Лекция 6

Деревья решений и композиции

Машинное обучение

Сергей Муравьёв / Андрей Фильченков

09.10.2020

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

- В презентации используются материалы курса «Машинное обучение» К.В. Воронцова
- Слайды доступны: **shorturl.at/hjyAX**
- Видео доступны: **shorturl.at/ltVZ3**

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Закономерность и правило

Предикат на множестве X:

$$\varphi: X \to \{0,1\}.$$

Предикат **покрывает** объект x, если $\varphi(x) = 1$.

Закономерность — предикат, покрывающий подавляющее большинство объектов одного класса и малое число объектов других классов.

Правило — легко *интерпретируемая* и *высокоинформативная* закономерность, описываемая простыми логическими формулами.

Интерпретируемые закономерности

Истоки открытия знаний в базах данных.

Закономерность φ называется **интерпретируемой**, если

1) сформулирована естественным языком;
2) зависит от небольшого числа параметров (1–7).

Пример (из русского языка): *Если слово является наречием и оканчивается на шипящую букву ("ж", "ч" или "ш"), в конце ставится "ь".*

Примеры: "вска**чь**", "насте**жь**".

Исключения: "уж", "замуж", "невтерпеж".

Концепция информативности

Закономерность φ является **информативной** для класса c, если

$$p(\varphi) = |\{x_i | \varphi(x_i) = 1, y_i = c\}| \to \max;$$
$$n(\varphi) = |\{x_i | \varphi(x_i) = 1, y_i \neq c\}| \to \min.$$

$p(\varphi)$ – истинно положительные;

$n(\varphi)$ – ложно положительные;

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Примеры построения правил(1/2)

Правило — это интерпретируемая, высокоинформативная закономерность или «одноклассовый» классификатор с отказами.

Примеры построения:

1. **Конъюнкция пороговых условий**:

$$R(x) = \bigwedge_{j \in J} [a_j \le f_j(x) \le b_j].$$

Примеры построения правил(2/2)

2. **Синдром** – выполнение не менее d условий из J:

$$R(x) = \left[\sum_{j \in J} \left[a_j \le f_j(x) \le b_j\right] \ge d\right],$$

(когда $d = |J|$ — это конъюнкция, когда $d = 1$ – дизъюнкция).

3. **Полуплоскость:**

$$R(x) = \left[\sum_{j \in J} w_j f_j(x) \ge w_0\right].$$

4. **Шар** – пороговая функция близости:

$$R(x) = [r(x, x_0) \le r_0].$$

Где брать правила

Правила можно:

- создавать самостоятельно;
- заимствовать у экспертов;
- обучать.

Правила можно обучать при помощи

- алгоритмов оптимизации;
- эвристических методов;
- специальных алгоритмов машинного обучения.

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Деревья решений

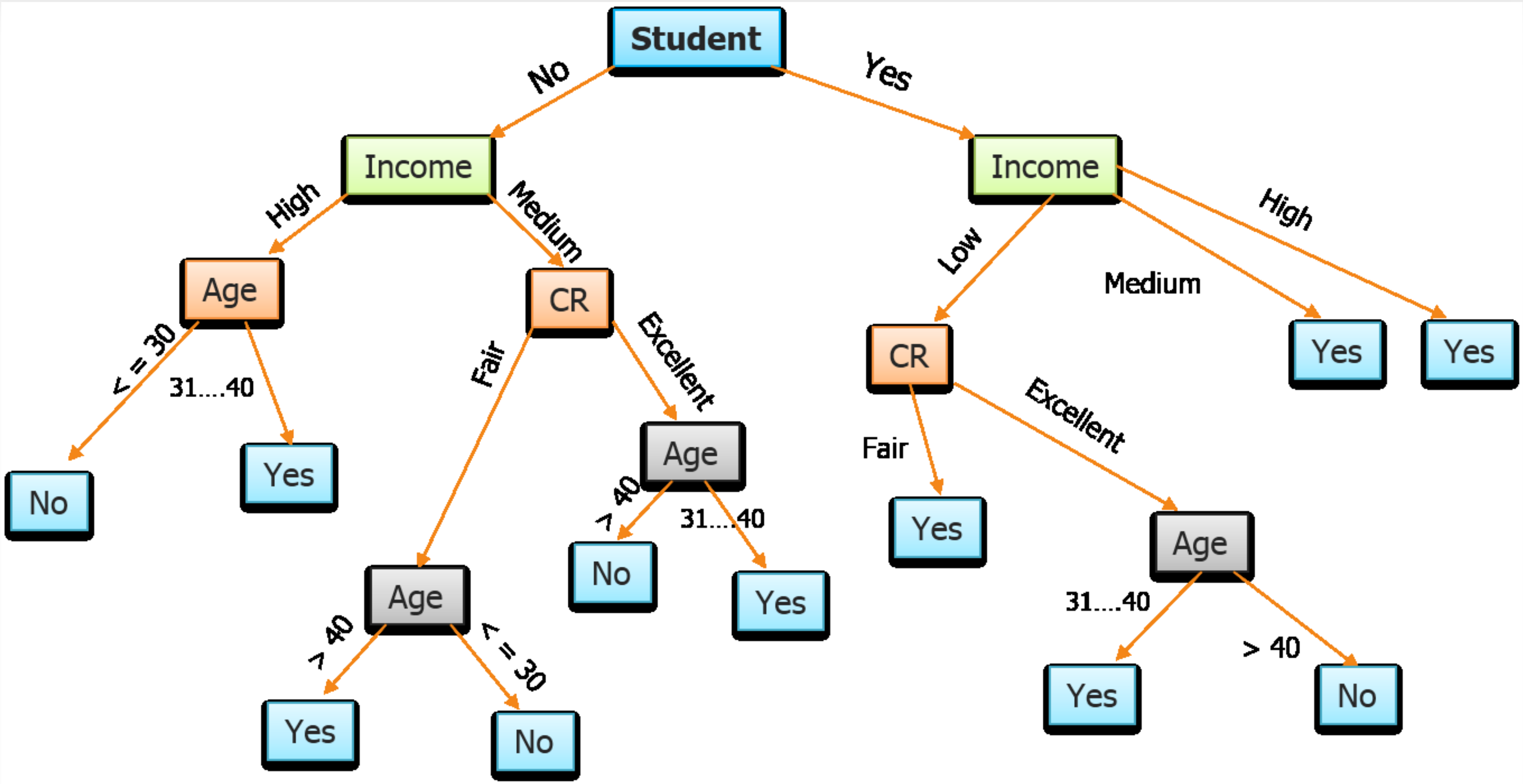
Дерево решений — алгоритм классификации и регрессии.

Вершины содержат разделяющие правила (вопросы).

Ребро — возможный ответ на вопрос в родительской вершине.

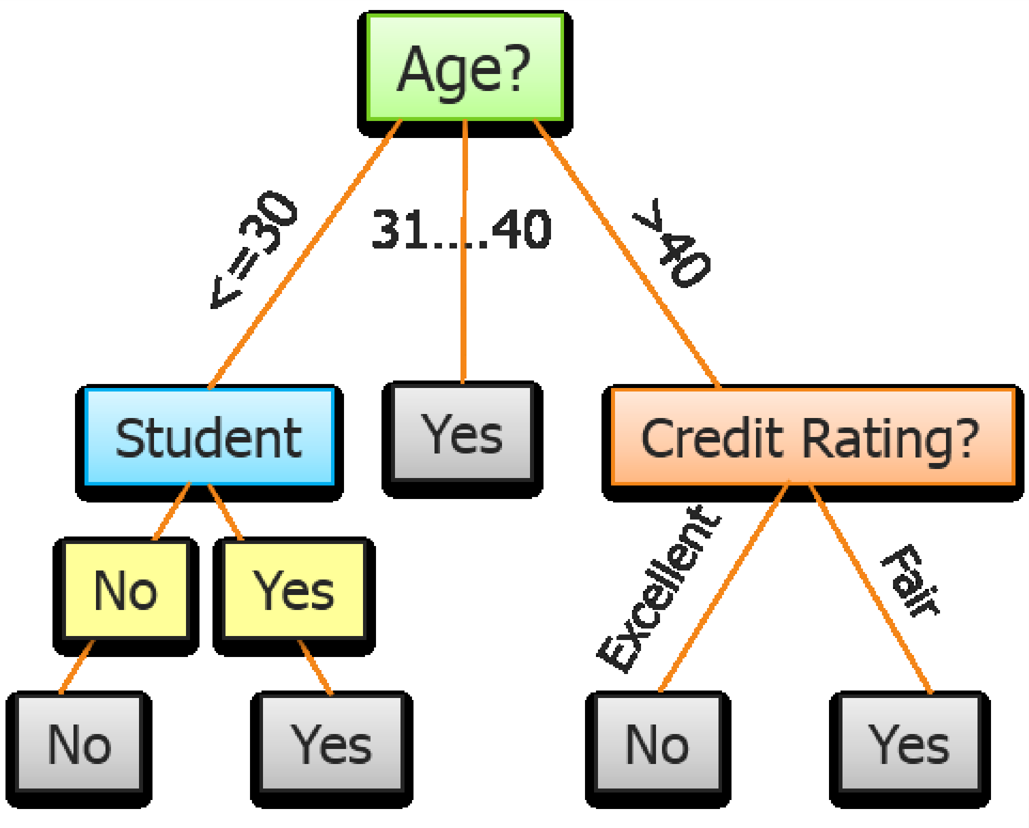
Листья содержат решения (класс объекта для задачи классификации и число для задачи регрессии).

Пример дерева решений (1/2)



Пример дерева решений (2/2)

Такой же классификатор может быть построен проще



Общая схема обучения

Множество разделяющих правил $\mathcal{B}$ и критерий ветвления Φ.

1. Выборка S в корне.

2. На каждом шаге рекурсивно обрабатываем S.

2.1. Если S содержит объекты только одного класса c, создаём лист с классом c и останавливаемся.

2.2. В противном случае выбираем правило $\mathscr{b} \in \mathcal{B}$, наилучшее с точки зрения Φ, и разделяем выборку на $S_1, \dots, S_k$.

2.3. Если выполнился критерий остановки, тогда возвращаем наиболее популярный класс в текущей выборке S, в противном случае создаём k детей вершины, соответствующих S_i, $i = [1, \dots, k]$.

3. Подрезаем итоговое дерево.

Ключевые вопросы

Как выбирать разделяющие правила?
Как выбрать критерий ветвления?
Как выбрать критерий остановки?
Как подрезать дерево?

Выбор семейства разделяющих правил

В общем случае является семейством классификаторов.

- В большинстве случаев выбираются правила для одного признака:

$$f_i(x) > m_i;$$
$$f_i(x) = d_i.$$

- Можно создавать комбинацию таких правил.

Выбор признаковых правил

Существует $|\mathcal{D}| - 1$ способов разбиения выборки.

- Рассмотреть каждое правило и выбрать наиболее информативные.
- Объединение диапазонов значений.
- Пропуск маленьких диапазонов.

Данным способом можно синтезировать правило для каждого признака.

Разбиение выборки

- Если выборка каждый раз разделяется на две части ($k = 2$), то $\mathcal{B}$ — семейство бинарных правил, а дерево бинарное.
- Если признак категориальный, можно строить несколько рёбер из вершины.
- Если признак численный, то применяется бинаризация / дискретизация.

На каждом шаге построения алгоритма, количество рёбер может быть различным, однако обычно k — фиксированное.

Выбор критерия ветвления

Критерий ветвления Ф задаётся следующей формулой:

$$\Phi(S) = \phi(S) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} \phi(S_i).$$

Наиболее популярные функционалы ϕ :

- $\phi_h(S)$ – энтропия, $\Phi_h(S)$ называется **IGain**;
- $\phi_g(S) = 1 - \sum_{i=1}^{m} p_i^2$, где p_i – вероятность (частота) присутствия элементов i-го класса в выборке S – **индекс Джини**. $\Phi_g(S)$ – **GiniGain**.

Используются также и другие критерии

GainRatio = IGain(S) / Enthropy(S).

Показатель качества разделения обычно не влияет на эффективность дерева.

Критерии остановки

Популярные критерии остановки:

- один из классов пустой после разбиения
- $\Phi(S)$ ниже определённого порога
- $|S|$ ниже определённого порога
- высота дерева выше определённого порога

Предподрезка

Другой тип критерия остановки.

Останавливаем рост дерева, когда нет статистически значимой связи между каким-либо признаком и классом в конкретном узле

Обычно, используется тест χ^2

Подрезка

Предпосылки: только верхние вершины влияют на качество алгоритма; Деревья решений склонны переобучаться.

Основная идея: отрезать нижние ветви.

Обрезка – это обработка созданных деревьев, когда последовательно применяются операторы упрощения, если они улучшают качество по определенному критерию (например, уменьшение количества ошибок).

Схема работы алгоритма подрезки

Разбить выборку на тренировочную и тестовую в отношении 2:1.

Для каждой вершины применяем определённый оператор упрощения, который является лучшим с точки зрения выбранного критерия качества:

1) Ничего не менять;
2) Заменить вершину «ребёнком» (для каждого «ребёнка»);
3) Заменить вершину листом (для каждого класса).

Примеры

ID3 (Quinlan, 1986):

IGain; $\Phi(S) < 0$; без подрезки.

C4.5 (Quinlan, 1993):

GainRatio; с подрезкой.

CART (Breinman, 1984):

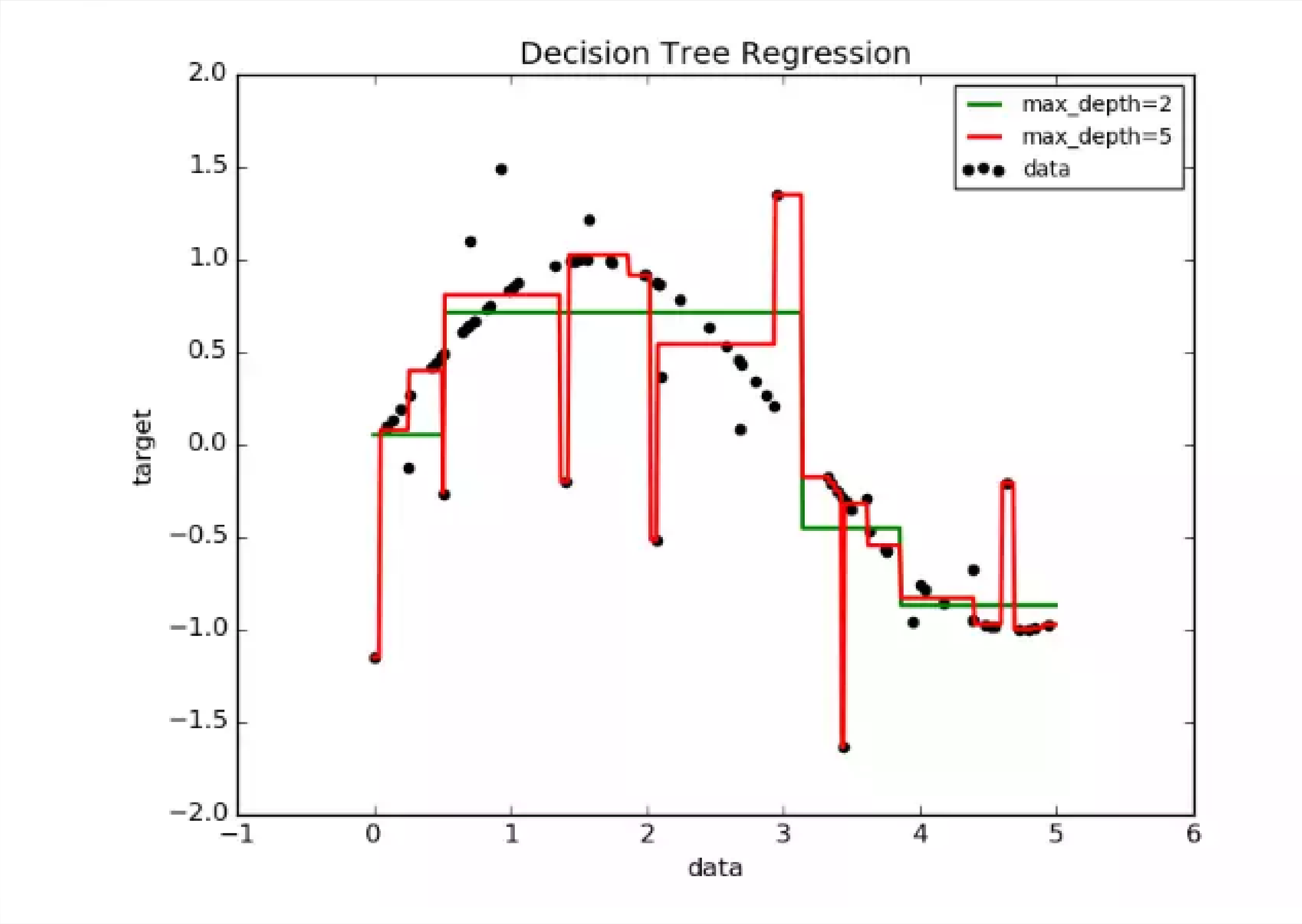
бинарное; GiniGain; подрезка. Предназначено для регрессии (значения в листьях).

Регрессионные деревья

Регрессионные деревья используются для решения задачи регрессии.

Они работают так же как и обычные деревья решений, но в листе возвращается среднее значение целевого признака вместо мажоритарного класса.

Пример регрессионных деревьев



Деревья моделей

Деревья моделей решают задачу регрессии.

Они работают так же, как деревья решений и регрессии, но каждый лист возвращает некоторые гиперпараметры модели.

Обычно используется для линейных моделей.

Анализ алгоритма

Преимущества:

- легко понять и интерпретировать;
- быстро обучается;
- может работать с разными типами данных;
- выполняет выбор признаков внутри себя.

Недостатки:

- чувствителен к шумам;
- быстро переобучается.

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Слабая и сильная обучаемость

Слабая обучаемость означает, что за полиномиальное время можно найти алгоритм, производительность которого будет больше 0,5.

Сильная обучаемость означает, что за полиномиальное время можно найти алгоритм, производительность которого будет сколь угодно высокой.

Теорема (Шапиро, 1990)
Сильная обучаемость эквивалентна слабой обучаемости, потому что любую модель можно усилить с помощью композиции алгоритмов.

Пример

Дано n алгоритмов с вероятностями корректной классификации $p_1, p_2, \dots, p_n \approx p$. Вероятности являются независимыми.

Новый алгоритм выберет метку класса исходя из наиболее предпочтительного выбранного класса в этих алгоритмах.

Тогда вероятность корректной классификации:

$$P_{vote} = p^n + np^{n-1}(1-p) + \frac{n(n-1)}{2}p^{n-2}(1-p)^2 + \dots +$$

$$+C_n^{n/2}p^{n/2}(1-p)^{n/2}.$$

Формулировка задачи

Множество объектов X, множество ответов Y.

Обучающая выборка $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{|\mathcal{D}|}$ с известными ответами.

Семейство **базовых алгоритмов**

$$H = \{h(x, a): X \to R | a \in A\},$$

a — вектор параметров, который описывает алгоритм, R является доменом (обычно $\mathbb{R}$ или $\mathbb{R}^M$), $h(x, a)$ зачастую вероятностная оценка.

Задача: найти (синтезировать) алгоритм, который является наиболее точным в прогнозировании ответа для объекта из X.

Композиция алгоритмов

Композиция из T базовых алгоритмов $h_1, \dots, h_T : X \to R$ выражается как:

$$H_T(x) = C(F(h_1(x), \dots, h_T(x))),$$

где $C : R \to Y$ — **решающее правило**, $F : R^T \to R$ — **корректирующая функция**.

R обычно больше Y.

Решающее правило

Решающее правило: $C\big(H(x)\big) \to Y$:

- для регрессии, $Y = \mathbb{R}$

$C(H(x)) = H(x)$, либо с какой-нибудь трансформацией.

- для классификации на k классах, $Y = \{1, \dots, k\}$

$$C(F(h_1(x), \dots, h_T(x))) = \operatorname{argmax}_{y \in Y} h_y(x)$$

- для бинарной классификации, $Y = \{-1, +1\}$

$$C(H(x)) = \operatorname{sign}(H(x))$$

Голосование

Самым простым примером корректирующей функции является **голосование**.

Два типа голосования:

- мажоритарное голосование (подсчёт «голосов»)
- мягкое голосование (подсчёт вероятностей)

Также можно добавлять веса для участников голосования ($\sum_{i=1..K} \alpha_i = 1$).

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Формулировка задачи бустинга

Будем синтезировать алгоритм, описываемый формулой

$$H_T(x) = \sum_{t=1}^{T} b_t h(x, a_t),$$

где $b_t \in \mathbb{R}$ – коэффициенты, минимизирующие эмпирический риск

$$\mathcal{L}(H_T, \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}(H_T(x_i), y_i) \to \min$$

для функции потерь $\mathcal{L}(H_T(x_i), y_i)$.

Градиентный спуск

Точное решение $\{(a_t, b_t)\}_{t=1}^{T}$ найти довольно проблематично.

Будем наращивать нашу композицию шаг за шагом

$$H_t(x) = H_{t-1}(x) + b_t h(x, a_t)$$

Для этого будем инкрементально оценивать градиент функции ошибки $\mathcal{L}^{(t)} = \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}(H_t(x_i), y_i)$.

Значение функции $\mathcal{L}^{(t)}$ — вектор ошибок на каждом объекте длины $|\mathcal{D}|$:

$$\mathcal{L}^{(t)} = \left(\mathcal{L}_1^{(t)}, \dots, \mathcal{L}_{|\mathcal{D}|}^{(t)}\right).$$

Градиент

Градиент (для i-й компоненты $\mathcal{L}^{(t-1)}$):

$$\nabla\mathcal{L}_i^{(t-1)} = \frac{\delta\mathcal{L}_i^{(t-1)}}{\delta H_{t-1}}(x_i) = \frac{\delta\left(\sum_i^{|\mathcal{D}|}\mathcal{L}(H_{t-1}, y_i)\right)}{\delta H_{t-1}}(x_i) =$$
$$= \frac{\delta\mathcal{L}(H_{t-1}, y_i)}{\delta H_{t-1}}(x_i).$$

Таким образом, мы получаем

$$H_t(x) = H_{t-1}(x) - b_t \nabla\mathcal{L}^{(t-1)}.$$

Выбор параметров

$$H_t(x) = H_{t-1}(x) - b_t \nabla\mathcal{L}^{(t-1)}$$

$$b_t = \operatorname{argmin}_b \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}\left(H_{t-1}(x_i) - b\nabla\mathcal{L}^{(t-1)}, y_i\right).$$

Вектор $\nabla\mathcal{L}^{(t-1)}$ не является базовым алгоритмом, поэтому

$$a_t = \operatorname{argmin}_{a\in A} \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}(h(x_i, a), \nabla\mathcal{L}^{(t-1)}) \equiv$$

$$\equiv \textsc{Learn}\left(\{x_i\}_{i=1}^{|\mathcal{D}|}, \left\{\nabla\mathcal{L}_i^{(t-1)}\right\}_{i=1}^{|\mathcal{D}|}\right).$$

Мы можем найти его, используя линейный поиск

$$b_t = \operatorname{argmin}_b \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}(H_{t-1}(x_i) - bh(x_i, a_t), y_i).$$

Обобщённый алгоритм

Input: $\mathcal{D}, T$

$H_0(x) = \textsc{Learn}\left(\{x_i\}_{i=1}^{|\mathcal{D}|}, \{y_i\}_{i=1}^{|\mathcal{D}|}\right)$

1. **for** $t = 1$ **to** T **do**
2. $\nabla \mathcal{L}^{(t-1)} = \left[\frac{\delta \mathcal{L}(H_{t-1}, y_i)}{\delta H_{t-1}}(x_i)\right]_{i=1}^{|\mathcal{D}|}$
3. $a_t = \textsc{Learn}\left(\{x_i\}_{i=1}^{|\mathcal{D}|}, \left\{\nabla \mathcal{L}_i^{(t)}\right\}_{i=1}^{|\mathcal{D}|}\right)$
4. $b_t = \operatorname{argmin}_b \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}(y_i, h_{t-1}(x_i) - b h(x_i, a_t))$
5. $H_t(x) = H_{t-1}(x) + b_t h(x, a_t)$
6. **return** H_T

Гладкость $\mathcal{L}$

Обычно $\mathcal{L}$ кусочно-линейная:

$$\mathcal{L} = \sum_{i=1}^{|\mathcal{D}|} M = \sum_{i=1}^{|\mathcal{D}|} \left[y_i \sum_{t=1}^{N} \alpha_t H_t(x_i) < 0 \right]$$

Гладкая аппроксимация пороговой функции $[M \leq 0]$:

- $E(M) = \exp(-M)$ в AdaBoost
- $L(M) = \log_2(1 + e - M)$ в LogitBoost
- $Q(M) = (1 - M)^2$ в GentleBoost
- $G(M) = \exp\big(-cM(M + s)\big)$ в BrownBoost

Известные алгоритмы

- AdaBoost
- LogitBoost
- BrownBoost
- AnyBoost
- Стохастический градиентный бустинг

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Основы AdaBoost

$$H_T(x) = \sum_{t=1}^{T} b_t h(x, a_t),$$

Алгоритм классификации, следовательно $\mathcal{L}(H(x), y) = \mathcal{L}(H(x)y)$.

Функция потерь $E(M) = \exp(-M)$

Понятие «**весов**» появилось ранее понятия «**градиента**».

Для вектора весов значимости объектов $U^{|\mathcal{D}|}$:

- $P(h, U^{|\mathcal{D}|})$ количество правильно классифицируемых объектов (TP+TN)
- $N(h, U^{|\mathcal{D}|})$ количество неправильно классифицируемых объектов (FP+FN)

Основная теорема бустинга

Теорема (Фронд, Шапиро, 1995)

Если для любого нормализованного вектора весов $U^{|\mathcal{D}|}$, существует такой алгоритм $H = h(x, a)$ такой, что он классифицирует выборку сколько-нибудь лучше чем случайно:

$$N = N\left(H, U^{|\mathcal{D}|}\right) < 1/2.$$

тогда минимум $Q^{(t)}$ достигается с

$$H_t = \operatorname{argmin}_H N\left(H, U^{|\mathcal{D}|}\right),$$

$$b_t = \frac{1}{2} \ln \frac{1 - N(H_t, U^{|\mathcal{D}|})}{N(H_t, U^{|\mathcal{D}|})}.$$

Веса объектов

Для $\mathcal{L}(H(x), y) = \mathcal{L}(H(x)y)$.

$$\nabla\mathcal{L}_i^{(t)} = \frac{\delta\mathcal{L}(H_{t-1}y_i)}{\delta H_{t-1}}(x_i) = y_i \frac{\delta\mathcal{L}(H_{t-1}y_i)}{\delta(H_{t-1}y_i)}(x_i) = y_i w_i,$$

где $w_i = \frac{\delta\mathcal{L}(H_{t-1}y_i)}{\delta(H_{t-1}y_i)}(x_i)$ $-$ **вес объекта** x_i.

Тогда четвёртый шаг алгоритма $a_t = \textsc{Learn}\left(\{x_i\}_{i=1}^{|\mathcal{D}|}, \left\{\nabla\mathcal{L}_i^{(t)}\right\}_{i=1}^{|\mathcal{D}|}\right)$:

$$h(x, a_t) = \operatorname{argmin}_{a\in A} \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}\left(h(x_i, a_t), \nabla\mathcal{L}_i^{(t)}\right) =$$

$$= \operatorname{argmin}_{a\in A} \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}(y_i w_i h(x_i, a_t)).$$

AdaBoost

Input: $\mathcal{D}, T$

1. **for** $i = 1$ **to** $|\mathcal{D}|$ **do**
2. $\quad w_i = \frac{1}{|\mathcal{D}|}$
3. **for** $t = 1$ **to** T **do**
4. $\quad a_t = \operatorname{argmin}_A N\big(h(x, a_t), U^{|\mathcal{D}|}\big)$
5. $\quad N_t = \sum_{i=1}^{|\mathcal{D}|} w_i [y_i h(x_i, a_t) < 0]$
6. $\quad b_t = \frac{1}{2} \ln \frac{1 - N_t}{N_t}$
7. $\quad$ **for** $i = 1$ **to** $|\mathcal{D}|$ **do**
8. $\quad\quad w_i = w_i \exp(-b_t y_i h(x_i, a_t))$
9. $\quad \textsc{Normalize}(\{w_i\}_{i=1}^{|\mathcal{D}|})$
10. **return** $H_N = \sum_{t=1}^{T} b_t h(x, a_t)$

Анализ бустинга

Преимущества:

- очень тяжело переобучить
- можно применять для различных функций сложности

Недостатки:

- нет обработки шумов
- не может применяться для мощного алгоритма
- трудно интерпретировать

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Эмпирические наблюдения

1. Веса алгоритмов не очень важны для достижения хорошего качества.

2. Вес предметов не очень важен для достижения разницы.

Ключевая идея

Идея: мы можем строить различные алгоритмы, обучая модели при разных условиях

Точная идея: мы можем обучать модели на различных «кусках данных».

Синтез случайных алгоритмов

- **Subsampling:** обучать алгоритм на подмножестве объектов.
- **Bagging:** обучать алгоритм на подмножествах одинакового размера с «бутстрапом» (случайный выбор с повторами)
- **Метод случайного подпространства:** обучать алгоритм на разных подпространствах заданного признакового описания
- **Фильтрация** (далее)

Фильтрация

Пусть у нас есть набор бесконечного размера.

Обучим первый алгоритм на X_1, который содержит первые m_1 элементов.

Бросаем монетку m_2 раз:

- орёл: добавляем в X_2 первый некорректно классифицированный элемент; ф
- решка: добавляем в X_2 первый корректно классифицированный элемент.

Обучаем второй алгоритм на X_2.

Добавляем в X_3 первые m_3 объектов, на которых первые два классификатора дадут разные результаты.

Обучаем третий алгоритм на X_3.

План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Случайный лес

Для набора $\mathcal{D} = \{x_i, y_i\}_{i=1}^{|\mathcal{D}|}$ с n признаками.

1. Выбираем подмножество размера $|\mathcal{D}|$ с повторениями.
2. Синтезируем дерево решений таким образом, что для каждой вершины выбирается n'(обычно $n' = \sqrt{n}$)случайных признаков.
3. Подрезка не применяется.

Повторяем 100, 1000, … раз.

Почему это работает?

- Это голосование
- Деревья легко переобучаются на разных подвыборках.
- По мере роста выборки его показатели качества сходятся

Прочие детали

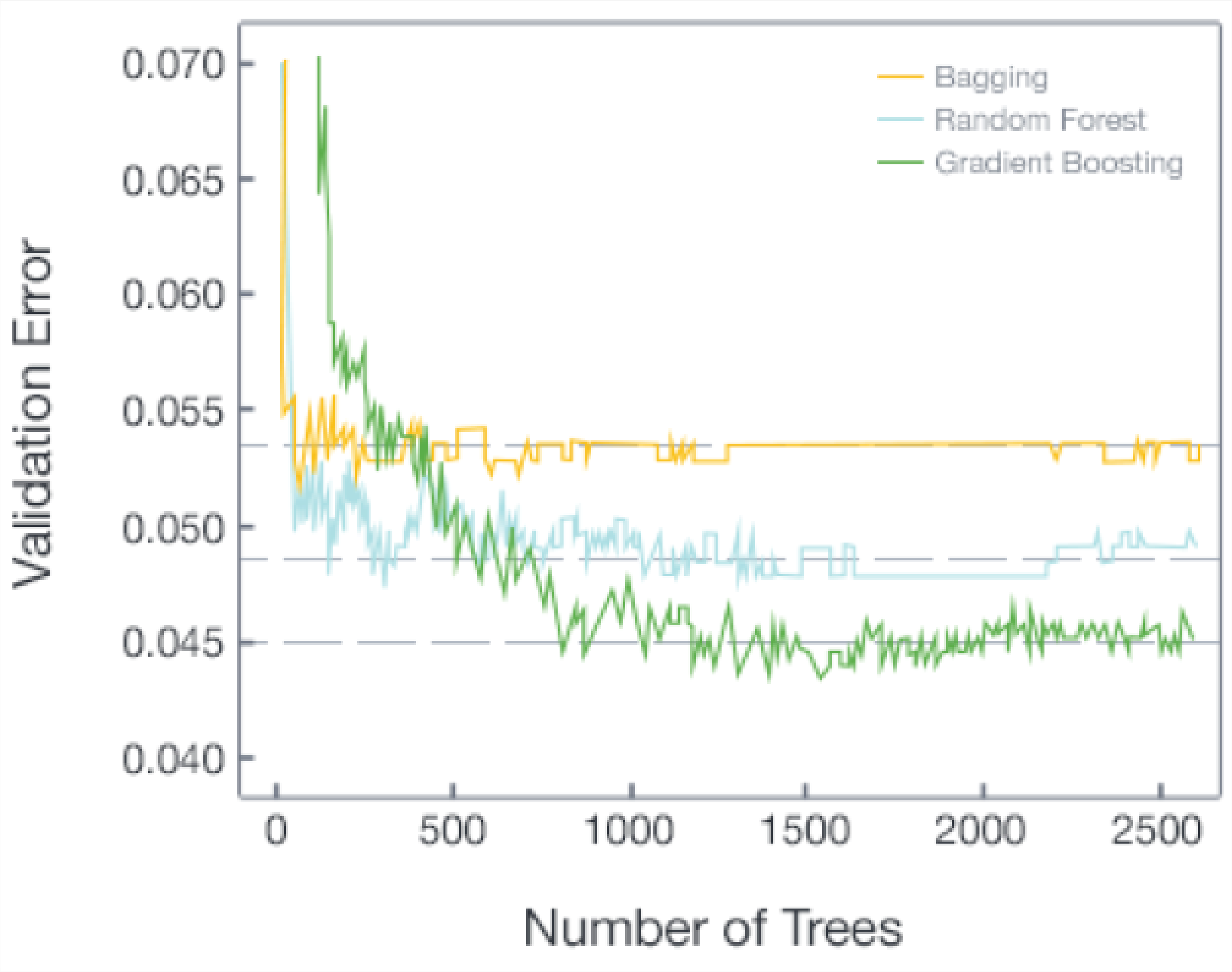
Как агрегировать?

- Мажоритарные голоса деревьев
- Вероятности (частота возникновения объекта класса в результирующем листе)

Как можно улучшить?

- **Экстремально рандомизированные деревья**: использовать случайные разделяющие правила (они быстрее).

Сравнение ансамблевых методов



План лекции

- Логические правила
- Индукция правил
- Деревья решений
- Композиция алгоритмов
- Бустинг
- AdaBoost
- Синтез случайных алгоритмов
- Случайный лес
- Стэкинг

Ключевая идея стэкинга

Вместо того, чтобы комбинировать алгоритмы, используйте их прогнозы как новые функции и изучите модель.

Эту идею можно обобщить на использование результатов классификации как новых характеристик объектов.

Что ещё может использоваться?

- Многоуровневый стэкинг
- Мета-обучение
- Комбинирование различных техник ансамблей