ЛЕКЦИЯ. КОМПОЗИЦИИ АЛГОРИТМОВ

Бутстрэп

Дана выборка X. Равномерно возьмём из выборки X ℓ объектов с возвращением (то есть, в новой выборке будут повторяющиеся объекты). Получим выборку X_1. Повторяем процедуру N раз, получаем выборки $X_1, \dots, X_N$

БУТСТРЭП - способ сгенерировать различные подвыборки из исходной выборки

Бэггинг (bootstrap aggregation):

С помощью бутстрэпа мы получили выборки $X_1, \dots, X_n$. Обучим по каждой выборке модель - получим базовые алгоритмы $b_1(x), \dots, b_N(x)$

Построим новую функцию регрессии:

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

Случайный лес

- Возьмём в качестве базовых алгоритмов для бэггинга решающие деревья, т.е. каждое случайное дерево $b_i(x)$ построено по своей подвыборке X_i.
- В каждой вершине дерева будем искать разбиение не по всем признакам, а по подмножеству признаков.
- Дерево строится до тех пор, пока в листе не окажется n_{min} объектов.

ЛЕКЦИЯ. БУСТИНГ

Идея: строим набор алгоритмов, каждый из которых исправляет ошибку предыдущих

Решаем задачу регрессии с минимизацией квадратичной ошибки: $\frac{1}{2}\sum_{i=1}^{l}(a(x_i)-y_i)^2 \to \min_a$

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

$$a(x)=\sum_{n=1}^{N} b_n(x),$$

где базовые алгоритмы $b_n(x)$ принадлежат некоторому семейству A.

Шаг 1: ищем алгоритм $b_1(x)$, минимизирующий ошибку:

$$b_1(x)=\underset{b\in A}{\operatorname{argmin}}\ \frac{1}{2}\sum_{i=1}^{l}(b(x_i)-y_i)^2$$

Ошибка на объекте x:

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

Следующий алгоритм должен настраиваться на эту ошибку, т.е. целевая переменная для следующего алгоритма — это вектор ошибок s (а не исходный вектор y)

Шаг 2: ищем алгоритм $b_2(x)$, настраивающийся на ошибки s первого алгоритма:

$$b_2(x)=\underset{b\in A}{\operatorname{argmin}}\ \frac{1}{2}\sum_{i=1}^{l}(b(x_i)-s_i)^2$$

Следующий алгоритм $b_3(x)$ будем выбирать так, чтобы он минимизировал ошибку предыдущей композиции (т.е. $b_1(x)+b_2(x)$) и т.д.

Каждый следующий алгоритм настраиваем на ошибку предыдущих.

Шаг N: Ошибка:

$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i)$$

Ищем алгоритм $b(x)$:

$$b_N(x) = \underset{b \in A}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{\ell} \left(b(x_i) - s_i^{(N)} \right)^2$$

Градиентный бустинг

Пусть $L(y, z)$ - произвольная дифференцируемая функция потерь. Строим алгоритм $a_N(x)$ вида

$$a_L(x) = \sum_{n=1}^{L} \gamma_n b_n(x), \text{ где на } N\text{-м шаге}$$

$$b_N(x) = \underset{b \in A}{\operatorname{argmin}} \sum_{i=1}^{\ell} \left(b(x_i) - s_i^{(N)} \right)^2,$$

$$\cancel{s_i^{(N)} = y_i - a_{N-1}(x_i)} \to s_i^{(N)} = -\left.\frac{\partial L}{\partial z}\right|_{z = a_{N-1}(x_i)}$$

Коэффициент γ_N должен минимизировать ошибку:

$$\gamma_N = \min_{\gamma \in R} \sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i) \right)$$

Выбор базовых алгоритмов:

- Если базовые алгоритмы очень простые, то они плохо приближают антиградиент функции потерь, т.е. градиентный бустинг может свестись к случайному блужданию
- Если базовые алгоритмы сложные, то за несколько шагов бустинг подгонится под обучающую выборку,

и получим переобученный алгоритм.

Чаще всего в качестве базовых алгоритмов используют решающие деревья.
В таком случае решающие деревья не должны быть очень маленькими, а также очень глубокими.
Оптимальная глубина — 3-6 (зависит от задачи).

Возможное решение – сокращение шага

Сокращение шага (регуляризация):

$$\boxed{a_N = a_{N-1}(x) + \eta \gamma_N b_N(x), \quad \eta \in (0; 1]}$$

Чем меньше темп обучения η, тем меньше степень доверия к каждому базовому алгоритму, и тем лучше качество итоговой композиции.