ТЕХНОСФЕРА

Лекция 2.5
Алгоритмические композиции
Развязка

Владимир Гулин
https://goo.gl/p8BjOD

19 февраля 2016 г.

План лекции

Задача обучения с учителем

Постановка задачи

Пусть дан набор объектов $\mathcal{D} = \{(\mathbf{x}_i, y_i)\},\ \mathbf{x}_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i \in 1, \ldots, N$, полученный из неизвестной закономерности $y = f(\mathbf{x})$. Необходимо построить такую $h(\mathbf{x})$, которая наиболее точно аппроксимирует $f(\mathbf{x})$.

Будем искать неизвестную

$$h(\mathbf{x}) = C(a_1(\mathbf{x}), \ldots, a_T(\mathbf{x}))$$

$a_i(\mathbf{x}) : \mathcal{X} \to \mathcal{R},\ \forall i \in \{1, \ldots, T\}$ - базовые модели
$C : \mathcal{R} \to \mathcal{Y}$ - решающее правило

Алгоритмические композиции

Simple Voting

$$h(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} a_i(\mathbf{x})$$

Weighted Voting

$$h(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} b_i a_i(\mathbf{x}),\ \ b_i \in \mathcal{R}$$

Mixture of Experts

$$h(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} b_i(\mathbf{x}) a_i(\mathbf{x}),\ \ b_i(\mathbf{x}) : \mathcal{X} \to \mathcal{R}$$

Недостатки градиентного бустинга

- Переобучается на “малом” количестве данных
- Медленно сходится к точке оптимума, из-за shrinkage
- Строит неоптимальный набор базовых моделей

XGBoost

eXtreme Gradient Boosting

Крайне широко используется на Kaggle соревнованиях в настоящее время.

- Прост в использовании
  - Легко установить
  - Есть пакеты для R и python
- Еффективен
  - Автоматически параллелится на одной машине
  - Имеется возможность запуска на кластере
- Показывает хорошие результаты
  - В среднем, для большинства датасетов
- Обладает гибкостью
  - Можно задавать собственную целевую функцию для оптимизации
  - Можно тюнить внутренние параметры алгоритма

XGBoost

Идея:

Ошибка композиции на обучающей выборке для градиентного бустинга

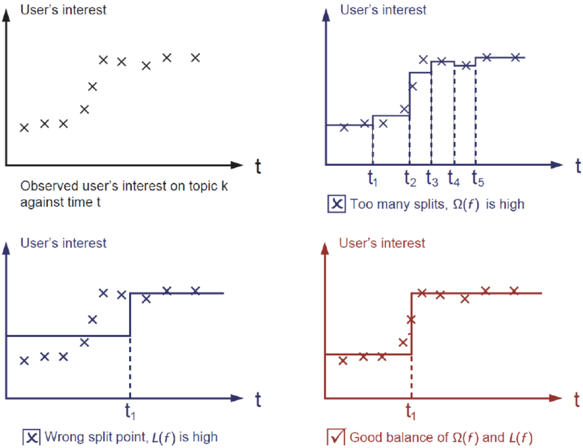
$$err(h) = \sum_{j=1}^{N} L(y_j, \sum_{i=1}^{T} b_i a_i(\mathbf{x}_j)))$$

XGBoost явно добавляет регуляризацию в этот функционал

$$err(h) = \sum_{j=1}^{N} L(y_j, \sum_{i=1}^{T} b_i a_i(\mathbf{x}_j))) + \sum_{i=1}^{T} \Omega(a_i)$$

XGBoost

Компромисс между функцией потерь и регуляризатором



XGBoost

Задача оптимизации:

Как мы обучаем обычный градиентный бустинг

$$err(h) = \sum_{j=1}^{N} L(y_j, \sum_{i=1}^{T-1} b_i a_i(\mathbf{x}_j) + b \cdot a(\mathbf{x}_j))) \to \min_{b,a}$$

Соответственно для XGBoost задача оптимизации модифицируется следующим образом:

$$err(h) = \sum_{j=1}^{N} L(y_j, \sum_{i=1}^{T-1} b_i a_i(\mathbf{x}_j) + b \cdot a(\mathbf{x}_j))) + \sum_{i=1}^{T-1} \Omega(a_i) + \Omega(a) \to \min_{b,a}$$

XGBoost

Наша текущая цель научиться подбирать базовую модель на каждой итерации бустинга

$$err(h^{(T)}) = \sum_{j=1}^{N} L(y_j, h^{(T-1)} + a(\mathbf{x}_j)) + \Omega(a) + const$$

Воспользуемся разложение в ряд Тейлора

$$f(x + \delta x) \approx f(x) + f'(x)\delta x + \frac{1}{2} f''(x)\delta x^2$$

Обозначим

$$g_j = \left[\frac{\partial L(y_j, h(\mathbf{x}_j))}{\partial h(\mathbf{x}_j)}\right]_{h=h^{(T-1)}}, \quad s_j = \left[\frac{\partial^2 L(y_j, h(\mathbf{x}_j))}{\partial h(\mathbf{x}_j)^2}\right]_{h=h^{(T-1)}}$$

XGBoost

Таким образом

$$err(h^{(T)}) \approx \sum_{j=1}^{N} \left[L(y_j, h^{(T-1)}) + g_j a(\mathbf{x}_j) + \frac{1}{2} s_j a^2(\mathbf{x}_j) \right] + \Omega(a) + const$$

Величины, не зависящие от a, можно исключить из выражения

$$\sum_{j=1}^{N} \left[g_j a(\mathbf{x}_j) + \frac{1}{2} s_j a^2(\mathbf{x}_j) \right] + \Omega(a) \to \min_a$$

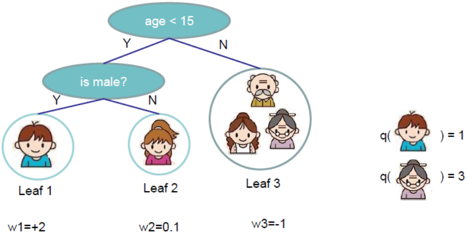
XGBoost

Что такое дерево решений?

Дерево решений состоит из набора предикатов и значий в листьях

$$a(\mathbf{x}) = w_{q(x)}, \quad w \in \mathcal{R}^Z, \quad q : \mathcal{R}^D \to \{1, .., Z\}$$

q - структура дерева, w - значения в листьях, Z - количество листьев



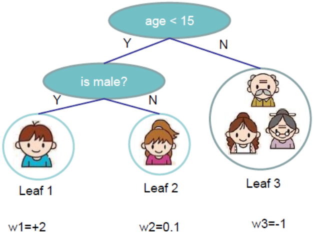
XGBoost

Сложность дерева

Определим сложность дерева как (например)

$$\Omega(a) = \gamma Z + \frac{1}{2}\lambda \sum_{i=1}^{Z} w_i^2$$

Средневзвешенная числа листьев и L_2 нормы значений в листьях



$$\Omega = \gamma 3 + \tfrac{1}{2}\lambda(4 + 0.01 + 1)$$

XGBoost

Обозначим факт поппаданию примера в $k-$ый лист

$$I_k = \{j | q(\mathbf{x_j} = k)\}$$

Перепишем целевую функцию в следующем виде:

$$\sum_{j=1}^{N} \left[g_j a(\mathbf{x}_j) + \frac{1}{2} s_j a^2(\mathbf{x}_j) \right] + \Omega(a) =$$

$$= \sum_{j=1}^{N} \left[g_j w_{q(\mathbf{x}_j)} + \frac{1}{2} s_j w_{q(\mathbf{x}_j)}^2 \right] + \gamma Z + \lambda \frac{1}{2} \sum_{k=1}^{Z} w_k^2 =$$

$$= \sum_{k=1}^{Z} \left[\left(\sum_{j \in I_k} g_j \right) w_k + \frac{1}{2} \left(\sum_{j \in I_k} s_j + \lambda \right) w_k^2 \right] + \gamma Z$$

Получили сумму Z назависимых квадратичных функций

- Как теперь получить значения, которые должны быть в листьях дерева?

XGBoost

Напоминание. Для квадратичной функции одной переменной

$$argmin_x Gx + \frac{1}{2}Sx^2 = -\frac{G}{S},\ S > 0, \quad \min_x Gx + \frac{1}{2}Sx^2 = -\frac{1}{2}\frac{G^2}{S}$$

Обозначим $G_k = \sum_{j \in I_k} g_j, \quad S_k = \sum_{j \in I_k} s_j$

$$\sum_{k=1}^{Z} \left[\left(\sum_{j \in I_k} g_j \right) w_k + \frac{1}{2} \left(\sum_{j \in I_k} s_j + \lambda \right) w_k^2 \right] + \gamma Z =$$

$$= \sum_{k=1}^{Z} \left[G_k w_k + \frac{1}{2}(S_j + \lambda) w_k^2 \right] + \gamma Z$$

Таким образом, для фиксированной структуры дерева $q(\mathbf{x})$, оптимальными значениями для листьев будут

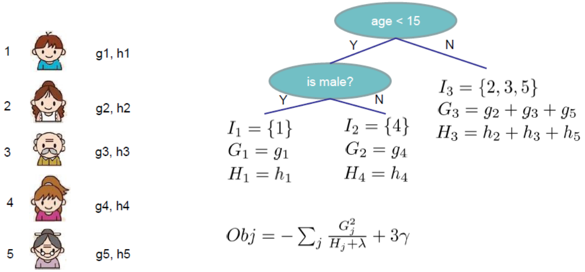
$$w_k^* = -\frac{G_k}{S_k + \lambda},\ k = 1, \dots, Z$$

XGBoost

При этом значение оптимизируемой функции, примет вид

$$err = -\frac{1}{2}\sum_{k=1}^{Z}\frac{G_k^2}{S_k+\lambda} + \gamma Z$$

Фактически, это мера того насколько “кошерное” дерево мы получили



XGBoost

Как же все таки вырастить дерево?

- Рассмотрим все возможные структуры дерева...
- Для каждой структуры вычислим score

$$err = -\frac{1}{2}\sum_{k=1}^{Z}\frac{G_k^2}{S_k+\lambda} + \gamma Z$$

- Выбрав лучшую структура дерева, рассчитаем оптимальные значения для листьев

$$w_k^* = -\frac{G_k}{S_k+\lambda},\ k = 1,\dots,Z$$

- Видите ли вы какие-нибудь проблемы в таком алгоритме?

XGBoost

Жадный рекурсивный алгоритм построения дерева

- Стартуем с дерева глубины 0
- Для каждого листа дерева пытаемся добавить новое разбиение

$$Gain = \frac{G_l^2}{S_l + \lambda} + \frac{G_r^2}{S_r + \lambda} - \frac{(G_l + G_r)^2}{S_l + S_r + \lambda} - \gamma$$

- Выбираем лучшее разбиение максимизируя *Gain*

XGBoost

Регуляризация

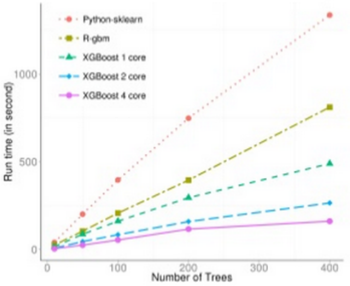
- Однако, *Gain* может быть отрицательным, если $\gamma >$ *loss reduction*

$$Gain = \frac{G_l^2}{S_l + \lambda} + \frac{G_r^2}{S_r + \lambda} - \frac{(G_l + G_r)^2}{S_l + S_r + \lambda} - \gamma$$

- Pre-stopping
  - Останавливаем ветвление, если лучшее разбиение имеет отризательный *Gain*
  - Правильно ли так поступать?
- Post-Pruning
  - Растим дерево до упора, а затем подрезаем листовые ветвления отризательным *Gain*

XGBoost

Хорошая реализация на C++



XGBoost

Результаты

- Kaggle CrowdFlower
- Kaggle Malware Prediction
- Kaggle Tradeshift
- Kaggle Higgs competition
- ...

Остальное можно почитать на `https://xgboost.readthedocs.org`

Вопросы