ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Машинное обучение

Лекция 9. Градиентный бустинг

Автор: Рустам Азимов

Санкт-Петербург, 2023г.

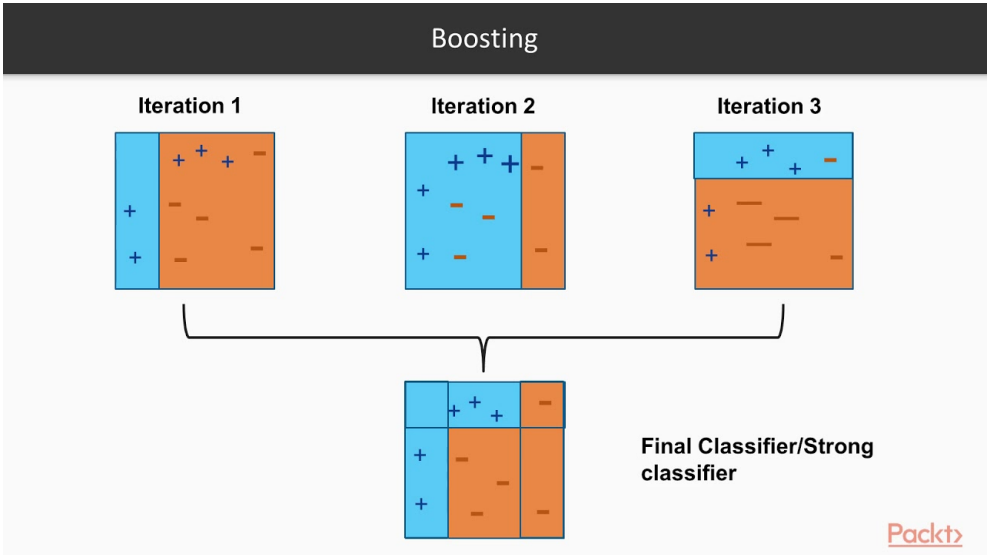
Введение

- Ранее мы рассмотрели бэггинг и случайные леса
- Мы строили композиции, которые независимо обучают каждый базовый алгоритм по некоторому подмножеству обучающих данных (подвыборок)
- Но можно было бы строить композицию так, чтобы каждая следующая модель исправляла ошибки предыдущих

AdaBoost

- Прародитель градиентного бустинга (1995 год)
- Жадный подход
- Строим линейную комбинацию слабых моделей, меняя веса у объектов выборки на каждой итерации
- Очередная модель (обычно решающее дерево) строится на ранее ошибочно предсказанных объектах, веса для которых увеличивают

AdaBoost


Boosting
Iteration 1
Iteration 2
Iteration 3
Final Classifier/Strong classifier
Packt›

Бустинг в задаче регрессии

- Теперь рассмотрим метод, который обобщает эту идею — **градиентный бустинг, GBM (Gradient Boosting Machine)**, предложенный Фридманом (1999 год), который работает для любых дифференцируемых функций потерь и является одним из наиболее мощных и универсальных на сегодняшний день
- Минимизируем квадратичный функционал:

$$\frac{1}{2}\sum_{i=1}^{l}(a(x_i) - y_i)^2 \to \min_a$$

- Наша композиция — это сумма базовых моделей $b_n(x)$ из некоторого семейства $\mathcal{A}$:

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

Бустинг в задаче регрессии

- Найдем первый базовый алгоритм, который является лучшим для нашей задачи из семейства $\mathcal{A}$

$$b_1(x) = arg \min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{l} (b(x_i) - y_i)^2$$

- Для многих семейств алгоритмов это сделать не сложно
- Если данное семейство не содержит безошибочную модель, то мы можем посчитать эти ошибки на каждом объекте:

$$s_i^{(1)} = y_i - b_1(x_i)$$

Бустинг в задаче регрессии

- Мы постарались приблизиться к истинным ответам с помощью алгоритма $b_1(x)$
- Но мы все-таки допускаем ошибки
- Так как наша композиция будет суммой базовых алгоритмов, то логично, что следующий алгоритм строим так, чтобы его ответы были как можно ближе к ошибкам первого алгоритма:

$$b_2(x) = arg \min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{l} (b(x_i) - s_i^{(1)})^2$$

Бустинг в задаче регрессии

- Каждый следующий алгоритм тоже будет пытаться исправить ошибки суммы всех предыдущих:

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

$$b_N(x) = arg \min_{b \in A} \frac{1}{2} \sum_{i=1}^{l} (b(x_i) - s_i^{(N)})^2$$

Бустинг в задаче регрессии

- Описанный метод можно просто реализовать, он показывает хорошие результаты и уже реализован во многих библиотеках, например, в scikit-learn
- Описанные остатки — это антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенной композиции
- Таким образом, выбирается каждый раз такой базовый алгоритм, который как можно сильнее уменьшит ошибку текущей композиции

Градиентный бустинг

- Пусть дана некоторая дифференцируемая функция потерь $L(y, z)$ и мы будем строить взвешенную сумму базовых алгоритмов:

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

- Как правило, коэффициент γ_0 берут равным единице, а базовый алгоритм b_0 выбирают очень простым:
  - нулевой $b_0(x) = 0$
  - в задачах классификации возвращающий самый популярный класс: $b_0(x) = arg\,\max_{y \in \mathbb{Y}} \sum_{i=1}^{l} [y_i = y]$
  - в задачах регрессии возвращающий средний ответ: $b_0(x) = \frac{1}{l} \sum_{i=1}^{l} y_i$

Градиентный бустинг

- Пусть, мы построили композицию $a_{N-1}(x)$ из $N-1$ алгоритма, и хотим выбрать следующий базовый алгоритм $b_N(x)$ так, чтобы как можно сильнее уменьшить ошибку:

$$\sum_{i=1}^{l} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{b_N, \gamma_N}$$

- Как уже говорилось, мы будет стараться исправить ошибки предыдущей композиции, то есть $\gamma_N b_N(x_i) = s_i$, где сдвиг s_i будет противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$:

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

- Мы сдвинемся в сторону скорейшего убывания функции потерь и вектор сдвигов $s = (s_1, \ldots, s_l)$ совпадает с антиградиентом

Градиентный бустинг

- При таком выборе сдвигов s_i мы сделаем один шаг градиентного спуска, двигаясь в сторону наискорейшего убывания ошибки на обучающей выборке
- Это будет градиентным спуском в l-мерном пространстве предсказаний алгоритма на объектах обучающей выборки
- Чтобы по этим s_i построить нужную функцию мы воспользуемся простой среднеквадратичной функцией потерь:

$$b_N(x) = arg \min_{b \in \mathcal{A}} \sum_{i=1}^{l} (b(x_i) - s_i)^2$$

- Тут мы уже не используем первоначальную функцию потерь L, информация о которой уже хранится в антиградиенте s_i, а на данном шаге лишь решается задача аппроксимации функции по l точкам
- Среднеквадратичной ошибки, как правило, оказывается достаточно

Градиентный бустинг

- Мы нашли базовый алгоритм b_N и теперь по аналогии можно подобрать коэффициент при нем:

$$\gamma_N(x) = arg \min_{\gamma \in \mathbb{R}} \sum_{i=1}^{l} L(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

- Описанный подход с аппроксимацией антиградиента базовыми алгоритмами и называется градиентным бустингом

Регуляризация

- На практике, градиентный бустинг очень быстро строит композицию, ошибка которой на обучении выходит на асимптоту, после чего начинает настраиваться на шум и переобучаться
- Этому может быть две причины:
  - Базовые алгоритмы очень простые (решающие деревья небольшой глубины), тогда они плохо приближают вектор антиградиента. Базовые алгоритмы будут соответствовать шагу по направлению сильно отличающегося от направления наискорейшего убывания и градиентный бустинг может свестись к случайному блужданию в пространстве
  - Базовые алгоритмы сложные (глубокие решающие деревья), тогда они способны за несколько шагов бустинга идеально подогнаться под обучающую выборку и получим переобучение, связанное с излишней сложностью семейства алгоритмов

Регуляризация: сокращение шага

- Одним хорошим способом решения данной проблемы является сокращение шага
- Вместо перехода в оптимальную точку в направлении антиградиента делается укороченный шаг

$$a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x)$$

- Здесь $\eta \in (0, 1]$ — это темп обучения
- Обычно, чем меньше темп обучения, тем лучше качество итоговой композиции
- Мы, по сути, понижаем доверие к направлению, восстановленному базовым алгоритмом

Число итераций

- Необходимо следить за числом итераций градиентного бустинга
- Ошибка на обучающей выборке монотонно стремится к нулю
- А вот ошибка на тестовой выборке, как правило, начинает увеличиваться после определенной итерации
- Выбрать оптимальное число итераций можно, например, по отложенной выборке или с помощью кросс-валидации

Стохастический градиентный бустинг

- Еще одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов
- Например, алгоритм b_N обучается не по всей выборке X, а лишь по ее какому-то случайному подмножеству $X_k \subset X$
- Так мы можем понизить уровень шума в обучении и повысить эффективность вычислений
- Существует рекомендация брать подвыборки X_k, размер которых вдвое меньше исходной выборки X

Функции потерь: регрессия

- При вещественном целевом векторе, как правило, используют квадратичную функцию потерь, которую мы уже рассматривали
- Другой вариант — модуль отклонения $L(y, z) = |y - z|$
- Антиградиент тогда вычисляется по формуле

$$s_i^{(N)} = -sign(a_{N-1}(x_i) - y_i)$$

Функции потерь: классификация

- В задаче бинарной классификации разумным выбором является логистическая функция потерь, с которой мы уже сталкивались при изучении линейных методов

$$L(y, z) = \log(1 + \exp(-yz))$$

- Тогда задача поиска базового алгоритма с ней принимает вид

$$b_N(x) = arg \min_{b \in \mathcal{A}} \sum_{i=1}^{l} (b(x_i) - \frac{y_i}{1 + \exp(y_i a_{N-1}(x_i))})^2$$

Градиентный бустинг над деревьями

- Считается, что градиентный бустинг над решающими деревьями — один из самых универсальных и сильных методов машинного обучения, известных на сегодняшний день
- В частности, на градиентном бустинге над деревьями основан алгоритм ранжирования компании Яндекс — MatrixNet

Градиентный бустинг над деревьями

- Как мы помним, решающее дерево разбивает все пространство на непересекающиеся области, в каждой из которых его ответ равен константе

$$b_n(x) = \sum_{j=1}^{J_n} b_{n_j}[x \in R_j]$$

- Здесь $j = 1, \ldots, J_n$ — индексы листьев, R_j — соответствующие области разбиения, b_{n_j} — значения в листьях
- Тогда, на N-й итерации бустинга композиция обновляется как

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{N_j}[x \in R_j]$$

Градиентный бустинг над деревьями

- Видно, что добавление в композицию одного дерева с J_N листьями равносильно добавлению J_N базовых алгоритмов, представляющих собой предикаты
- Мы можем улучшить качество композиции, подобрав свой коэффициент при каждом из предикатов:

$$\sum_{i=1}^{l} L(y_i, a_{N-1}(x_i)) + \sum_{j=1}^{J_N} \gamma_{N_j} b_{N_j}[x_i \in R_j]) \to \min_{\{\gamma_{N_j}\}_{j=1}^{J_N}}$$

Градиентный бустинг над деревьями

- Поскольку области разбиения R_j не пересекаются, данная задача распадается на J_N независимых подзадач:

$$\gamma_{N_j} = arg \min_{\gamma} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma)$$

- В некоторых случаях оптимальные коэффициенты могут быть найдены аналитически — например, для квадратичной и абсолютной ошибки
- В остальных случаях коэффициенты можно найти с помощью итерационных методов, например, для логистической функции потерь

Источники

- http://www.machinelearning.ru/
- https://scikit-learn.org