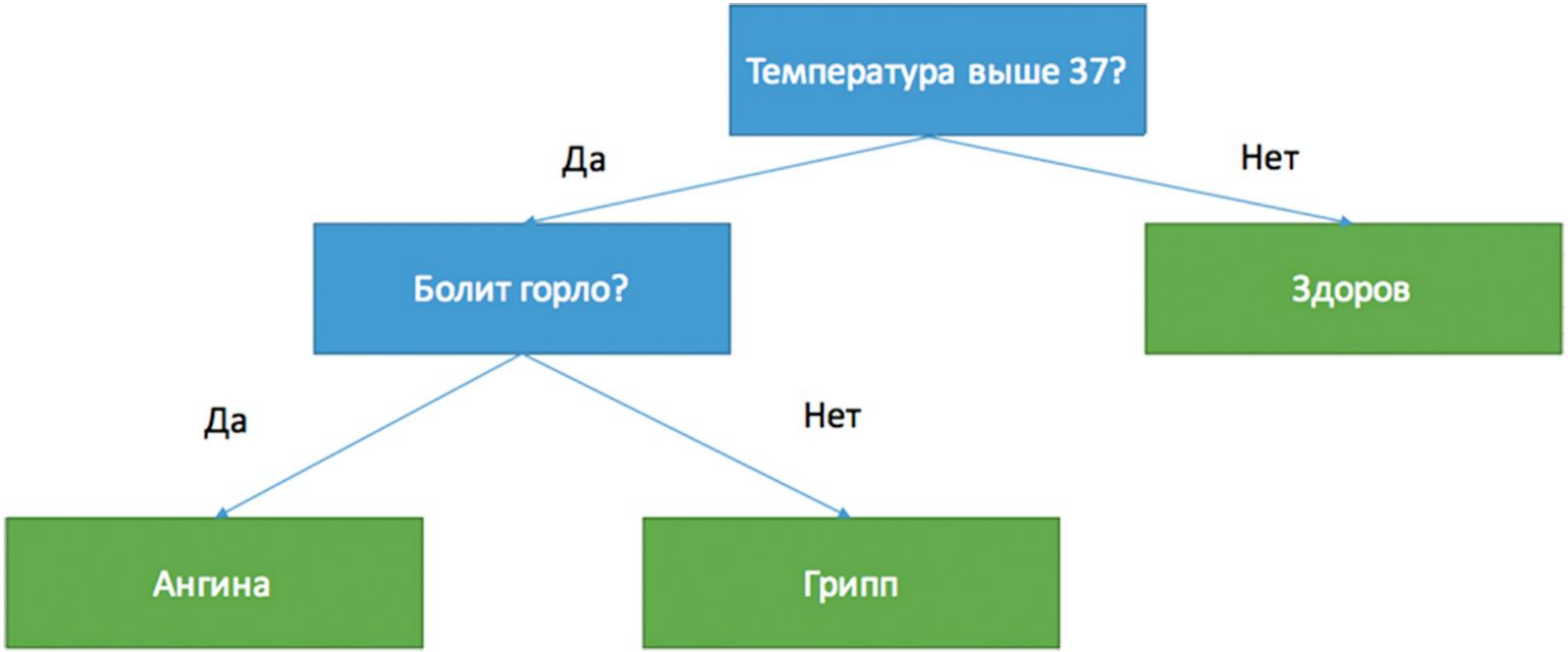
Решающие деревья: Random Forest, Gradient Boosting

Решающие деревья



Решающие деревья

- В каждой вершине дерева записано условие
- Условие - признак и порог для него
- Сыновья вершины - варианты для этого условия: признак больше порога, признак меньше порога
- В листьях записаны исходы и, возможно, их вероятности

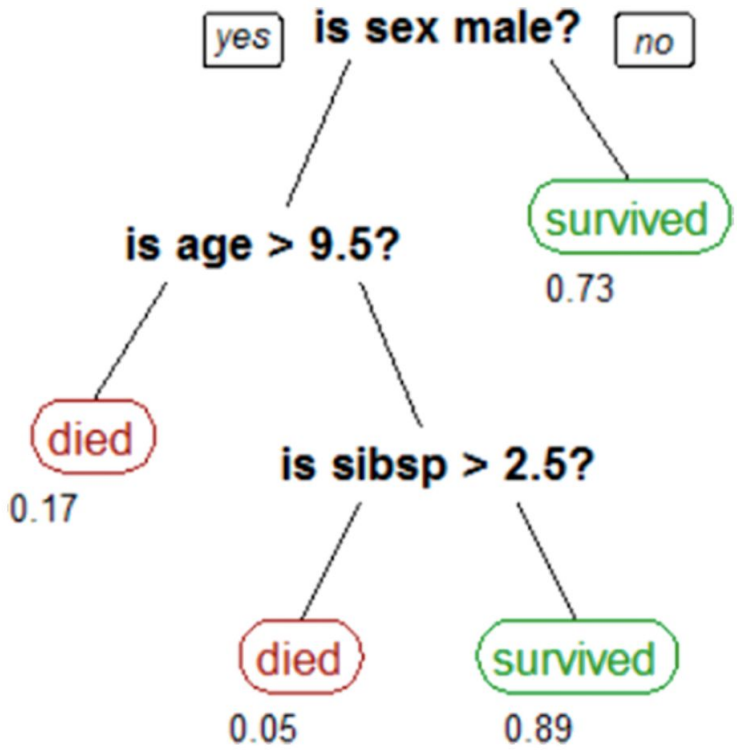
Плюсы:

- Чрезвычайно просты
- Очень легко интерпретируемы

Минусы

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

Решающие деревья на примере Титаника



- Имеем вероятность выживания в каждом листе
- Как при этом определить исходы?

Выбор исхода для листа в задаче классификации

$$a_m = \underset{y \in \mathbb{Y}}{\operatorname{argmax}} \sum_{i \in X_m} [y_i = y]$$

- Будем брать в качестве исхода самый часто встречающийся исход среди примеров обучающей выборки, попавших в этот лист
- В этом случае можно также определить вероятность как отношение числа примеров данного класса ко всем остальным

Выбор исхода для листа в задаче регрессии

$$a_m = \frac{1}{|X_m|} \sum_{i \in X_m} y_i$$

- Будем брать в качестве предсказания среднее по всем примерам обучающей выборки, попавшим в этот лист

Алгоритм построения решающих деревьев

- Имеем корень дерева и полную обучающую выборку
- Найдём для данной вершины условие разбиения - выберем признак и порог
- Разделим выборку на две согласно выбранному условию и запустим рекурсивную процедуру построения от детей нашей вершины
- Будем выполнять эту процедуру до некоторого момента

Алгоритм построения решающих деревьев

Имеем 2 важных вопроса:

- Как выбрать условие разбиения для вершины?
- Когда остановить рекурсивную процедуру построения?

Алгоритм построения решающих деревьев

Имеем 2 важных вопроса:

- **Как выбрать условие разбиения для вершины?**
- Когда остановить рекурсивную процедуру построения?

Как выбрать условие разбиения в вершине

- Рассмотрим, как найти условие для разбиения в вершине
- Пусть в данную вершину попало множество примеров X_m
- Тогда будем минимизировать ошибку $Q(X_m, j, t)$ для условия $[x^j \leq t]$
- Минимизировать будем среди всех возможных j и t

Как выбрать условие разбиения в вершине

- Перебирать все возможные j и t слишком ресурсозатратно
- Для повышения случайности дерева можно выбирать признак j случайно

Алгоритм построения решающих деревьев

- Как выбрать ошибку разбиения $Q(X_m, j, t)$?
- Пусть мы разбили X_m на два множества:

$$X_\ell = \{x \in X_m | [x^j \leq t]\}$$

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

- Тогда возьмём в качестве функции ошибки:

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

Алгоритм построения решающих деревьев

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

Доля объектов в листьях

Алгоритм построения решающих деревьев

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

Разброс ответов в левом листе

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

Разброс ответов в правом листе

Как можно считать разброс?

- Как посчитать разброс значения в числовой выборке?

Разброс ответов для регрессии

$$\bar{y}(X) = \frac{1}{|X|} \sum_{i \in X} y_i$$

$$H(X) = \frac{1}{|X|} \sum_{i \in X} (y_i - \bar{y}(X))^2$$

- По сути мы как раз имеем дисперсию

Как можно считать разброс для классификации?

- Почему не подойдёт дисперсия?

Разброс ответов для классификации

- Для каждого класса подсчитаем долю вхождения:

$$p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k]$$

- Критерий Джини: $H(X) = \sum_{k=1}^{K} p_k (1 - p_k)$

  Минимизируя этот критерий мы максимизируем число пар объектов одного класса

- Энтропийный критерий: $H(X) = -\sum_{k=1}^{K} p_k \ln p_k$

  Будем считать, что $0 \ln 0 = 0$

Алгоритм построения решающих деревьев

Имеем 2 важных вопроса:

- Как выбрать условие разбиения для вершины?
- **Когда остановить рекурсивную процедуру построения?**

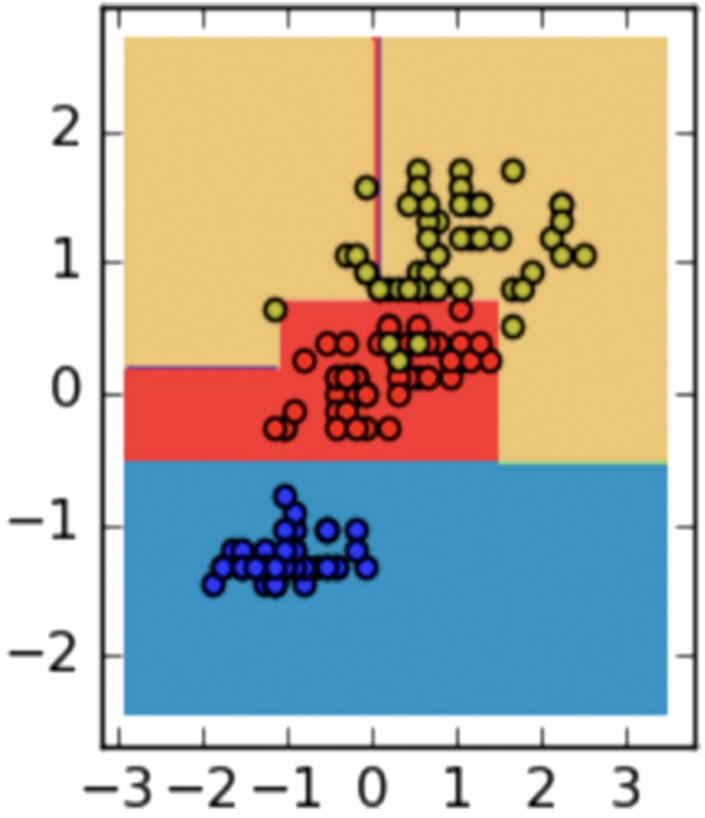
Переобучение решающих деревьев

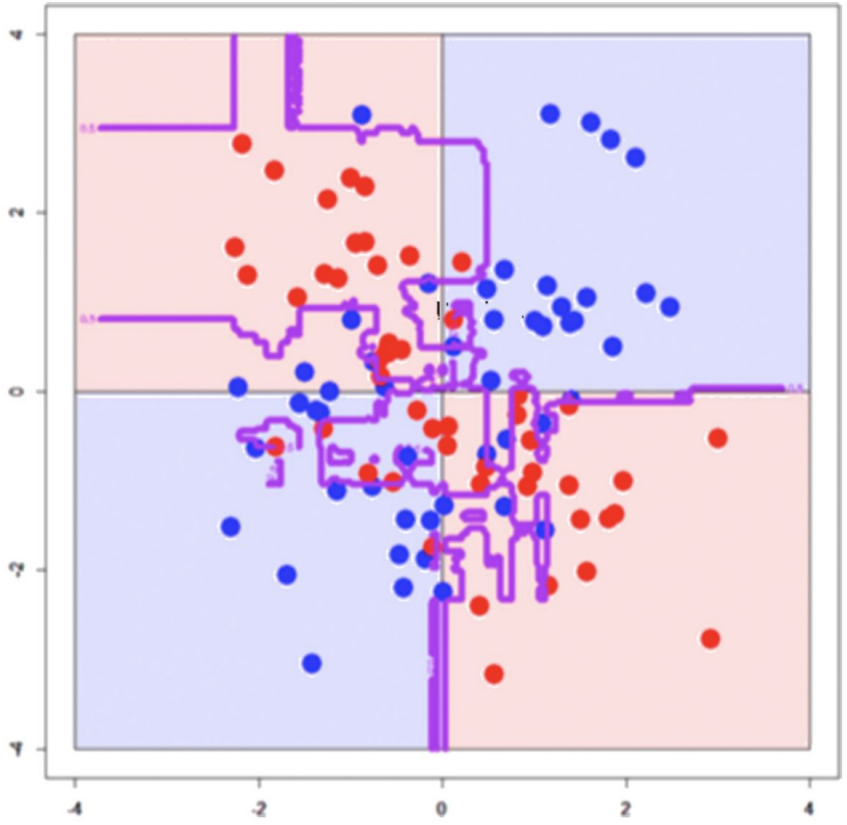
- Решающее дерево может достичь нулевой ошибки на любой непротиворечивой выборке (например, отправив каждый пример в отдельный лист)
- В таком случае очевидно, что чем меньше размер дерева, тем меньше риск переобучения

Примеры для задачи классификации

- Что можно сказать о качестве этих моделей?

Примеры для задачи классификации

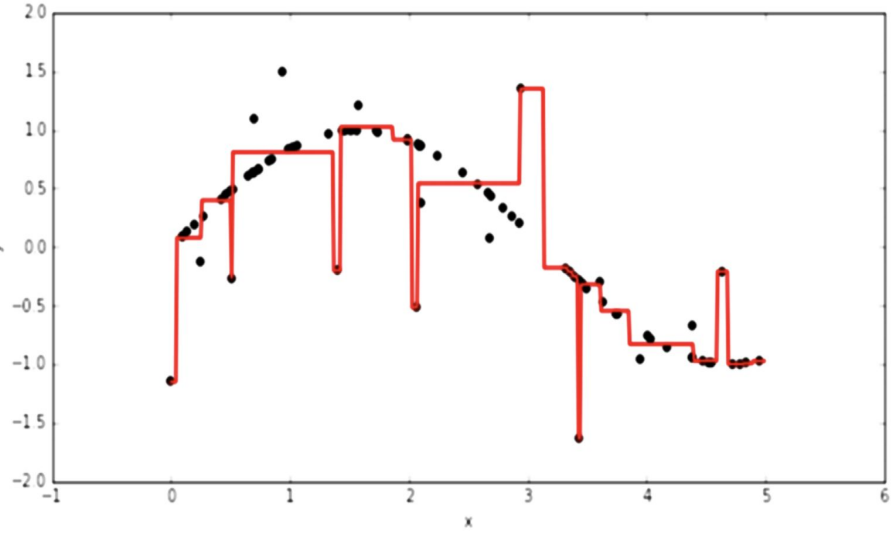


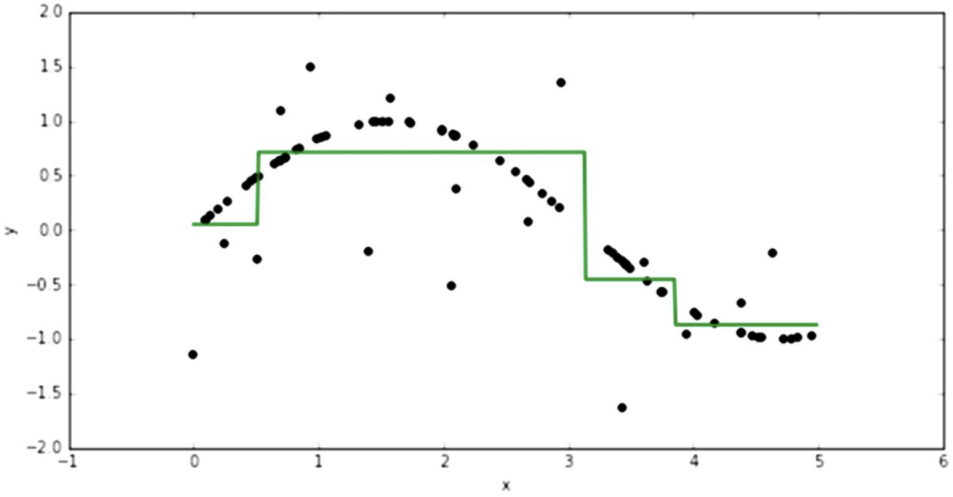


Примеры для задачи регрессии

- Что можно сказать о качестве этих моделей?

Примеры для задачи регрессии





Критерий останова при разбиении

- По размеру выборки в вершине:
  - При **n=1** получаем свой лист на каждый пример и переобучаемся
  - При слишком большом **n** получим недостоверный прогноз
  - Рекомендуется использовать небольшие **n** (напрмер, в диапазоне 3-5)
- Ограничение по глубине:
  - Очень хорошо работает в ансамблях
  - Ограничение глубины должно подбираться индивидуально для каждой задачи и очень сильно зависит от обучающей выборки

Решающие деревья в Python

- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.DecisionTreeRegressor

Полный список и описание доступны здесь:
https://scikit-learn.org/stable/modules/classes.html#module-sklearn.tree

Построение ансамблей

- Вместо построения только одной модели будем строить сразу несколько
- Далее объединим ответы в один общий

Ансамбль

- Пример для бинарной классификации на классы {-1, 1}
  - Обучим сразу много моделей $b_1(x), ..., b_N(x)$
  - Усредним ответы:

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

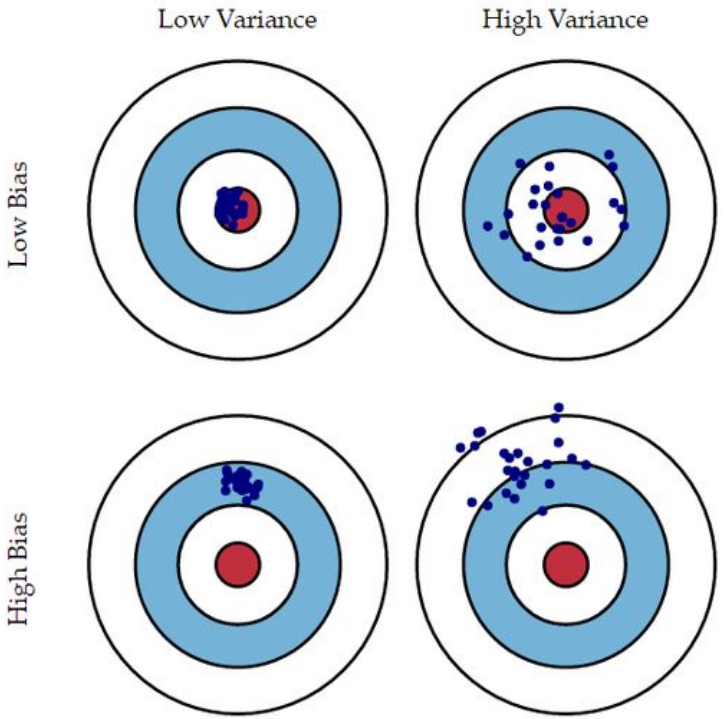
- Для разнообразия будем обучать модели на различных подвыборках обучающей выборки

Бутстрап

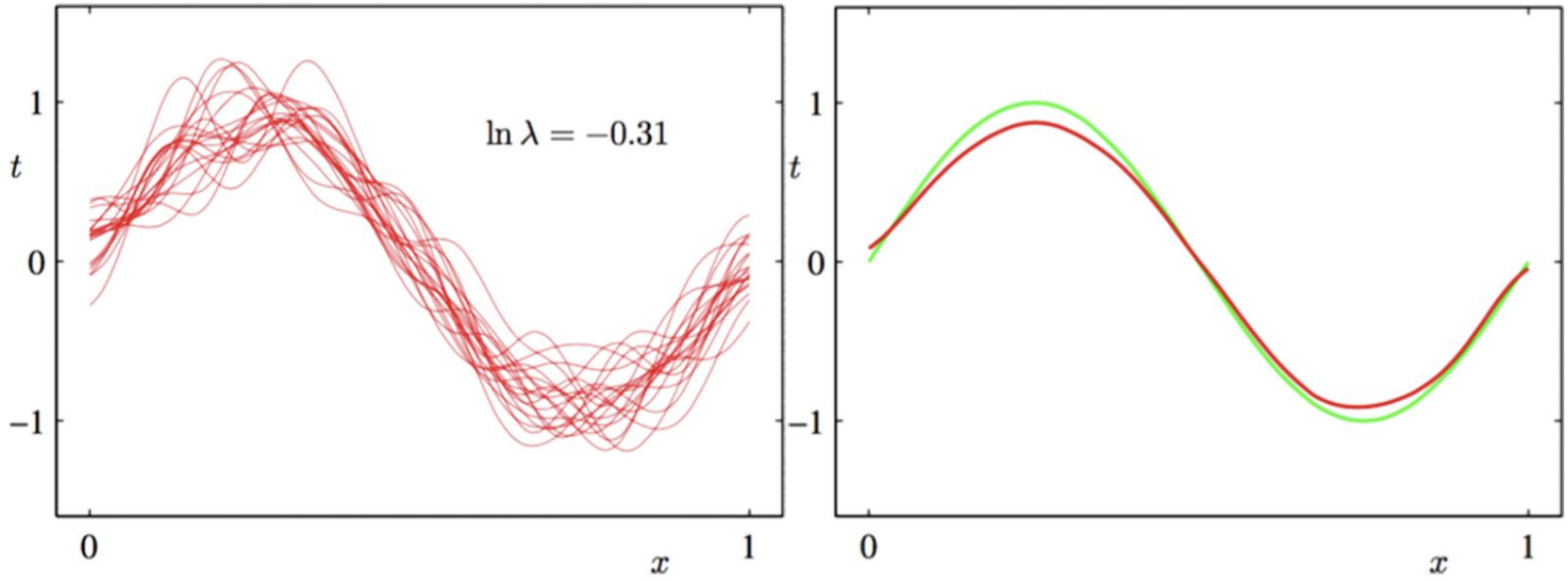
- Будем каждый раз случайно выбирать ℓ элементов с возвращением из выборки размера ℓ
- Например, из множества {x1, x2, x3, x4} -> {x1, x1, x2, x3}
- В таком случае каждый раз мы будем получать порядка $0.632 * \ell$ различных элементов

Качество ансамбля

- Выделим два параметра ошибки: разброс и смещение
- Очевидно, что ансамбли не позволит изменить смещение
- При этом ансамбль позволит сократить разброс за счёт усреднения



Качество композиции



Качество ансамбля

- Смещение ансамблей = смещению базовых алгоритмов
- Разброс ансамблей = (разбросу базовых алгоритмов) / N + (корреляция между базовыми алгоритмами), где N - число алгоритмов в ансамбле
- Таким образом, можно сократить разброс в N раз при использовании N независимых алгоритмов

Построение максимально независимых алгоритмов в композиции

- Бэггинг - обучение на случайном подмножестве обучающих семплов
- Метод случайных подпространств - обучение на случайном множестве признаков
- Объединение двух предыдущих методов
- Доли семплов и признаков - гиперпараметры
- Больше случайности в построение самих базовых алгоритмов

Ансамбли над решающими деревьями

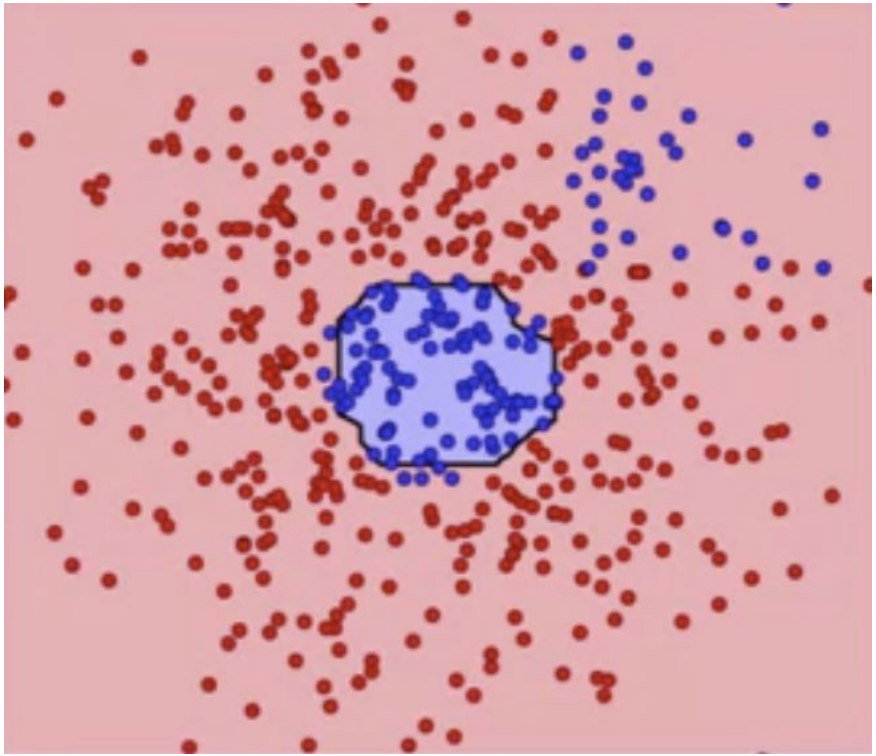
- Случайный лес
- Бустинг
- Градиентный бустинг

Случайный лес

- Ансамбль решающих деревьев
- Решающие деревья должны быть максимально непохожими
- Будем использовать случайный выбор признаков при разбиении вершин
- Будем использовать также беггинг, метод случайных подпространств
- Будем ограничивать число примеров в вершине или глубину дерева для борьбы с переобучением
- Ответ будем просто усреднять
- При этом решающие деревья должны быть глубокими, чтобы улавливать сложные закономерности

Случайный лес

- Построим случайный лес из неглубоких решающих деревьев
- Каждое решающее дерево может найти только простую закономерность
- Следовательно, композиция тоже не сможет найти сложных закономерностей



Случайный лес

Плюсы:

- Низкая корреляция между деревьями
- Не переобучается при росте числа деревьев
- Каждое дерево строится независимо => можно очень хорошо распараллелить

Минусы:

- Для сложных задач нужно слишком много деревьев

Случайный лес в Python

- sklearn.ensemble.RandomForestClassifier
- sklearn.ensemble.RandomForestRegressor

Полный список и описание доступны здесь:
https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble

Бустинг

- Базовые алгоритмы обучаются последовательно
- Каждый последующий алгоритм исправляет ошибки ансамбля из предыдущих
- Благодаря этому не требуются сложные алгоритмы

Бустинг

- Имеем задачу регрессии
- Построим некоторый базовый алгоритм
- Получим следующие результаты:

Ответы	y_1	y_2	...	y_ℓ
Прогнозы	$b_1(x_1)$	$b_1(x_2)$	...	$b_1(x_\ell)$
Поправка	$y_1 - b_1(x_1)$	$y_2 - b_1(x_2)$	...	$y_\ell - b_1(x_\ell)$

- Будем далее предсказывать поправку
- Продолжим процесс рекурсивно до некоторого момента

Градиентный бустинг

- Для начала необходимо построить самый первый базовый алгоритм
- Лучше взять его максимально простым:

$$b_0(x) = 0$$

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

$$b_0(x) = \operatorname*{argmax}_{y \in \mathbb{Y}} \sum_{i=1}^{\ell} [y_i = y]$$

Градиентный бустинг

- Будем действовать итеративно
- Пусть у нас уже построено $N-1$ базовых алгоритмов
- Тогда имеем текущее предсказание:

$$a_{N-1}(x) = \sum_{n=0}^{N-1} b_n(x)$$

Градиентный бустинг

- На текущей итерации требуется найти новый базовый алгоритм $b(x_i)$
- Данный базовый алгоритм должен минимизировать следующий функционал:

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + b(x_i)\right) \to \min_b$$

Градиентный бустинг

- Немного упростим задачу
- Найдём прогнозы $s = (s_1, \dots, s_\ell)$, которые будут оптимальны для обучающей выборки:

$$F(s) = \sum_{i=1}^{\ell} L\big(y_i, a_{N-1}(x_i) + s_i\big) \to \min_s$$

- Как найти такой вектор прогнозов?

Градиентный бустинг

- Найдём данный вектор с помощью антиградиента функционала ошибки на обучающей выборке:

$$s = \Big(-L'_z(y_1, a_{n-1}(x_1)), \dots \\ \dots, -L'_z(y_\ell, a_{n-1}(x_\ell)) \Big)$$

- Однако, наша задача найти функцию для всего пространства объектов, а не только для обучающей выборки

Градиентный бустинг

- Поэтому обучим алгоритм, который приближает антиградиент на обучающей выборке:

$$b_N(x) = \underset{b}{\operatorname{argmin}} \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2$$

- Добавим данный алгоритм в композицию:

$$a_n(x) = \sum_{m=1}^{n} b_m(x)$$

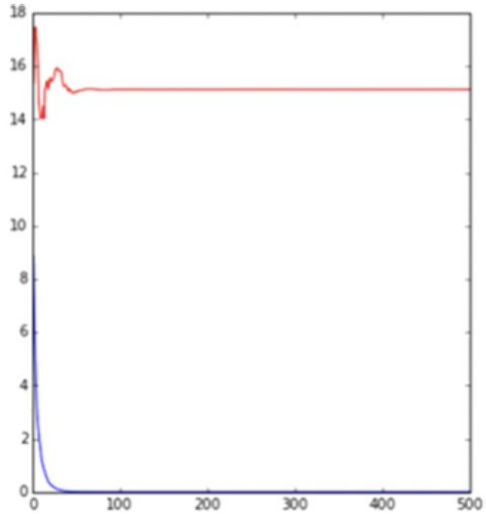
Борьба с переобучением

- Обучение на подвыборках семплов и подпространствах признаков
- Регуляризация самих базовых моделей
- Сокращение шага:
  - Зафиксируем длину шага $\eta \in (0, 1]$
  - Будем добавлять ответ очередного алгоритма, умноженный на шаг:

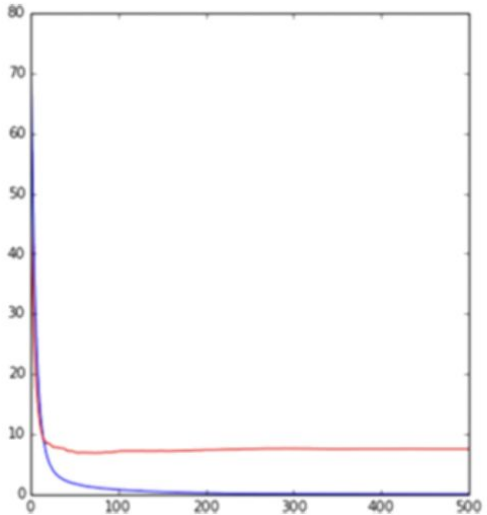
$$a_N(x) = a_{N-1}(x) + \eta b_N(x)$$

Правильный подбор шага

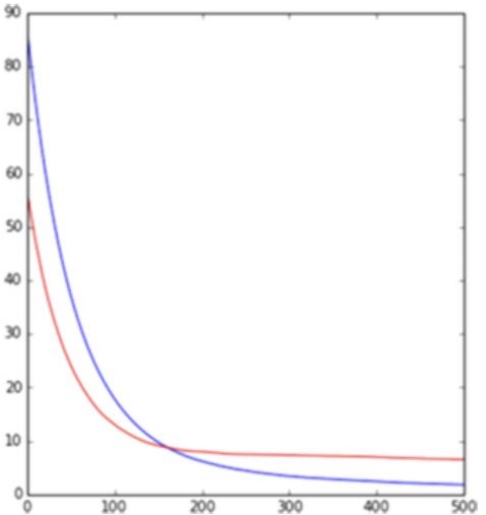
- Правильный подбор шага очень важен:



$\eta = 1$



$\eta = 0.1$



$\eta = 0.01$

Правильный подбор шага

- Чем меньше шаг, тем больше нужно базовых алгоритмов
- Нужно подбирать наравне с другими гиперпараметрами
- Шаг можно также подбирать минимизируя функционал ошибки итогового ансамбля

Стохастический градиентный бустинг

- Обучаем каждый алгоритм не на всей выборке, а на случайном подмножестве

Градиентный бустинг над решающими деревьями

- Выберем в качестве базовой модели решающее дерево
- Будем с помощью регуляризаций стараться строить простые деревья
  - Ограничиваем высоту (обычно 2-20)
  - Учим на подвыборке семплов и подпространстве признаков

Градиентный бустинг в Python

- XGBoost - наиболее распространенный
- LightGBM - очень быстрый, обычно немного проигрывает XGBoost в точности
- CatBoost - довольно быстрый и точный имеет множество полезных фич, но на практике чаще всего проигрывает двум предыдущим

Основные гиперпараметры в XGBoost

- max_depth - ограничивает максимальную глубину деревьев в ансамбле
- learning_rate - устанавливает скорость обучения
- n_estimators - количество деревьев в ансамбле
- min_child_weight - ограничивает снизу сумму весов сыне
- subsample - определяет долю семплов для обучения каждого дерева
- colsample_bytree - определяет долю фичей для обучения каждого дерева