Градиентый бустинг — просто о сложном

27 ноября 2018 Базовый курс

Хотя большинство победителей соревнований на Kaggle используют композицию разных моделей, одна из них заслуживает особого внимания, так как является почти обязательной частью. Речь, конечно, про **Градиентный бустинг** (GBM) и его вариации. Возьмем, например. победителя Safe Driver Prediction, Michael Jahrer. Его решение — это комбинация шести моделей.
Одна LightGBM (вариация GBM) и пять нейронных сетей. Хотя его успех в большей мере принадлежит полуконтролируемому обучению, которое он использовал для упорядочивания данных, градиентный бустинг сыграл свою роль.

Даже несмотря на то, что градиентный бустинг используется повсеместно, многие практики до сих пор

предустановленных библиотек. Цель этой статьи — дать понимание как же работает градиентный бустинг. Разбор будет посвящен чистому “vanilla” GMB.

Ансамбли, бэггинг и бустинг

Когда мы пытаемся предсказать целевую переменную с помощью любого алгоритма машинного обучения, главные причины отличий реальной и предсказанной переменной — это noise, variance и bias. Ансамбль помогает уменьшить эти факторы (за исключением noise — это неуменьшаемая величина).

Ансамбль

Ансамбль — это набор предсказателей, которые вместе дают ответ (например, среднее по всем). Причина почему мы используем ансамбли — несколько предсказателей, которые пытаюсь получить одну и ту же переменную дадут более точный результат, нежели одиночный предсказатель. Техники ансамблирования впоследствии классифицируются в Бэггинг и Бустинг.

Бэггинг

Бэггинг — простая техника, в которой мы строим независимые модели и комбинируем их, используя некоторую модель усреднения (например, взвешенное среднее, голосование большинства или нормальное среднее).

Обычно берут случайную подвыборку данных для каждой модели, так все модели немного отличаются друг от друга. Выборка строится по модели выбора с возвращением. Из-за

модели, это уменьшает variance. Примером бэггинга служит модель случайного леса (Random Forest, RF)

Бустинг

Бустинг — это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно

Это техника использует идею о том, что следующая модель будет учится на ошибках предыдущей. Они имеют неравную вероятность появления в последующих моделях, и чаще появятся те, что дают наибольшую ошибку. Предсказатели могут быть выбраны из широкого ассортимента моделей, например, деревья решений, регрессия, классификаторы и т.д. Из-за того, что предсказатели обучаются на ошибках, совершенных предыдущими, требуется меньше времени для того, чтобы добраться до реального ответа. Но мы должны выбирать критерий остановки с осторожностью, иначе это может привести к переобучению. Градиентный бустинг — это пример бустинга.

Алгоритм градиентного бустинга

Градиентный бустинг — это техника машинного обучения для задач классификации и регрессии, которая строит модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений.

Цель любого алгоритма обучения с учителем — определить функцию потерь и минимизировать её. Давайте обратимся к математике градиентного бустинга. Пусть, например, в

$$Loss = MSE = \sum (y_i - y_i^p)^2$$

where, y_i = ith target value, y_i^p = ith prediction, $L(y_i, y_i^p)$ is Loss function

Мы хотим, чтобы построить наши предсказания таким образом, чтобы MSE была минимальна. Используя градиентный спуск и обновляя предсказания, основанные на скорости обучения (learning rate), ищем значения, на которых MSE минимальна.

$$y_i^p = y_i^p + \alpha * \delta \sum (y_i - y_i^p)^2 / \delta y_i^p$$

which becomes, $y_i^p = y_i^p - \alpha * 2 * \sum (y_i - y_i^p)$

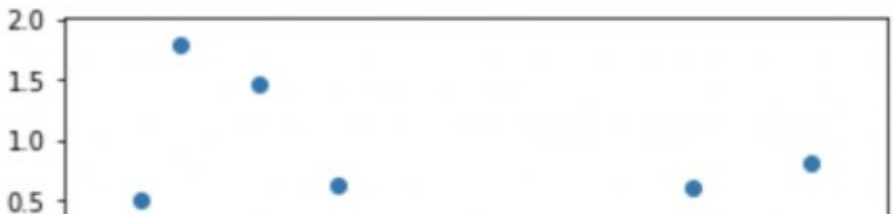
where, α is learning rate and $\sum (y_i - y_i^p)$ is sum of residuals

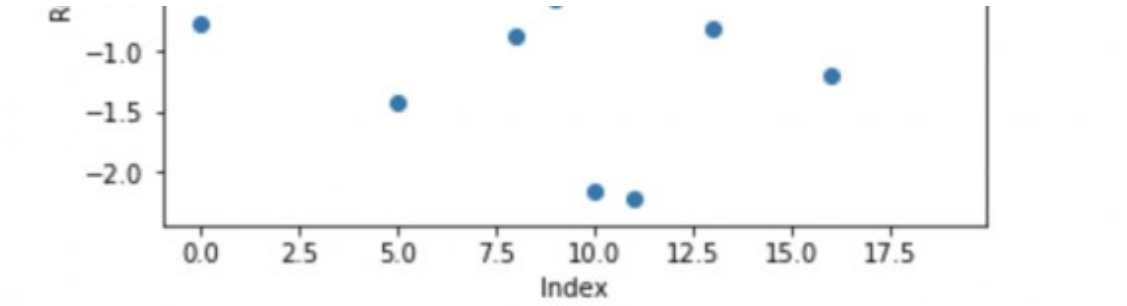
Итак, мы просто обновляем предсказания таким образом, что сумма наших отклонений стремилась к нулю и предсказанные значения были близки к реальным.

Интуиция за градиентным бустингом

Логика, что стоит за градиентым бустингом, проста, ее можно понять интуитивно, без математического формализма. Предполагается, что читатель знаком с простой линейной регрессией.

Первое предположение линейной регресии, что сумма отклонений = 0, т.е. отклонения должны быть случайно распределены в окрестности нуля.





Нормальное распределение выборки отклонений со средним 0

Теперь давайте думать о отклонениях, как об ошибках, сделанных нашей моделью. Хотя в моделях основанных на деревьях не делается такого предположения, если мы будем размышлять об этом предположении логически (не статистически), мы можем понять, что увидив принцип распределения отклонений, сможем использовать данный паттерн для модели.

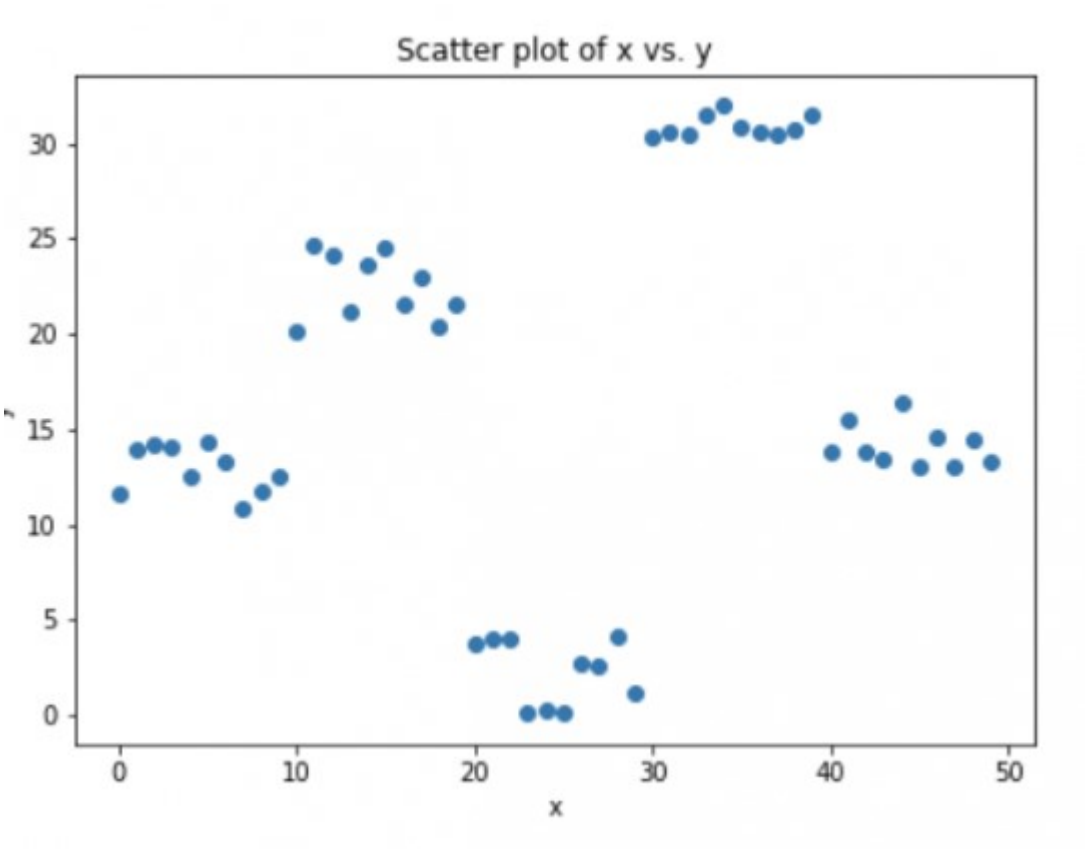
Итак, интуиция за алгоритмом градиентного бустинга — **итеративно применять паттерны отклонений и улучшать предсказания**. Как только мы достигли момента, когда отклонения не имеют никакого паттерна, мы прекращаем достраивать нашу модель (иначе это может привести к переобучению). Алгоритмически, мы минимизируем нашу функцию потерь.

В итоге,

- Сначала строим простые модели и анализируем ошибки;
- Определяем точки, которые не вписываются в простую модель;
- Добавляем модели, которые обрабатывают сложные случаи, которые были выявлены на начальной модели;
- Собираем все построенные модели, определяя вес

Шаги построения модели градиентного спуска

Рассмотрим смоделированные данные, как показано на диаграмме рассеивания ниже с 1 входным (x) и 1 выходной (y) переменными.



Данные для показанного выше графика генерируются с использованием кода python:

```
x = np.arange(0,50)
x = pd.DataFrame({'x':x})

# just random uniform distributions in differnt range

y1 = np.random.uniform(10,15,10)
y2 = np.random.uniform(20,25,10)
y3 = np.random.uniform(0,5,10)
y4 = np.random.uniform(30,32,10)
y5 = np.random.uniform(13,17,10)
```

random_data hosted with ❤ by GitHub view raw

1. Установите линейную регрессию или дерево решений на данные (здесь выбрано дерево решений в коде) [вызов *x* как input и *y* в качестве output]

```
xi = x # initialization of input
yi = y # initialization of target
# x,y --> use where no need to change original y
ei = 0 # initialization of error
n = len(yi)  # number of rows
predf = 0 # initial prediction 0

for i in range(30): # loop will make 30 trees (n_estimators).
    tree = DecisionTree(xi,yi) # DecisionTree scratch code can be found in sha
                               # It just create a single decision tree with pr
    tree.find_better_split(0)  # For selected input variable, this splits (<n
                               # target variable in both splits is minimum as

    r = np.where(xi == tree.split)[0][0]   #  finds index where this best spli

    left_idx = np.where(xi <= tree.split)[0] # index lhs of split
    right_idx = np.where(xi > tree.split)[0] # index rhs of split
```

find_split hosted with ❤ by GitHub view raw

2. Вычислите погрешности ошибок. Фактическое целевое значение, минус прогнозируемое целевое значение [e1 = y — y_predicted1]

3. Установите новую модель для отклонений в качестве целевой переменной с одинаковыми входными переменными [назовите ее e1_predicted]

4. Добавьте предсказанные отклонения к предыдущим прогнозам
[y_predicted2 = y_predicted1 + e1_predicted]

5. Установите еще одну модель оставшихся отклонений. т.е.

Управление overfitting-ом может контролироваться путем постоянной проверки точности на данных для валидации.

```
1       # predictions by ith decisision tree
2
3       predi = np.zeros(n)
4       np.put(predi, left_idx, np.repeat(np.mean(yi[left_idx]), r))  # replace le
5       np.put(predi, right_idx, np.repeat(np.mean(yi[right_idx]), n-r))  # right
6
7       predi = predi[:,None]  # make long vector (nx1) in compatible with y
8       predf = predf + predi  # final prediction will be previous prediction valu
9
10      ei = y - predf  # needed originl y here as residual always from original y
11      yi = ei # update yi as residual to reloop
```

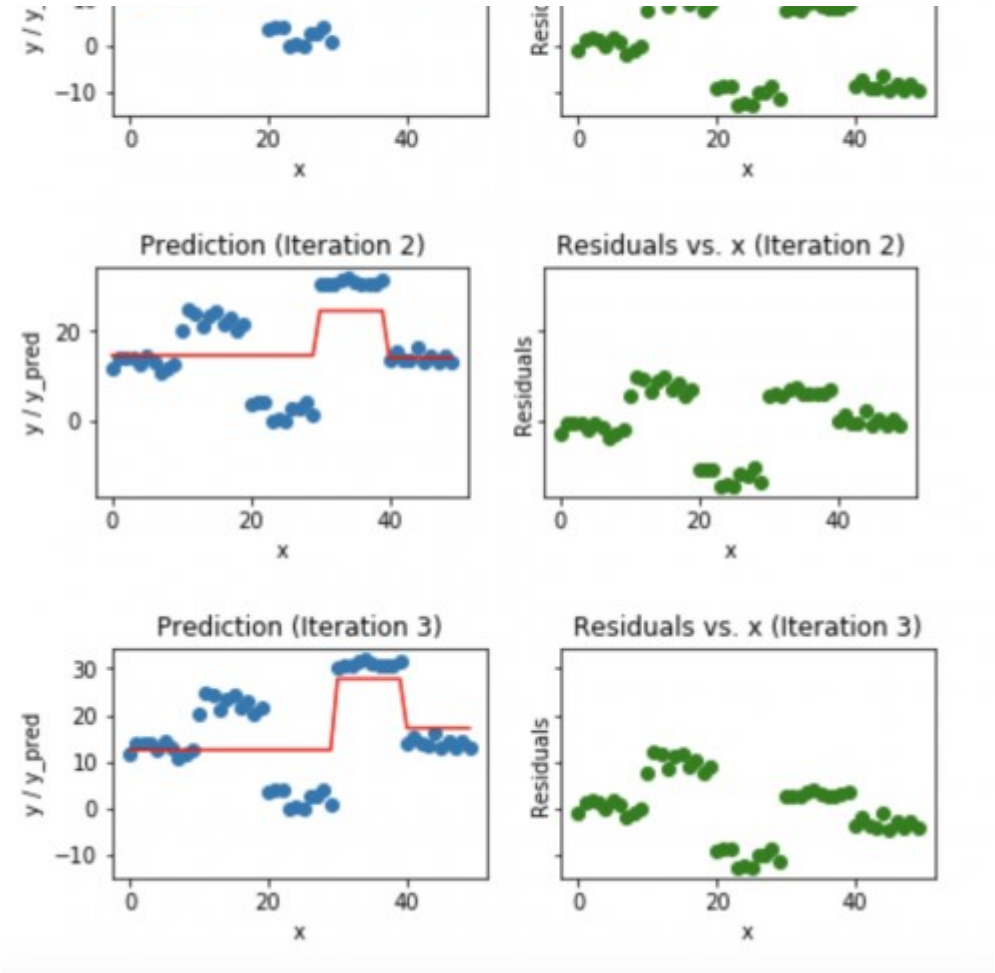
error_residual hosted with ❤ by GitHub view raw

Чтобы помочь понять базовые концепции, вот ссылка с полной реализацией простой модели градиентного бустинга с нуля.

Приведенный код — это неоптимизированная vanilla реализация повышения градиента. Большинство моделей повышения градиента, доступных в библиотеках, хорошо оптимизированы и имеют множество гиперпараметров.

Визуализация работы Gradient Boosting Tree:

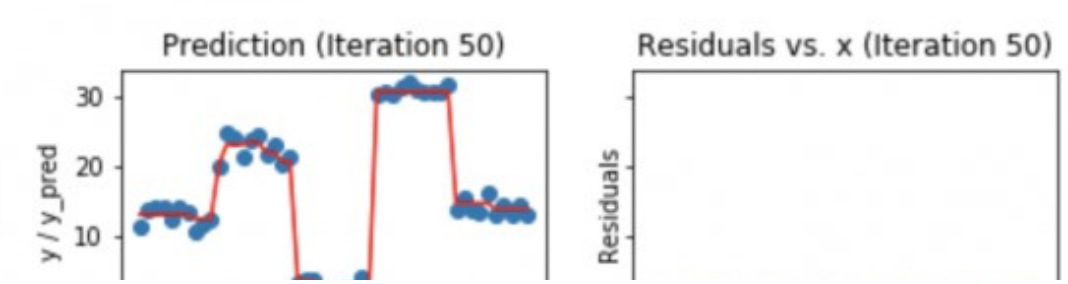
- Синие точки (слева) отображаются как вход (x) по сравнению с выходом (y);
- Красная линия (слева) показывает значения, предсказанные деревом решений;
- Зеленые точки (справа) показывают остатки по сравнению с вводом (x) для i-й итерации;
- Итерация представляет собой последовательное заполнения дерева Gradient Boosting.



Визуализация предсказаний (18-20 итерации)

Заметим, что после 20-й итерации отклонения распределены случайным образом (здесь не говорим о случайной норме) около 0, и наши прогнозы очень близки к истинным значениям (итерации называются n_estimators в реализации sklearn). Возможно, это хороший момент для остановки, или наша модель начнет переобучаться.

Посмотрим, как выглядит наша модель после 50-й итерации.



Визуализация градиентного бустинга после 50 итераций

Мы видим, что даже после 50-й итерации отклонения по сравнению с графиком x похожи на то, что мы видим на 20-й итерации. Но модель становится все более сложной, и предсказания перерабатывают данные обучения и пытаются изучить каждый учебный материал. Таким образом, было бы лучше остановиться на 20-й итерации.

Фрагмент кода Python, используемый для построения всех вышеперечисленных графиков.

```
# plotting after prediction
xa = np.array(x.x) # column name of x is x
order = np.argsort(xa)
xs = np.array(xa)[order]
ys = np.array(predf)[order]

#epreds = np.array(epred[:,None])[order]

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize = (13,2.5))

ax1.plot(x,y, 'o')
ax1.plot(xs, ys, 'r')
ax1.set_title(f'Prediction (Iteration {i+1})')
ax1.set_xlabel('x')
ax1.set_ylabel('y / y_pred')

ax2.plot(x, ei, 'go')
ax2.set_title(f'Residuals vs. x (Iteration {i+1})')
ax2.set_xlabel('x')
ax2.set_ylabel('Residuals')
```

visualization hosted with ❤ by GitHub view raw

Видео Александра Ихлера:

Автор: Станислав Литвинов

Источник: https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d