1 Расскажите, как работает регуляризация в решающих деревьях, какие параметры мы штрафуем в данных алгоритмах?

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Элементы контролируемого обучения

XGBoost используется для задач контролируемого обучения, где мы используем данные обучения (с несколькими функциями) xi прогнозировать целевую переменную yi Прежде чем мы узнаем конкретно о деревьях, давайте начнем с обзора основных элементов контролируемого обучения.

Модель и параметры

Модель в контролируемом обучении обычно относится к математической структуре, с помощью которой прогноз y_i делается на основе входных данных x_i . Типичным примером является линейная модель, где прогноз задается как

$$\hat{y}_i = \sum_j \theta_j \times x_{ij}$$

линейная комбинация взвешенных входных функций. Значение прогноза может иметь разные интерпретации в зависимости от задачи, например, регрессии или классификации. Например, его можно логистически преобразовать, чтобы получить вероятность положительного класса в логистической регрессии, а также его можно использовать в качестве рейтингового балла, когда мы хотим ранжировать выходные данные.

Целевая функция: Потери в обучении + Регуляризация

При разумном выборе y_i мы можем решать множество задач, таких как регрессия, классификация и ранжирование. Задача обучения модели сводится к нахождению наилучших параметров θ , которые наилучшим образом соответствуют обучающим данным x_i и меткам y_i . Чтобы обучить модель, нам нужно определить целевую функцию, чтобы измерить, насколько хорошо модель соответствует обучающим данным.

Отличительной особенностью целевых функций является то, что они состоят из двух частей: потери при обучении и члена регуляризации:

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

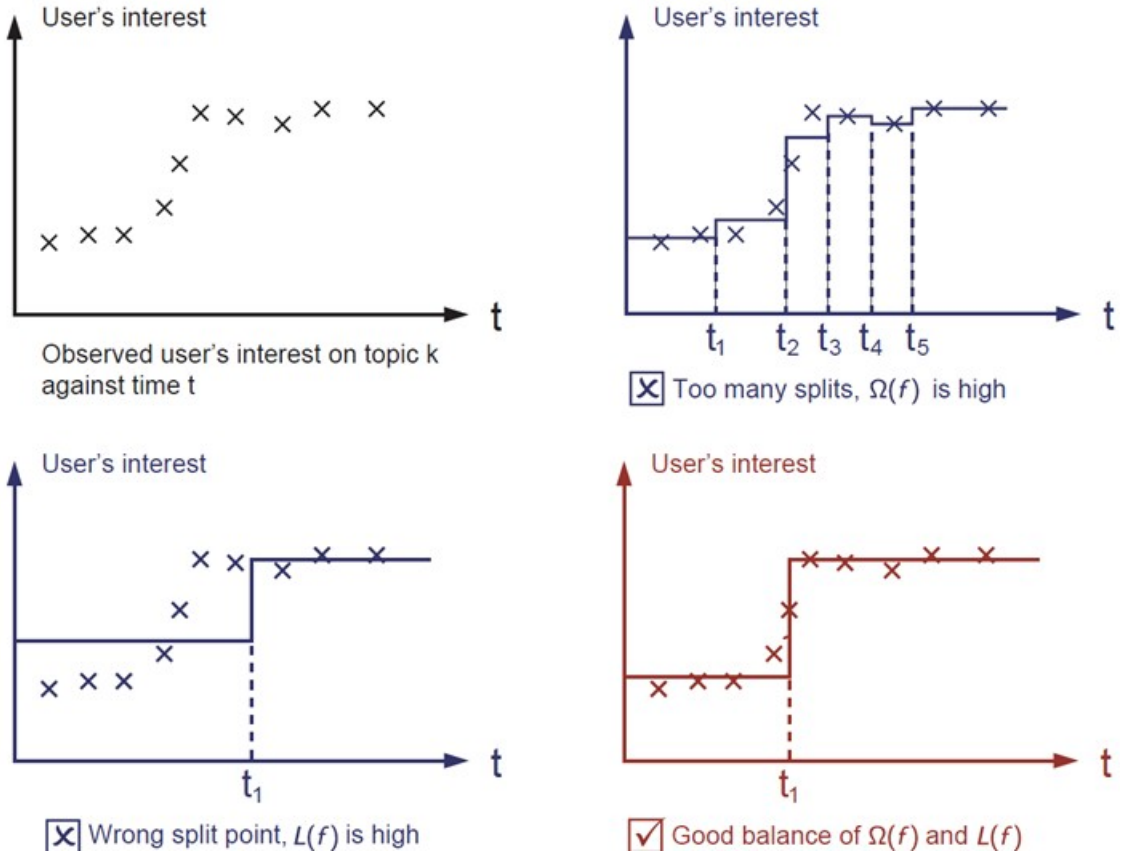
где L - функция обучающих потерь, а Ω - член регуляризации. Потери при обучении измеряют, насколько наша модель предсказуема по отношению к обучающим данным. Обычный выбор L - это среднеквадратичная ошибка, которая определяется как

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

Еще одна часто используемая функция потерь - это логистические потери, которые следует использовать для логистической регрессии:

$$L(\theta) = \sum_i (y_i \ln(1 + e^{\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i}))$$

Термин регуляризации - это то, что люди обычно забывают добавить. Термин регуляризации контролирует сложность модели, что помогает избежать переобучения. Это звучит немного абстрактно, поэтому давайте рассмотрим следующую проблему на следующем рисунке. Вам предлагается визуально подогнать пошаговую функцию с учетом точек входных данных в верхнем левом углу изображения. Какое решение из трех, по вашему мнению, лучше всего подходит?

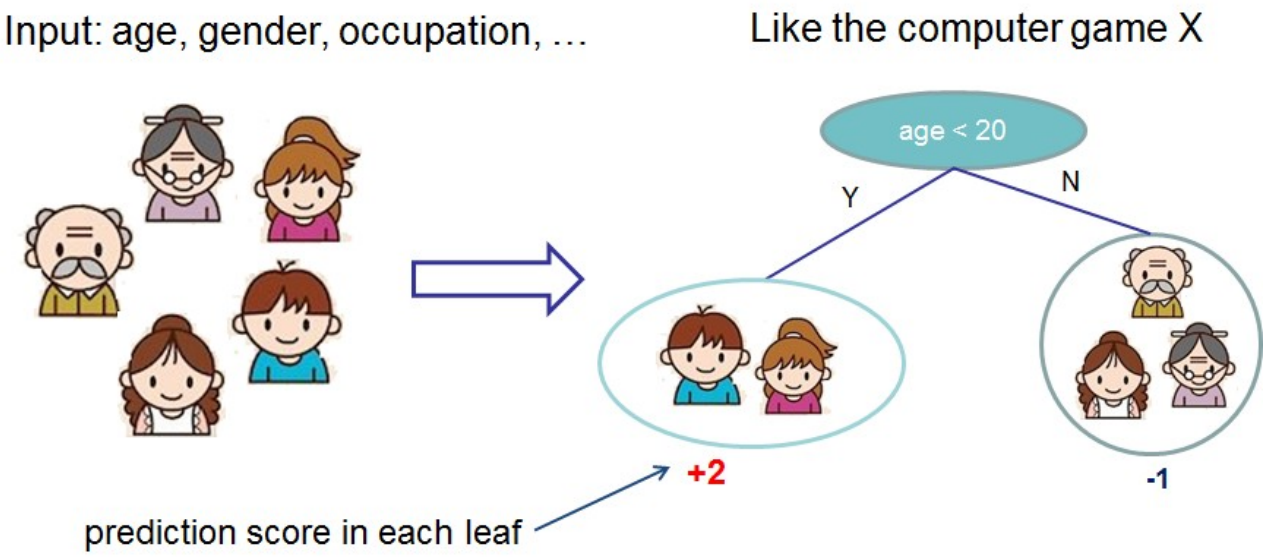


Правильный ответ отмечен красным. Пожалуйста, подумайте, подходит ли вам это визуально. Общий принцип заключается в том, что нам нужна как простая, так и прогностическая модель. Компромисс между ними также называется компромиссом смещения и дисперсии в машинном обучении.

2 По какому принципу рассчитывается "важность признака (feature_importance)" в ансамблях деревьев?

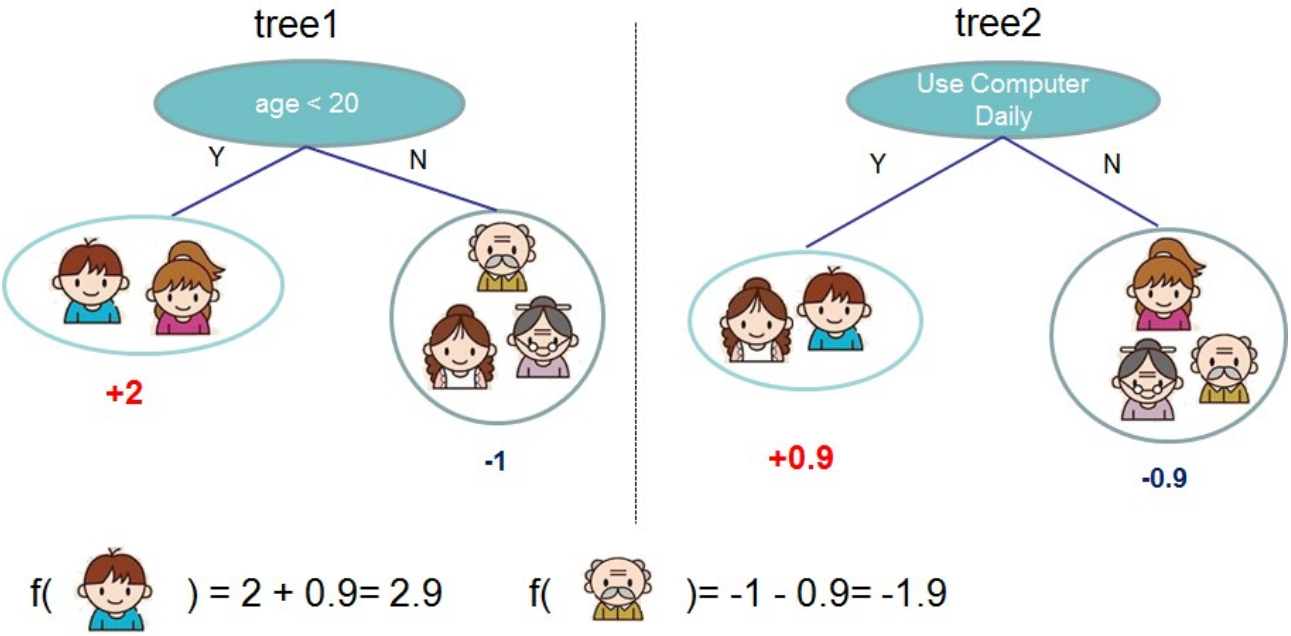
Ансамбли дерева решений

Теперь, когда мы ввели элементы контролируемого обучения, давайте начнем с настоящих деревьев. Для начала давайте сначала узнаем о выборе модели ансамблей XGBoost: дерево решений. Модель ансамбля деревьев состоит из набора деревьев классификации и регрессии (CART). Вот простой пример CART, которая определяет, понравится ли кому-то гипотетическая компьютерная игра X.



Мы классифицируем членов семьи по разным листам и присваиваем им оценку на соответствующем листе. CART немного отличается от деревьев решений, в которых лист содержит только значения решений. В CART реальная оценка связана с каждым листом, что дает нам более богатые интерпретации, выходящие за рамки классификации. Это также позволяет использовать принципиальный единый подход к оптимизации, как мы увидим в более поздней части этого руководства.

Обычно одного дерева недостаточно, чтобы его можно было использовать на практике. На самом деле используется модель ансамбля, которая суммирует предсказания нескольких деревьев вместе.



Вот пример ансамбля из двух деревьев. Баллы прогнозов каждого отдельного дерева суммируются, чтобы получить окончательный результат. Если вы посмотрите на пример, важным фактом является то, что два дерева пытаются дополнять друг друга. Математически мы можем записать нашу модель в виде

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

где K - количество деревьев, f - функция в функциональном пространстве F, а F - множество всех возможных CART. Оптимизируемая целевая функция определяется выражением

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

Теперь возникает вопрос с подвохом: какая модель используется в случайных лесах? Ансамбли деревьев! Так что случайные леса и усиленные деревья на самом деле одни и те же модели; разница в том, как мы их тренируем. Это означает, что, если вы пишете службу прогнозирования для ансамблей деревьев, вам нужно написать только один, и он должен работать как для случайных лесов, так и для деревьев с градиентным усилением. (См. Реальный пример в Treelite.) Один из примеров того, почему элементы контролируемого обучения рушатся.

Улучшение деревьев (Tree Boosting)

Теперь, когда мы представили модель, давайте перейдем к обучению: как нам изучать деревья? Ответ, как и всегда для всех моделей контролируемого обучения: определите целевую функцию и оптимизируйте ее!

Пусть следующая целевая функция (помните, что она всегда должна содержать потерю обучения и регуляризацию):

$$obj(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Аддитивное обучение

Первый вопрос, который мы хотим задать: каковы параметры деревьев? Вы можете обнаружить, что нам нужно изучить эти функции f_i , каждая из которых содержит структуру дерева и оценки листьев. Изучение древовидной структуры намного сложнее, чем традиционная задача оптимизации, когда вы можете просто взять градиент. Выучить сразу все деревья сложно. Вместо этого мы используем аддитивную стратегию: исправляем то, что мы узнали, и добавляем по одному новому дереву за раз. Мы записываем значение прогноза на шаге t как $\hat{y}_t^{(t)}$. Тогда у нас есть:

$$\hat{y}_t^{(0)} = 0$$

$$\hat{y}_t^{(1)} = f_1(x_i) = \hat{y}_t^{(0)} + f_1(x_i)$$

$$\hat{y}_t^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_t^{(1)} + f_2(x_i)$$

…

$$\hat{y}_t^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_t^{(1)} + f_2(x_i)$$

Остается спросить: какое дерево мы хотим на каждом шаге? Естественно добавить тот, который оптимизирует нашу цель.

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_k)$$

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

Если мы рассмотрим использование среднеквадратичной ошибки (MSE) в качестве нашей функции потерь, цель станет

$$\begin{aligned} \text{obj}^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{constant} \end{aligned}$$

Форма MSE удобна, с членом первого порядка (обычно называемым остатком) и квадратичным членом. Для других потерь процентов (например, логистических потерь) получить такую красивую форму не так-то просто. Итак, в общем случае мы берем разложение Тейлора функции потерь до второго порядка:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

где g_i и h_i определены как

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned}$$

После удаления всех констант конкретная цель на шаге t становится:

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Это становится нашей целью оптимизации для нового дерева. Одним из важных преимуществ этого определения является то, что значение целевой функции зависит только от g_i и h_i . Таким образом XGBoost поддерживает настраиваемые функции потерь. Мы можем оптимизировать каждую функцию потерь, включая логистическую регрессию и попарное ранжирование, используя точно такой же решатель, который принимает на вход g_i и h_i !