ML BD

Katya Koshchenko

Spring 2020

1 Лекция 6. Tree Models

1.1 Деревья и построение

Задача регрессии: $D* = (x, y)_{i=1}^{n}, x_i \in R^m, y_i \in R$.

Loss: $L(y, f(x)) = (f(x) - y)^2$. MSE: $E_{xy}[L(y, f(x))] - > min_\theta$.

Модель: $f(x) = w_{q(x)}, w \in R^T, q = R_1, ..., R_T$.

Node дерева: $1x_R < S$.

Из теории знаем, что поиск оптимального дерева NP полно, так что обычно строят деревья сверху вниз. Алгоритм построения (n — node D$\subset D*$) :

(n -> split D_L, D_R) <- FindBestSplit(D)
if StopingCriteria(D_L):
  n->LeftPredict<-FindPredict(D_L)
else:
  Algorithm(n-> left, D_L)

FindBestSplit. Выбираем его по значению Impurity. Оно равно Imp текущих объектов в ситуации, когда ничего не сплитуем. То есть считаем предикт на текущих объектах:

$$I = Imp(D) - [\frac{|D_L|}{|D|} \cdot Imp(D_L) + \frac{|D_R|}{|D|} \cdot Imp(D_R)] - > max$$

То есть хотим минимизировать эту вот сумму вероятностную в скобках.

Как выглядит Impurity в случае регрессии, что происходит с предиктом. Пусть предикт это лист, в нем хотим простую функцию. Самый простой вариант — число. Хотим такое число, чтобы MSE по объектам в этом листе было минимальным.

$$a* = argmin_a \sum_{i \in D} (a - y_i)^2$$

откуда получаем

$$a* = \frac{1}{|D|}\sum_{i \in D} y_i$$

Теперь посчитаем значение Impurity:

$$Imp(D) = \frac{1}{|D|}\sum_{i \in D}(a - y_i)^2 = Var(D)$$

и подставим это в нашу формулу для сплита:

$$I = |D|Var(D) - (|D_L|Var(D_L) + |D_R|Var(D_R))$$

1.2 Mean Target Embedding

У нас на категориях не задан порядок, а мы говорили, что в узле есть индикаторы, где значения фичей сравниваются. Для категориальных фичей можно делать ванхот, но если на нем просто обучать дерево, то будет очень много фичей, а значит долго и глубоко. Итого, проблемы с one-hot encoding: 1) Длинные пути для каждой категориальной фичи, и тогда не обучишь. 2) В решающих деревьях для фичей можно считать feature importance (всякие gain, color и тд). Скажем, что важность для конкретной фичи — как сильно она позваляет уменьшить ошибку в результате сплита. Если сделали сплит по фиче в нескольких местах дерева, то просуммировали уменьшение impurity во всех этих местах после сплита. И так получается, что ближе к корню дерева в этих терминах будут более важные фичи, тк выгодна близко к корню разбивать дерево так, чтобы слева и справа была маленькая ошибка. А тк фича теперь ванхот, то маловероятно, что сплиты окажутся близко к корню. Каждая из ванхот фичей будет маленький вклад вносить, а значит важность категориальной фичи будет маленькой. P.S.В топе обычно оказываются исторические фичи: как часто кликали на эту рекламу пользователи, или как часто сам юзер кликает на рекламу и тд.

То есть ванхот для деревьев не подходит. Придумали, как категорию заменить на вещественное число — Mean Target Embedding. Есть категориальная фича, x_{ik} — для объекта i фичу k категориальную хотим заменить на вещественное:

$$z_{ik} = \frac{\sum_{j=1}^{n} 1(x_{jk} = x_{ik}) \cdot y_i}{\sum_{j=1}^{n} 1(x_{jk} = x_{ik})}$$

Что это такое: смотрим на всех юзеров, у которых значение фичи k совпало с нашим и суммируем их предикт, нормализуя на количество. Проблема: если какое-то значение фичи встречатся редко, то эта штука оч шумно себя ведет. Борятся с этим сглаживанием:

$$z_{ik} = \frac{\sum_{j=1}^{n} 1(x_{jk} = x_{ik}) \cdot y_i + \alpha \cdot P\_apriori}{\sum_{j=1}^{n} 1(x_{jk} = x_{ik}) + \alpha}$$

Откуда взять априорное: из истории, среднее по всему датасету, или среднее значение по рекламодателю данного объявления. Или, как другой вариант, есть CatBoost, они берут перестановку датасета и считают сумму по всем объектам в перестановке ДО рассматриваемого.
Когда вообще полезно смотреть на предыдущие? Если упорядочен датасет по времени, то не хотим включать информацию из будущего.

1.3 Большие деревья

Как строить дереьвя для больших данных: бьем на бакеты, то есть для каждой фичи строим гистаграмму распределения, делаем бакеты (множество кандидатов). Дальше MapReduce. Map(D*, M — текущее состояние модели, N — nodes). И храним табличку, где строки — узлы, столбцы — категориальные фичи:

$$T_{nk}[split\_point] = \{\sum y, \sum y^2, \sum 1\}$$

Сумма берется по всем объектам, которые были приняты мапом изначально, которые добрались до узла n и у которых значение фичи k меньше, чем split_point.

На выход Map выкидывает список пар ключ-значение. Ключ — (n, k, split point). Значение — фрагмент таблицы ($T_{nk}[s]$), то есть после reduce посчитаем настоящее значение, саггрегировав все значения по этому ключу.

Научили обучать одно дерево. Теперь можно ансамбли. Random forest, boosting и тд.

1.4 Gradient Boosting

$$f*(x) = argmin_{f \in F} E_{xy}[L(y, f(x))]$$

$$f(x) = \sum_{m=0}^{M} \beta_m \cdot h(x, a_m)$$

$$h_m = \frac{dL}{df_{m-1}} = [\frac{dL(y, f(x))}{df(x)}]_{f(x)=f_{m-1}(x)}$$

Stage-wise:

$$(\beta_m, a_m) = argmin_{\beta,a} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta \cdot h(x_i, a))$$

$$f_{m-1} = (h_0, ..., h_{m-1})$$

$$f_m(x) = f_{m-1}(x) + \beta_m \cdot h(x, a_m) \forall m \beta_m = \theta - const$$

Пусть $L(y, f(x)) = 0.5(f(x) - y)^2$.

$$h_m = [\frac{dL(y, f(x))}{df(x)}]_{f(x)=f_{m-1}(x)} = f_{m-1}(x) - y$$