра влияет на устойчивость модели к шуму и на ее способность к обобщению.
- `gamma`: минимальное снижение функции ошибки, которое необходимо достичь, чтобы продолжать делить узел. Выбор этого параметра влияет на устойчивость модели к шуму и на скорость обучения.
- `subsample`: доля выборки, используемая для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- `colsample_bytree`: доля признаков, используемых для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- `alpha`: коэффициент L1-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.
- `lambda`: коэффициент L2-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.
- `num_boosted_round`: количество итераций обучения. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.

Выбор оптимальных значений этих гиперпараметров может существенно повлиять на точность и обобщающую способность модели XGBoost.

0.2.4 Оценка качества алгоритма

Для оценки качества алгоритма XGBoost часто используют следующие метрики:
Классификация:

- **accuracy**: $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision**: $\frac{TP}{TP+FP}$
- **recall**: $\frac{TP}{TP+FN}$
- **F1**: $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- **MSE**
- **MAE**

0.2.5 Интерпретация признаков с помощью алгоритма

XGBoost имеет несколько методов для интерпретации важности признаков, которые могут помочь понять, какие признаки вносят наибольший вклад в модель. Рассмотрим несколько таких методов:

- Важность признаков на основе деревьев: этот метод основан на том, как деревья в модели XGBoost используют признаки для принятия решений. Для каждого признака суммируется значение важности, которое это свойство получает при разделении данных на каждом узле дерева. Чем выше суммарное значение важности, тем важнее признак.

- SHAP значения: SHAP (SHapley Additive exPlanations) - это метод, который предоставляет объяснение для каждого предсказания, показывая, насколько каждый признак вносит в конечный результат. Он использует теорию игр Шепли для распределения вклада между признаками.
- Partial Dependence Plot (PDP): PDP - это график, который показывает, как модель меняет свои прогнозы в зависимости от значений одного признака, при этом все остальные признаки остаются на своих местах. Это позволяет понять, какой эффект на прогноз оказывает каждый отдельный признак.
- Feature Interaction: XGBoost также предоставляет возможность выявления взаимодействия между признаками. Это может быть полезно, когда важность признаков взаимозависима, и один признак вносит больший вклад только вместе с другими признаками.

Все эти методы могут помочь понять, какие признаки вносят наибольший вклад в модель XGBoost и как они взаимодействуют между собой.

0.2.6 Применение алгоритма

XGBoost - это мощный алгоритм машинного обучения, который может применяться во многих областях. Вот несколько примеров его применения:

- Классификация: XGBoost может использоваться для классификации в различных областях, таких как медицина, финансы, обработка естественного языка и многих других.
- Регрессия: XGBoost может использоваться для решения задач регрессии, таких как прогнозирование цен на недвижимость, оценка риска финансовых инструментов и т.д.
- Ранжирование: XGBoost может использоваться для построения систем рекомендаций, ранжирования результатов поиска и т.д.
- Анализ данных: XGBoost может использоваться для поиска закономерностей и трендов в данных, а также для выявления аномалий и выбросов.
- Обработка изображений: XGBoost может использоваться для классификации изображений, определения объектов на изображении, обнаружения дефектов и т.д.
- Обработка звука: XGBoost может использоваться для классификации звуковых сигналов, распознавания речи, анализа звукового спектра и т.д.
- Промышленность: XGBoost может использоваться для оптимизации процессов в промышленности, таких как управление производственными линиями, оптимизация логистики и т.д.

В целом, XGBoost широко используется в различных областях, где требуется высокая точность и скорость работы алгоритма машинного обучения.

0.2.7 Плюсы и минусы алгоритма

Плюсы:

- Высокая производительность и масштабируемость на больших наборах данных.
- Превосходные результаты на различных задачах машинного обучения.
- Поддержка распределенных вычислений для обучения на кластерах.
- Реализация регуляризации для борьбы с переобучением.
- Возможность обработки разных типов данных (числовые, категориальные, текстовые и т.д.).

- Возможность обработки пропущенных значений.

Минусы:

- Большое количество гиперпараметров может быть сложным для настройки.
- Не так хорошо работает на маленьких выборках данных.
- Может потребоваться больше времени для обучения, чем для простых моделей машинного обучения, таких как линейные модели.

0.2.8 Реализация алгоритма в Python

XGBoost реализован в библиотеке xgboost (<https://xgboost.readthedocs.io/en/stable/>). Для задачи классификации с использованием XGBoost в библиотеке xgboost существует классификатор **XGBClassifier**, а для задач регрессии — **XGBRegressor**. Основные параметры этих моделей:

- **n_estimators**: количество деревьев в лесу. По умолчанию `n_estimators=100`.
- **objective**: функция потерь, которую требуется минимизировать при обучении модели. Поддерживаются различные функции потерь в зависимости от задачи (например, «binary:logistic» для бинарной классификации или «reg:squarederror» для регрессии). По умолчанию `objective='reg:squarederror'`.
- **max_depth**: максимальная глубина каждого дерева в лесу. По умолчанию `max_depth=6`.
- **learning_rate**: шаг обучения (также называемый темпом обучения). По умолчанию `learning_rate=0.3`.
- **min_child_weight**: минимальный вес, необходимый для разделения узла. По умолчанию `min_child_weight=1`.
- **subsample**: доля обучающих данных, которые используются для каждого дерева. По умолчанию `subsample=1`.
- **colsample_bytree**: доля признаков, которые используются для каждого дерева. По умолчанию `colsample_bytree=1`.
- **reg_alpha**: коэффициент L1-регуляризации. По умолчанию `reg_alpha=0`.
- **reg_lambda**: коэффициент L2-регуляризации. По умолчанию `reg_lambda=1`.

Классы XGBClassifier/XGBRegressor также имеют методы **fit(X, y)** для обучения модели на данных `X` и `y`, метод **predict(X)** для предсказания целевых значений для новых данных `X`, методы **score(X, y)** и **get_params()** для получения оценки точности модели и параметров модели соответственно. Кроме того, класс XGBClassifier имеет метод **predict_proba(X)**, который возвращает вероятности принадлежности каждому классу.

0.2.9 Вопросы для самопроверки

Как происходит построение дерева в XGBoost по умолчанию?

1. Level-wise
2. Leaf-wise
3. Oblivious decision tree

Правильные ответы: 1

Как XGBoost выбирает предикаты для разделения выборки?

1. XGBoost выбирает предикаты для разделения выборки случайным образом.
2. XGBoost выбирает предикаты для разделения выборки на основе их важности.
3. XGBoost выбирает предикаты для разделения выборки путем перебора всех возможных предикатов.
4. XGBoost выбирает предикаты для разделения выборки на основе Information Gain.

Правильные ответы: 4

Выберите плюсы алгоритма XGBoost:

1. Большое количество гиперпараметров может быть сложным для настройки.
2. Возможность обработки пропущенных значений.
3. Возможность обработки разных типов данных (числовые, категориальные, текстовые и т.д.).
4. Высокая производительность и масштабируемость на больших наборах данных.
5. Может потребоваться больше времени для обучения, чем для простых моделей машинного обучения, таких как линейные.
6. Не так хорошо работает на маленьких выборках данных.
7. Поддержка распределенных вычислений для обучения на кластерах.
8. Превосходные результаты на различных задачах машинного обучения.
9. Реализация регуляризации для борьбы с переобучением.

Правильные ответы: 2, 3, 4, 7, 8, 9

0.2.10 Резюме по разделу

XGBoost (Extreme Gradient Boosting) - это эффективный и мощный алгоритм машинного обучения, используемый для задач классификации и регрессии.

Основными преимуществами XGBoost являются высокая скорость обучения и предсказания, возможность работы с большими объемами данных, устойчивость к переобучению и возможность интерпретации результатов.

В XGBoost используется ансамбль деревьев решений, которые объединяются с помощью градиентного бустинга. Этот метод позволяет улучшать качество модели путем последовательного добавления новых деревьев и корректировки ошибок предыдущих.

XGBoost предлагает множество параметров для настройки модели, включая параметры дерева и параметры бустинга. Эти параметры позволяют контролировать глубину деревьев, скорость обучения, количество деревьев в ансамбле и другие параметры, влияющие на качество модели.

В целом, XGBoost является одним из наиболее эффективных и гибких инструментов машинного обучения, которые можно использовать для решения различных задач классификации и регрессии.

0.3 LightGBM

Цель занятия: ученик может применить алгоритм LightGBM для решения задач классификации и регрессии на подготовленных и неподготовленных данных.

План занятия:

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

0.3.1 Визуальная демонстрация алгоритма

LightGBM - это алгоритм градиентного бустинга деревьев решений, который разработан Microsoft и считается одним из наиболее быстрых и эффективных алгоритмов градиентного бустинга.

Основным преимуществом LightGBM является его способность работать с большими объемами данных и быстрая скорость обучения модели. Это достигается за счет использования нескольких техник оптимизации, таких как основанная на гистограммах обработка данных, локальной оценки градиента (GOSS) и сжатия данных (компактное представление данных). В LightGBM используется стратегия обучения по листьям (leaf-wise) в отличие от стратегии обучения по слоям (level-wise), используемой в других алгоритмах градиентного бустинга (Рисунок 4). Это означает, что при построении дерева на каждом шаге алгоритм выбира-

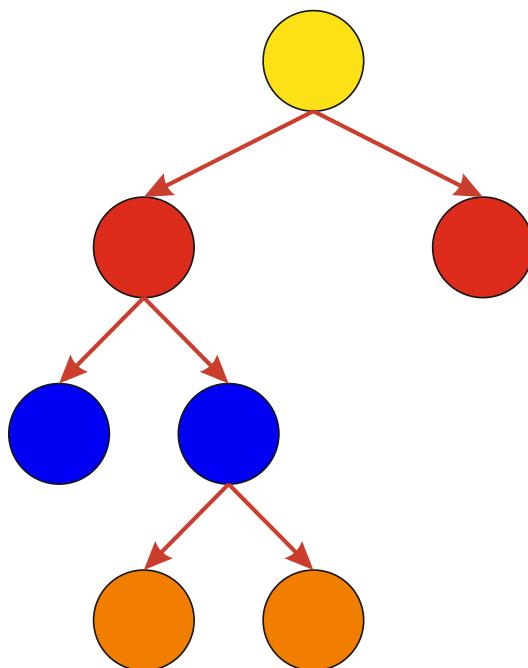


Рисунок 4: Leaf-wise построение дерева в LightGBM

ет тот лист, который максимально уменьшает функцию потерь, и строит дочерние узлы от этого листа. Таким образом, в LightGBM каждое дерево может иметь разную глубину, что позволяет модели улавливать более сложные зависимости в данных. Также это позволяет LightGBM строить деревья с меньшим количеством узлов, что уменьшает сложность модели и ускоряет время обучения. Однако, leaf-wise стратегия может привести к переобучению, особенно если выборка содержит выбросы или шумы. Поэтому, в LightGBM реализован ряд механизмов для предотвращения переобучения, таких как регуляризация и early stopping.

Кроме того, LightGBM поддерживает распределенное обучение на нескольких компьютерах, поддерживает многоклассовую классификацию и регрессию.

Общие шаги алгоритма LightGBM:

- Построение гистограмм для фичей
- Обучение деревьев решений с использованием градиентного бустинга
- Пересчет градиента и гессиана на каждой итерации для обновления весов деревьев
- Вычисление прогноза модели на новых данных

Таким образом, LightGBM - это быстрый и эффективный алгоритм градиентного бустинга, который может использоваться для решения задач классификации и регрессии на больших объемах данных.

0.3.2 Подготовка данных для алгоритма

Подготовка данных для алгоритма LightGBM может включать в себя следующие шаги:

- Обработка выбросов: для достижения лучшей производительности рекомендуется предварительно обработать выбросы.
- Обработка пропущенных значений: если есть пропущенные значения, заполните их средними значениями, медианами или другими подходящими значениями.

0.3.3 Процесс обучения

Процесс обучения LightGBM включает в себя следующие шаги:

1. Подготовка данных.
2. Подготовка параметров модели. LightGBM имеет множество гиперпараметров, которые могут быть настроены для достижения наилучшей производительности модели.
3. Создание деревьев. LightGBM использует алгоритм, основанный на градиентном бустинге деревьев решений. На каждой итерации модель добавляет новое дерево, которое пытается уменьшить остаточную ошибку предыдущей модели.
4. Прогнозирование. После построения модели можно использовать ее для прогнозирования новых значений. LightGBM позволяет делать прогнозы как для задач классификации, так и для задач регрессии.
5. Тюнинг гиперпараметров. Как и в случае с другими моделями машинного обучения, можно провести тюнинг гиперпараметров для достижения наилучшей производительности модели. LightGBM имеет множество гиперпараметров, которые могут быть настроены для улучшения качества модели.
6. Повторение. Шаги 2-4 могут быть повторены несколько раз для достижения наилучшей производительности модели.

В результате обучения LightGBM получается ансамбль деревьев решений, которые были построены поэтапно с помощью градиентного бустинга. LightGBM обладает высокой скоростью работы и хорошей производительностью на больших наборах данных. Кроме того, LightGBM поддерживает параллельное выполнение и распределенное обучение.

Особенности обучения LightGBM:

- Обработка отсутствующих значений: LightGBM также поддерживает обработку отсутствующих значений в данных, но лучше делать предобработку данных.
- Регуляризация: LightGBM использует регуляризацию для контроля сложности модели и избежания переобучения, но также использует методы сокращения данных и ограничения глубины деревьев.
- Подбор гиперпараметров: LightGBM позволяет настраивать множество гиперпараметров, таких как глубина дерева, скорость обучения, количество деревьев и другие параметры. Это позволяет найти оптимальные настройки модели для каждой конкретной задачи.
- Автоматический выбор признаков: LightGBM может автоматически выбирать наиболее важные признаки для моделирования. Это происходит путем оценки важности признаков на основе их вклада в улучшение целевой функции, но также можно использовать внешние алгоритмы выбора признаков.
- Параллельное обучение: LightGBM может обучаться параллельно на многих ядрах процессора, что позволяет сократить время обучения моделей и ускорить процесс выбора оптимальных гиперпараметров.

Важные параметры при обучении LightGBM:

- learning_rate: скорость обучения, которая определяет, насколько сильно корректируются веса при обновлении модели на каждой итерации. Выбор этого параметра влияет на скорость сходимости модели и на ее способность к обобщению.
- max_depth: максимальная глубина дерева, которая определяет количество уровней дерева решений. Выбор этого параметра влияет на способность модели к обобщению и на скорость обучения.
- min_child_samples: минимальное количество образцов, необходимых для создания нового узла. Выбор этого параметра влияет на устойчивость модели к шуму и на ее способность к обобщению.
- min_child_weight: минимальный вес дочернего узла, который определяет, какую минимальную величину должен иметь вес дочернего узла, чтобы продолжать делить узел. Выбор этого параметра влияет на устойчивость модели к шуму и на ее способность к обобщению.
- subsample: доля выборки, используемая для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- colsample_bytree: доля признаков, используемых для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- reg_alpha: коэффициент L1-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.
- reg_lambda: коэффициент L2-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.

0.3.4 Оценка качества алгоритма

Для оценки качества алгоритма LightGBM часто используют следующие метрики:

Классификация:

- **accuracy:** $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision:** $\frac{TP}{TP+FP}$
- **recall:** $\frac{TP}{TP+FN}$
- **F1:** $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- **MSE**
- **MAE**

0.3.5 Интерпретация признаков с помощью алгоритма

Чтобы интерпретировать признаки в LightGBM, можно использовать следующие методы:

- Важность признаков: LightGBM предоставляет встроенную функцию для оценки важности признаков. Эта функция рассчитывает важность признаков на основе их использования в деревьях решений. Важность признаков можно использовать для определения наиболее значимых признаков, влияющих на целевую переменную.
- SHAP значения: SHAP (SHapley Additive exPlanations) - это метод интерпретации модели, который позволяет определить влияние каждого признака на предсказание модели. LightGBM поддерживает SHAP значения, которые могут быть рассчитаны для каждого образца в данных. SHAP значения можно использовать для объяснения причин, по которым модель дает определенные предсказания.
- Визуализация деревьев решений: LightGBM позволяет визуализировать деревья решений, которые были созданы в ходе обучения модели. Визуализация деревьев позволяет легче понимать, какие признаки используются в модели и как они влияют на предсказание.
- Предсказания на новых данных: LightGBM позволяет сделать предсказания на новых данных. Если предсказания на новых данных достаточно точны, можно использовать эти предсказания для определения важности признаков. Если признаки не важны для предсказания на новых данных, то они могут быть удалены из модели.

0.3.6 Применение алгоритма

Основные области применения LightGBM включают в себя:

- **Финансы:** прогнозирование кредитных рисков, анализ финансовых рынков.
- **Реклама:** прогнозирование эффективности рекламы, оптимизация бюджета рекламных кампаний.
- **Интернет-магазины:** рекомендательные системы, прогнозирование продаж, сегментация аудитории.
- **Обработка естественного языка:** классификация текстов, анализ тональности, машинный перевод.

- **Изображения и видео:** распознавание объектов, классификация изображений, анализ видео.
- **Промышленность:** мониторинг и управление качеством продукции, прогнозирование отказов оборудования.
- **Транспорт и логистика:** прогнозирование спроса на перевозки, оптимизация маршрутов, анализ логистических данных.

В целом, LightGBM может быть полезным в любой области, где требуется классификация или регрессия на основе большого количества признаков и где есть достаточно данных для обучения модели. Он также может быть полезен для задач, связанных с обработкой естественного языка и изображений, и для задач прогнозирования в различных отраслях.

0.3.7 Плюсы и минусы алгоритма

Плюсы:

- Высокая скорость обучения: LightGBM использует алгоритм градиентного бустинга, который позволяет быстро обучать модель на больших объемах данных.
- Высокая точность предсказаний: LightGBM способен достичь высокой точности предсказаний, особенно при использовании большого количества деревьев.
- Эффективное использование памяти: LightGBM использует специальную структуру данных, которая позволяет эффективно использовать память и ускоряет обучение модели.
- Гибкость в настройке: LightGBM позволяет настраивать множество гиперпараметров, что позволяет добиться наилучшей производительности модели.

Минусы:

- Чувствительность к шуму и выбросам: LightGBM может быть чувствителен к шуму и выбросам в данных, что может приводить к переобучению модели.
- Неэффективность при наличии большого количества категориальных признаков с большим количеством уникальных значений: в этом случае может возникнуть проблема переобучения модели.
- Требуется подбор гиперпараметров: для достижения наилучшей производительности модели необходимо подобрать оптимальные гиперпараметры.
- Неинтерпретируемость: модель LightGBM не обеспечивает полного понимания, как именно происходит принятие решения.

0.3.8 Реализация алгоритма в Python

LightGBM реализован в библиотеке lightgbm (<https://lightgbm.readthedocs.io/en/latest/>). В библиотеке LightGBM для задач классификации существует класс **LGBMClassifier**, а для задач регрессии - класс **LGBMRegressor**. Основные параметры классов LGBMClassifier/LGBMRegressor:

- **objective:** целевая функция, которую оптимизирует LightGBM. Для задач классификации поддерживаются следующие целевые функции: «binary», «multiclass», «multiclassova», «cross_entropy», «cross_entropy_lambda», «xentropy», «xentlambda», «lambdarank», «rank_xendcg», «rank_xendcg_5», «rank_xendcg_10», «rank_ndcg»,

«rank_ndcg_5», «rank_ndcg_10», «regression», «regression_l1», «huber», «fair», «poisson», «gamma», «tweedie», «quantile», «mape», «gamma_regression», «tweedie_regression». Для задач регрессии поддерживаются следующие целевые функции: «regression», «regression_l1», «huber», «fair», «poisson», «quantile», «mape», «gamma», «tweedie», «gamma_regression», «tweedie_regression».

- **n_estimators:** количество деревьев в градиентном бустинге. По умолчанию n_estimators=100.
- **learning_rate:** шаг обучения, который контролирует вклад каждого дерева. По умолчанию learning_rate=0.1.
- **max_depth:** максимальная глубина каждого дерева. По умолчанию max_depth=-1, что означает, что деревья разрастаются до тех пор, пока не будет достигнуто минимальное количество элементов для разделения.
- **num_leaves:** максимальное количество листьев в каждом дереве. По умолчанию num_leaves=31.
- **min_data_in_leaf:** минимальное количество элементов, которые должны быть в листьях дерева. По умолчанию min_data_in_leaf=20.
- **min_child_samples:** минимальное количество элементов, необходимое для того, чтобы узел мог быть разделен на два подузла. По умолчанию min_child_samples=20.
- **feature_fraction:** количество признаков, которые должны быть рассмотрены при каждом разделении. По умолчанию feature_fraction=1.0.
- **bagging_fraction:** доля элементов, которые должны быть использованы при построении каждого дерева. По умолчанию bagging_fraction=1.0.
- **bagging_freq:** частота использования случайных элементов для построения каждого дерева. По умолчанию bagging_freq=0.

Классы LGBMClassifier/LGBMRegressor имеют методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, классы LGBMClassifier/LGBMRegressor имеют методы **score(X, y)** и **get_params()** для получения оценки точности модели и параметров модели соответственно.

0.3.9 Вопросы для самопроверки

Как происходит построение дерева в LightGBM по умолчанию?

1. Level-wise
2. Leaf-wise
3. Oblivious decision tree

Правильные ответы: 2

Выберите плюсы алгоритма LightGBM:

1. Высокая скорость обучения: LightGBM использует алгоритм градиентного бустинга, который позволяет быстро обучать модель на больших объемах данных.
2. Высокая точность предсказаний: LightGBM способен достичь высокой точности предсказаний, особенно при использовании большого количества деревьев.
3. Гибкость в настройке: LightGBM позволяет настраивать множество гиперпараметров, что позволяет добиться наилучшей производительности модели.

4. Неинтерпретируемость: модель LightGBM не обеспечивает полного понимания, как именно происходит принятие решения.
5. Неэффективность при наличии большого количества категориальных признаков с большим количеством уникальных значений: в этом случае может возникнуть проблема переобучения модели.
6. Чувствительность к шуму и выбросам: LightGBM может быть чувствителен к шуму и выбросам в данных, что может приводить к переобучению модели.
7. Эффективное использование памяти: LightGBM использует специальную структуру данных, которая позволяет эффективно использовать память и ускоряет обучение модели.

Правильные ответы: 1, 2, 3, 7

0.3.10 Резюме по разделу

LightGBM - это быстрый и эффективный алгоритм градиентного бустинга деревьев решений, который использует уникальные техники построения деревьев и распределения данных по группам для достижения высокой скорости и точности предсказаний. LightGBM имеет ряд преимуществ перед другими алгоритмами градиентного бустинга, включая меньшее время обучения, более высокую скорость предсказания и более низкое потребление памяти. LightGBM поддерживает работу с категориальными признаками, автоматическую настройку гиперпараметров и распределенное обучение на нескольких процессорах.

0.4 Catboost

Цель занятия: ученик может применить алгоритм Catboost для решения задач классификации и регрессии на подготовленных и неподготовленных данных.

План занятия:

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

0.4.1 Визуальная демонстрация алгоритма

CatBoost - это алгоритм градиентного бустинга деревьев решений от компании Yandex. Он может обрабатывать большие объемы данных и имеет высокую скорость обучения модели, за счет использования нескольких техник оптимизации, таких как категориальная обработка данных, симметричный случайный выбор (Symmetric Random Selection) и использование градиентов второго порядка (Hessian).

Одним из основных преимуществ CatBoost является его способность автоматически обрабатывать категориальные признаки, не требуя их предварительной обработки, что может значительно ускорить процесс обучения модели.

В отличие от LightGBM, в CatBoost используется стратегия обучения по слоям (level-wise), которая состоит в том, что на каждом уровне дерева выбираются все узлы для разделения одновременно, что позволяет более полно использовать информацию о данных. Это может привести к более точной модели, но может также увеличить время обучения.

В Catboost используется структура дерева «Oblivious Decision Tree» - это алгоритм построения дерева решений, который относится к классу «неведущих» (oblivious) алгоритмов. В отличие от обычных деревьев решений, которые могут использовать различные признаки на каждом уровне дерева, неведущие деревья используют только один и тот же признак на каждом уровне дерева (Рисунок 5).

CatBoost поддерживает распределенное обучение на нескольких компьютерах, многоклассовую классификацию и регрессию.

Общие шаги алгоритма CatBoost:

1. Предобработка данных и подготовка набора признаков
2. Обучение базовых моделей на первоначальном наборе данных
3. Создание и обучение ансамбля базовых моделей с использованием градиентного бустинга
4. Вычисление ошибки на каждой итерации градиентного бустинга и пересчет весов моделей с учетом ошибки
5. Применение ансамбля моделей на новых данных для решения задачи классификации или регрессии

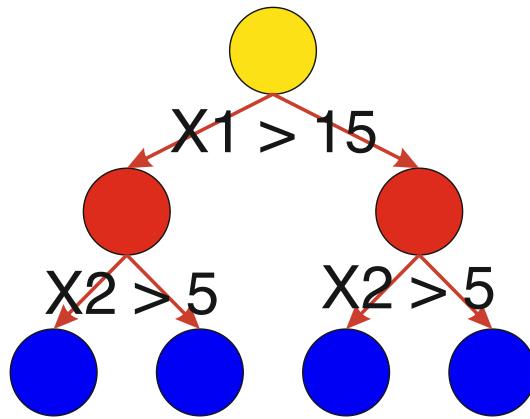


Рисунок 5: Oblivious Decision Tree в Catboost

CatBoost отличается от других алгоритмов градиентного бустинга тем, что он автоматически обрабатывает категориальные признаки и не требует их предварительного преобразования.

0.4.2 Подготовка данных для алгоритма

Подготовка данных для алгоритма Catboost может включать в себя следующие шаги:

- Обработка пропущенных значений. Если есть пропущенные значения, заполните их средними значениями, медианами или другими подходящими значениями.

0.4.3 Процесс обучения

Процесс обучения CatBoost включает в себя следующие шаги:

1. Подготовка данных и определение набора признаков.
2. Определение гиперпараметров модели, таких как количество деревьев, глубина деревьев, скорость обучения и т. д.
3. Обучение базовых моделей на первоначальном наборе данных.
4. Создание ансамбля базовых моделей с использованием градиентного бустинга.
5. Оценка качества модели на проверочном наборе данных для определения оптимального числа итераций и избежания переобучения.
6. Применение обученной модели для решения задачи классификации или регрессии.

CatBoost также поддерживает автоматическое кодирование категориальных признаков, встроенную кросс-валидацию и техники борьбы с переобучением, такие как регуляризация и сокращение градиента. Кроме того, CatBoost позволяет использовать несколько процессоров и графические процессоры для ускорения обучения модели.

Особенности обучения CatBoost:

- Обработка отсутствующих значений: CatBoost поддерживает обработку отсутствующих значений в данных, используя специальный токен для пропущенных значений, чтобы модель могла корректно обработать эти данные.

- Регуляризация: CatBoost использует регуляризацию для контроля сложности модели и избежания переобучения, но также использует методы сокращения данных и ограничения глубины деревьев.
- Подбор гиперпараметров: CatBoost позволяет настраивать множество гиперпараметров, таких как глубина дерева, скорость обучения, количество деревьев и другие параметры. Для нахождения оптимальных настроек модели можно использовать встроенные функции подбора гиперпараметров.
- Параллельное обучение: CatBoost также может обучаться параллельно на многих ядрах процессора, что позволяет сократить время обучения моделей и ускорить процесс выбора оптимальных гиперпараметров.
- Обработка категориальных признаков: CatBoost имеет уникальный алгоритм для работы с категориальными признаками, который позволяет автоматически преобразовывать категориальные признаки в числовые значения, не требуя предварительной обработки данных.

Важные параметры при обучении CatBoost:

- **learning_rate**: скорость обучения, которая определяет, насколько сильно корректируются веса при обновлении модели на каждой итерации. Этот параметр влияет на скорость сходимости модели и на ее способность к обобщению.
- **depth**: максимальная глубина дерева, которая определяет количество уровней дерева решений. Этот параметр влияет на способность модели к обобщению и на скорость обучения.
- **l2_leaf_reg**: коэффициент L2-регуляризации весов листьев деревьев. Этот параметр влияет на скорость обучения и на способность модели к обобщению.
- **min_data_in_leaf**: минимальное количество образцов в листе дерева. Этот параметр влияет на устойчивость модели к шуму и на ее способность к обобщению.
- **max_bin**: максимальное количество корзин для гистограммного метода построения деревьев. Этот параметр влияет на скорость обучения и на качество модели.
- **subsample**: доля выборки, используемая для обучения каждого дерева. Этот параметр влияет на способность модели к обобщению и на устойчивость к переобучению.
- **random_strength**: сила случайности, используемая при выборе случайных признаков для обучения каждого дерева. Этот параметр влияет на устойчивость к переобучению и на качество модели.
- **bagging_temperature**: температура распределения Больцмана, используемая при выборе объектов для обучения каждого дерева. Этот параметр влияет на способность модели к обобщению и на устойчивость к переобучению.

0.4.4 Оценка качества алгоритма

Для оценки качества алгоритма Catboost часто используют следующие метрики:

Классификация:

- **accuracy**: $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision**: $\frac{TP}{TP+FP}$
- **recall**: $\frac{TP}{TP+FN}$
- **F1**: $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- MSE
- MAE

0.4.5 Интерпретация признаков с помощью алгоритма

CatBoost предлагает различные методы интерпретации признаков, которые могут помочь в понимании важности признаков и их влияния на предсказания модели:

- Важность признаков: CatBoost предоставляет встроенную функцию для оценки важности признаков. Эта функция рассчитывает важность признаков на основе их использования в деревьях решений. Важность признаков можно использовать для определения наиболее значимых признаков, влияющих на целевую переменную.
- SHAP значения: CatBoost также поддерживает расчет SHAP значений для интерпретации модели. SHAP значения позволяют определить влияние каждого признака на предсказание модели и объяснить, почему модель дает определенные предсказания. Это может быть полезным для понимания важности и влияния признаков.
- Визуализация деревьев: CatBoost позволяет визуализировать деревья решений, построенные в ходе обучения модели. Визуализация деревьев позволяет лучше понять, какие признаки используются в модели и как они влияют на предсказание.
- Предсказания на новых данных: При сделанных предсказаниях на новых данных в CatBoost можно оценить важность признаков. Если признаки не важны для предсказания на новых данных, то они могут быть удалены из модели.

Эти методы интерпретации признаков помогают разобраться в модели и понять, какие признаки наиболее важны для ее предсказаний.

0.4.6 Применение алгоритма

CatBoost широко используется в различных областях и может быть применен для решения следующих задач:

- **Реклама и маркетинг:** прогнозирование кликов и конверсий, оптимизация рекламных кампаний, персонализация рекомендаций.
- **Финансы:** оценка кредитного scoringа, мошенническое обнаружение, прогнозирование рыночных трендов.
- **Здравоохранение:** диагностика заболеваний, прогнозирование рисков, анализ медицинских изображений.
- **Интернет-магазины:** рекомендательные системы, прогнозирование спроса, управление ассортиментом.
- **Телекоммуникации:** прогнозирование оттока клиентов, оптимизация сетей связи.
- **Транспорт и логистика:** маршрутизация, прогнозирование задержек, управление логистическими процессами.
- **Энергетика:** прогнозирование потребления энергии, оптимизация энергетических сетей.

CatBoost показывает хорошую производительность в задачах с большим числом категориальных признаков и может эффективно работать с разреженными данными. Он также может быть применен в задачах обработки естественного языка, компьютерного зрения и других областях, где требуется классификация и регрессия.

0.4.7 Плюсы и минусы алгоритма

Плюсы

- Обработка категориальных признаков: CatBoost автоматически обрабатывает категориальные признаки без необходимости их предварительного преобразования. Это упрощает процесс подготовки данных и позволяет моделировать задачи с большим количеством категориальных признаков.
- Устойчивость к переобучению: CatBoost включает в себя встроенные методы регуляризации, которые помогают предотвратить переобучение модели. Это делает его более устойчивым к шуму и выбросам в данных.
- Высокая точность и обобщающая способность: CatBoost показывает хорошие результаты на различных задачах и способен обобщать на новые данные. Он обладает высокой предсказательной точностью при правильной настройке гиперпараметров.
- Эффективное использование категориальных признаков: CatBoost использует специальный алгоритм для эффективной обработки категориальных признаков, что позволяет получать более точные предсказания.

Минусы

- Длительное время обучения: в некоторых случаях обучение CatBoost может требовать больше времени по сравнению с другими алгоритмами градиентного бустинга, особенно при большом количестве категориальных признаков.
- Требуется настройка гиперпараметров: CatBoost имеет множество гиперпараметров, которые необходимо настроить для достижения наилучшей производительности модели. Это может потребовать времени и вычислительных ресурсов.
- Высокое потребление памяти: CatBoost требует большого объема памяти для обработки больших объемов данных с большим количеством признаков, особенно при использовании GPU.

В целом, CatBoost является мощным алгоритмом градиентного бустинга с хорошей поддержкой категориальных признаков, но требует настройки гиперпараметров и может потребовать больше вычислительных ресурсов.

0.4.8 Реализация алгоритма в Python

CatBoost реализован в библиотеке `catboost` (<https://catboost.ai/en/docs/>). В библиотеке CatBoost для задач классификации существует класс **CatBoostClassifier**, а для задач регрессии - класс **CatBoostRegressor**. Основные параметры классов `CatBoostClassifier`/`CatBoostRegressor`:

- **loss_function:** целевая функция, которую оптимизирует CatBoost. Для задач классификации поддерживаются следующие целевые функции: «Logloss», «CrossEntropy», «MultiClass», «MultiClassOneVsAll», «AUC», «F1», «Accuracy», «PRAUC», «Recall»,

«Precision», «BalancedAccuracy». Для задач регрессии поддерживаются следующие целевые функции: «MAE», «MAPE», «Poisson», «Quantile», «RMSE», «SquaredLoss», «Huber», «LogLinQuantile».

- **n_estimators:** количество деревьев в градиентном бустинге. По умолчанию n_estimators=1000.
- **learning_rate:** шаг обучения, который контролирует вклад каждого дерева. По умолчанию learning_rate=0.03.
- **max_depth:** максимальная глубина каждого дерева. По умолчанию max_depth=6.
- **num_leaves:** максимальное количество листьев в каждом дереве. По умолчанию num_leaves=31.
- **min_data_in_leaf:** минимальное количество элементов, которые должны быть в листьях дерева. По умолчанию min_data_in_leaf=1.
- **colsample_bytree:** количество признаков, которые должны быть рассмотрены при каждом разделении на каждом уровне дерева. По умолчанию colsample_bytree=1.0.
- **subsample:** доля элементов, которые должны быть использованы при построении каждого дерева. По умолчанию subsample=1.0.
- **random_seed:** начальное значение для генератора случайных чисел. По умолчанию random_seed=None.

Классы CatBoostClassifier/CatBoostRegressor имеют методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, классы CatBoostClassifier/CatBoostRegressor имеют методы **score(X, y)** и **get_params()** для получения оценки точности модели и параметров модели соответственно.

0.4.9 Вопросы для самопроверки

Как происходит построение дерева в Catboost по умолчанию?

1. Level-wise
2. Leaf-wise
3. Oblivious decision tree

Правильные ответы: 3

Выберите плюсы алгоритма Catboost:

1. Высокая точность и обобщающая способность: CatBoost показывает хорошие результаты на различных задачах и способен обобщать на новые данные. Он обладает высокой предсказательной точностью при правильной настройке гиперпараметров.
2. Высокое потребление памяти: CatBoost требует большого объема памяти для обработки больших объемов данных с большим количеством признаков, особенно при использовании GPU.
3. Длительное время обучения: в некоторых случаях обучение CatBoost может требовать больше времени по сравнению с другими алгоритмами градиентного бустинга, особенно при большом количестве категориальных признаков.
4. Обработка категориальных признаков: CatBoost автоматически обрабатывает категориальные признаки без необходимости их предварительного преобразования. Это упрощает процесс подготовки данных и позволяет моделировать задачи с большим количеством категориальных признаков.

5. Требуется настройка гиперпараметров: CatBoost имеет множество гиперпараметров, которые необходимо настроить для достижения наилучшей производительности модели. Это может потребовать времени и вычислительных ресурсов.
6. Устойчивость к переобучению: CatBoost включает в себя встроенные методы регуляризации, которые помогают предотвратить переобучение модели. Это делает его более устойчивым к шуму и выбросам в данных.
7. Эффективное использование категориальных признаков: CatBoost использует специальный алгоритм для эффективной обработки категориальных признаков, что позволяет получать более точные предсказания.

Правильные ответы: 1, 4, 6, 7

0.4.10 Резюме по разделу

CatBoost - это еще один быстрый и эффективный алгоритм градиентного бустинга деревьев решений, который также использует уникальные техники построения деревьев и обработки данных. Он также имеет ряд преимуществ перед другими алгоритмами, включая автоматическую обработку категориальных признаков, быструю скорость обучения и предсказания, а также поддержку распределенного обучения.

Одним из основных преимуществ CatBoost является его способность работать с категориальными признаками без необходимости их предварительной обработки, что может существенно упростить процесс подготовки данных. CatBoost также обеспечивает автоматическую настройку гиперпараметров, что может помочь улучшить качество модели. Кроме того, CatBoost может использоваться для задач классификации, регрессии и ранжирования.

0.5 Резюме по модулю

Бустинг является методом машинного обучения, который позволяет улучшить качество прогнозов, создаваемых слабыми моделями машинного обучения. При использовании этого метода каждая последующая базовая модель фокусируется на неправильно классифицированных примерах данных предыдущих моделей, что позволяет создать сильную модель, которая дает более точные прогнозы.

Существует несколько подходов для реализации бустинга, такие как Adaboost и градиентный бустинг. Adaboost работает путем создания последовательности слабых взвешенных моделей машинного обучения и комбинирования их в единую сильную модель. Градиентный бустинг использует градиент, настраиваемый на остатки (разность между предсказаниями текущей модели и правильными ответами предыдущих моделей), чтобы определить, какие примеры данных были неправильно классифицированы предыдущими моделями, и насколько сильно эти примеры влияют на функцию потерь.

XGBoost, LightGBM и CatBoost являются эффективными и мощными алгоритмами градиентного бустинга, используемыми для задач классификации и регрессии. Они используют градиентный бустинг деревьев решений для улучшения качества модели путем последовательного добавления новых деревьев и корректировки ошибок предыдущих.

Основными преимуществами XGBoost являются высокая скорость обучения и предсказания, возможность работы с большими объемами данных, устойчивость к переобучению и возможность интерпретации результатов. LightGBM имеет ряд преимуществ перед другими алгоритмами градиентного бустинга, включая меньшее время обучения, более высокую скорость предсказания и более низкое потребление памяти. CatBoost также имеет ряд преимуществ перед другими алгоритмами, включая автоматическую обработку категориальных признаков, быструю скорость обучения и предсказания, а также поддержку распределенного обучения.

В целом, все три алгоритма предоставляют множество параметров для настройки модели, влияющих на качество модели. Они также могут использоваться для задач классификации, регрессии и ранжирования. Однако, выбор конкретного алгоритма зависит от конкретных задач и требований к модели.