

МЕГАФОН



# АНСАМБЛИ

ВОЛОДКИНА ЕКАТЕРИНА

# Методы



- Bagging
- Random Forest
- Boosting
- Blending
- Stacking



<https://www.youtube.com/watch?v=5tkMi72w8j0>

# Bais Variance Decomposition



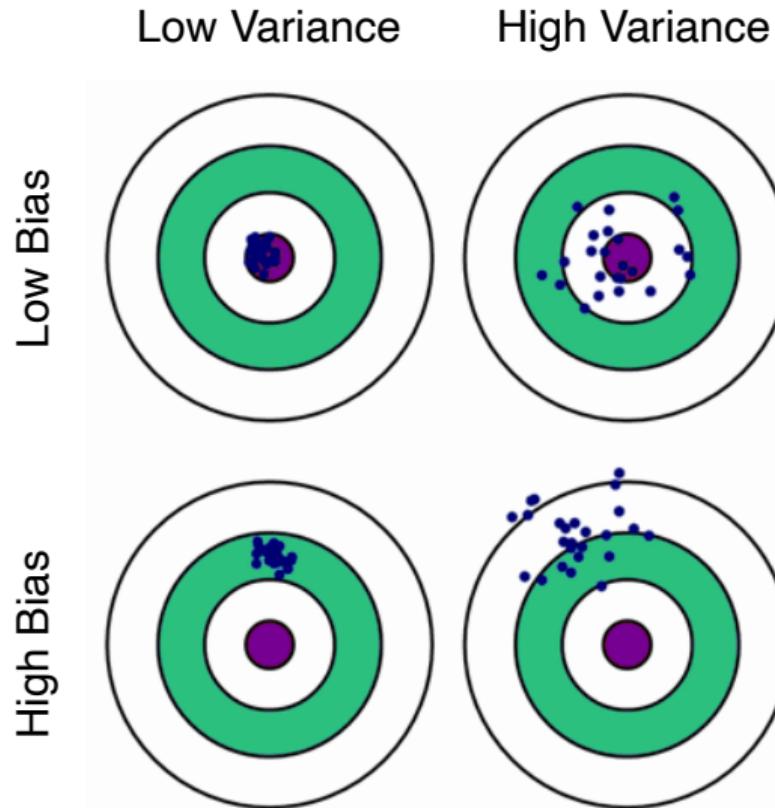
$$L(\mu) = \underbrace{\mathbb{E}_{x,y} \left[ (y - \mathbb{E}[y | x])^2 \right]}_{\text{шум}} + \underbrace{\mathbb{E}_x \left[ (\mathbb{E}_X [\mu(X)] - \mathbb{E}[y | x])^2 \right]}_{\text{смещение}} + \underbrace{\mathbb{E}_x \left[ \mathbb{E}_X \left[ (\mu(X) - \mathbb{E}_X [\mu(X)])^2 \right] \right]}_{\text{разброс}}.$$

ошибка  
идеального  
алгоритма

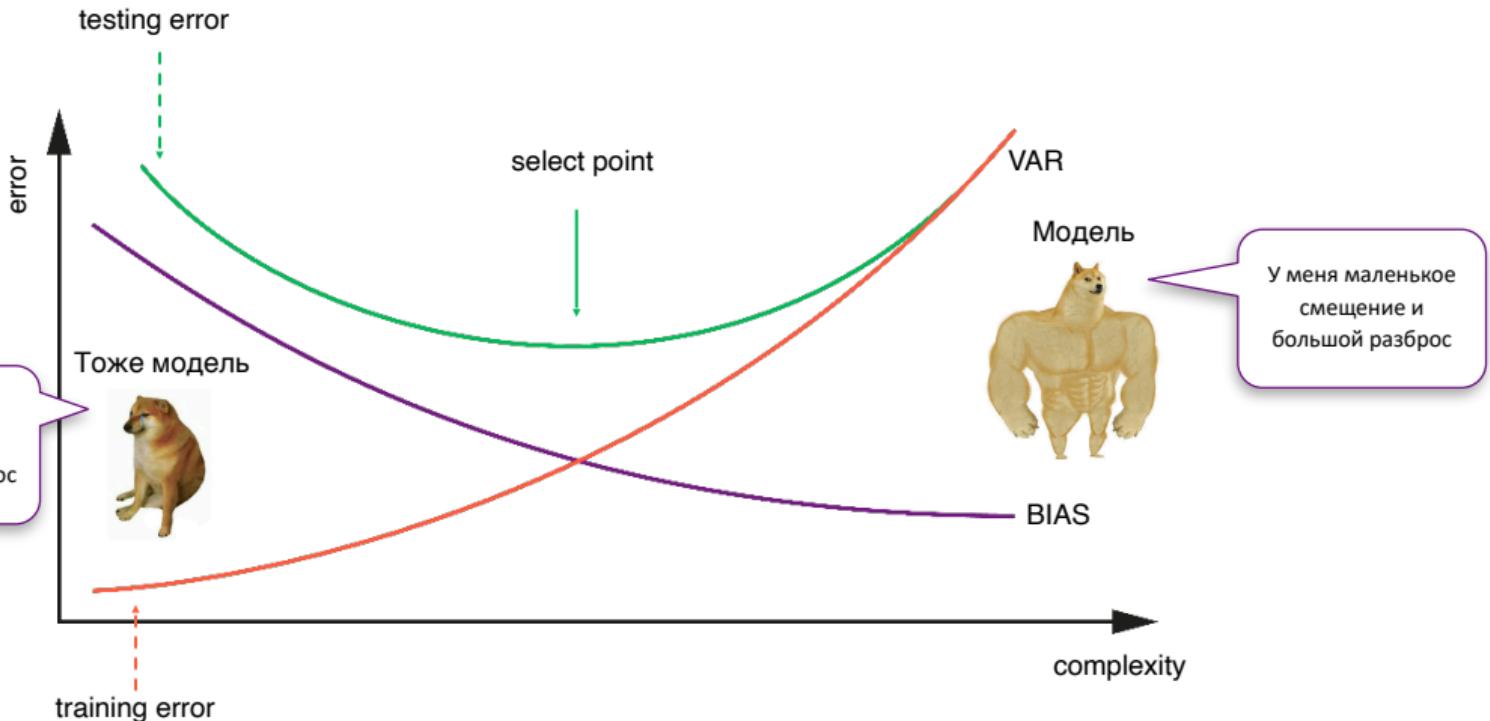
отклонение среднего  
ответа обученного  
алгоритма от ответа  
идеального алгоритма

разброс ответов  
обученных алгоритмов  
относительно среднего  
ответа на выборке

# Мишени



# Связь с (недо/пере)обучением



# Простейший пример композиции 1



$$a(x) = \frac{1}{N} \sum_{j=1}^N b_j(x)$$

$$\mathbb{E}_x (b_j(x) - y(x))^2 = \mathbb{E}_x \varepsilon_j^2(x)$$

$$E_1 = \frac{1}{N} \sum_{j=1}^N \mathbb{E}_x \varepsilon_j^2(x)$$

$$\begin{aligned}\mathbb{E}_x \varepsilon_j(x) &= 0 \\ \mathbb{E}_x \varepsilon_i(x) \varepsilon_j(x) &= 0, \quad i \neq j\end{aligned}$$

Предположим, что ошибки несмещены и некоррелированы



## Простейший пример композиции 2

$$\begin{aligned} E_N &= \mathbb{E}_x \left( \frac{1}{N} \sum_{j=1}^n b_j(x) - y(x) \right)^2 = \\ &= \mathbb{E}_x \left( \frac{1}{N} \sum_{j=1}^N \varepsilon_j(x) \right)^2 = \\ &= \frac{1}{N^2} \mathbb{E}_x \left( \sum_{j=1}^N \varepsilon_j^2(x) + \underbrace{\sum_{i \neq j} \varepsilon_i(x) \varepsilon_j(x)}_{=0} \right) = \\ &= \frac{1}{N} E_1 \end{aligned}$$

Усреднение ответов набора базовых алгоритмов уменьшило средний квадрат ошибки в N раз!

# Bagging



Bagging = Bootstrap\* + Aggregation

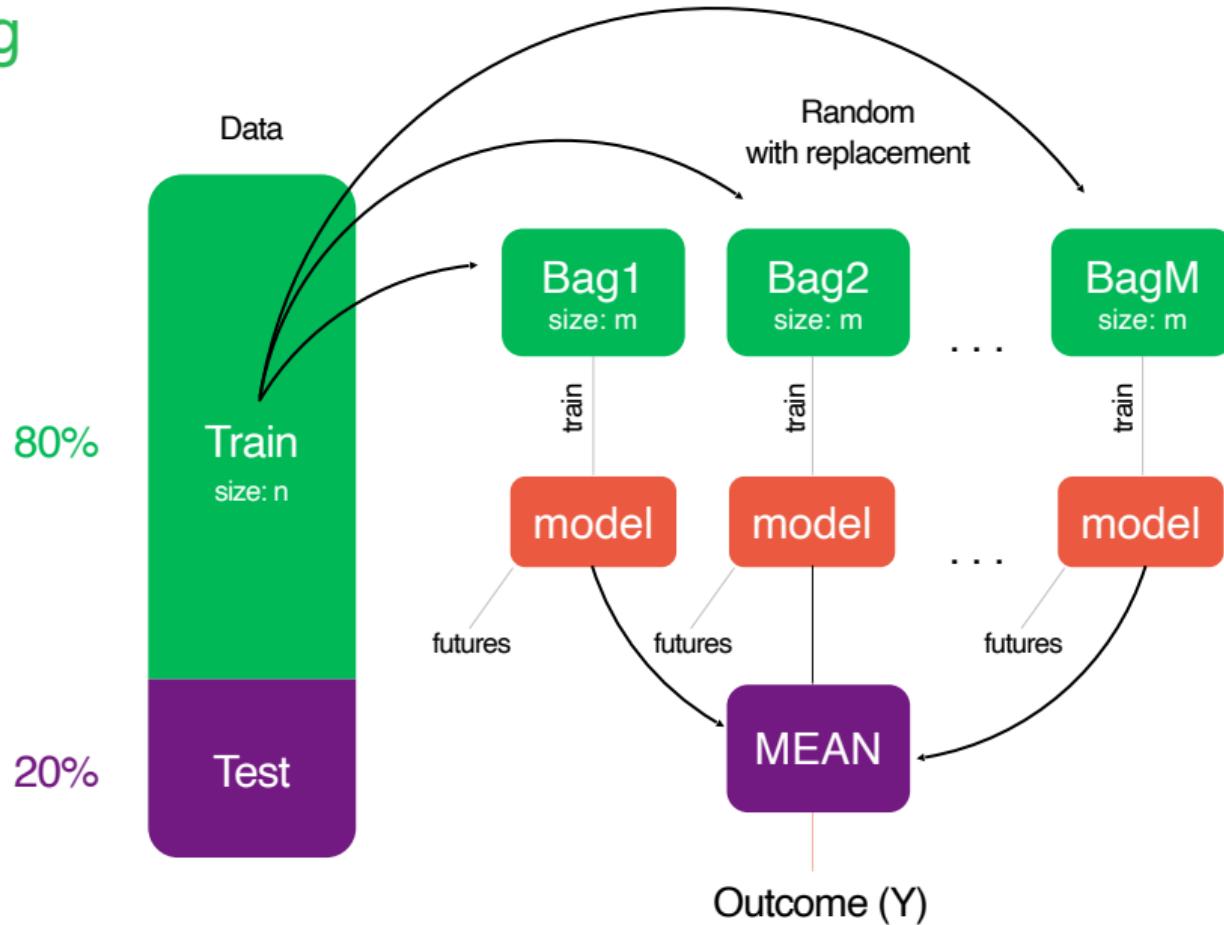
\*Bootstrap - сэмплирование объектов из выборки с возвращением

$$a(x) = \frac{1}{N} \sum_{j=1}^N b_j(x) = \frac{1}{N} \sum_{j=1}^N h(X_i)(x)$$

$X_i$  - получили бутстррапом



# Bagging



# Bagging



- смещение композиции, полученной с помощью бэггинга, совпадает со смещением одного базового алгоритма
- если базовые алгоритмы некоррелированы, то дисперсия композиции в  $N$  раз меньше дисперсии отдельных алгоритмов

Нужны слабо скоррелированные алгоритмы с маленьким смещением, при этом разброс мы уменьшим за счет bagging



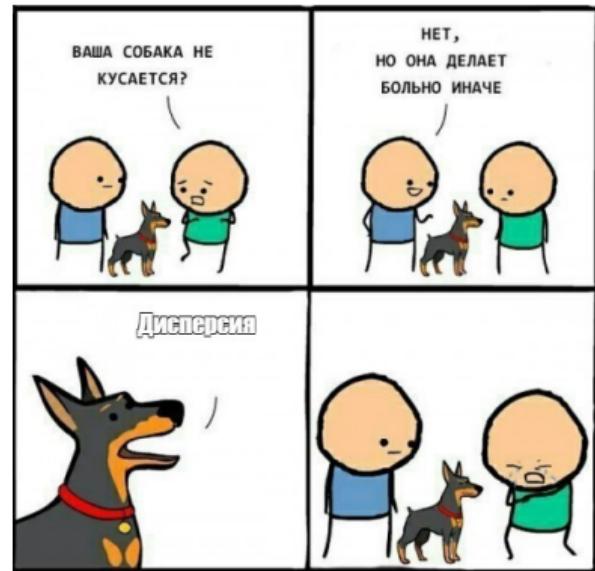
# Random Forest



Решающие деревья:

- достаточно сложны и могут достигать нулевой ошибки на любой выборке (имеют низкое смещение)
- легко переобучаются

Снизим дисперсию за счет bagging и корреляцию алгоритмов за счет рандомного семплирования не только объектов (сам bagging), но и признаков!





## Случайный лес - bagging над решающими деревьями

N раз:

1. Сгенерировать выборку X с помощью бутсрапа
2. Выбрать m случайных признаков
3. Обучить классификатор

Ответом является композиция  $a_N(x) = \frac{1}{N} \sum_{n=1}^N b_n(x)$

# Связь понятий



**Bootstrap** - сэмплирование объектов из выборки с возвращением

**Bagging = Bootstrap + Aggregation** - агрегирование ответов алгоритмов, обученных на бустрапированных выборках

**Random Forest = Bagging trees + sampling of features** -  
рандомизация и на уровне объектов, и на уровне признаков  
(чтобы алгоритмы не коррелировали)



«Может ли набор слабых  
обучающих алгоритмов  
создать сильный  
обучающий алгоритм?»



У нас большое  
смещение и  
маленький разброс



Кернс и Вэлиант



## Идея:

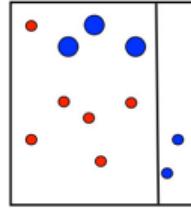
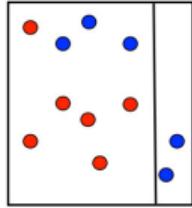
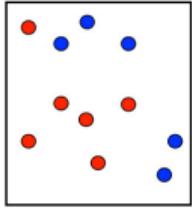
итеративно добавляем слабые  алгоритмы,  
наращивая наш ансамбль постепенными улучшениями  
тех участков данных, где предыдущие модели больше  
ошибались, тем самым **уменьшая смещение**



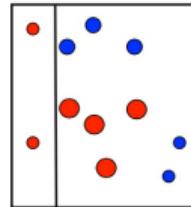
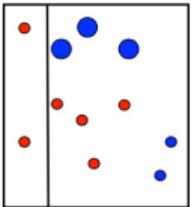
Подход заключается в жадном построении линейной комбинации простых моделей (базовых алгоритмов) путем перевзвешивания входных данных.

Каждая последующая модель (как правило, дерево решений) строится таким образом, чтобы **придавать больший вес** и предпочтение ранее некорректно предсказанным наблюдениям.

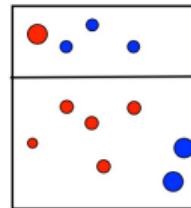
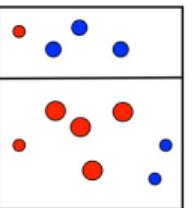
# AdaBoost



$t = 1$



$t = 2$

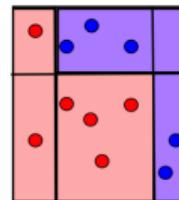


$t = 3$



$$\alpha_1 \begin{array}{|c|c|}\hline \text{Red} & \text{Purple} \\ \hline \end{array} + \alpha_2 \begin{array}{|c|c|}\hline \text{Red} & \text{Purple} \\ \hline \end{array} + \alpha_3 \begin{array}{|c|c|}\hline \text{Purple} & \text{Red} \\ \hline \end{array}$$

=



# Gradient Boosting (Machine)



Рассмотрим задачу регрессии с оптимизацией MSE:

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

Будем искать итоговый алгоритм как сумму слабых алгоритмов



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

Построим первый базовый алгоритм:

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



# Gradient Boosting (Machine)

Остатки после первой итерации:  $s_i^{(1)} = y_i - b_1(x_i)$

Если приближать ответы нового алгоритма к остаткам, то алгоритм не будет допускать ошибок на обучающей выборке:

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

Каждый следующий алгоритм будем настраивать на остатки предыдущего:



$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i), \quad i = 1, \dots, \ell;$$

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

# Gradient Boosting (Machine)



Заметим, что для MSE, остатки могут быть найдены как **антиградиент функции потерь**:

$$s_i^{(N)} = y_i - a_{N-1}(x_i) = -\frac{\partial}{\partial z} \frac{1}{2} (z - y_i)^2 \Big|_{z=a_{N-1}(x_i)}$$

**Но** подгон под остатки никак не учитывает особенности функции потерь.

**Идея:** настраивать новый базовый алгоритм на антиградиент функции потерь. Тогда добавляя такой алгоритм в сумму, мы будем сдвигаться в сторону скорейшего убывания функции потерь.

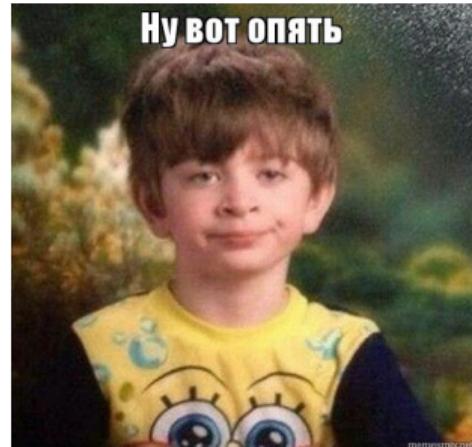
# Gradient Boosting (Machine)



На что это похоже? **На градиентный спуск!**  
В  $\ell$ -мерном пространстве предсказаний!

По аналогии с learning rate в градиентном спуске можно взвешивать каждый базовый алгоритм:

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





# Gradient Boosting (Machine)

- Инициализация композиции оптимальным константным значением

$$f_0 = \arg \min_{c \in \mathbb{R}} \sum_{i=1}^n \mathcal{L}(c, y_i)$$

- Для всех  $t = 1, \dots, T$ :

- Вычислить остатки предыдущей композиции:

$$g_i^t = - \left[ \frac{\partial \mathcal{L}(f_t, x_i, y_i)}{\partial f_t(x_i)} \right]_{i=1}^N$$

- Настроить базовую регрессию  $b_t(x)$  на полученные остатки, т.е. обучить его по выборке  $\{(x_i, g_i^t), i = 1, \dots, n\}$ . (МНК)
- Вычислить коэффициент  $\alpha_t$  перед базовым алгоритмом  $b_t(x)$  как решение следующей задачи оптимизации:

$$\alpha_t = \arg \min_{\alpha} \sum_{i=1}^N \mathcal{L}(f_{t-1}(x_i) + \alpha b_t(x_i), y_i)$$

Решение находится из уравнения

$$\frac{\partial Q_t(f_t, X, y)}{\partial \alpha_t} = \sum_{i=1}^N \frac{\partial \mathcal{L}(f_t(x_i), y_i)}{\partial f_t(x_i)} b_t(x_i) = 0$$

- Добавить полученное слагаемое в композицию:  $f_t(x) = f_{t-1}(x) + \alpha_t b_t(x)$ .

В результате получаем алгоритм  $a(x) = f_T(x) = \sum_{t=1}^T \alpha_t b_t(x)$

Чтобы получить предсказание достаточно получить результат:  $y_{pred} = a(x_{pred})$



# Популярные реализации GBM



*dmlc*  
**XGBoost**

 LightGBM

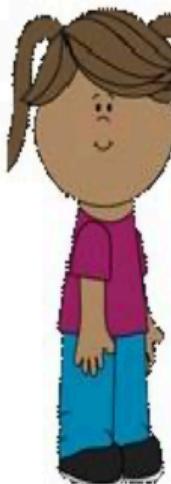


Yandex  
CatBoost

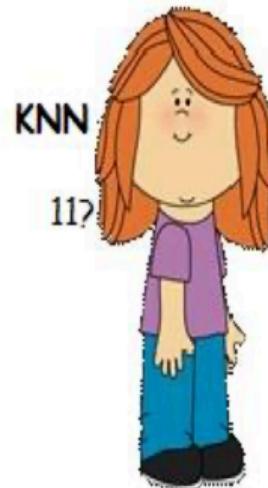
# Blending And Stacking



LR  
13?

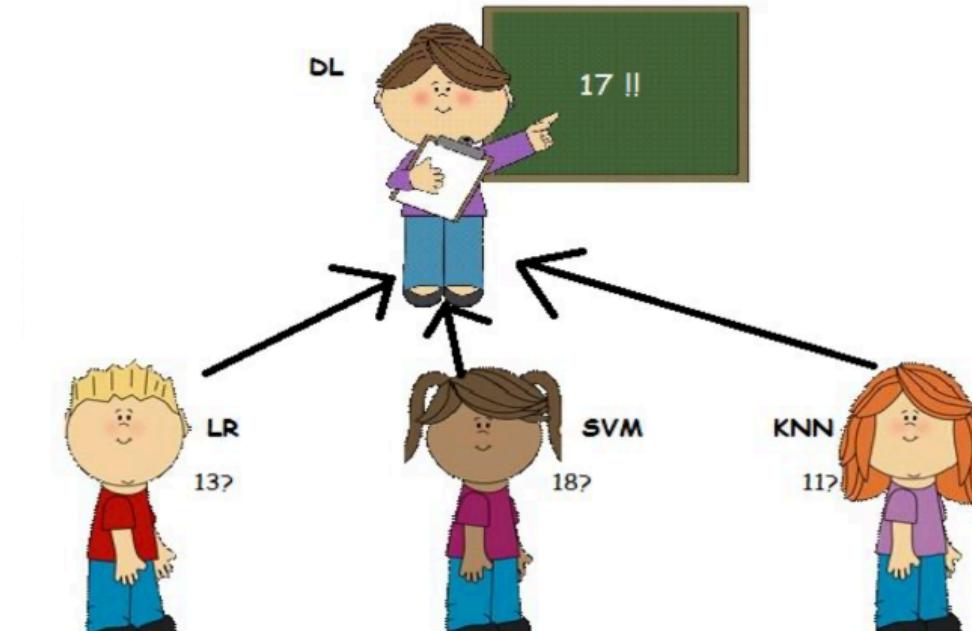


SVM  
18?

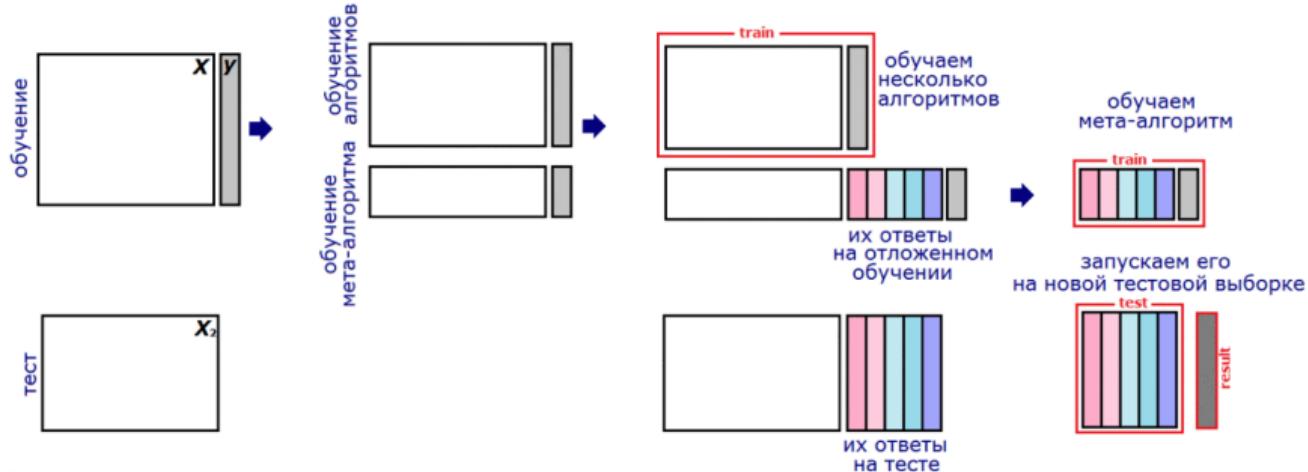


KNN  
11?

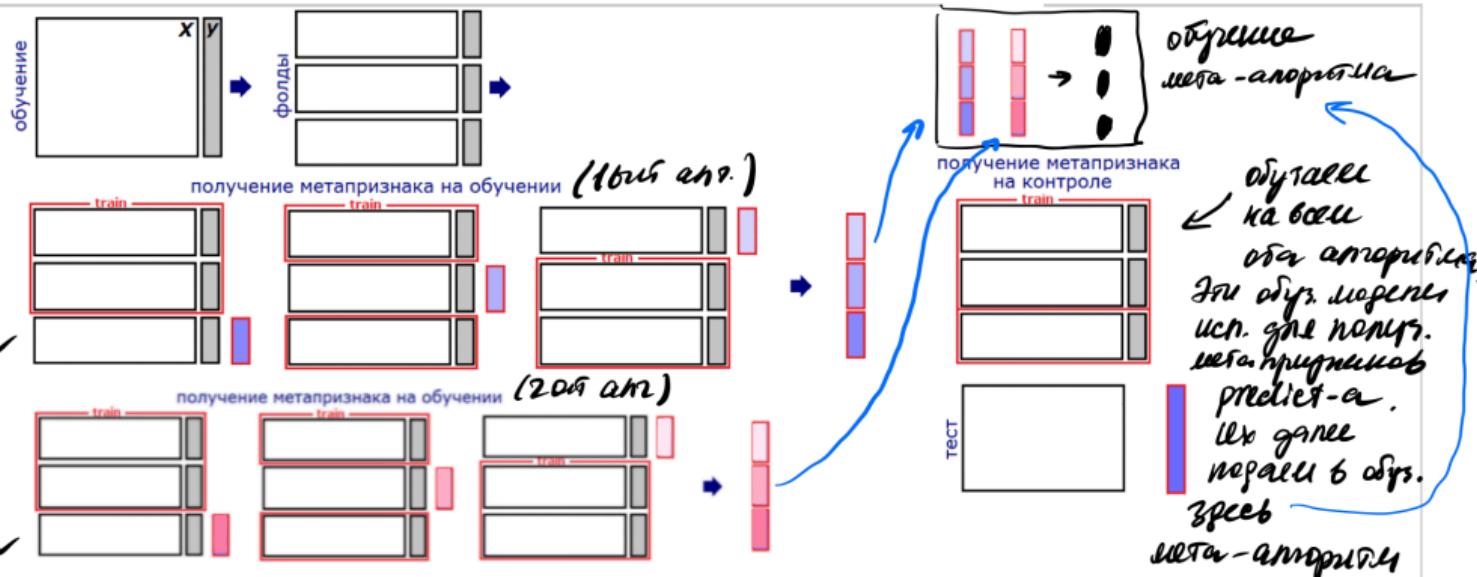
# Blending And Stacking



# Blending



# Stacking





## Выводы

- Ансамбли позволяют снижать смещение или разброс
- Самые популярные ансамбли: Random Forest и GBM
- Также можно использовать bagging, blending, stacking

# Практика



Ensembles.ipynb



# Почитать



Соколов: <https://github.com/esokolov/ml-course-hse/tree/master/2019-fall/lecture-notes>

Coursera Kaggle (4 неделя-ансамбли): <https://www.coursera.org/learn/competitive-data-science/>

Дьяконов стекинг, блендинг: <https://dyakonov.org/2017/03/10/стекинг-stacking-и-блэндинг-blending/>

XGBoost tutorial: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

Хорошая лекция про XGBoost: [https://www.youtube.com/watch?v=2DiOwj-X1ic&list=PLwdBkWbW0oHEUmY07a0G5jabP\\_fWfGQet&index=14](https://www.youtube.com/watch?v=2DiOwj-X1ic&list=PLwdBkWbW0oHEUmY07a0G5jabP_fWfGQet&index=14)

Wiki AdaBoost: <https://ru.wikipedia.org/wiki/AdaBoost>

Курс ODS: <https://habr.com/ru/company/ods/blog/327250/>



# Почитать. AdaBoost

- Инициализировать веса объектов  $w_i^{(0)} = \frac{1}{N}, i = 1, \dots, N.$
- Для всех  $t = 1, \dots, T$ 
  - Обучить базовый алгоритм  $b_t$  на **взвешенных объектах**, пусть  $\epsilon_t$  – его ошибка на обучающей выборке.

$$\epsilon_t = Q(b_t, X, y) = \sum_{i=1}^N w_i^{(t-1)} [y_i \neq b_t(x)]$$

- Если  $\epsilon \geq 0.5$ , то не имеет смысла дальше продолжать работать с данным семейством решений (оно работает хуже рандома)
- Коэффициент

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t} > 0$$

- Обновить веса объектов:

$$w_i^{(t)} = w_i^{(t-1)} e^{-\alpha_t y_i b_t(x_i)}, \quad i = 1, \dots, N$$

- Веса объектов, которые неверно классифицировались – увеличиваются, другие – уменьшаются

$$\begin{cases} e^{-\alpha_t y_i b_t(x_i)} < 1 & \text{если } y_i = b_t(x_i) \\ e^{-\alpha_t y_i b_t(x_i)} > 1 & \text{если } y_i \neq b_t(x_i) \end{cases}$$

- Нормировать веса объектов:  $w_i^{(t)} = \frac{w_i^{(t)}}{\sum_{j=1}^k w_j^{(t)}}, i = 1, \dots, N.$
- Вернуть  $a = \sum_t \alpha_t b_t$

Вес объекта увеличивается в  $e^{\alpha_t}$  раз, когда  $b_t$  допускает на нём ошибку, и уменьшается во столько же раз, когда  $b_t$  правильно классифицирует этот объект. Таким образом, непосредственно перед настройкой базового алгоритма наибольший вес накапливается у тех объектов, которые



# Почитать. XGBoost

- Предположим, что мы хотим обучить ансамбль из  $K$  деревьев:

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \text{ где } f_k \in F.$$

- Целевая функция – это потери + регуляризаторы:

$$\text{Obj} = \sum_{i=1}^N l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k).$$

- Например, для регрессии  $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$ .
- А для классификации в AdaBoost было

$$l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i}).$$



# Почитать. XGBoost

- Мы не можем просто взять и минимизировать общую ошибку
  - трудно минимизировать по всевозможным деревьям.
- Так что опять продолжаем жадным образом:

$$\hat{y}_i^{(0)} = 0,$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i),$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_1(x_2),$$

...,



а предыдущие деревья всегда остаются теми же самыми, они фиксированы.

# Почитать. XGBoost



- Чтобы добавить следующее дерево, нужно оптимизировать

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i), \text{ так что}$$

$$\text{Obj}^{(t)} = \sum_{i=1}^N l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + \text{Const.}$$

- Например, для квадратов отклонений

$$\text{Obj}^{(t)} = \sum_{i=1}^N \left(2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2\right) + \Omega(f_t) + \text{Const.}$$



# Почитать. XGBoost

- Чтобы оптимизировать

$$\text{Obj}^{(t)} = \sum_{i=1}^N l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{Const},$$

давайте заменим это на аппроксимацию второго порядка.

- Обозначим

$$g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial (\hat{y}^{(t-1)})^2},$$

тогда



$$\text{Obj}^{(t)} \approx \sum_{i=1}^N \left( l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t) + \text{Const.}$$

- Например, для квадрата отклонения  $g_i = 2(\hat{y}^{(t-1)} - y_i)$ ,  $h_i = 2$ .



# Почитать. XGBoost

- Итак,

$$\text{Obj}^{(t)} \approx \sum_{i=1}^N \left( l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t) + \text{Const.}$$

- Уберём константы:

$$\text{Obj}^{(t)} \approx \sum_{i=1}^N \left( g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t).$$

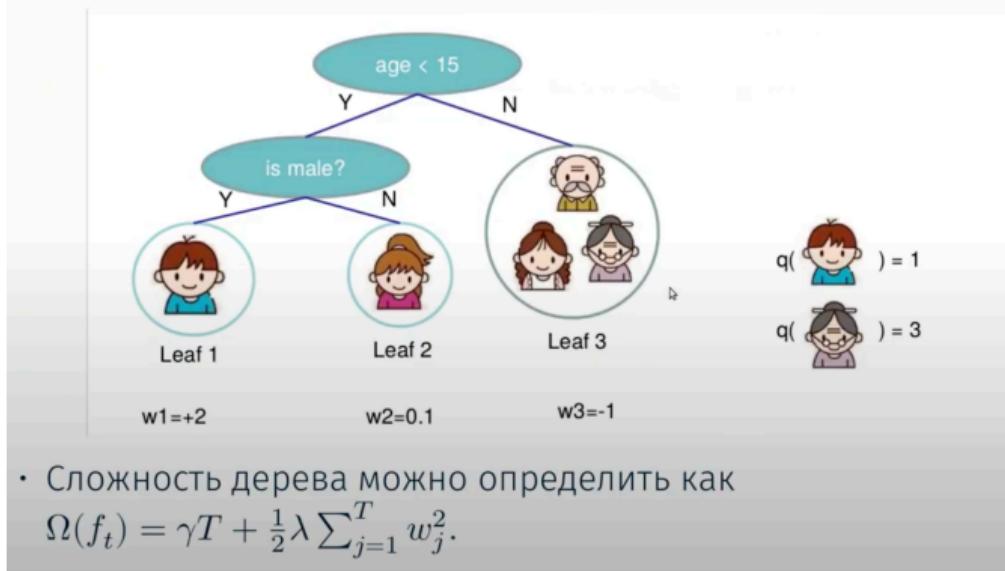
- И это и есть основная идея градиентного бустинга.
- Теперь давайте вернёмся к обучению деревьев и сложим всё воедино.



# Почитать. XGBoost

- Дерево – это вектор оценок в листьях и функция, которая вход отображает в лист:

$$f_t(x) = w_{q(x)}, \text{ где } w \in \mathbb{R}^T, q : \mathbb{R}^d \rightarrow \{1, \dots, T\}.$$



- Сложность дерева можно определить как

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2.$$



# Почитать. XGBoost

- Теперь перегруппируем слагаемые относительно листьев:

$$\begin{aligned}\text{Obj}^{(t)} &\approx \sum_{i=1}^N \left( g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t) \\ &= \sum_{i=1}^N \left( g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right) + \Omega(f_t) \\ &= \sum_{j=1}^T \left( w_j \sum_{i \in I_j} g_i + \frac{1}{2} w_j^2 \left( \sum_{i \in I_j} h_i + \lambda \right) \right) + \gamma T \\ &= \sum_{j=1}^T \left( G_j w_j + \frac{1}{2} w_j^2 (H_j + \lambda) \right) + \gamma T,\end{aligned}$$

где  $I_j = \{i \mid q(x_j) = j\}$ ,  $G_j = \sum_{i \in I_j} g_i$ ,  $H_j = \sum_{i \in I_j} h_i$ .

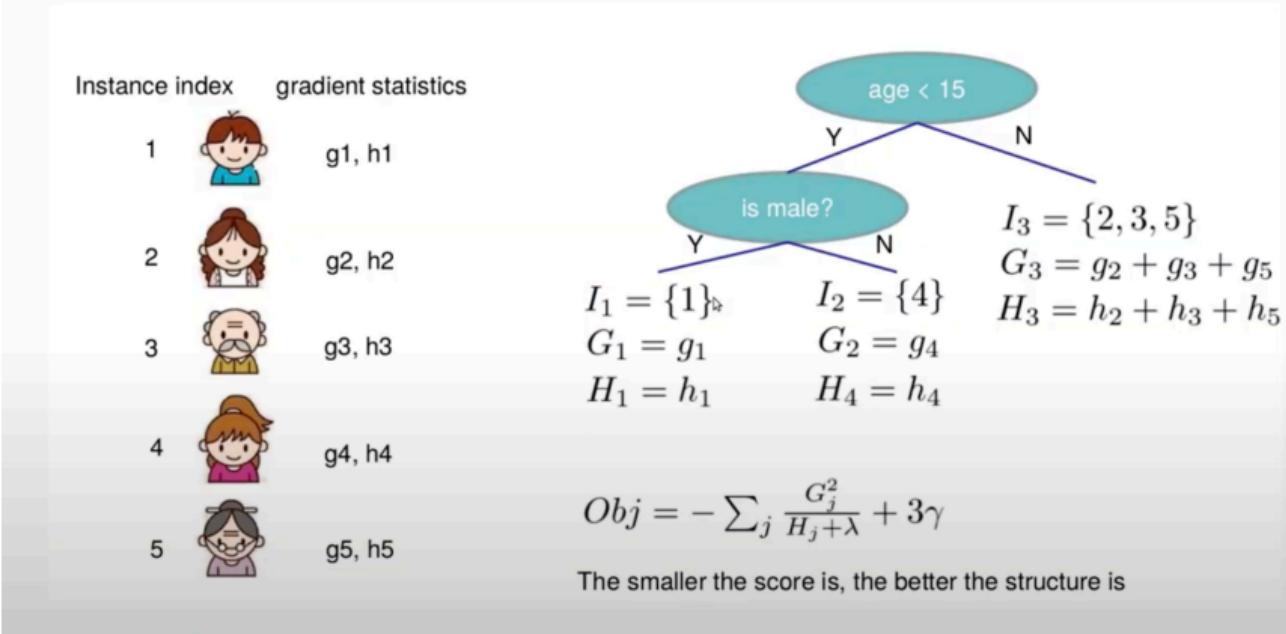
- Это сумма  $T$  независимых квадратичных функций, так что

$$w_j^* = -\frac{G_j}{H_j + \lambda}, \quad \text{Obj}^{(t)} \approx -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T.$$

# Почитать. XGBoost



- Пример из (Chen, Guestrin):



## Почитать. XGBoost



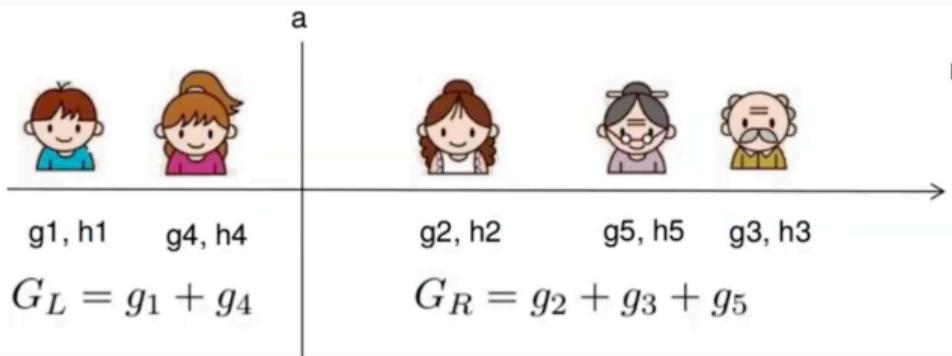
- Так что мы находим наилучшую структуру дерева относительно  $-\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$  и используем оптимальные веса листьев  $w_j^* = -\frac{G_j}{H_j + \lambda}$ .
- Как найти структуру? Жадно: для каждого листа попробуем добавить разбиение, и целевая функция меняется на

$$\text{Gain} = \frac{1}{2} \left( \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma.$$

# Почитать. XGBoost



- Самое лучшее разбиение – то, которое максимизирует gain:



- Обрезание: сначала вырастим дерево до максимальной глубины, потом рекурсивно обрежем листья с отрицательным gain.