Решающие деревья

Ансамбли, случайный лес, градиентный бустинг

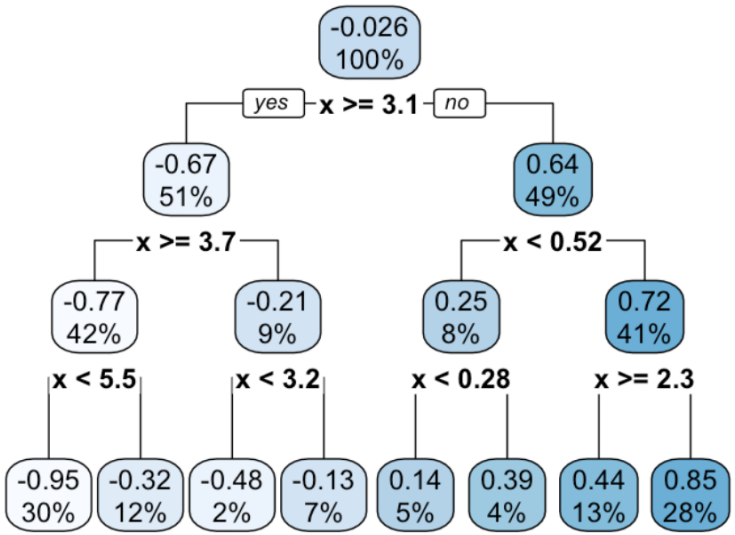
16/03/2020

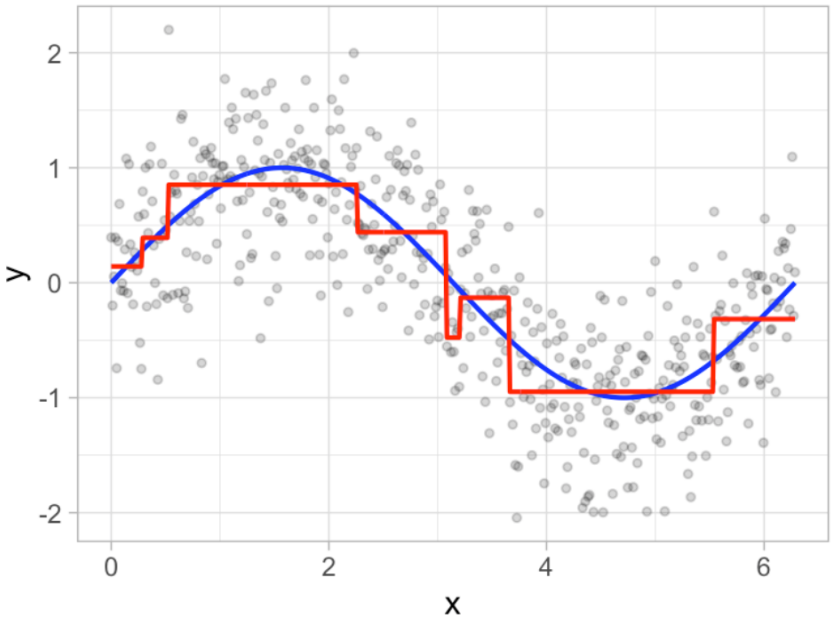
Саратовский Государственный Университет

Table of contents

Решающие деревья

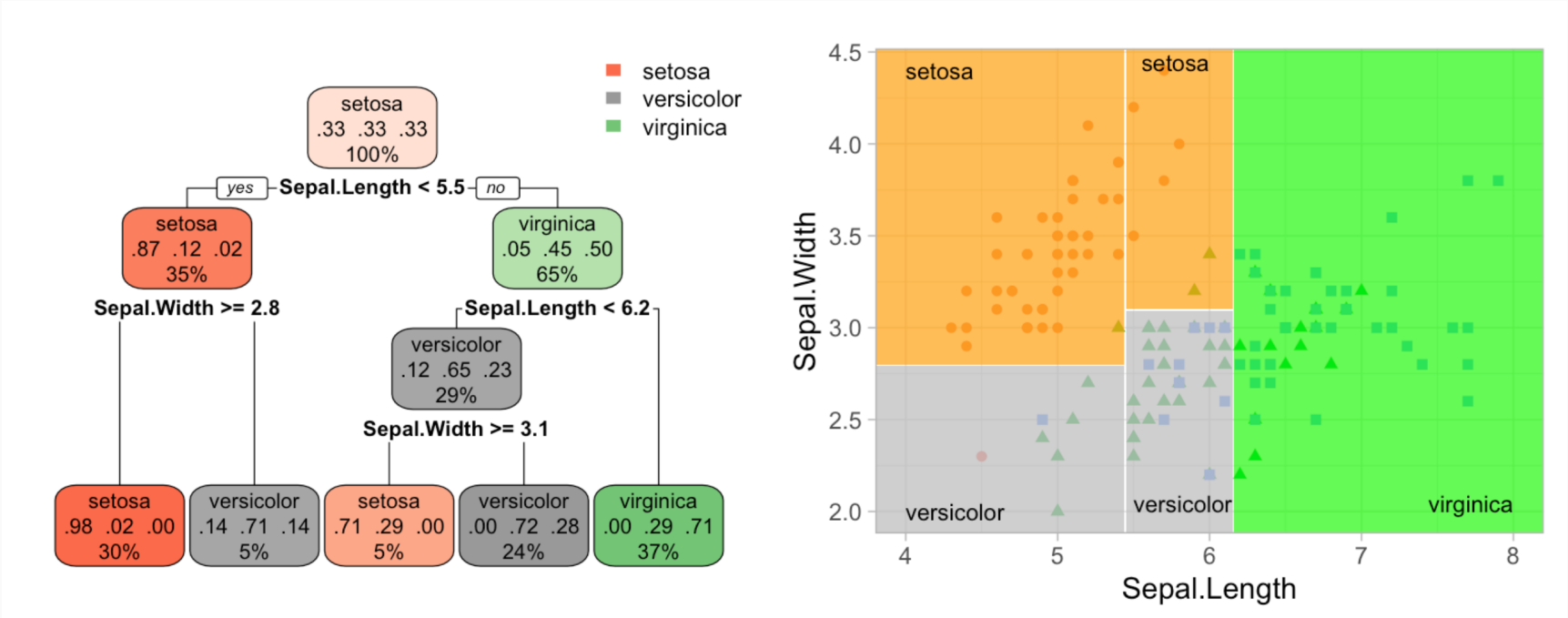
Решающее дерево для задачи регрессии





Корневой узел $\rightarrow$ Внутренние узлы $\rightarrow$ Лист

Решающее дерево для задачи классификации



Корневой узел $\rightarrow$ Внутренние узлы $\rightarrow$ Лист

Выбор исхода для листа

Пусть X_p - примеры обучающей выборки попавшие в лист p и v_p предсказанное значение в листе p

Задача регрессии

Среднее по всем примерам обучающей выборки, попавшим в лист p

$$v_p = \frac{1}{|X_p|} \sum_{(x_i, y_i) \in X_p} y_i$$

Задача классификации

Самый часто встречающийся класс c среди примеров обучающей выборки, попавших в лист p

$$v_p = \arg\max_c \sum_{(x_i, y_i) \in X_p} [y_i = c]$$

Алгоритм построения решающих деревьев

NP-полная задача, поэтому применим жадный алгоритм

Алгоритм

- Имеем корень дерева, обучающую выборку $\{(x_m,\ y_m)\}_{m=1}^{M}$
- Найдем для данного узла условие разбиения - выберем признак j и порог t

$$[x^{\,j} \leq t]$$

- Разделим выборку на две согласно выбранному условию и запустим рекурсивную процедуру построения от потомков текущего узла
- Будем выполнять эту процедуру пока не выполнится условие останова

Основные вопросы в алгоритме

Как выбрать условие разбиения для узла?

Когда остановить рекурсивную процедуру построения?

Выбор условия разбиения в узле

Что подсказывает интуиция?

Выбор условия разбиения в узле

Пусть X_p - примеры обучающей выборки попавшие в лист p

Пусть x^j - j-й признак из входного вектора, t - некоторое число

Пусть разбили X_p на два множества (левое и правое)

$$X_l = \{x \in X_p | [x^j \leq t]\}$$

$$X_r = \{x \in X_p | [x^j > t]\}$$

Будем минимизировать функцию ошибки $Q(X_p, j, t)$ для условия $[x^j \leq t]$ среди всех возможных j и t

Как определить вид Q?

Выбор функции ошибки

Возьмем в качестве функции ошибки Q

$$Q(X_p, j, t) = \frac{|X_l|}{|X_p|} H(X_l) + \frac{|X_r|}{|X_p|} H(X_r)$$

$|X_p|$ - количество объектов в узле p

$|X_l|/|X_p|$ - доля объектов в левом листе

$|X_r|/|X_p|$ - доля объектов в правом листе

$H(X_l)$ - разброс ответов в левом листе

$H(X_r)$ - разброс ответов в правом листе

Как определить вид H?

Разброс ответов для регрессии

Пусть $\overline{y}$ - среднее значение обучающих примеров в узле

$$\overline{y} = \frac{1}{|X|} \sum_{(x_i,\, y_i) \in X} y_i$$

Средняя квадратичная ошибка

$$H(X) = \frac{1}{|X|} \sum_{(x_i,\, y_i) \in X} (y_i - \overline{y})^2$$

Средняя абсолютная ошибка

$$H(X) = \frac{1}{|X|} \sum_{(x_i,\, y_i) \in X} |y_i - \overline{y}|$$

Разброс ответов для классификации

Подсчитаем долю вхождения класса $c \in K$ в узел

$$\pi_c = \frac{1}{|X|} \sum_{(x_i, y_i) \in X} [y_i = c],$$

Вероятность ошибки

$$H(X) = 1 - \pi_{\hat{c}}, \quad \hat{c} = \arg\max_c \pi_c$$

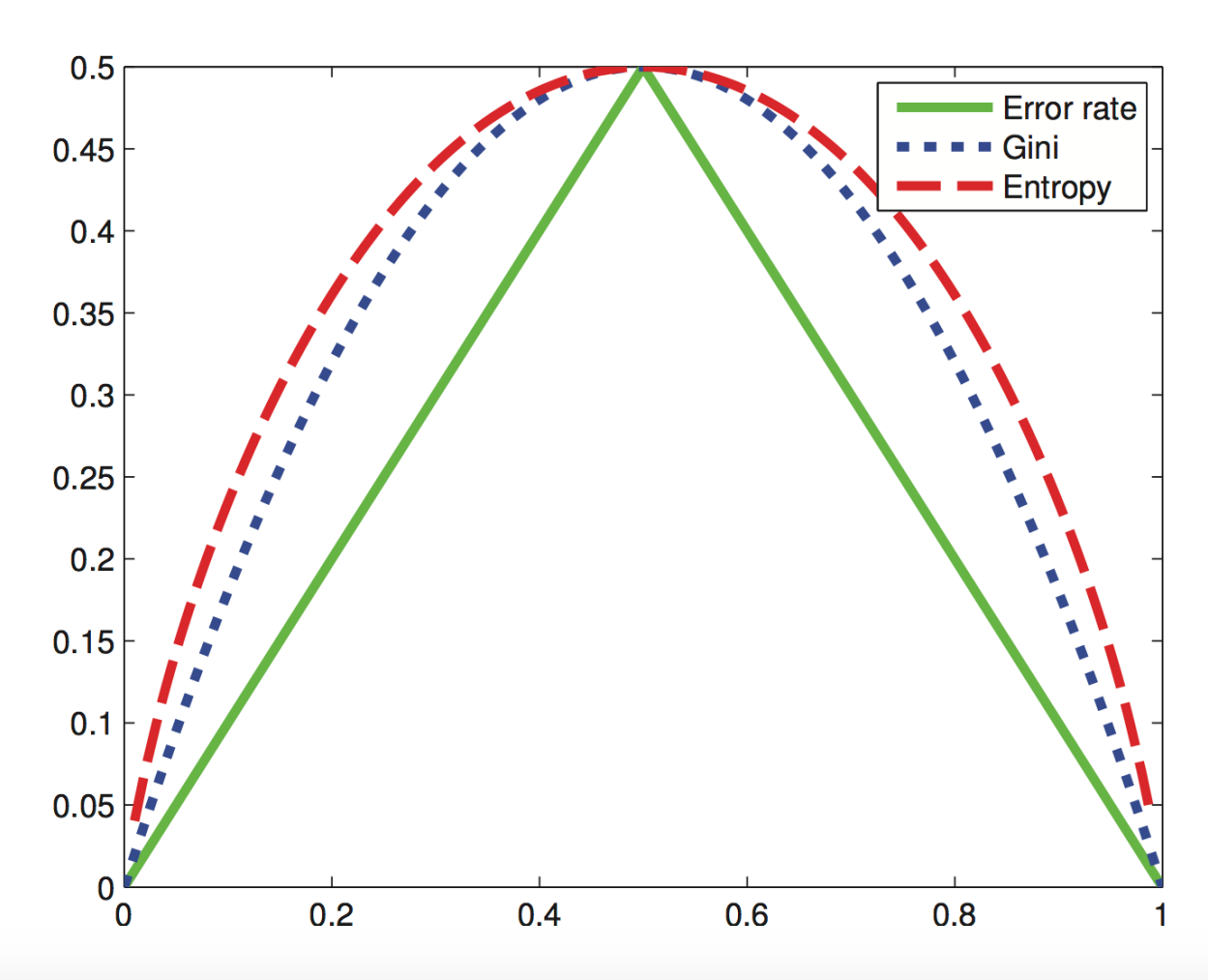
Энтропия

$$H(X) = -\sum_{c=1}^{K} \pi_c \log \pi_c$$

Индекс Джини

$$H(X) = \sum_{c=1}^{K} \pi_c (1 - \pi_c) = 1 - \sum_{c=1}^{K} \pi_c^2$$

Разброс ответов для классификации

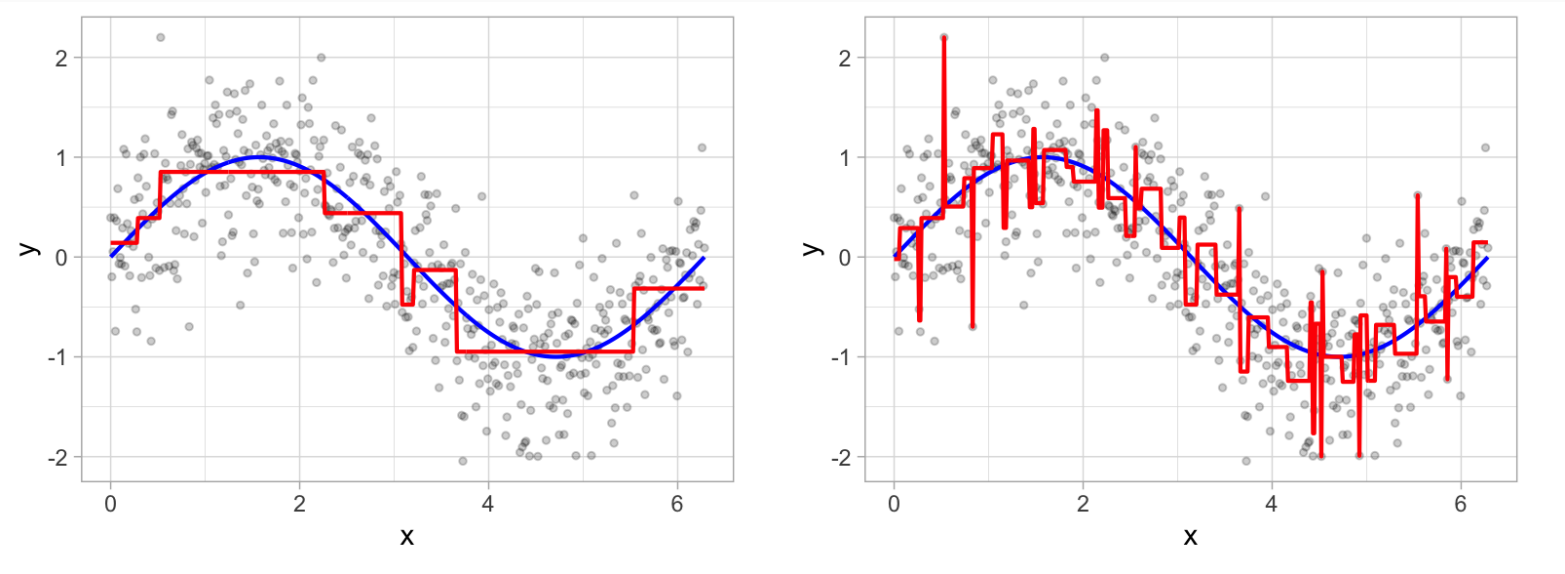


Для задачи бинарной классификации, на горизонтальной оси вероятность того, что узел принадлежит классу 1

Переобучение решающих деревьев

Решающее дерево может достичь нулевой ошибки на любой непротиворечивой выборке (например, отправив каждыи пример в отдельныи лист)

Чем меньше размер дерева, тем меньше риск переобучения



Критерии останова при обучении

Ограничение по размеру выборки в узле

- Мало примеров в узле = переобучение, больше не делим
- Много примеров в узле = недостоверный прогноз, продолжаем обучение
- Рекомендуется небольшое количество примеров (3-5) в качестве порогового значения, зависит от данных

Ограничение по глубине дерева

- Чем глубже дерево, тем выше вероятность переобучения
- Оптимальная глубина сильно зависит от обучающей выборки

Достоинства и недостатки решающих деревьев

Плюсы

- Чрезвычайно просты
- Очень легко интерпретируемы

Минусы

- Могут определять только простые зависимости
- Легко переобучаются
- Сильно меняются даже при небольшом изменении выборки

Решающие деревья в Python

Решающие деревья

- Классификация: sklearn.tree.DecisionTreeClassifier
- Регрессия: sklearn.tree.DecisionTreeRegressor

Полный список и описание

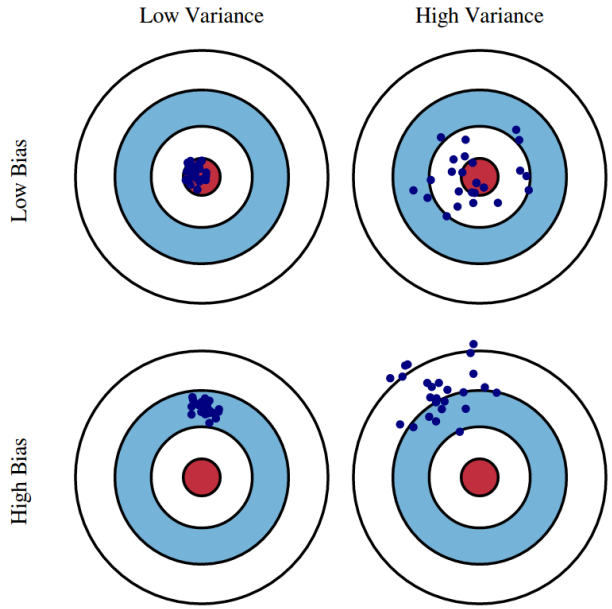
- https://scikit-learn.org/stable/modules/classes.htmlmodule-sklearn.tree

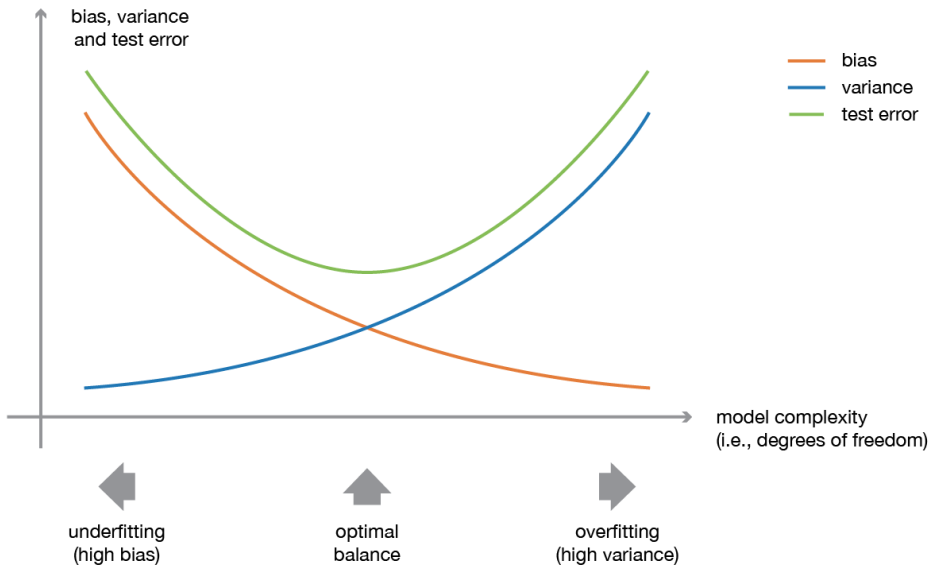
Ансамбли моделей

Дилемма смещения-дисперсии

Смещение (bias): ошибка при наличии бесконечных данных

Дисперсия (variance): как сильно меняется модель при тренировке на другом наборе данных; переобучение





Ансамбль моделей

- Ансамбль комбинирует несколько моделей для создания лучшей модели
- Генерируют несколько моделей с помощью одного и того же базового учителя
- Обычно в ансамбле используются быстрые базовые алгоритмы
- Ансамбли можно рассматривать как способ компенсации плохого алгоритма обучения путём дополнительных вычислений

Виды ансамблей

Параллельные ансамбли

- Цель: **снизить дисперсию** базовых моделей
- Базовые сложные модели генерируются параллельно
- Примеры: бэггинг решающих деревьев, случайный лес
- Не склонны к переобучению

Последовательные ансамбли (бустинг)

- Цель: **снизить смещение** базовых моделей
- Базовые простые модели генерируются последовательно
- Примеры: Adaboost, Gradient Boosting
- Могут переобучаться

Стекинг

Параллельные ансамбли

Параллельный ансамбль

Обучим параллельно несколько моделей $b_1(x), \ldots b_N(x)$

Усредним выводы моделей для получения предсказания

Для задачи регрессии

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

Для задачи классификации (majority voting)

$$f(x) = \arg\max_{c \in K} \left[\text{card}(n | b_n(x) = c) \right],$$

где K - множество всех классов

Бутстрап

- Случайная выборка S из набора данных D с возвращением
- Репрезентативность: $|S|$ велико
- Независимость: $|S| < |D|$
- Создаем почти i.i.d. тренировочные наборы данных

Беггинг (bagging)

Сгенерируем N почти i.i.d. тренировочных наборов данных с помощью бутстрап

Обучаем базовые почти i.i.d модели $\{b_n\}_{n=1}^N$

Итоговая модель f есть усреднение по всем моделям $\{b_n\}_{n=1}^N$

$$f(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x),$$

где x - тестовый входной вектор

Усреднение моделей не изменяет их смещение, но снижает их дисперсию (аналогично i.i.d. случайным величинам)

$$\text{Дисперсия f} = \frac{\text{Дисперсия } b_n}{N} + \text{Корреляция между } \{b_n\}_{n=1}^N$$

Метод случайных подпространств

Еще один способ уменьшить корреляцию между деревьями

Обеспечивает устойчивость к отсутствующим данным

Также называется attribute bagging

Метод случайных подпространств

- D - множество признаков в исходном наборе данных
- N - количество моделей в ансамбле
- (бэггинг) Для каждой из базовых моделей $b_n,\ n = 1, \ldots N$ сгенерируем обучающий набор бутстрап методом
- (метод с.п.) Для каждой из базовых моделей b_n случайным образом выберем набор признаков $d \subset D$ и проведем тренировку моделей $b_n,\ n = 1, \ldots N$. Часто $|d| = \sqrt{|D|}$

Случайный лес (random forest)

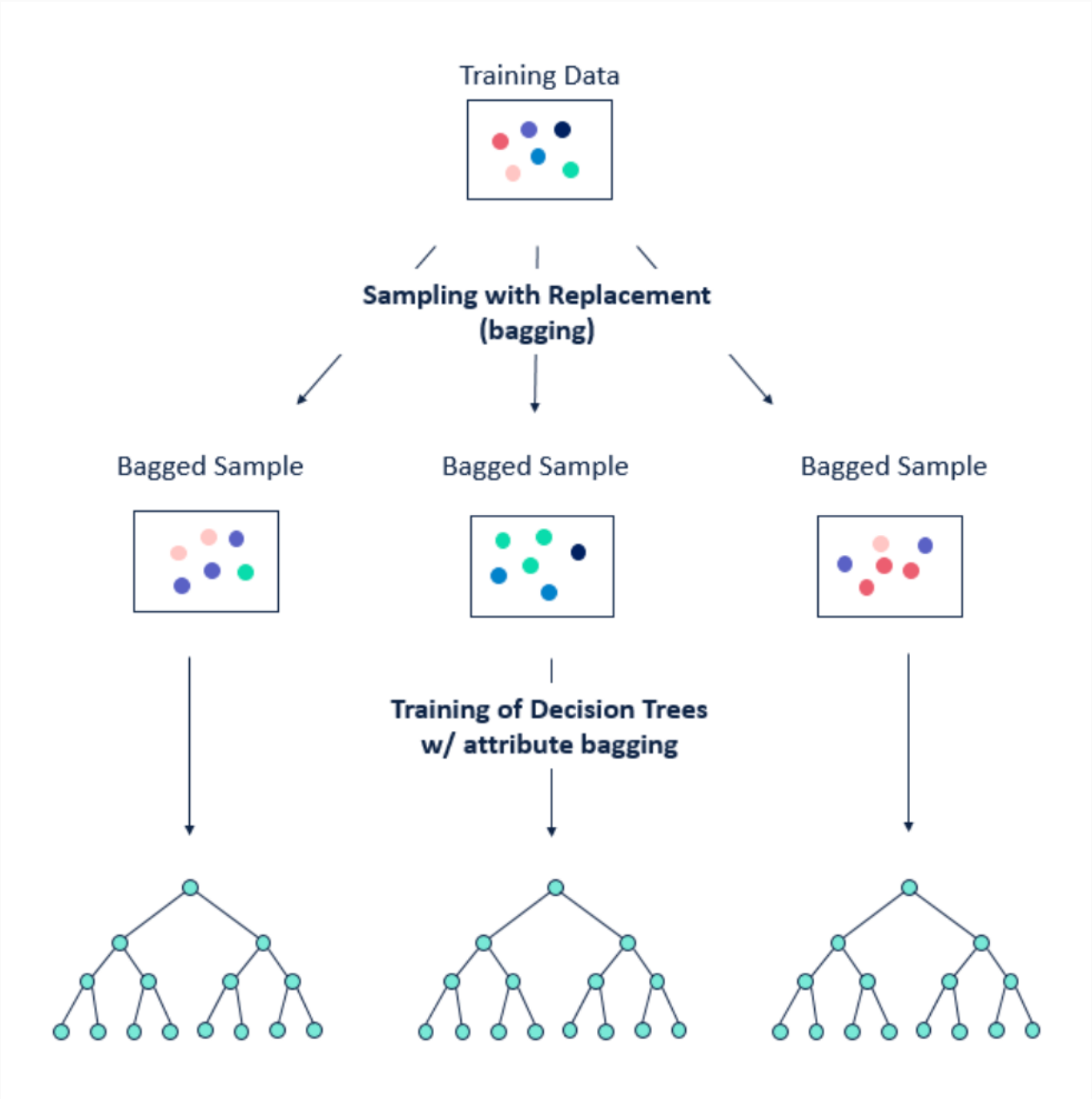
Случайный лес

- Глубокие деревья в качестве базовых моделей
- Беггинг
- Метод случайных подпространств

Вариация метода случайных подпространств

В классической формулировке алгоритма случайного леса
Breiman, Leo. "Random Forests" Machine learning 45.1 (2001)
все деревья имеют доступ ко всем признакам, но в каждом узле дерева рассматривается только подмножество признаков

Тренировка случайного леса



Достоинства и недостатки случайного леса

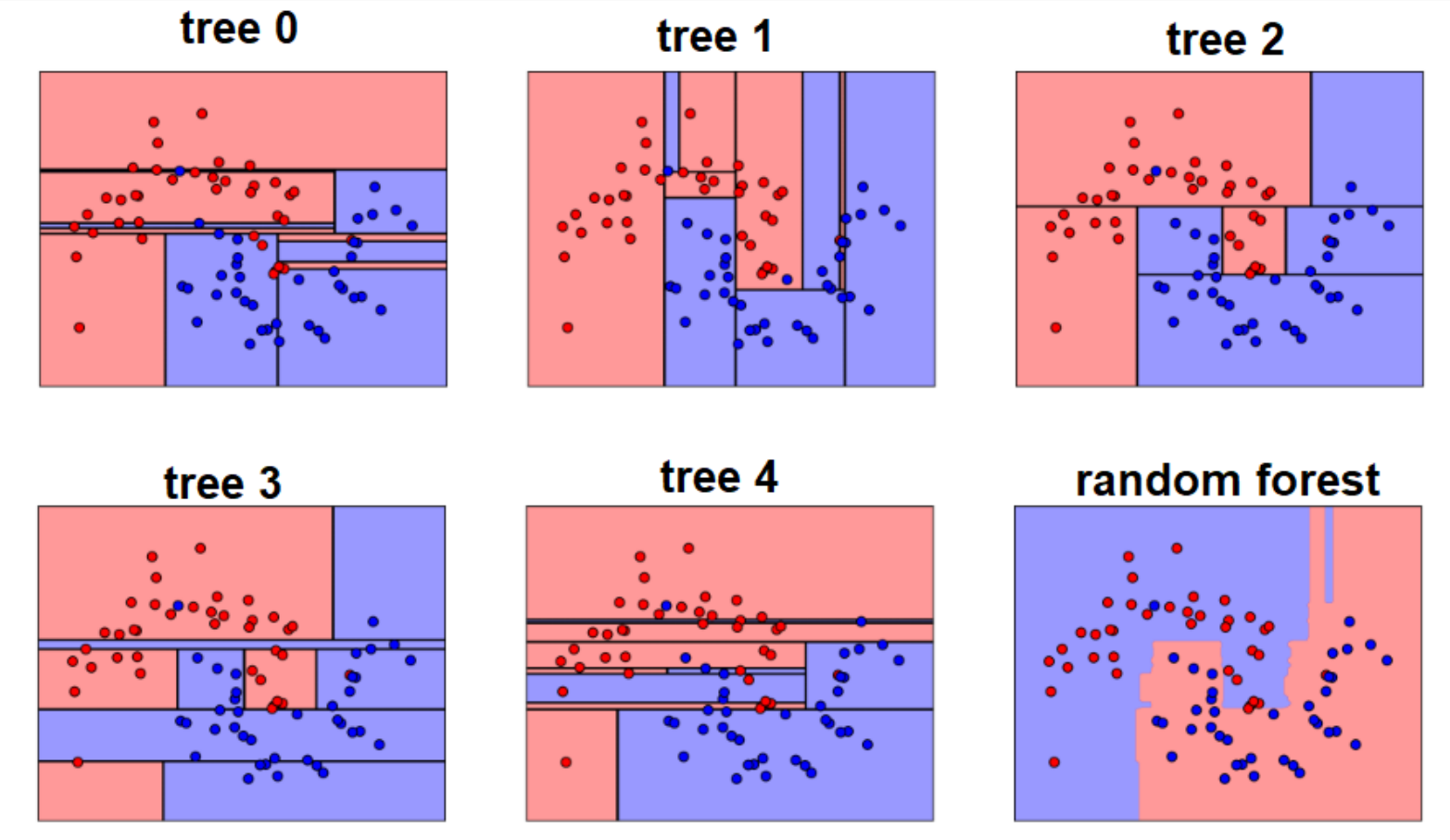
Плюсы

- Могут обрабатывать данные с большим количеством признаков
- Не переобучается при росте числа деревьев
- Каждое дерево строится независимо поэтому можно распараллелить процесс обучения
- Не чувствительны к масштабированию признаков

Минусы

- Для сложных задач нужно слишком много деревьев

Пример классификации с помощью случайного леса



Беггинг и случайный лес в Python

Беггинг

- Классификация: sklearn.ensemble.BaggingClassifier
- Регрессия: sklearn.ensemble.BaggingRegressor

Случайный лес

- Классификация: sklearn.ensemble.RandomForestClassifier
- Регрессия: sklearn.ensemble.RandomForestRegressor

Полный список и описание

- https://scikit-learn.org/stable/modules/classes.htmlmodule-sklearn.ensemble

Последовательные ансамбли (бустинг)

Бустинг (базовая идея)

Пусть имеем задачу регрессии $x \to y$

Построим некоторую базовую модель b_1

Получим следующие результаты

Пример	Истинное	Прогноз	Поправка
1	y_1	$b_1(x_1)$	$y_1 - b_1(x_1)$
2	y_2	$b_1(x_2)$	$y_2 - b_1(x_2)$
...	...	...	...
m	y_m	$b_1(x_m)$	$y_m - b_1(x_m)$

Обучим следующую модель b_2 для предсказания поправки

Продолжим процесс рекурсивно до некоторого момента

Бустинг (постановка задачи)

Оптимизационная задача

$$\min_f \sum_{m=1}^{M} L\left(y_m, f(x_m)\right), \quad f(x) = \sum_{n=1}^{N} b(x;\ \gamma_n),$$

где L - функция потерь, f - итоговая модель, $b(x;\ \gamma_n)$ - базовая модель с параметрами γ_n, $\{(x_m, y_m)\}_{m=1}^{M}$ - обучающие примеры

Сложно оптимизировать сразу по всем $\{\gamma_n\}_{n=1}^{N}$, поэтому применим **итеративный подход**

Функция потерь L может быть разная в зависимости от задачи

Бустинг (итеративный подход)

Итеративный подход

- На шаге n найдем оптимальные γ_n - параметры базовой модели (M - количество обучающих примеров)

$$\gamma_n = \arg\min_{\gamma} \sum_{m=1}^{M} L(y_m,\ f_{n-1}(x_m) + b(x_m; \gamma))$$

- Добавим базовую модель $b(x;\ \gamma_n)$ к итоговой модели f

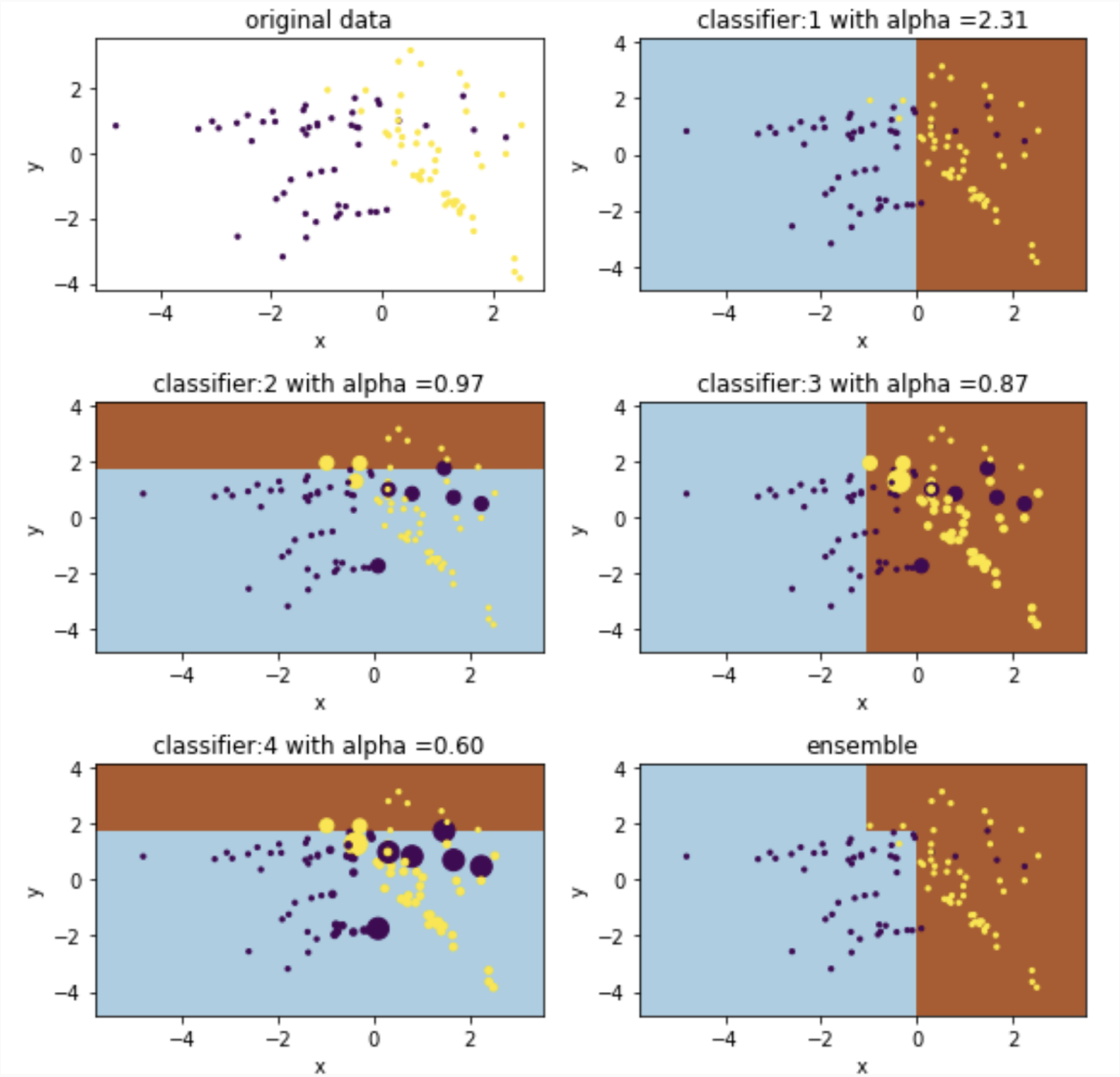
$$f_n(x) = f_{n-1}(x) + b(x;\ \gamma_n)$$

Возьмем в качестве функции L **квадратичную ошибку**, тогда

$$L(y_m,\ f_{n-1}(x_m) + b(x_m;\ \gamma)) = (r_{mn} - b(x_m;\ \gamma))^2,$$

где поправка $r_{mn} = y_m - f_{n-1}(x_m)$, т.о. приближаем поправку

Пример работы бустинга (Adaboost)



Градиентный бустинг

Пусть мы сразу хотим оптимизировать итоговую модель f

$$\hat{f} = \arg\min_f L(f),$$

где $f = (f(x_1), \dots f(x_M))$ - 'параметры', M - количество обучающих примеров

Будем решать используя градиентный спуск по 'параметрам'

$$g_{mn} = \left[\frac{\partial L(y_m, f(x_m))}{\partial f(x_m)}\right]_{f=f_{n-1}}$$

причем для квадратичной функции потерь $g_{mn} = y_m - f(x_m)$

Шаг функционального градиентного спуска

$$f_n = f_{n-1} - \nu g_n$$

НО мы оптимизируем f только в фиксированных M точках!

Алгороитм градиентного бустинга

- Инициализировать $f_0(x) = \arg\min_\gamma \sum_{m=1}^{M} L(y_m, b(x_m; \gamma))$
- Цикл $n = 1 : N$ (количество шагов бустинга)
  - Расчет градиентной поправки (разность в случае квадратичной L)

$$r_{mn} = -\left[\frac{\partial L(y_m, f(x_m))}{\partial f(x_m)}\right]_{f(x_m)=f_{n-1}(x_m)}$$

  - Найти параметры γ_n следующей базовой модели b, которая минимизирует выражение (приближая градиент)

$$\sum_{m=1}^{M}(r_{mn} - b(x_m; \gamma_n))^2$$

  - Обновить итоговую модель $f_n(x) = f_{n-1}(x) + \nu b(x; \gamma_n)$
- Вернуть $f(x) = f_N(x)$

Борьба с переобучением градиентного бустинга

Способы снижения вероятности переобучения

- Плавно уменьшать скорость обучения (параметр шага градиентного спуска ν)
- Правильно выбрать количество базовых моделей (большое количество моделей приводит к переобучению)
- Регуляризация самих базовых моделей (ограничить максимальную глубину деревьев)
- Стохастический градиентный спуск - используем только определенную подвыборку данных на каждом шаге градиентного спуска

Градиентный бустинг над обучающими деревьями

Бустинг над обучающими деревьями

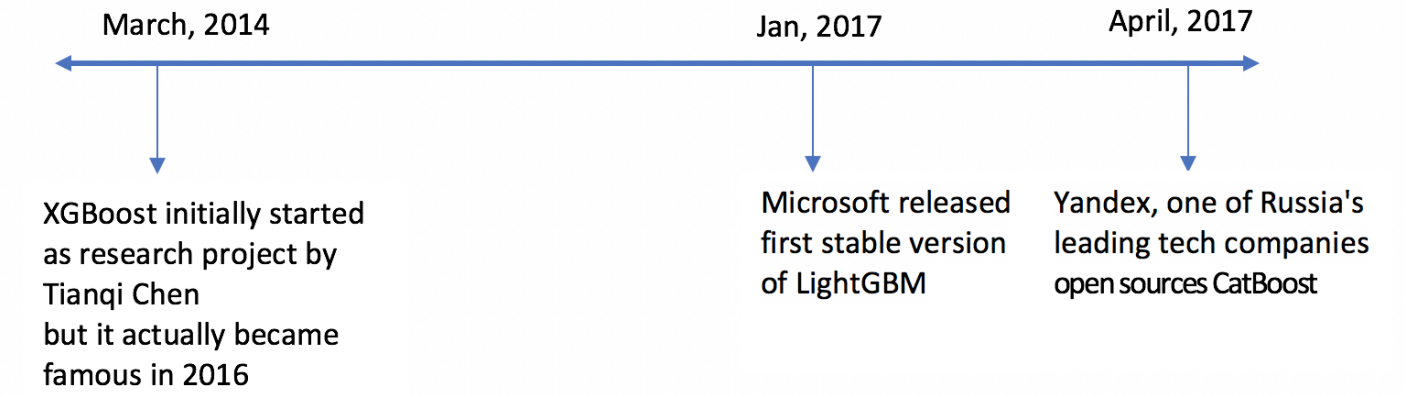
Выберем в качестве базовои модели решающее дерево

Стараемся строить простые деревья:

- Ограничиваем высоту (обычно 2-20)
- Учим на подвыборке примеров и подпространстве признаков

Градиентный бустинг в Python

- XGBoost - наиболее распространенный
- LightGBM - быстрый
- Catboost - тоже довольно быстрый, хорошо работает с категориальными признаками



Достоинства и недостатки градиентного бустинга

Плюсы

- Отличные характеристики на задачах регрессии и классификации
- Обычно качество предсказания выше, чем у случайного леса
- Не чувствительны к масштабированию признаков

Минусы

- Долгая тренировка
- Требуется подбор гиперпараметров
- Возможно переобучение
- Чувствительны к выбросам (outliers)

Спасибо за внимание!