

# Деревья решений и ансамбли

Занятие №7

Журавлев Вадим и Ярошенко Ангелина



образование

# Ансамбли



# Основная идея

Суть ансамблей состоит в построении нескольких независимых/зависимых моделей и объединении их предсказаний с целью повышения качества.



# Простое голосование

Решение принимается на основании большинства голосов в **классификации** или среднего значения в **регрессии**.



# Взвешенное голосование

Решение принимается на основании  
взвешенного большинства голосов в  
**классификации** или  
Взвешенного среднего значения в  
**регрессии.**

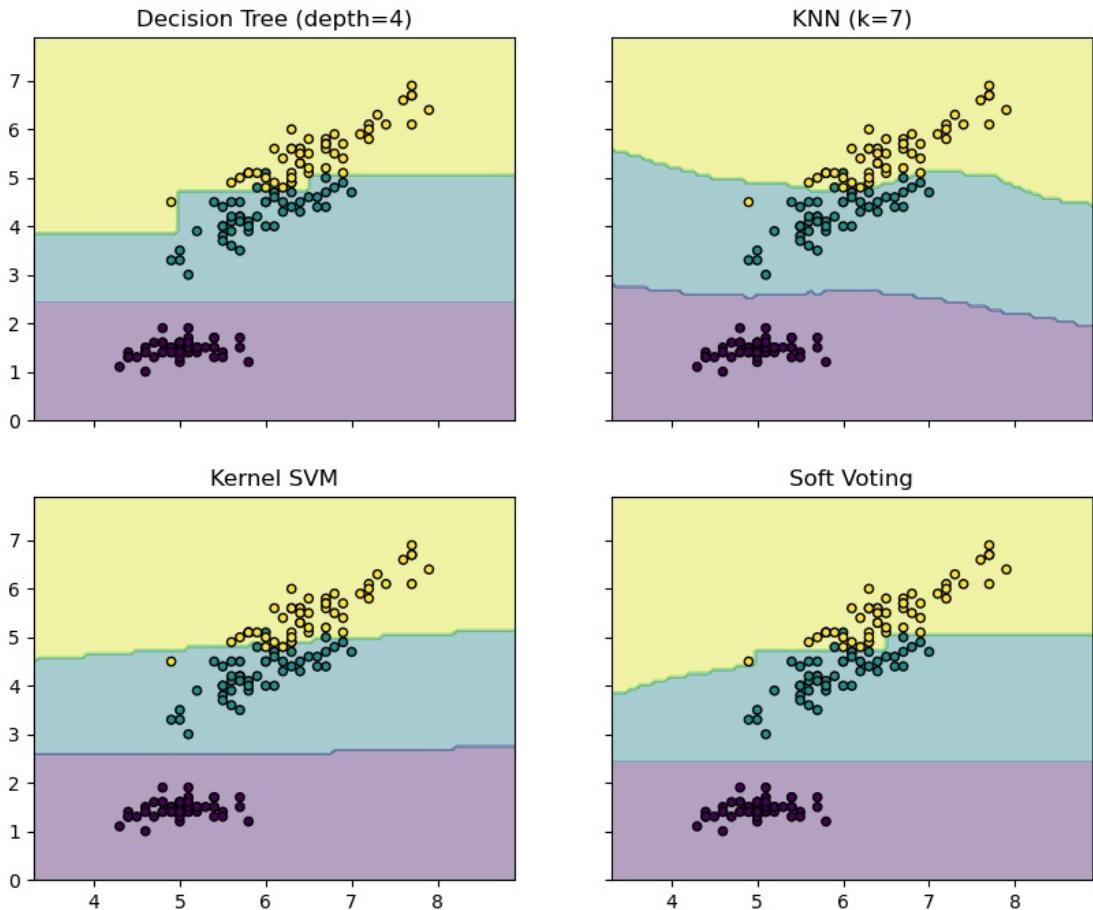
Но как выбрать коэффициенты?  
*Например, посмотреть на точность/  
степень переобучения каждого из  
алгоритмов*



# Пример голосования

В данном голосовании веса моделей

[2, 1, 2]



<https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier>

# Недостатки метода

- Не учитываются особенности каждой из моделей
- Модель получается простой



# Стекинг

Вместо голосования агрегация голосов будет происходить с помощью мета-алгоритма.



# Блендинг

При простейшей вариации стекинга данные делятся на **train**(обучение набора базовых моделей) и **test**(предсказание базовых моделей и обучение мета-алгоритма)

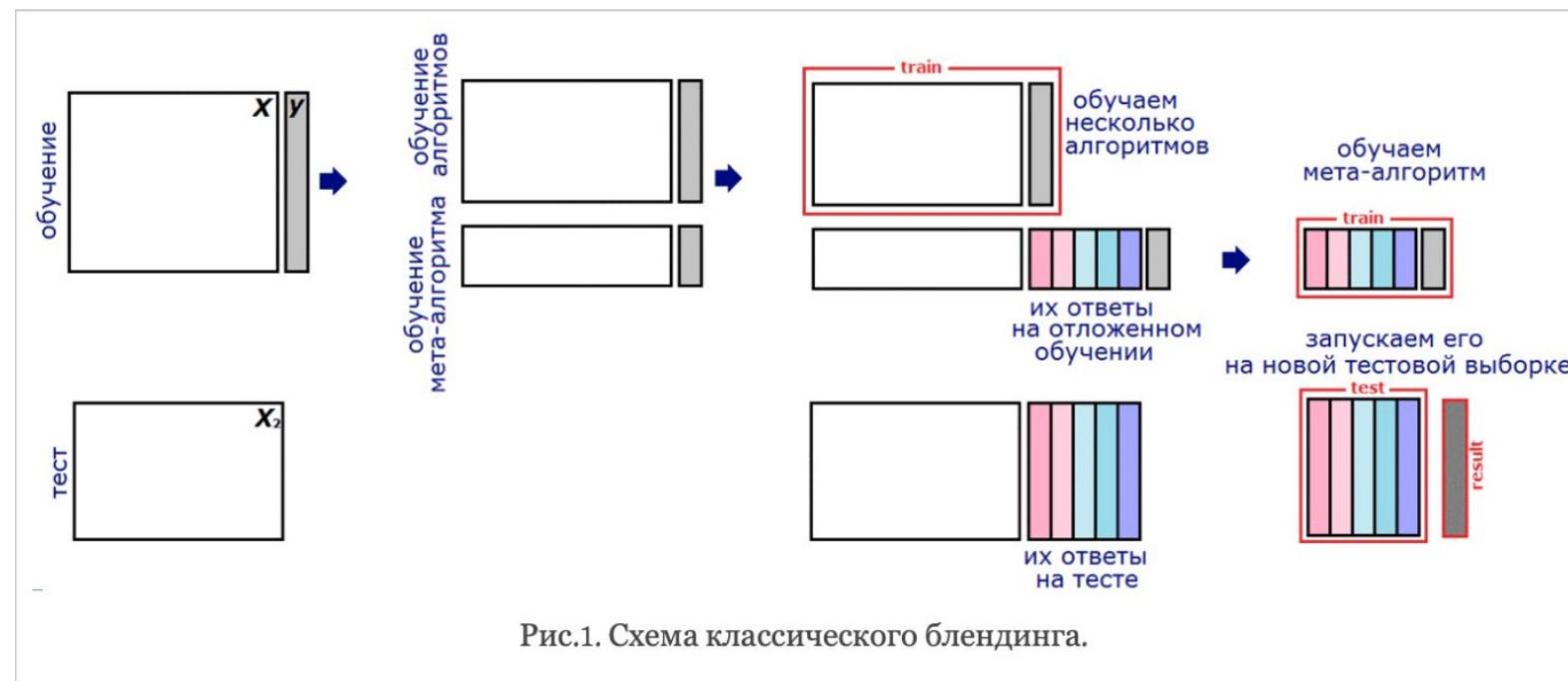


Рис.1. Схема классического блендинга.

# Блендинг

Для улучшения качества можно повторить блендинг с несколькими разбиениями

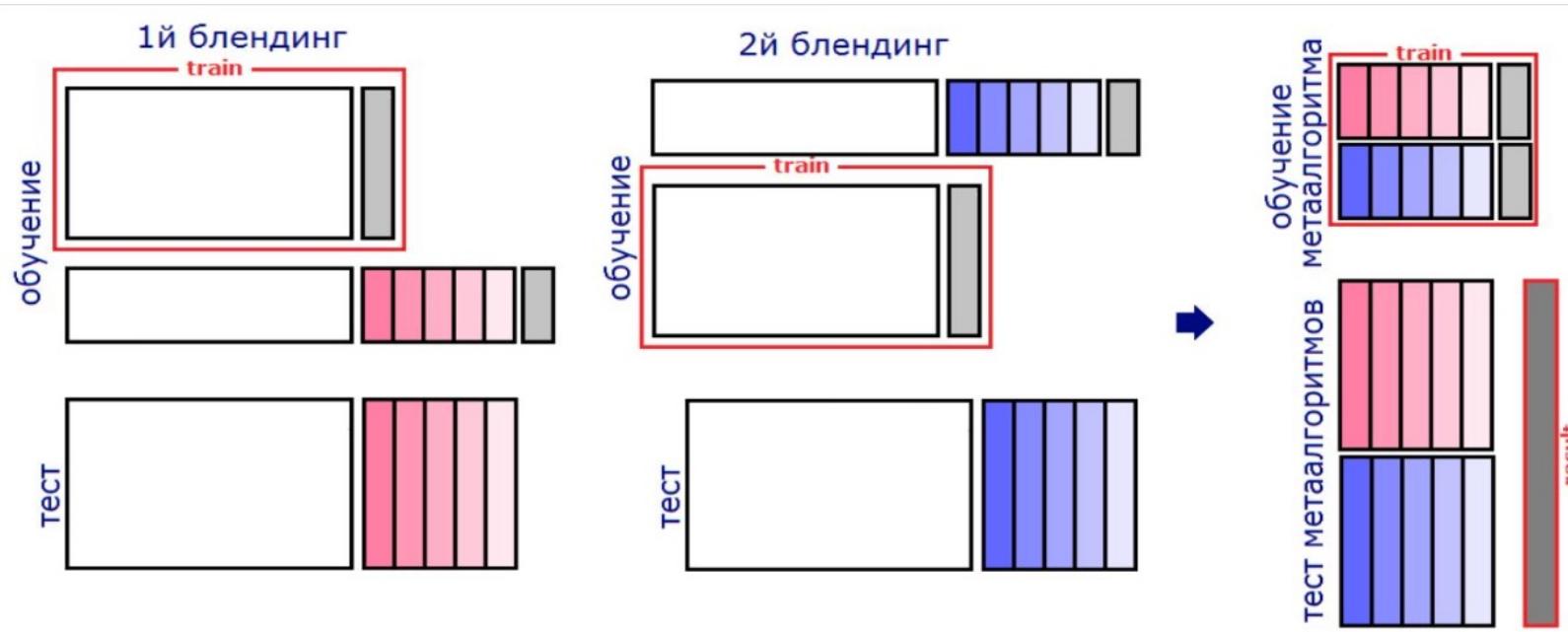


Рис.2. Возможная модификация блендинга.



# Стекинг

В более общем случае мета-модель обучается на cross-val predict-ах базовых моделей – такое предсказание будет выступать как мета-признак.

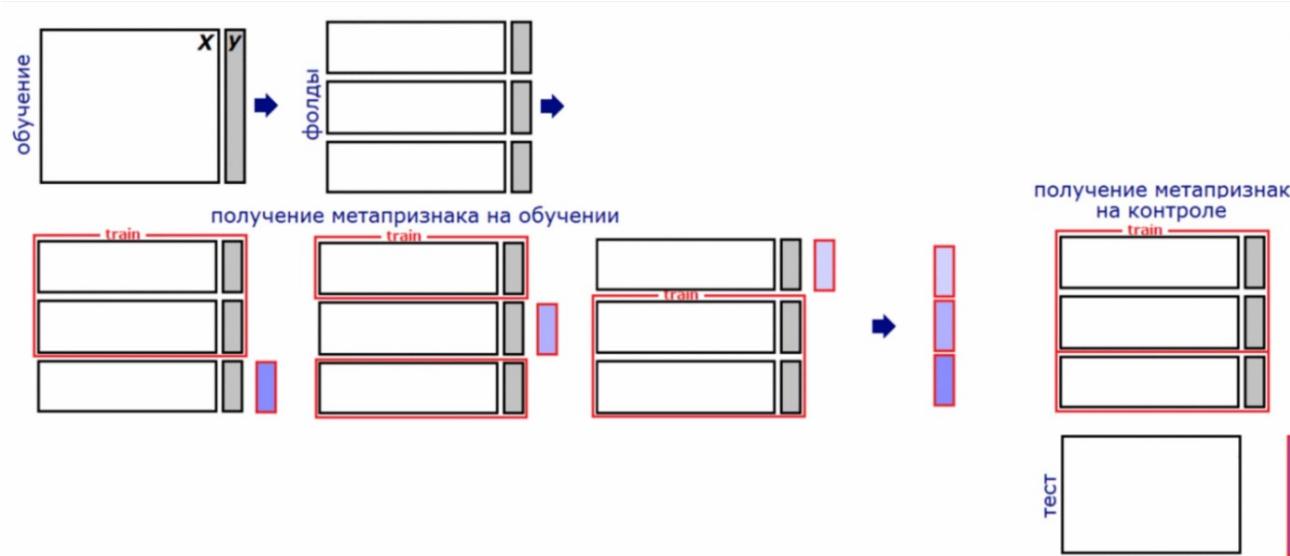


Рис.3. Получение метапризнака в классическом стекинге.



# Бэггинг и бустинг

## Бэггинг

Строим много сильных независимых моделей  
Усредняем результаты

## Бустинг

Строим каждую модель на основе предыдущей,  
предсказывая не таргет, а ее ошибку



# Бутстрэп

- метод исследования распределения статистик вероятностных распределений, основанный на многократной генерации псевдовыборок на базе имеющейся выборки.



Генерируем подвыборки из исходной методом случайного выбора с возвращением.



На псевдовыборках считаем целевую статистику



Анализируем распределение целевой статистики на псевдовыборках



# Типичная задача на собеседовании

В обучающей выборке  $n$  объектов. Из этих  $n$  объектов мы сэмплируем с возвращением  $m$  объектов. Какова доля различных объектов, которые попадают в выборку бустрэпа?



# Спойлер

примерно 63%

Это можно легко доказать: пусть в выборке  $\ell$  объектов. На каждом шаге все объекты попадают в подвыборку с возвращением равновероятно, т.е отдельный объект — с вероятностью  $\frac{1}{\ell}$ .

Вероятность того, что объект НЕ попадет в подвыборку (т.е. его не взяли  $\ell$  раз):  $(1 - \frac{1}{\ell})^\ell$ . При  $\ell \rightarrow +\infty$  получаем один из "замечательных" пределов  $\frac{1}{e}$ . Тогда вероятность попадания конкретного объекта в подвыборку  $\approx 1 - \frac{1}{e} \approx 63\%$ .





## Метод случайных подпространств / feature bagging

- Из Train генерим методом случайного выбора признаков без возвращения  $\text{Train}'_1 \dots \text{Train}'_N$
- На каждом  $\text{Train}'$  строим модель
- Итоговое предсказание получаем усреднением предсказаний всех моделей



# Наконец, бэггинг

- Bootstrap aggregation (Leo Breiman, 1994)

1

Генерируем N подвыборок из исходной методом бутстрэпа.

2

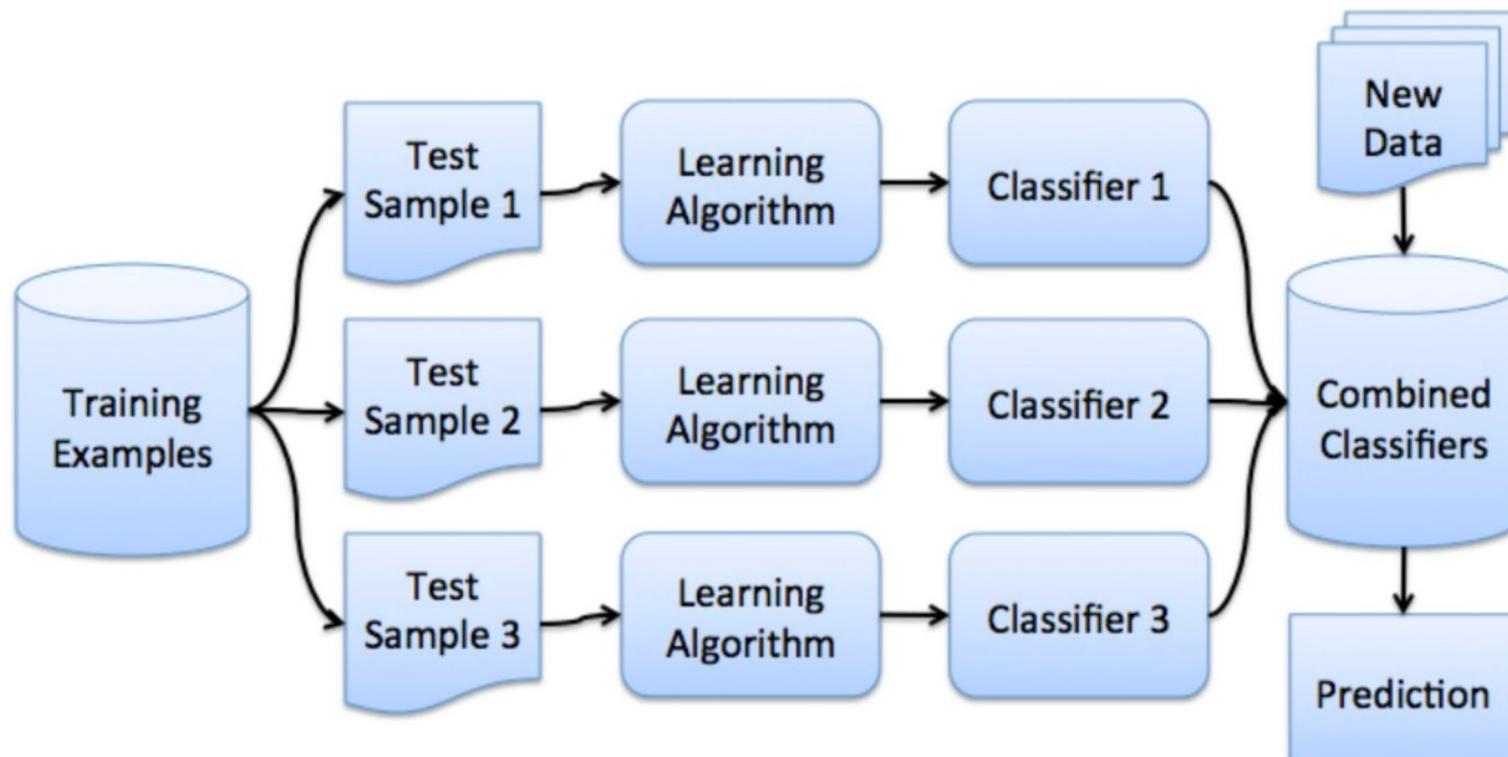
На каждой подвыборке строим модель

3

Итоговое предсказание получаем усреднением/голосованием по предсказаниям всех моделей



# Схема бэггинга



# БЭГГИНГ

Пусть

$$a(x) = \frac{1}{k}(b_1(x) + \cdots + b_k(x)).$$

Покажем, что бэггинг не влияет на bias.

$$\begin{aligned}\text{bias}_X a(x, X) &= f(x) - \mathbb{E}_X[a(x, X)] = f(x) - \mathbb{E}_X \left[ \frac{1}{k} \sum_{i=1}^k b(x, X^i) \right] = \\ &= f(x) - \frac{1}{k} \sum_{i=1}^k \mathbb{E}_X [b(x, X^i)] = f(x) - \frac{1}{k} \sum_{i=1}^k \mathbb{E}_X [b(x, X)] = f(x) - \mathbb{E}_X b(x, X) \\ &= f(x) - \mathbb{E}_X b(x, X) = \text{bias}_X b(x, X)\end{aligned}$$



# БЭГГИНГ

Теперь покажем, что такой метод уменьшает variance.

$$\begin{aligned}\mathbb{V}_X[a(x, X)] &= \mathbb{E}_X[a(x, X) - \mathbb{E}_X[a(x, X)]]^2 = \\ &= \mathbb{E}_X \left[ \frac{1}{k} \sum_{i=1}^k b(x, X^i) - \mathbb{E}_X \left[ \frac{1}{k} \sum_{i=1}^k b(x, X^i) \right] \right]^2 = \\ &= \frac{1}{k^2} \mathbb{E}_X \left[ \sum_{i=1}^k (b(x, X^i) - \mathbb{E}_X b(x, X^i))^2 \right] = \\ &= \frac{1}{k^2} \sum_{i=1}^k \mathbb{E}_X (b(x, X^i) - \mathbb{E}_X b(x, X^i))^2 + \\ &\quad + \frac{1}{k^2} \sum_{k_1 \neq k_2} \mathbb{E}_X [(b(x, X^{k_1}) - \mathbb{E}_X b(x, X^{k_1})) (b(x, X^{k_2}) - \mathbb{E}_X b(x, X^{k_2}))] = \\ &= \frac{1}{k^2} \sum_{i=1}^k \mathbb{V}_X b(x, X^i) + \frac{1}{k^2} \sum_{k_1 \neq k_2} \text{cov}(b(x, X^{k_1}), b(x, X^{k_2}))\end{aligned}$$



# Случайный лес

- *Бэггинг над решающими деревьями*

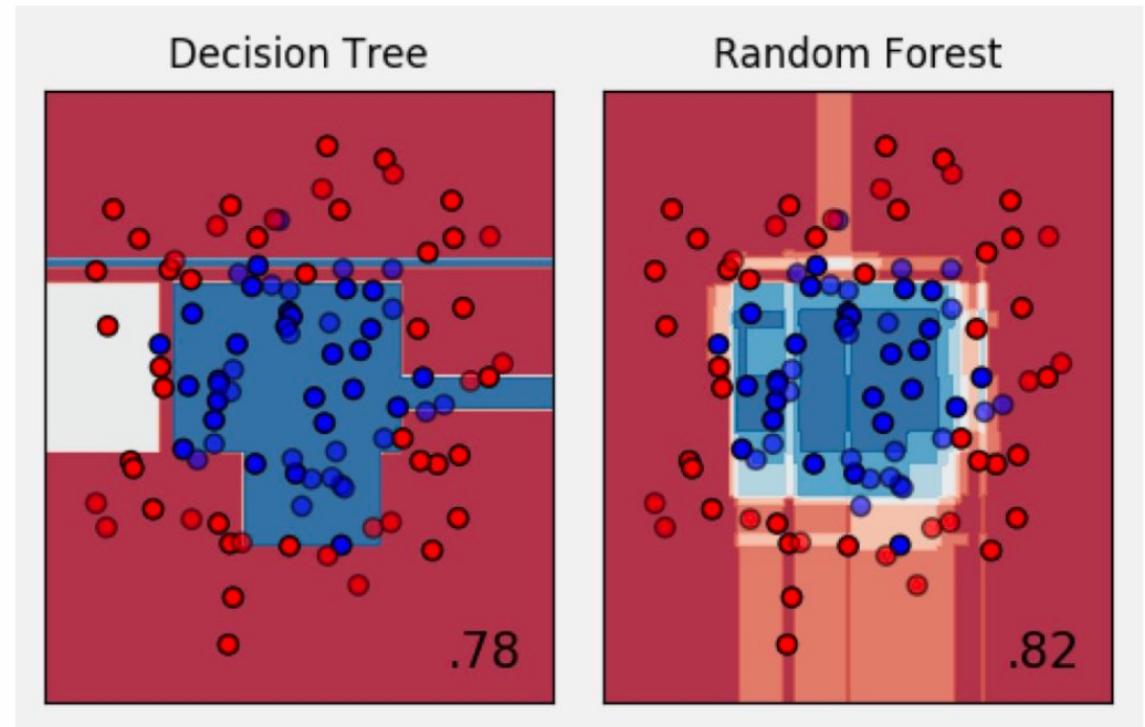
Алгоритм.

- Берем подмножество выборки
- Берем подмножество признаков
- Строим много переобученных моделей
- Усредняем/голосуем



# Случайный лес

- Получаем лучшее качество, чем у единичного дерева
  - Глубиной деревьев контролируем смещение
  - Количество деревьев регулируем дисперсию
- Помним про корреляцию: для уменьшения дисперсии должны быть как можно более различными



# Случайный лес в sklearn

**RandomForestRegressor(*n\_estimators*, *criterion*, *max\_depth*, *min\_samples\_split*,  
*min\_samples\_leaf*, *min\_weight\_fraction\_leaf*, *max\_features*, *max\_leaf\_nodes*, *min\_im\_purity\_decrease*, *bootstrap*, *oob\_score*, *n\_jobs*, *random\_state*, *verbose*, *warm\_start*,  
*ccp\_alpha*, *max\_samples*)**

- Параметры функции потерь
- Параметры ансамбля
- Параметры дерева
- Параметры технические

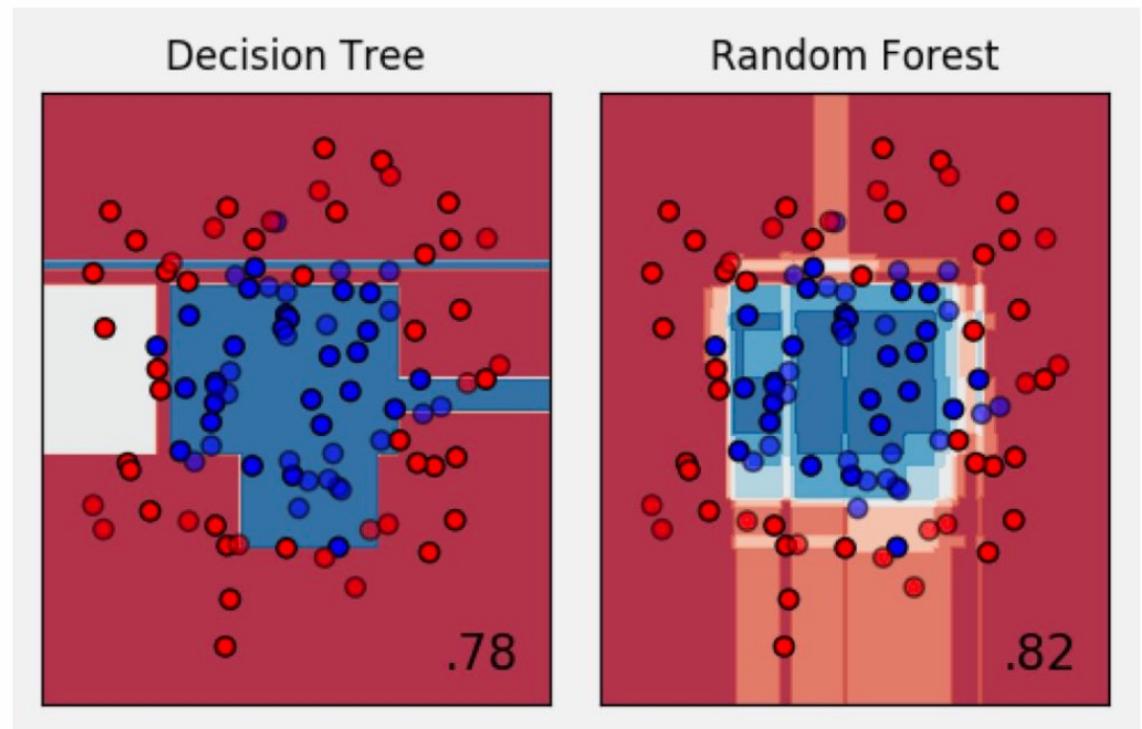


[Ссылка на класс](#)

# Оценка важности признаков

Чем выше в среднем признак в дереве решений, тем он важнее.

Можно отдельно посчитать [permutation Importance](#): насколько ухудшится модель если случайно перемешать признак (работает очень долго).



# Случайный лес

Плюсы:

- Алгоритм прост
- Нет чувствительности к выбросам
- Не переобучается
- Хорошо параллелизуется
- Не требует сложной настройки параметров
- Не требует нормализации данных



# Случайный лес

Минусы:

- Модели не интерпретируемы
- Работает хуже линейных моделей, когда есть разреженные признаки
- Большой размер получающихся моделей, как следствие долгое предсказание



# Случайный лес

Минусы:

- Модели не интерпретируемы
- Работает хуже линейных моделей, когда есть разреженные признаки
- Большой размер получающихся моделей, как следствие долгое предсказание



# Градиентный бустинг

Идея:

1. Представляем итоговую модель  $f(x)$  как сумму слабых моделей  $h(x)$  (обычно решающие деревья малой глубины).
2. Пусть задана дифференцируемая функция потерь  $L(y, f(x))$
3. На каждом шаге мы ищем модель  $h(x)$ , которая бы аппроксимировала вектор антиградиента  $L$



# Градиентный бустинг

1. Инициализировать GBM константным значением  $\hat{f}(x) = \hat{f}_0$ ,  $\hat{f}_0 = \gamma$ ,  $\gamma \in \mathbb{R}$

$$\hat{f}_0 = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

2. Для каждой итерации  $t = 1, \dots, M$  повторять:

1. Посчитать псевдо-остатки  $r_t$

$$r_{it} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x)=\hat{f}(x)}, \quad \text{for } i = 1, \dots, n$$

2. Построить новый базовый алгоритм  $h_t(x)$  как регрессию на псевдо-остатках

$$\{(x_i, r_{it})\}_{i=1, \dots, n}$$

3. Найти оптимальный коэффициент  $\rho_t$  при  $h_t(x)$  относительно исходной функции потерь

$$\rho_t = \arg \min_{\rho} \sum_{i=1}^n L(y_i, \hat{f}(x_i) + \rho \cdot h_t(x_i, \theta))$$

4. Сохранить  $\hat{f}_t(x) = \rho_t \cdot h_t(x)$

5. Обновить текущее приближение  $\hat{f}(x)$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{f}_t(x) = \sum_{i=0}^t \hat{f}_i(x)$$

3. Скомпоновать итоговую GBM модель  $\hat{f}(x)$

$$\hat{f}(x) = \sum_{i=0}^M \hat{f}_i(x)$$



# Градиентный бустинг

Построили алгоритм  $a(x)$ , построим алгоритм  $b(x)$  такой, что

$$a(x_i) + b(x_i) = y_i, \quad i \in \{1, 2, \dots, m\},$$

Алгоритм  $b(x)$  поправляет ошибки алгоритма  $a(x)$  на невязку:  $\varepsilon_i = y_i - a(x_i)$



# Градиентный бустинг

$$F = \sum_{i=1}^m L(y_i, a(x_i) + b_i) \rightarrow \min_{(b_1, \dots, b_m)}$$

Функция многих переменных максимально убывает



# Градиентный бустинг

$$F(b_1, \dots, b_m) = F = \sum_{i=1}^m L(y_i, a(x_i) + b_i) \rightarrow \min_{(b_1, \dots, b_m)}$$

Функция многих переменных максимально убывает в направлении своего антиградиента:

$$-(L'(y_1, a(x_1)), \dots, L'(y_m, a(x_m)))$$

102



# Библиотечный градиентный бустинг

**GradientBoostingClassifier**(loss, learning\_rate, n\_estimators, subsample, criterion, min\_samples\_split, min\_samples\_leaf, min\_weight\_fraction\_leaf, max\_depth, min\_imprurity\_decrease, init, random\_state, max\_features, verbose, max\_leaf\_nodes, warm\_start, validation\_fraction, n\_iter\_no\_change, tol, ccp\_alpha)

- Параметры функции потерь
- Параметры ансамбля
- Параметры дерева
- Параметры технические



# Недостатки бустинга

- **Проблема переобучения градиентного бустинга**

По мере увеличения числа деревьев ошибка на обучающей выборке постепенно уходит в 0. Ошибка на контрольной выборке существенно больше ошибки на обучающей выборке, достигает минимума примерно на 10 итерации, а затем начинает опять возрастать. Имеет место переобучение.

- **Сокращение размера шага**

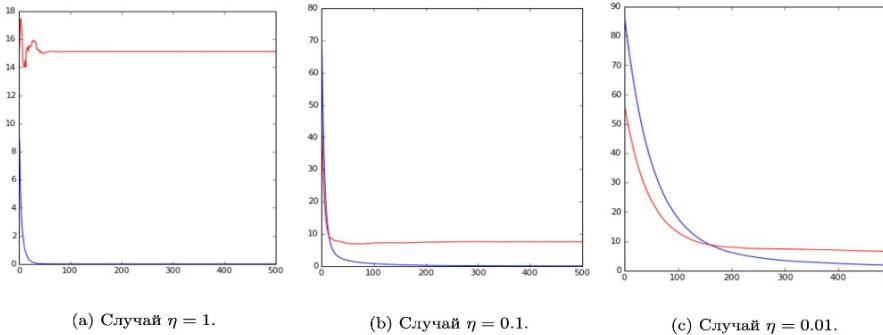
Чтобы решить эту проблему, нужно «не доверять» направлению, которое построил базовый алгоритм и лишь чуть-чуть смещаться в сторону этого вектора:

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

где  $\eta \in (0, 1]$  — длина шага. Это обеспечивает очень аккуратное движение в пространстве, что делает возможным нахождение локального минимума.



# Недостатки бустинга



Как видно по графикам, при  $\eta = 0.1$  качество на контрольной выборке уже существенно лучше, то есть в некотором смысле удалось побороть переобучение. При еще меньшей длине шага  $\eta = 0.01$  градиентному бустингу требуется существенно больше итераций, чтобы достичь чуть-чуть большего качества. Таким образом:

- Чем меньше размер шага, тем больше нужно базовых алгоритмов, чтобы достичь хорошего качества, и тем больше времени занимает процесс.
- Чем меньше размер шага, тем лучшего качества можно достичь.

Другими словами, приходится выбирать: или быстро получить достаточно хорошее качество, или получить качество чуть-чуть лучше за большее время.



# Недостатки бустинга

- Идея бустинга обычно плохо применима к построению композиции из достаточно сложных и мощных алгоритмов. Построение такой композиции занимает очень много времени, а качество существенно не увеличивается.
- Результаты работы бустинга сложно интерпретируемы, особенно если в композицию входят десятки алгоритмов.



# Преимущества бустинга

- Хорошая обобщающая способность. В реальных задачах (не всегда, но часто) удаётся строить композиции, превосходящие по качеству базовые алгоритмы. Обобщающая способность может улучшаться (в некоторых задачах) по мере увеличения числа базовых алгоритмов.
- Простота реализации.
- Собственные накладные расходы бустинга невелики. Время построения композиции практически полностью определяется временем обучения базовых алгоритмов.
- Возможность идентифицировать объекты, являющиеся шумовыми выбросами.



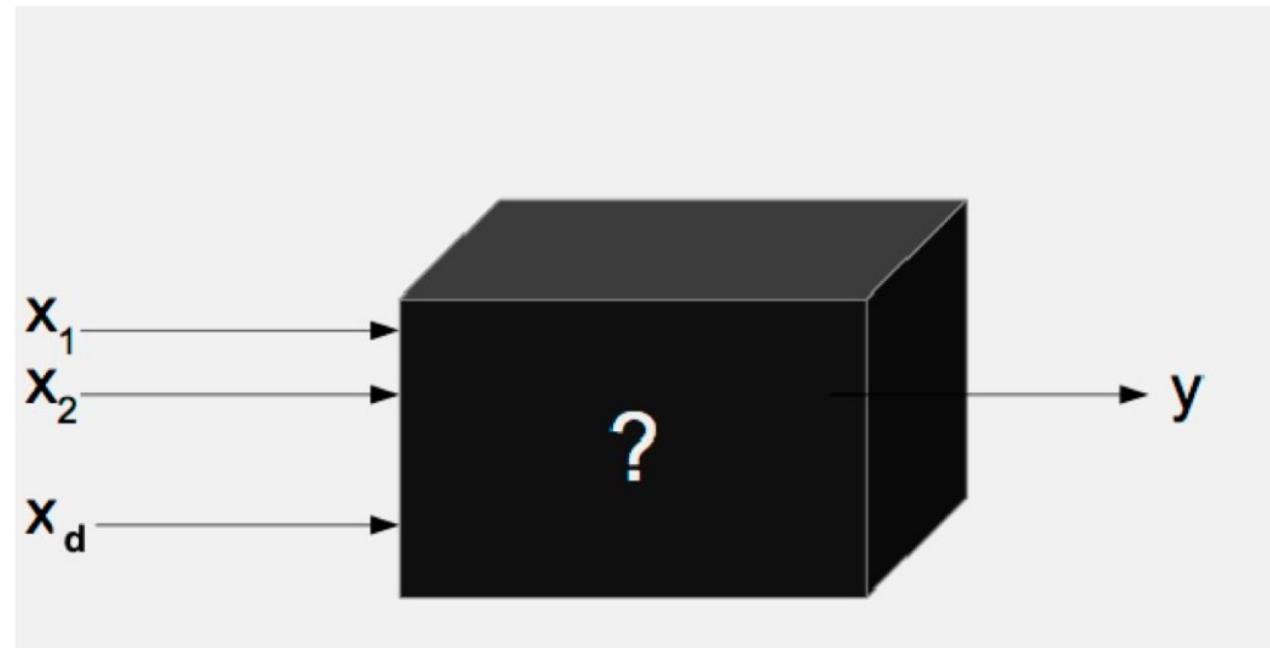
# Открытые реализации градиентного бустинга

*dmlc*  
**XGBoost**



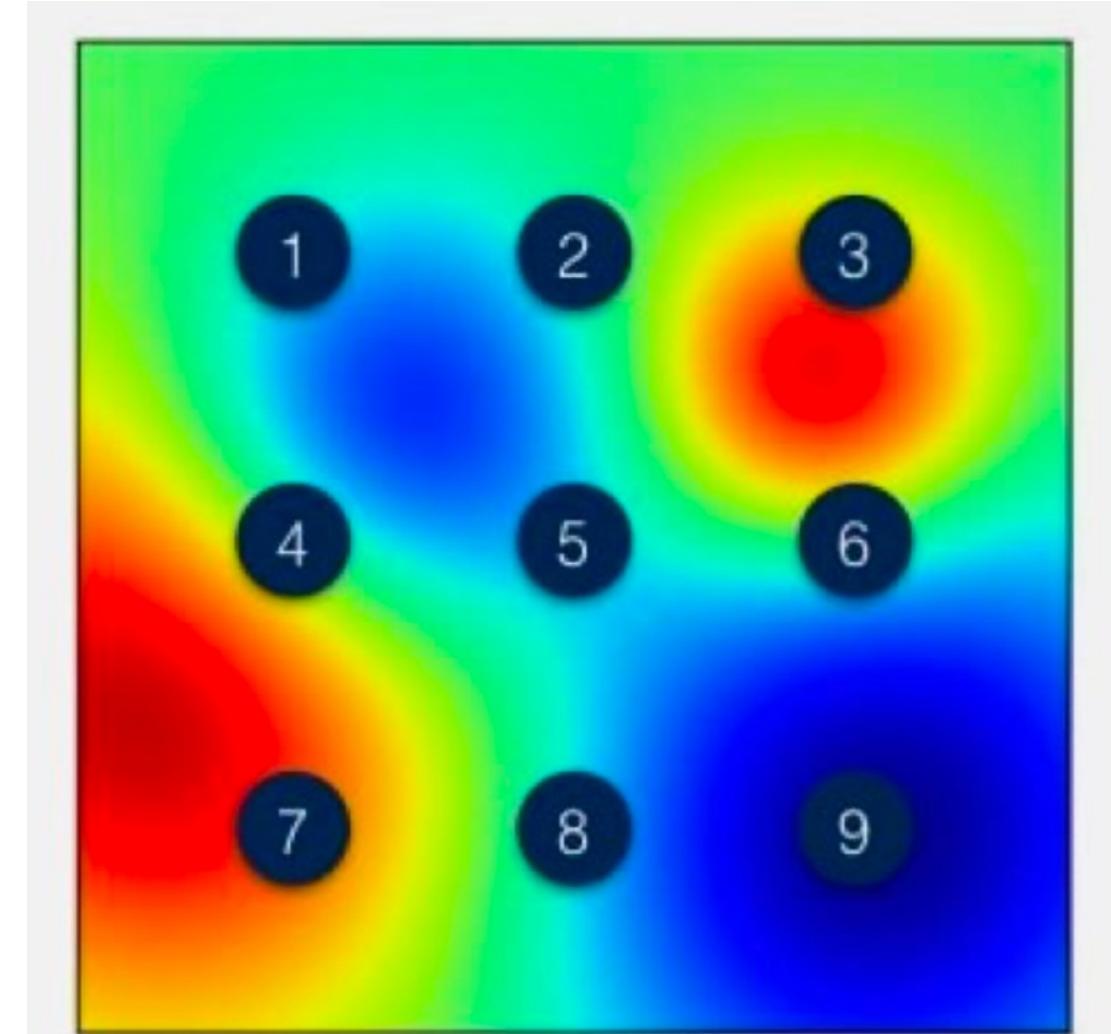
# Автоматический подбор гиперпараметров

Было бы здорово не менять руками гиперпараметры при каждом построении модели, а запустить сразу большую задачу, которая выдаст лучший алгоритм.



# GridSearch

- Перебираем параметры по решетке
- Запоминаем параметры, дающие лучшее качество

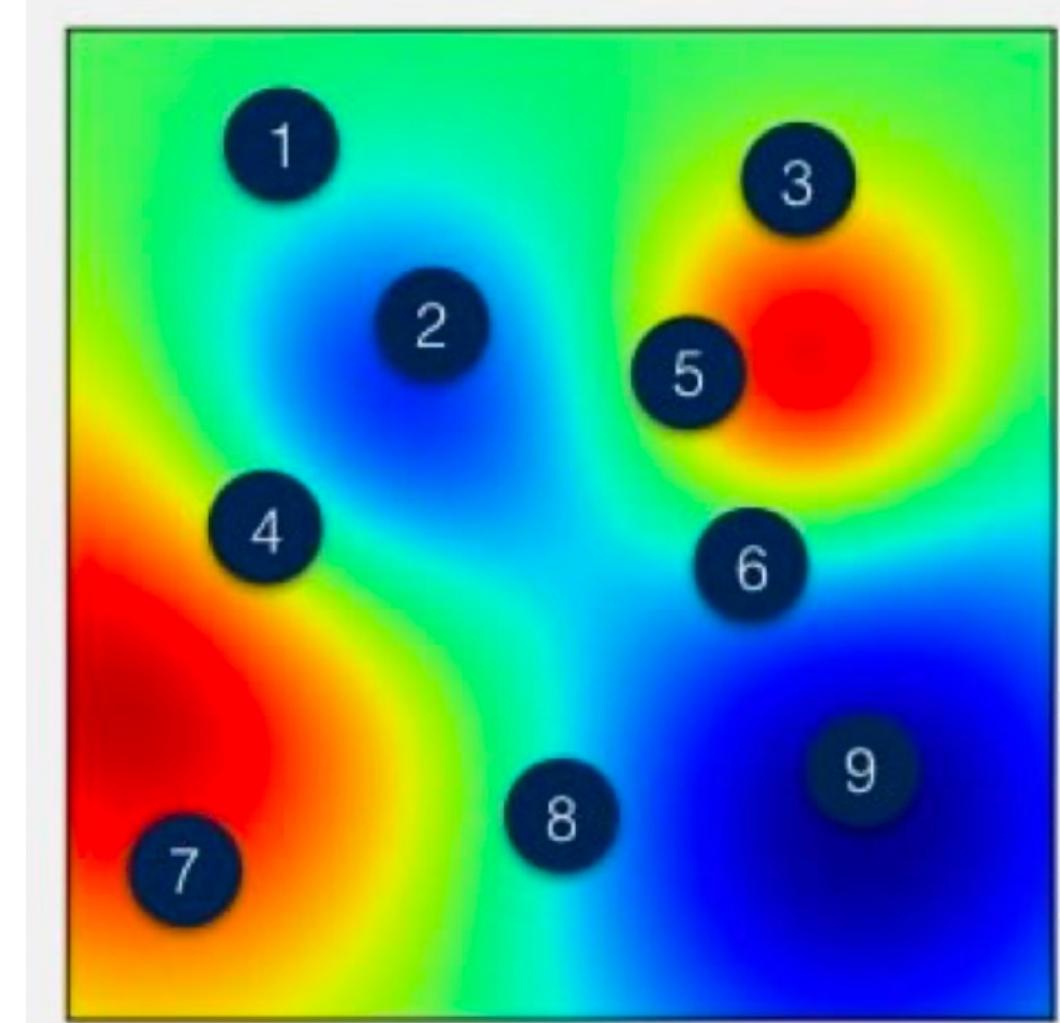


Grid Search



# RandomSearch

- Сэмплируем параметры модели
- Запоминаем параметры, дающие лучшее качество



Random Search



# Инструменты для оптимизации гиперпараметров

- sklearn
- Hyperopt
- BayesianOptimization
- Hyperparameter
- Optuna
- Optunity



# Слайд для ваших вопросов

THANK YOU FOR YOUR ATTENTION,  
YOU CAN CLAP NOW

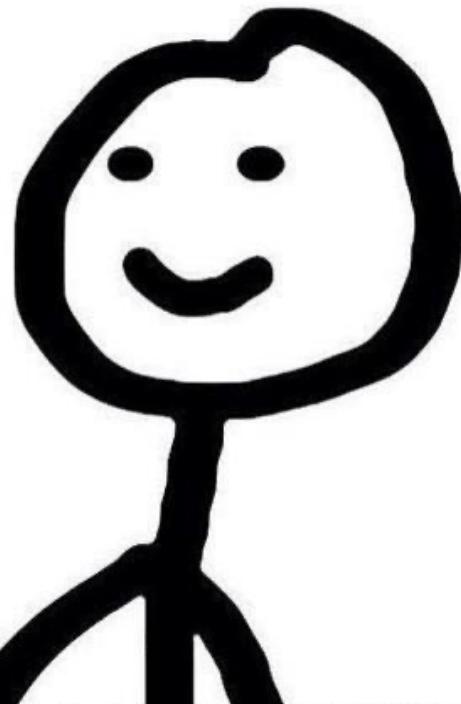


IF YOU HAVE ANY QUESTIONS, PLEASE  
ASK MY FRIEND GOOGLE

[makeameme.org](http://makeameme.org)



# Слайд для вашего отзыва :)



Поставьте ~~хорошую~~  
оценку, пожалуйста

