

ЗЕЛЕНКОВ Ю. А. (С) 2024

2. ОБУЧЕНИЕ С УЧИТЕЛЕМ НА ТАБЛИЧНЫХ ДАННЫХ



СОДЕРЖАНИЕ

- Общая постановка задачи обучения с учителем.
- Нейронные сети.
- Деревья решений.
- Компромисс: смещение VS дисперсия.
- Ансамбли моделей.
- Метрики качества и валидация моделей.

ОБЩАЯ ПОСТАНОВКА ЗАДАЧИ

ДАННЫЕ

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

ПРЕДИКТИВНАЯ ФУНКЦИЯ

$$\hat{y} = f(x, \theta) \\ f \in \mathcal{F}$$

ФУНКЦИЯ ПОТЕРЬ

$$L(f(x, \theta), y)$$



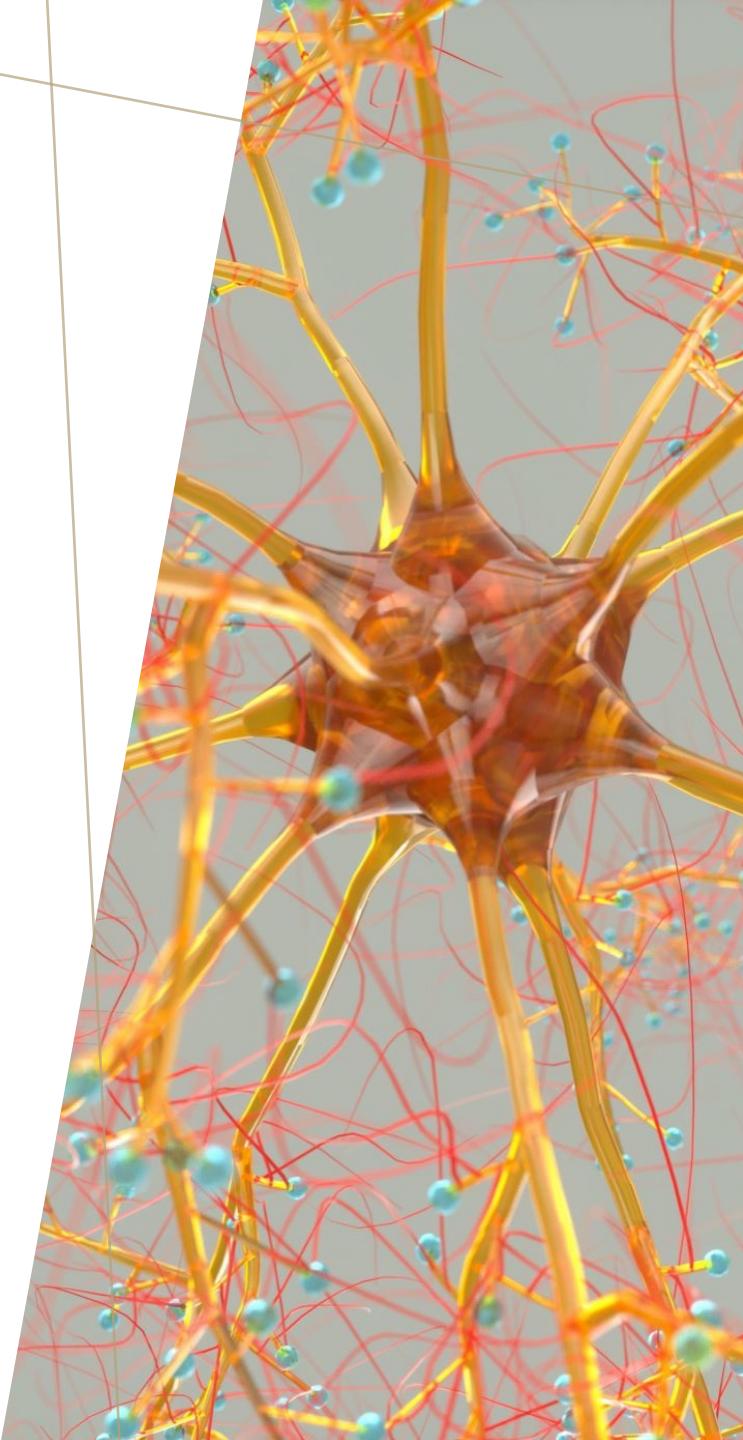
Алгоритм, оптимизирующий $L(f(x, \theta), y)$

$$\theta \leftarrow \operatorname{argmin}_{\theta} L(f(x, \theta), y)$$

ПРЕДИКТИВНАЯ ФУНКЦИЯ

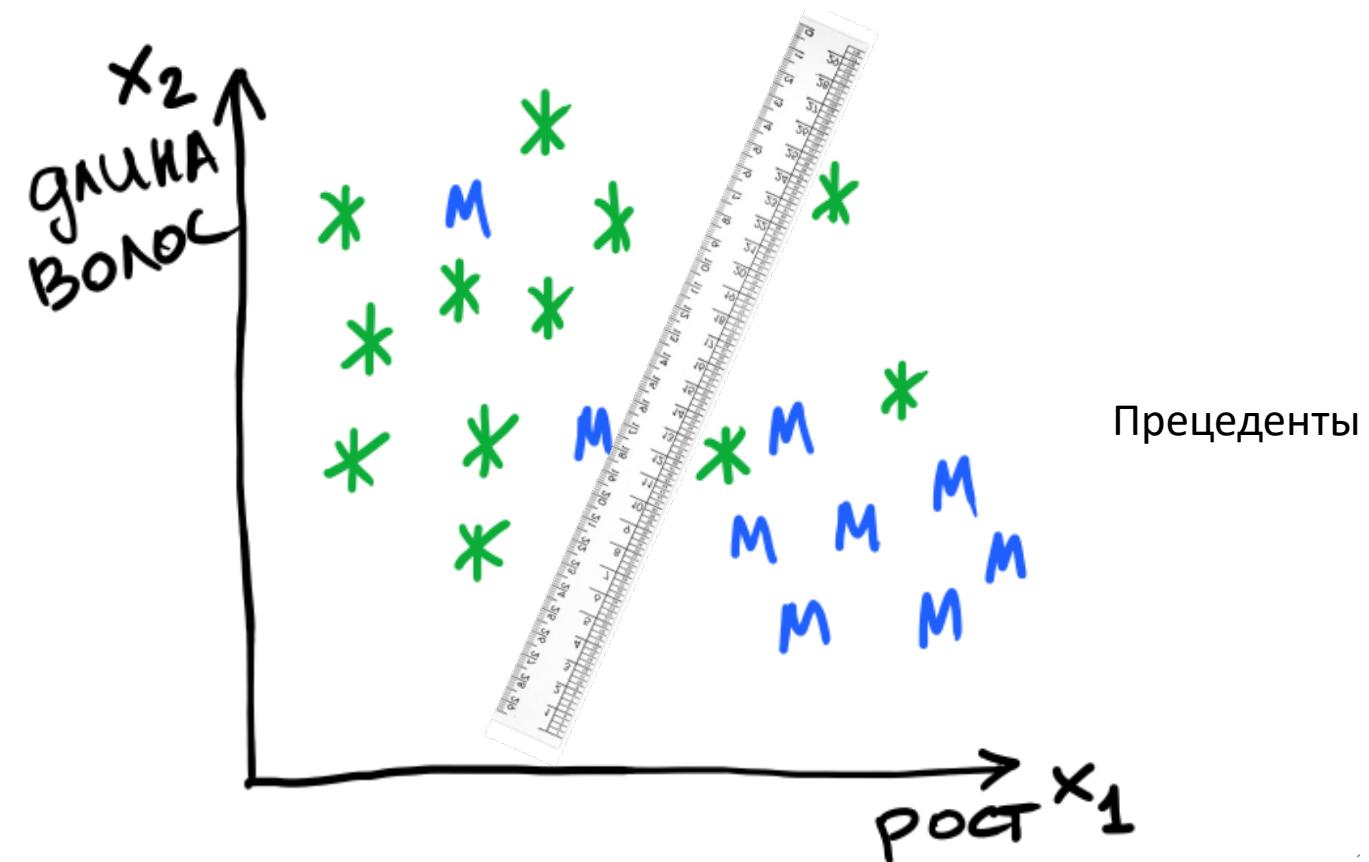
$$f: X \rightarrow Y$$

ДИФФЕРЕНЦИРУЕМЫЕ МОДЕЛИ. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ (ANN)

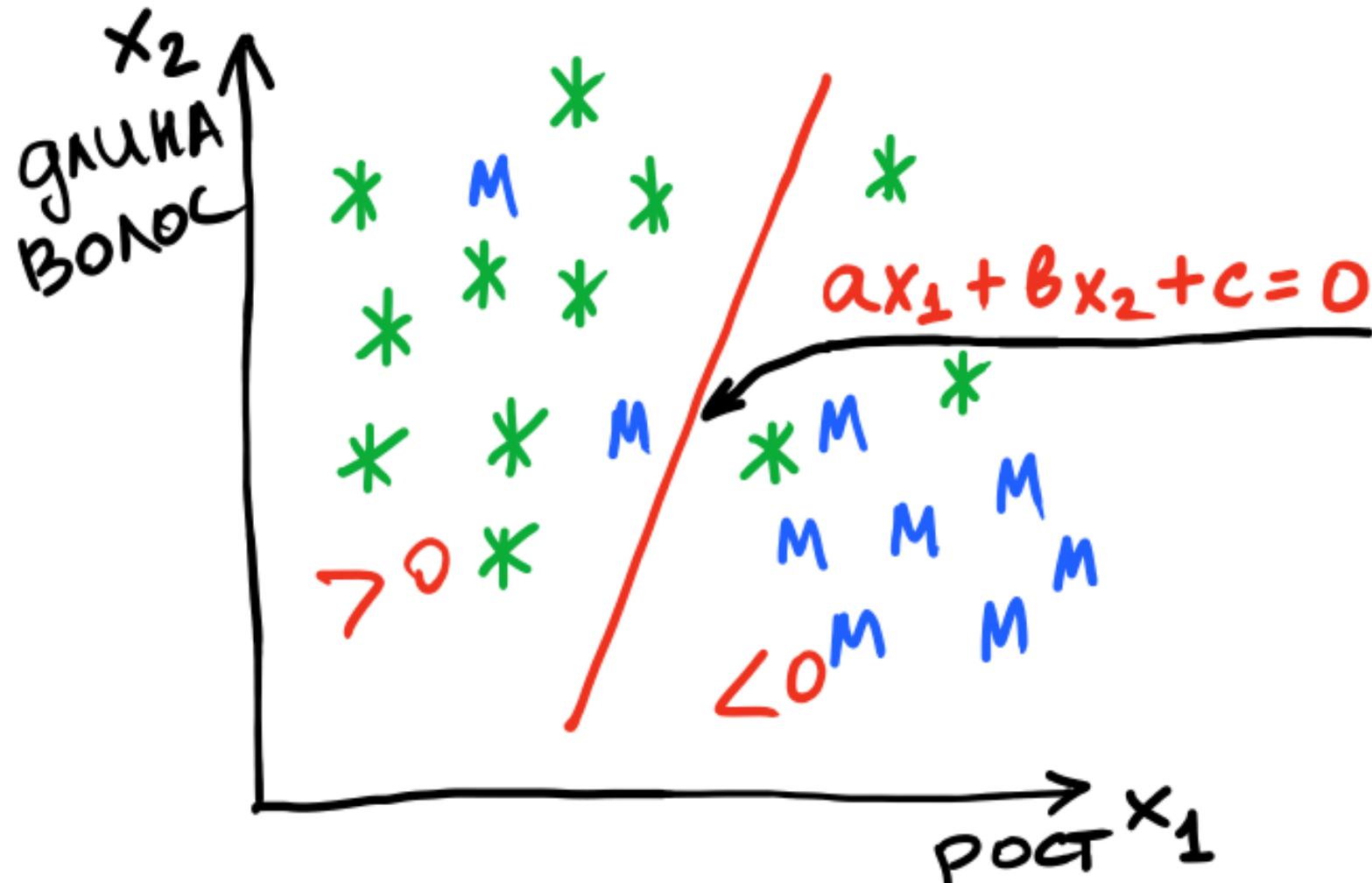


ЛИНЕЙНАЯ МОДЕЛЬ ДЛЯ КЛАССИФИКАЦИИ

x_1	x_2	y
180	5	М
170	20	Ж
160	5	М
190	30	?

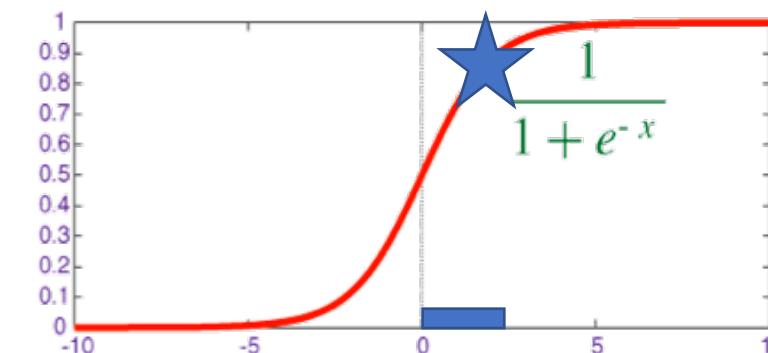
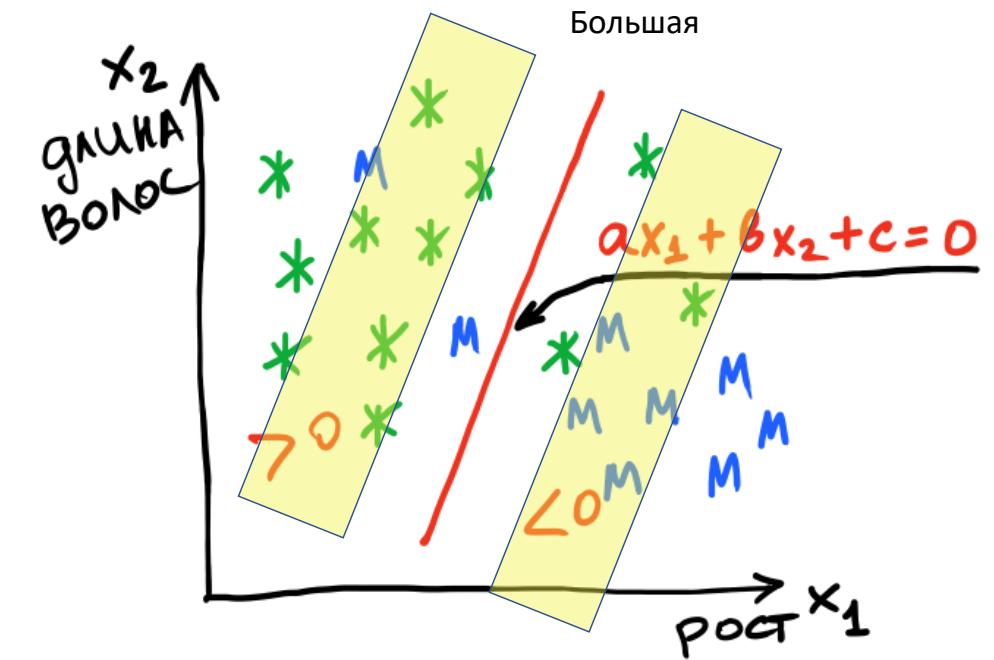
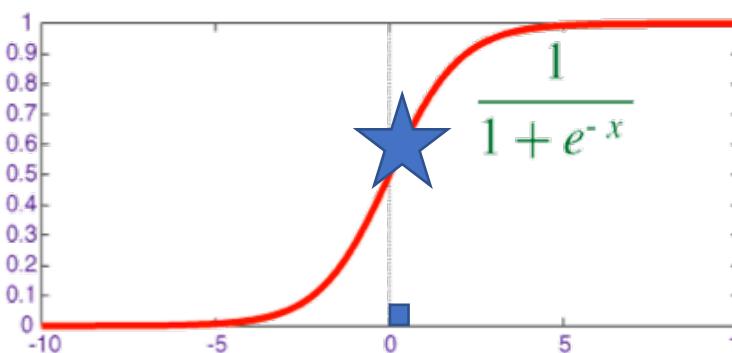
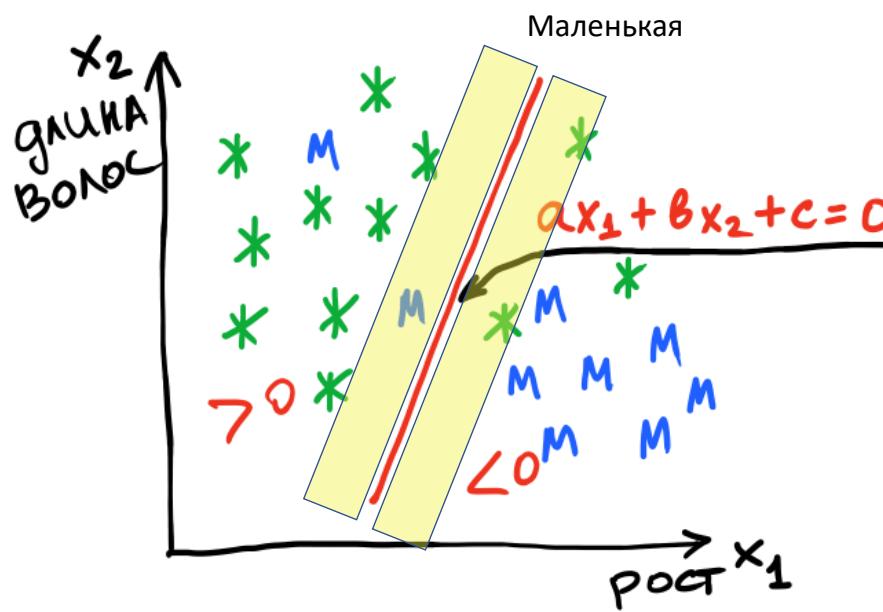


x_1	x_2	y
180	5	М
170	20	Ж
160	5	М
190	30	?

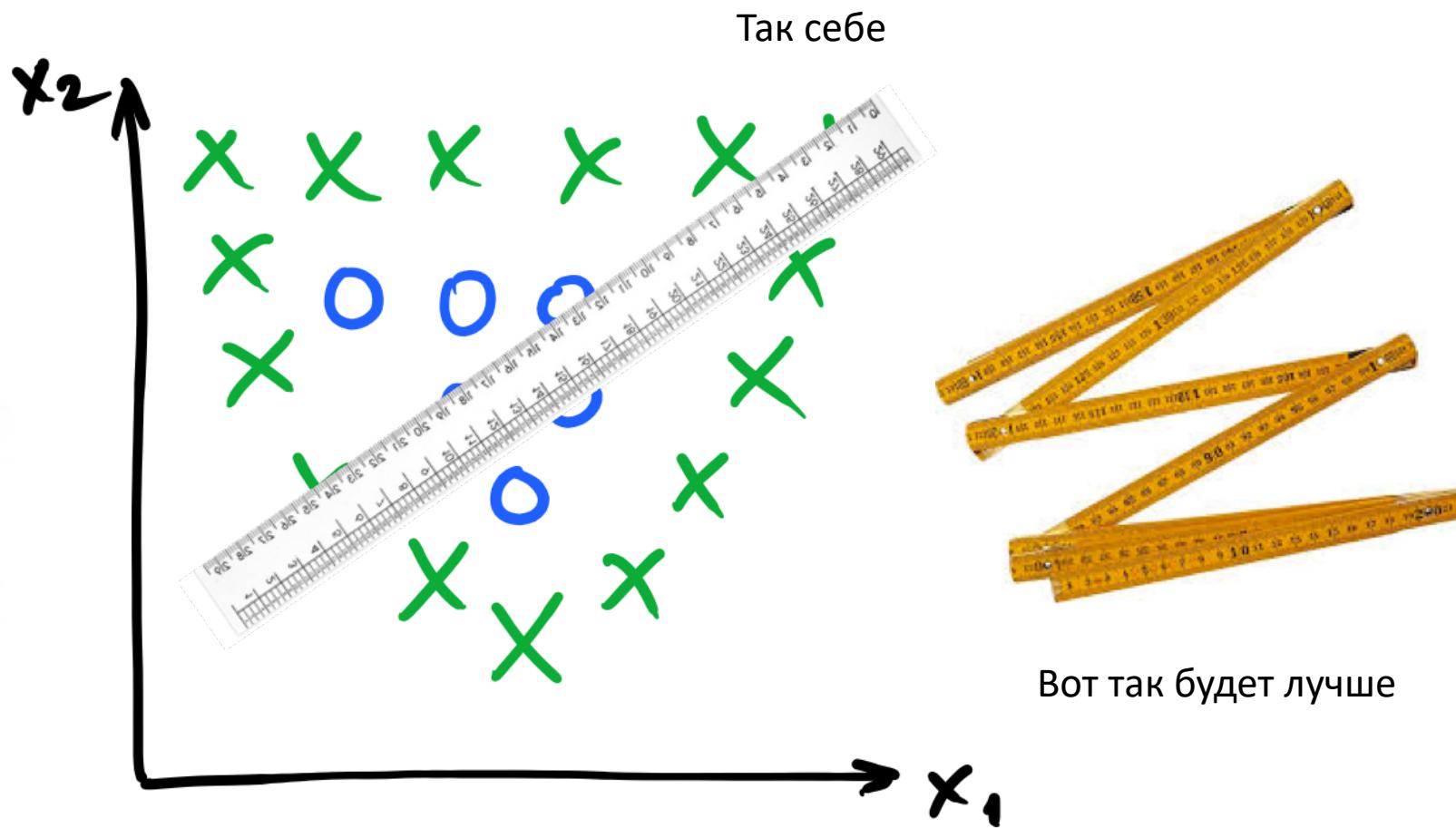


5 ошибок, accuracy = 0.77

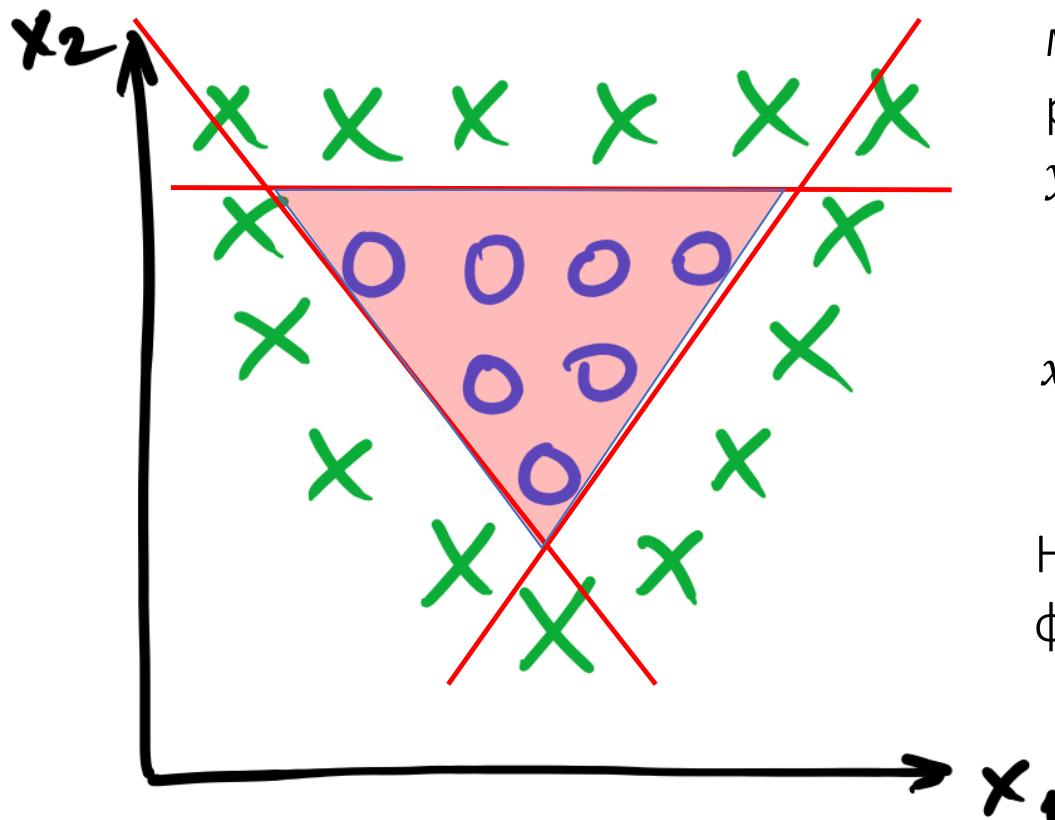
УВЕРЕННОСТЬ ПРЕДСКАЗАНИЯ



ЧТО ДЕЛАТЬ В ТАКОЙ СИТУАЦИИ?



КОМПОЗИЦИЯ ЛИНЕЙНЫХ ФУНКЦИЙ

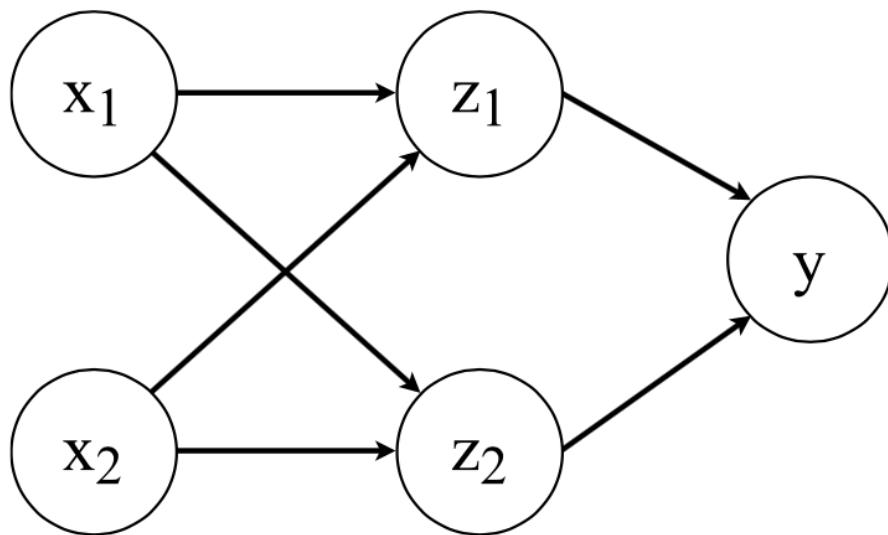


Построим три **линейные** модели, отвечающие за разделение в разных областях $y_1(x), y_2(x), y_3(x)$.

	$y_1(x)$	$y_2(x)$	$y_3(x)$	y
x	0.5	0.1	0.6	1
	0.1	0.8	0.2	0

На их предсказаниях построим финальную линейную модель
 $y(x) = \alpha y_1(x) + \beta y_2(x) + \gamma y_3(x)$

КОМПОЗИЦИЯ ЛИНЕЙНЫХ ФУНКЦИЙ



$$z_1 = w_{11}x_1 + w_{12}x_2$$

$$z_2 = w_{21}x_1 + w_{22}x_2$$

$$y = w_{31}z_1 + w_{32}z_2$$

Проблема: сохраняется линейность

$$y = (w_{31}w_{11} + w_{32}w_{21})x_1 + (w_{31}w_{12} + w_{32}w_{22})x_2$$

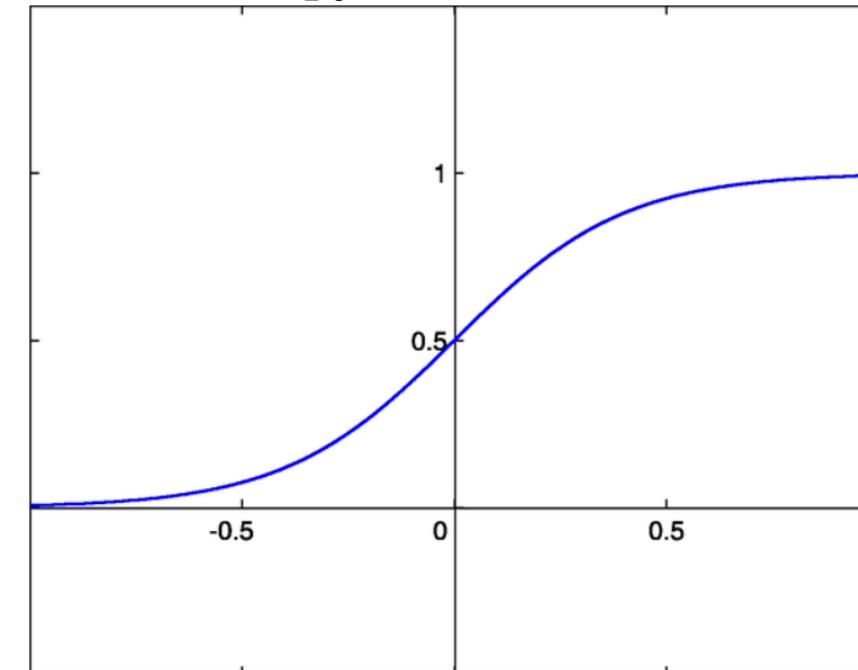
НЕЛИНЕЙНЫЕ ФУНКЦИИ

$h : \mathbb{R} \rightarrow \mathbb{R}$ — некоторая нелинейная функция

Пример: сигмоида

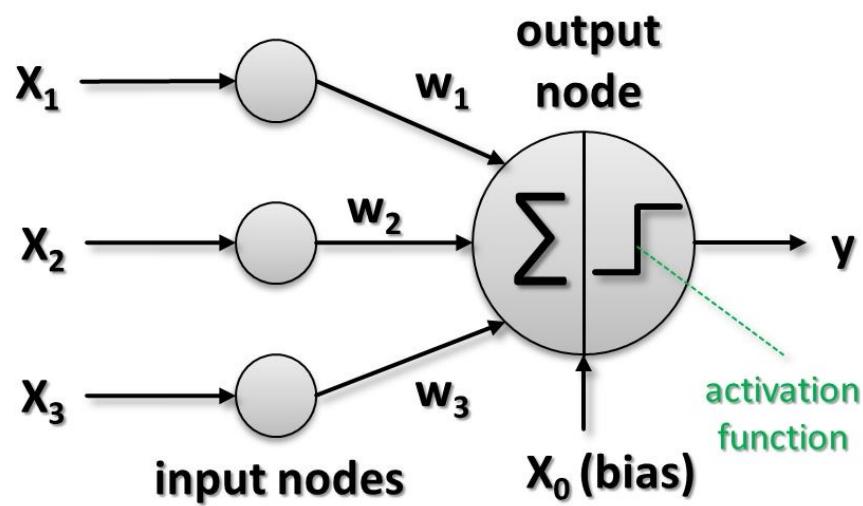
$$h(x) = \frac{1}{1 + e^{-x}}$$

В каждом узле:

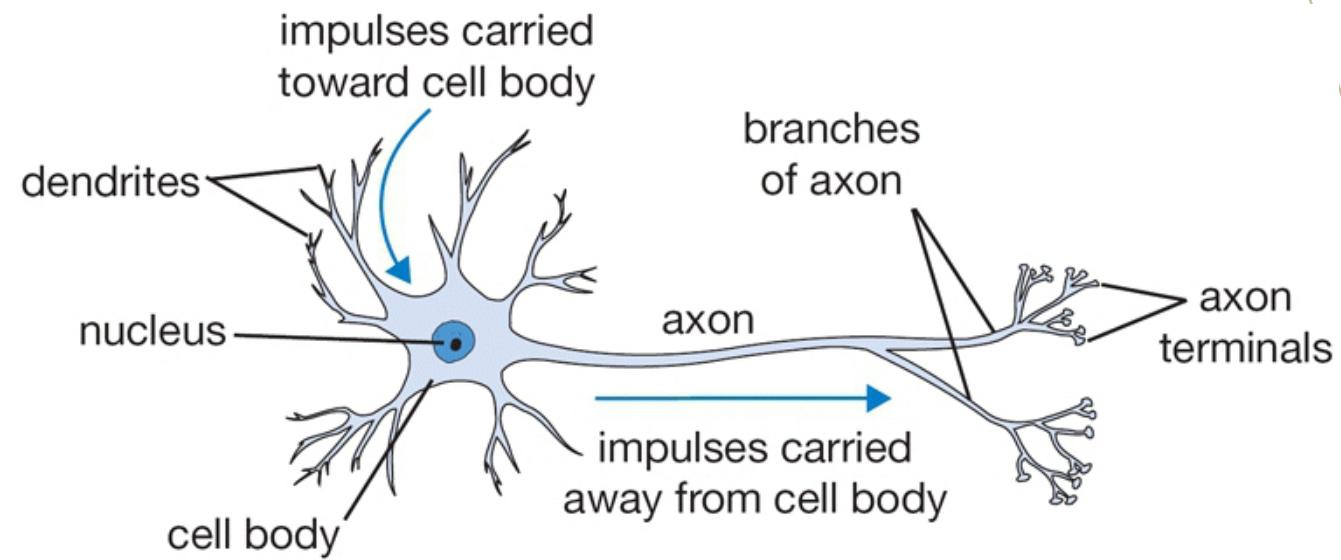


$$f(x_1, \dots, x_k) = h \left(\sum_{i=1}^k w_i x_i + w_0 \right)$$

ARTIFICIAL NEURON

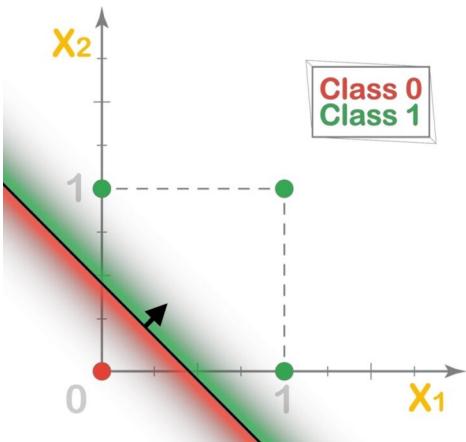


$$y = h \left(\sum_{i=1}^k w_i x_i + w_0 \right)$$



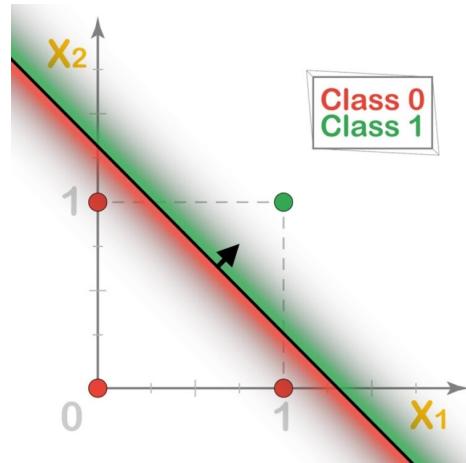
OR

X_1	X_2	$X_1 \text{ OR } X_2$
0	0	0
0	1	1
1	0	1
1	1	1



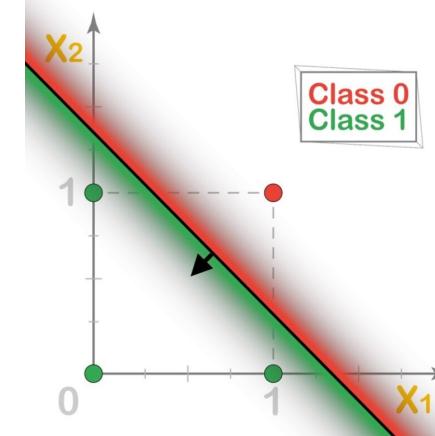
AND

X_1	X_2	$X_1 \text{ AND } X_2$
0	0	0
0	1	0
1	0	0
1	1	1



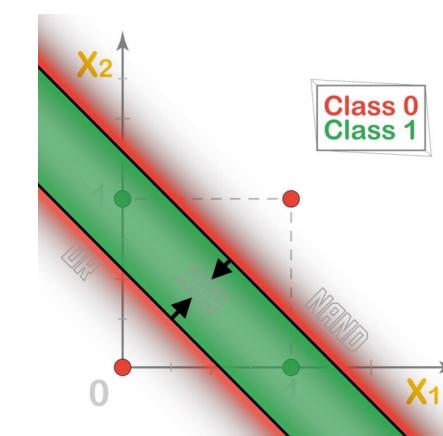
NAND

X_1	X_2	$X_1 \text{ NAND } X_2$
0	0	1
0	1	1
1	0	1
1	1	0



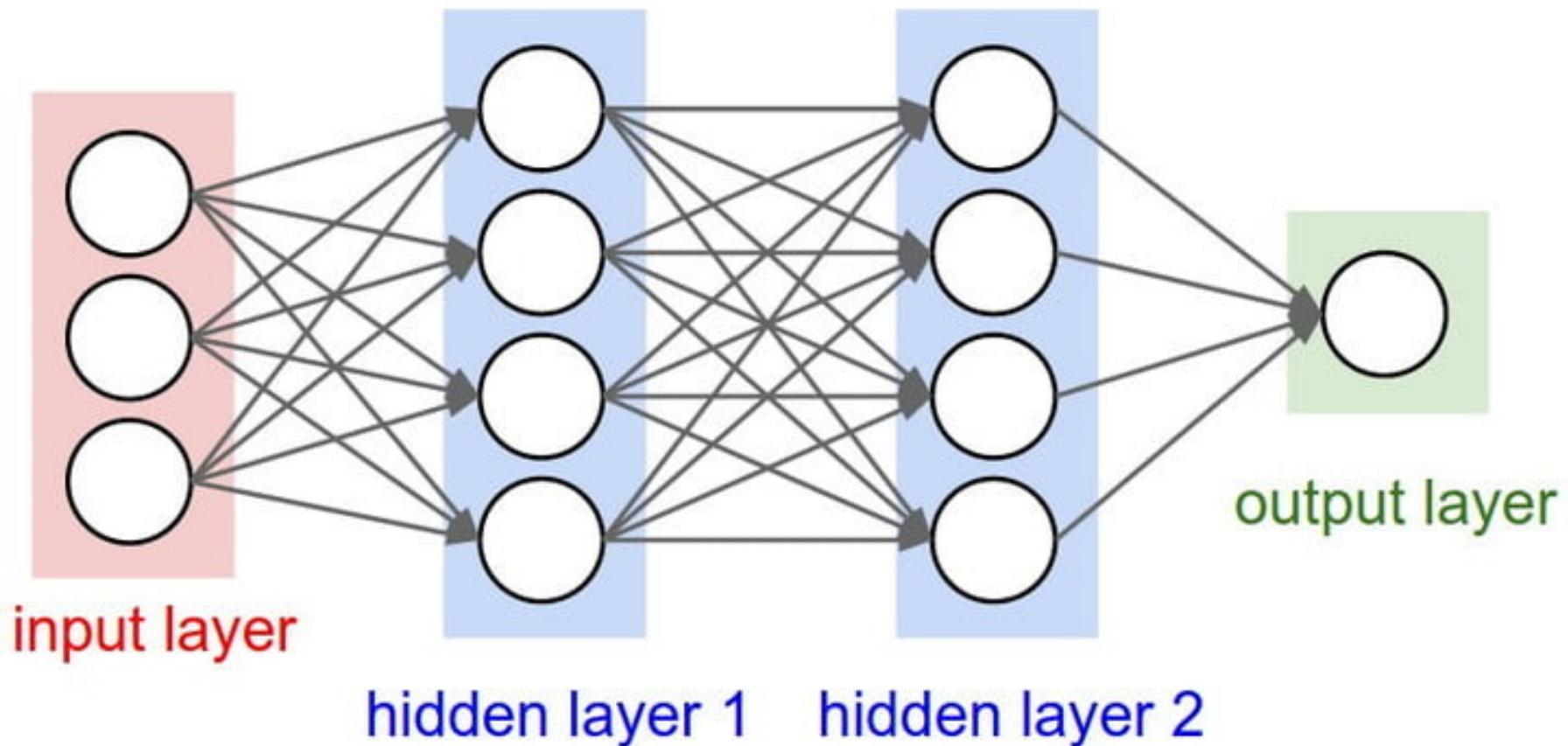
XOR

X_1	X_2	$X_1 \text{ XOR } X_2$
0	0	0
0	1	1
1	0	1
1	1	0



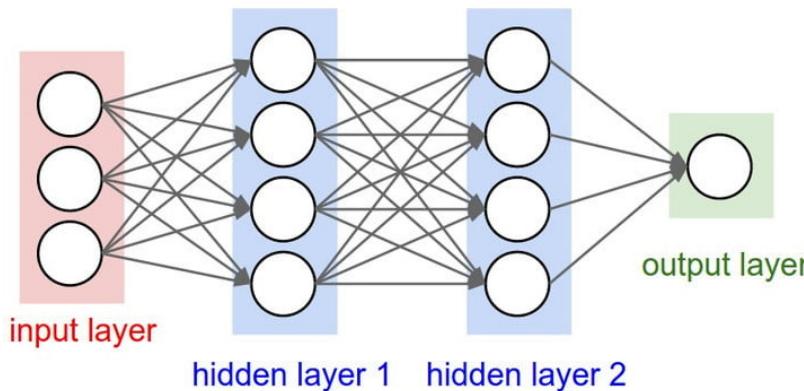
$$\text{XOR } (x_1, x_2) = \text{AND}(\text{OR}(x_1, x_2), \text{NAND}(x_1, x_2))$$

ARTIFICIAL NEURAL NETWORK



source: <https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/>

ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ



$\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ - обучающая выборка

$a(x; w)$ - модель (ANN)

x – входной объект

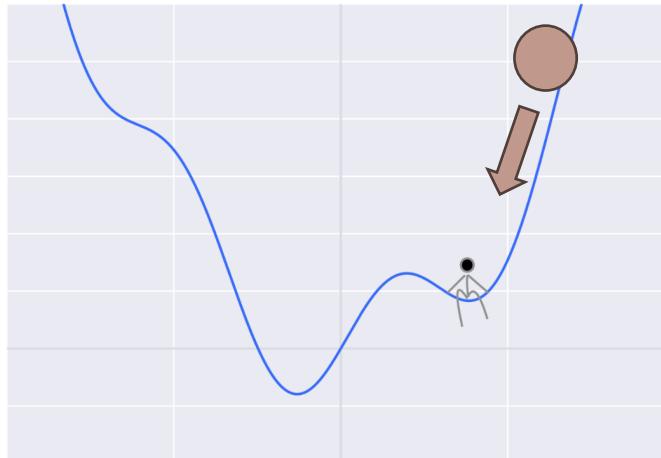
w – параметры (веса нейронов)

Гиперпараметры: архитектура сети (число слоев и нейронов в них)

$L(y, \hat{y}) \rightarrow \min$ - функция потерь

Оптимизация градиентным спуском: $\frac{\partial L(y, \hat{y})}{\partial w}$

ГРАДИЕНТНЫЙ СПУСК

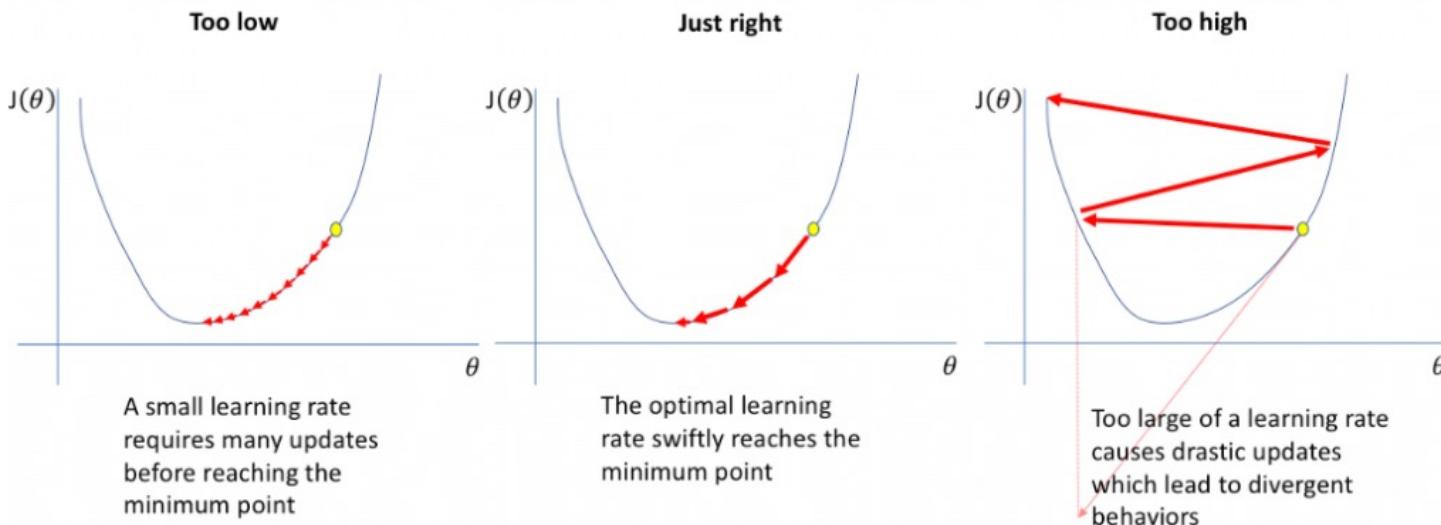


Может застрять в локальном минимуме

- Пусть $f = f(x, y)$

- Частная производная: $\frac{\partial f}{\partial x}$

- Градиент: $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$



Чувствителен к величине шага обучения

РАСШИРЕНИЯ:

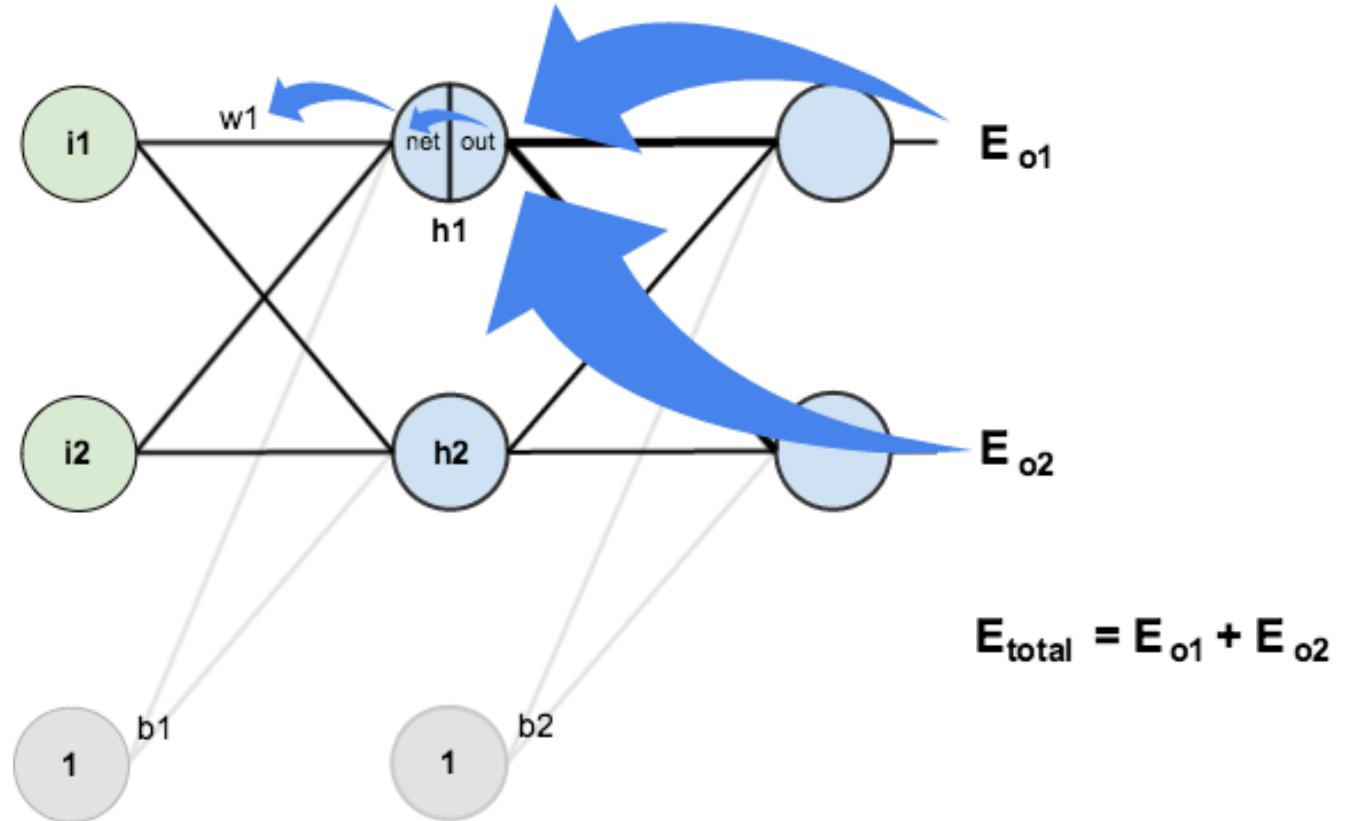
- Стохастический градиентный спуск (SGD)
- Градиентный спуск с инерцией
- Адаптивный градиентный спуск / снижение скорости обучения
- ADAM – индивидуальное управление скоростью обучения по каждому параметру

BACKPROPAGATION ALGORITHM

Модификация классического градиентного спуска
Функция активации **должна быть** дифференцируема

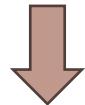
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

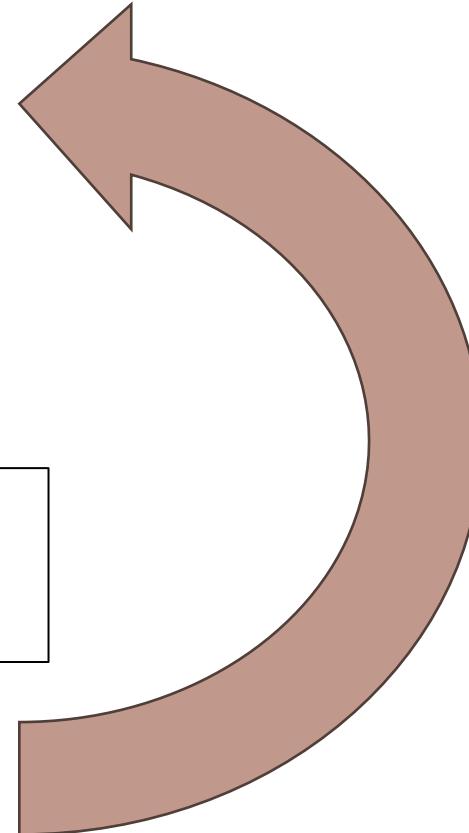


PIPELINE

Определяем гиперпараметры ANN:
 K – число внутренних слоев
 n_K – число нейронов в каждом слое

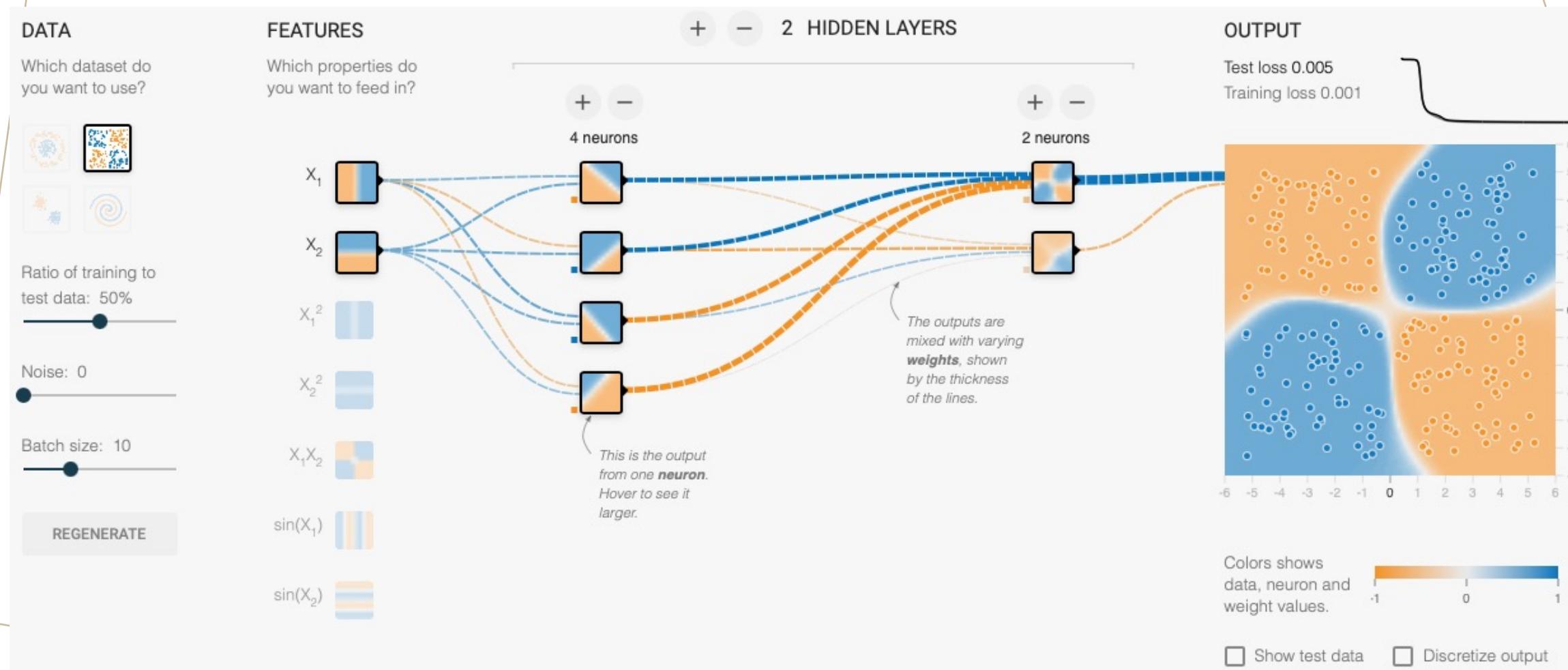


Вычисление весов с
помощью метода обратного
распространения ошибки



Оптимизация
гиперпараметров

<http://playground.tensorflow.org>



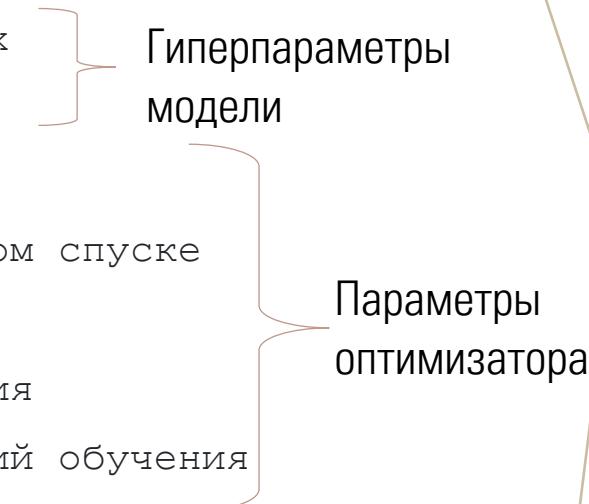
КЕЙСЫ ИЗ ПРЕДЫДУЩЕЙ ЛЕКЦИИ

См. jupyter блокнот Lecture2.ipynb

		Линейные модели		Multi-Layer Perceptron (10,2)	
		Train	Test	Train	Test
Регрессия (стоимость жилья)	MSE	0.518	0.720	0.475	0.689
	RMSE	0.556	0.746	0.494	0.703
Классификация (банкроты)	Accuracy	0.824	0.837	0.835	0.831
	AUC ROC	0.540	0.580	0.610	0.620

MLP IN SCIKT-LEARN

```
from sklearn.neural_network import MLPRegressor  
  
mlp = MLPRegressor(  
    hidden_layer_sizes =(10,2),      # кол-во нейронов в скрытых слоях  
    activation = "relu",            # функция активации  
    solver = "adam",               # оптимизатор  
    batch_size = "auto",           # размер пакета при стохастическом спуске  
    learning_rate = "constant",    # план управления шагом обучения  
    learning_rate_init = 0.001,     # начальное значения шага обучения  
    max_iter = 1000                # максимальное количество итераций обучения  
)  
  
mlp.fit(X_train, y_train)  
  
y_pred = mlp.predict(X_test)
```

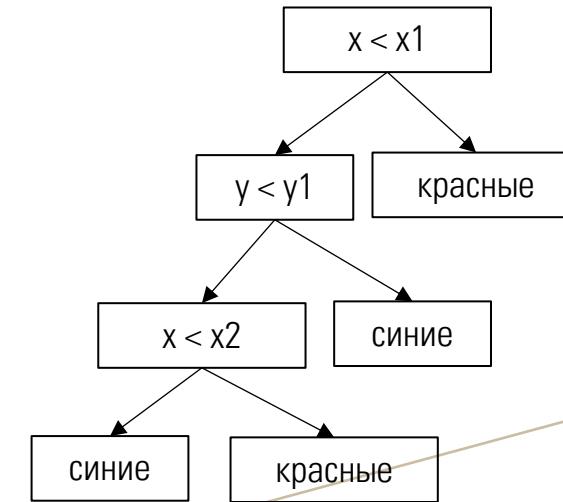
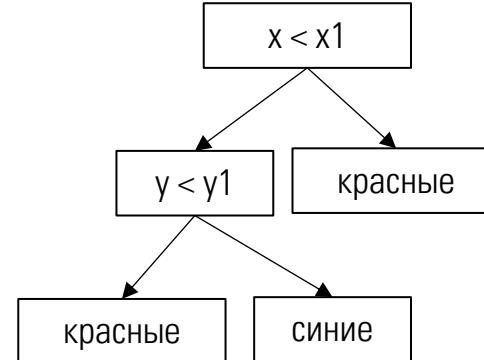
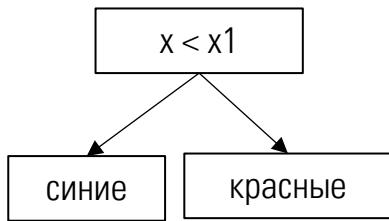
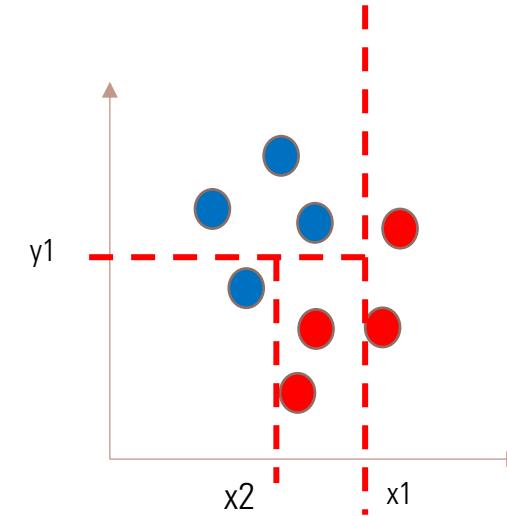
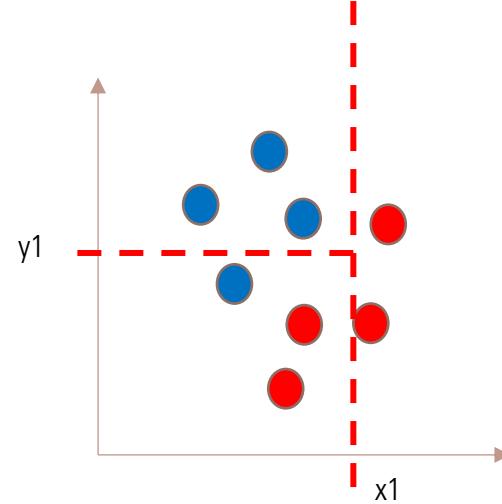
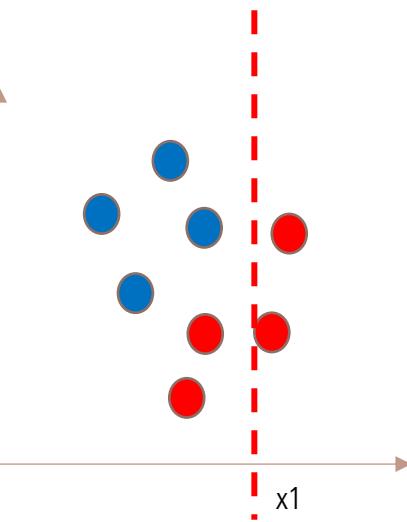


Подробнее см. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

НЕДИФЕРЕНЦИРУЕМЫЕ МОДЕЛИ.
ДЕРЕВЬЯ РЕШЕНИЙ
(DECISION TREE)



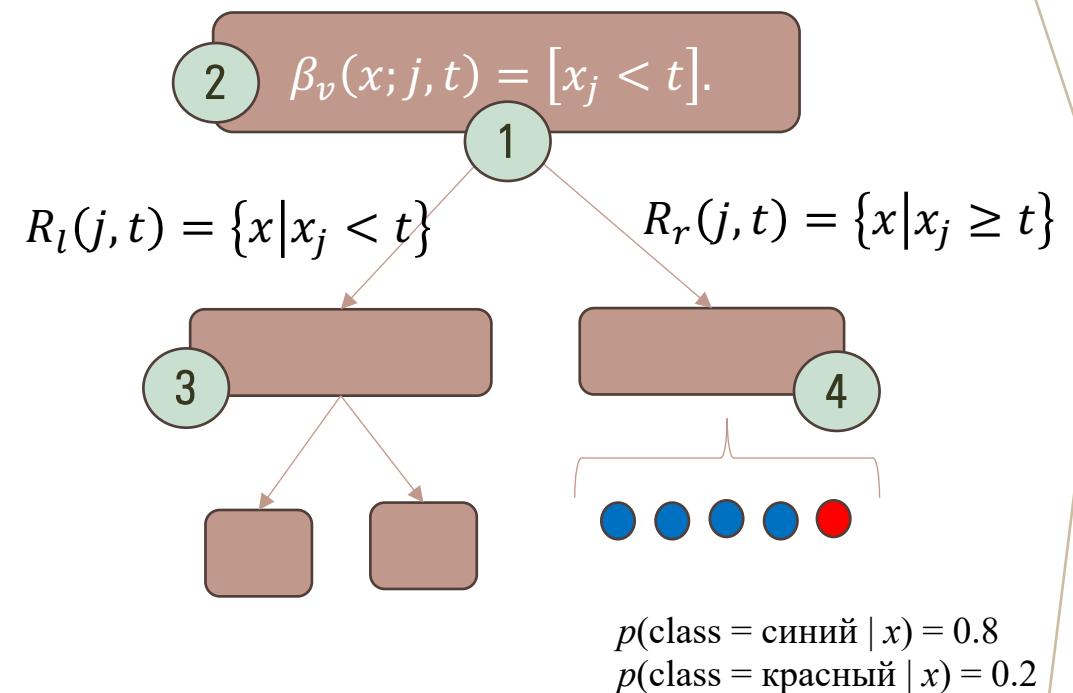
DECISION TREE



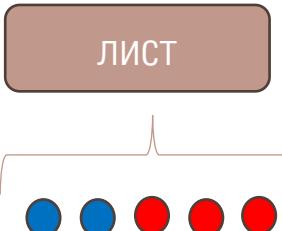
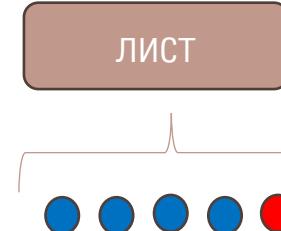
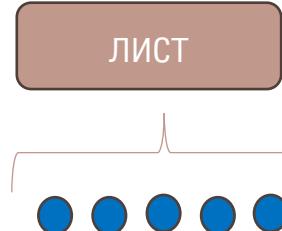
АЛГОРИТМ

$Q(X, j, t)$ - заданный функционал качества
 X – обучающая выборка
 j – номер признака / атрибута
 t – пороговое значение

- 1 Находим наилучшее разбиение выборки X на две части $R_l(j, t) = \{x | x_j < t\}$ и $R_r(j, t) = \{x | x_j \geq t\}$ с точки зрения заданного функционала качества $Q(X, j, t)$
- 2 Найдя наилучшие значения j и t , создаем вершину v , которой ставим в соответствие функцию $\beta_v(x; j, t) = [x_j < t]$.
- 3 Для каждой из подвыборок R_l и R_r рекурсивно повторяем шаги 1-2 пока не будет выполнено условие останова.
- 4 Если условие останова выполнено, рекурсия прекращается, текущая вершина объявляется листом. Листу ставится в соответствие ответ $c_v \in Y$:
 - классификация – вероятность, равная доле экземпляров класса в листе.
 - регрессия – среднее (медиана) целевой переменной экземпляров в листе.



КАК ВЫБРАТЬ ПОРОГ РАЗБИЕНИЯ?

Классификация	Высокая неопределенность	Низкая неопределенность	Полная определенность
Множество объектов в листе R			
$\text{Энтропия } H(R) = - \sum_{i=1}^n p_i \log p_i$	0.292	0.217	0
$\text{Критерий Джини } H(R) = \sum_{i=1}^n p_i(1 - p_i)$	0.480	0.320	0

$H(R)$ - критерий информативности (impurity criterion), который оценивает качество распределения целевой переменной среди объектов множества R . Чем меньше разнообразие целевой переменной во множестве R , тем меньше должно быть значение $H(R)$.

КАК ВЫБРАТЬ ПОРОГ РАЗБИЕНИЯ?

Регрессия

Множество объектов в листе R

$$H(R) = -\frac{1}{|R|} \sum_{(x_i, y_i \in R)} (y_i - y_{mean})^2$$

Высокая
неопределенность

лист

1.0; 1.5; 2.0; 2.0; 2.5

Низкая
неопределенность

лист

1.0; 1.5; 1.5; 2.0; 2.0

Высокая
определенность

лист

1.5; 1.5; 1.5; 2.0; 2.0

0.26

0.14

0.06

ВЫБОР ОПТИМАЛЬНОГО РАЗБИЕНИЯ

Множество объектов в вершине
(значения атрибута x_j и целевой переменной y)
 $x_j = \{5.0; 5.0; 5.1; 5.5; 5.6; 5.8; 5.7; 6.0\}$
 $y = \{1.0; 1.5; 1.6; 2.0; 2.1; 2.5; 2.5; 3.0\}$

$$H(R) = 0.364 \quad \text{-- вычисляется на основе значений } y$$

Варианты
разбиения

$$\beta_v = [x_j < 5.5]$$

$$R_l = \{1.0; 1.5; 1.6; \} \\ R_r = \{2.0; 2.1; 2.5; 2.5; 3.0\}$$

$$H(R_l) = 0.069 \quad H(R_r) = 0.126 \quad Q(R, x_j, t) = 0.260$$

$$\beta_v = [x_j < 5.8]$$

$$R_l = \{1.0; 1.5; 1.6; 2.0; 2.1\} \\ R_r = \{2.5; 2.5; 3.0\}$$

$$H(R_l) = 0.154 \quad H(R_r) = 0.056 \quad Q(R, x_j, t) = 0.247$$

Максимизируем функционал

$$Q(R, x_j, t) = H(R) - \frac{|R_l|}{|R|} H(R_l) - \frac{|R_r|}{|R|} H(R_r)$$

РЕШАЕМ ЗАДАЧУ РЕГРЕССИИ

California Housing

2. DecisionTree

```
# регрессия

from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(criterion='squared_error', splitter='best', max_depth=None, random_state = 1)

dtr.fit(X_train, y_train)

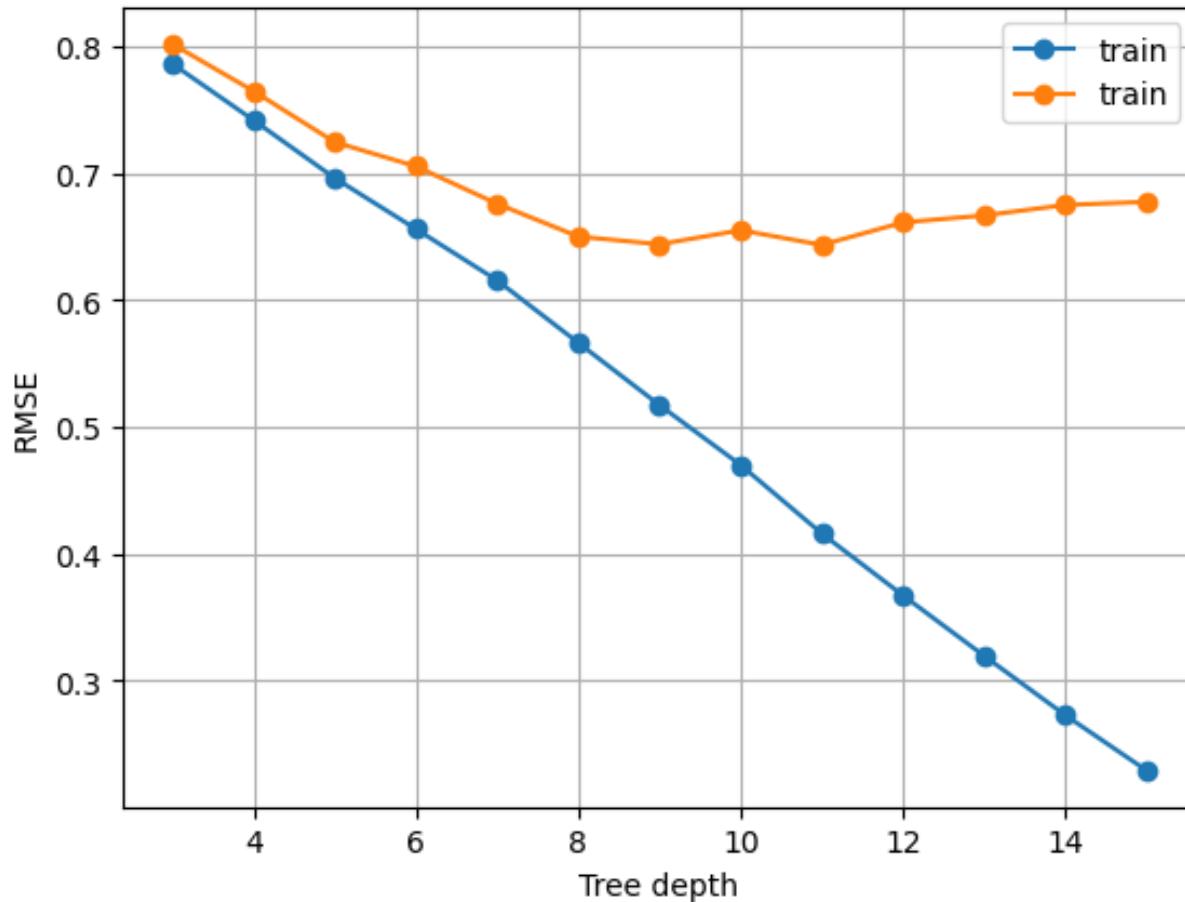
y_train_pred = dtr.predict(X_train)
mse = mean_squared_error(y_train, y_train_pred)
print(f'Ошибка на обучающих данных: MSE = {mse :.3f}, RMSE = {np.sqrt(mse) :.3f}')

y_test_pred = dtr.predict(X_test)
mse1 = mean_squared_error(y_test, y_test_pred)
print(f'Ошибка на тестовых данных : MSE = {mse1 :.3f}, RMSE = {np.sqrt(mse1) :.3f}'')
```

Ошибка на обучающих данных: MSE = 0.000, RMSE = 0.000
Ошибка на тестовых данных : MSE = 0.501, RMSE = 0.708

- модель переобучилась (overfitted)!

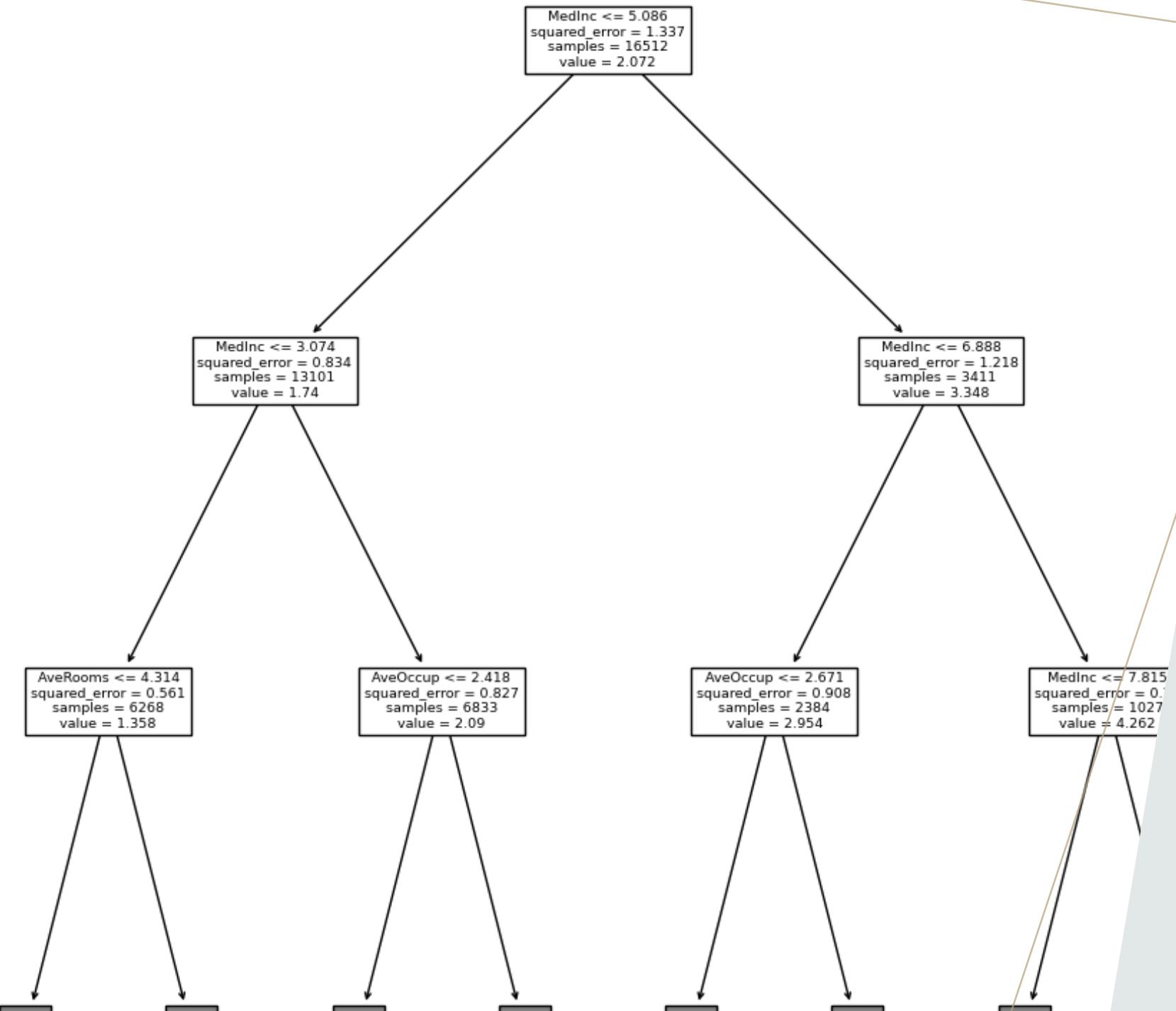
ПРОБЛЕМА ПЕРЕОБУЧЕНИЯ



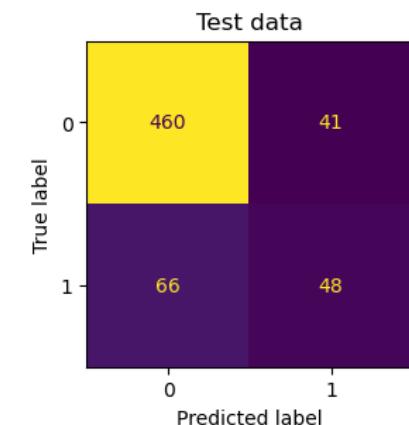
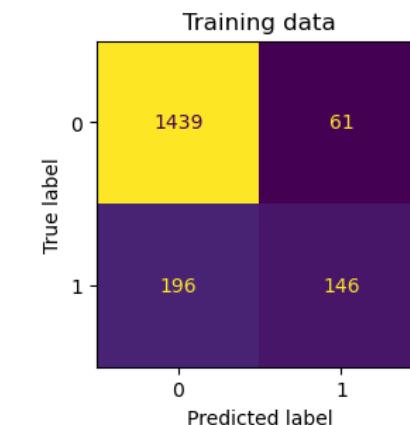
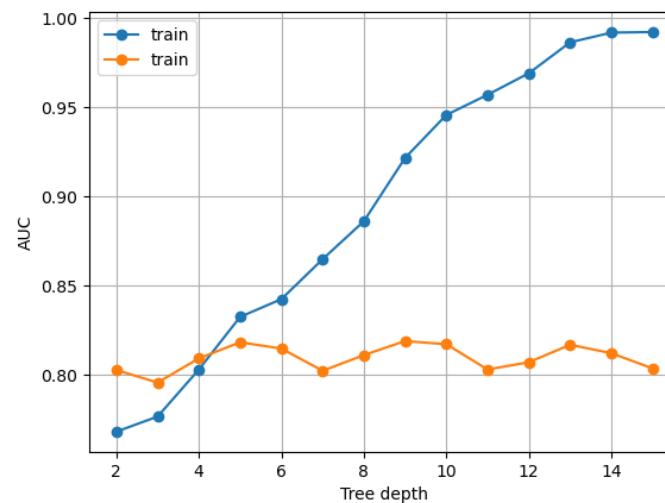
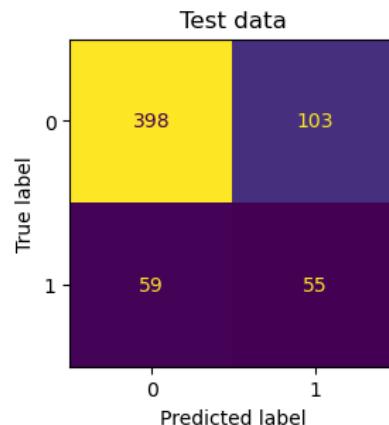
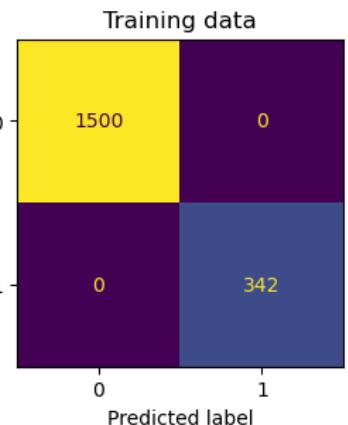
Ограничиваем глубину дерева

```
dtr = DecisionTreeRegressor(  
    criterion='squared_error',  
    splitter='best',  
    max_depth = 8,  
    random_state = 1)
```

ИНТЕРПРЕТАЦИЯ ДЕРЕВА



РЕШАЕМ ЗАДАЧУ РЕГРЕССИИ



Дерево неограниченной глубины

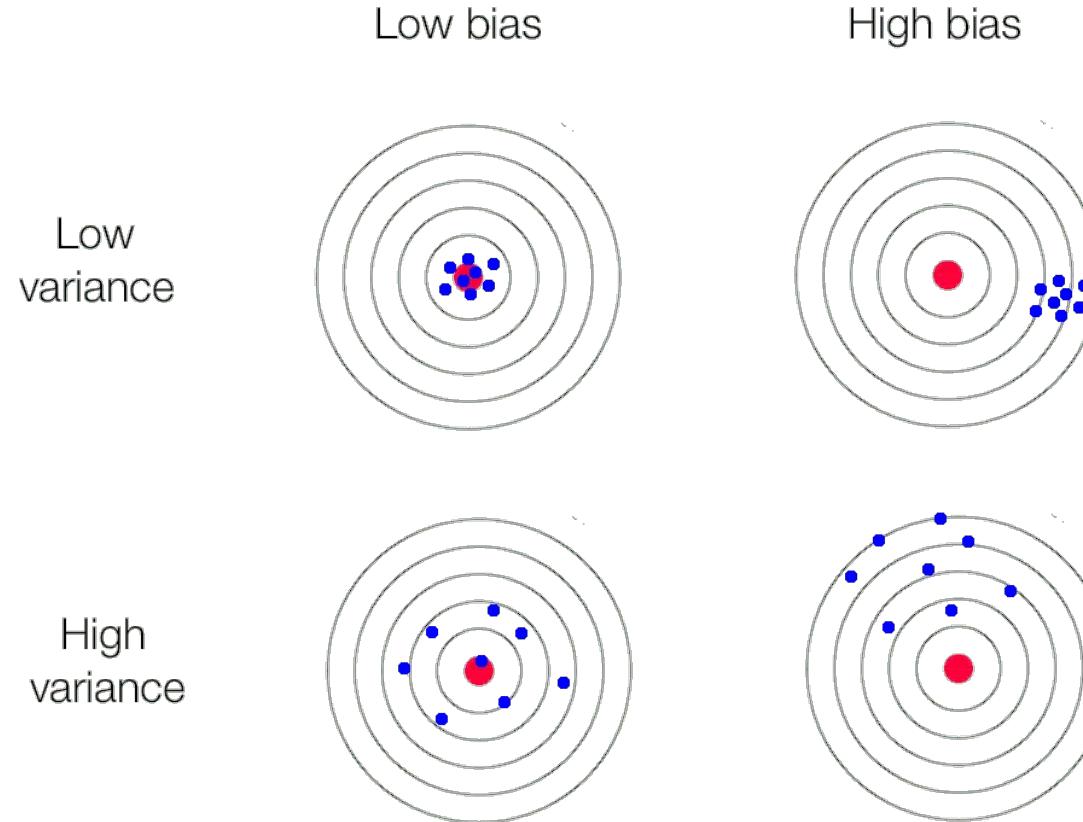
Дерево глубиной `max_depth = 5`

КЕЙСЫ ИЗ ПРЕДЫДУЩЕЙ ЛЕКЦИИ

См. jupyter блокнот Lecture2.ipynb

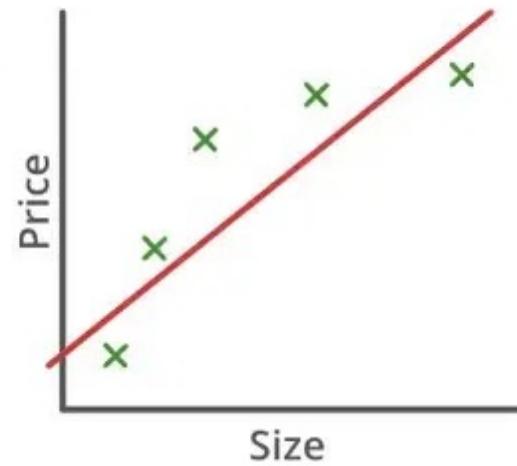
		Линейные модели		MLP (10,2)		Decision Tree (max_depth = None)		Decision Tree (max_depth = d)	
		Train	Test	Train	Test	Train	Test	Train	Test
Регрессия (стоимость жилья)	MSE	0.518	0.720	0.475	0.689	0.0	0.501	0.321	0.422
	RMSE	0.556	0.746	0.494	0.703	0.0	0.708	0.566	0.650
Классификация (банкроты)	Accuracy	0.824	0.837	0.835	0.831	1.000	0.737	0.860	0.826
	AUC ROC	0.540	0.580	0.610	0.620	1.000	0.638	0.693	0.670

КОМПРОМИСС: СМЕЩЕНИЕ И ДИСПЕРСИЯ

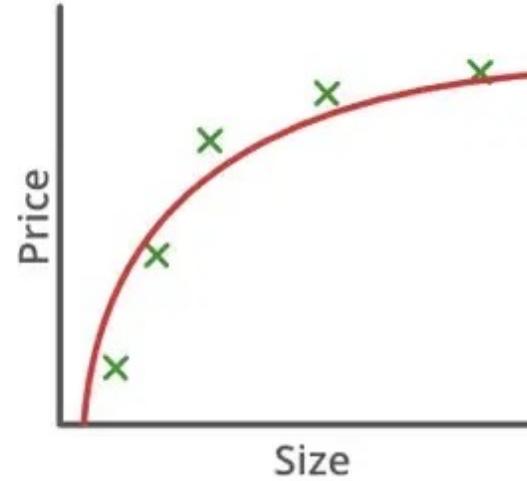


Смещение (bias) – это погрешность оценки, возникающая в результате ошибки в алгоритме обучения. Большое смещение – результат **недообучения (underfitting)**, когда пропущены связи между признаками и целевой переменной.

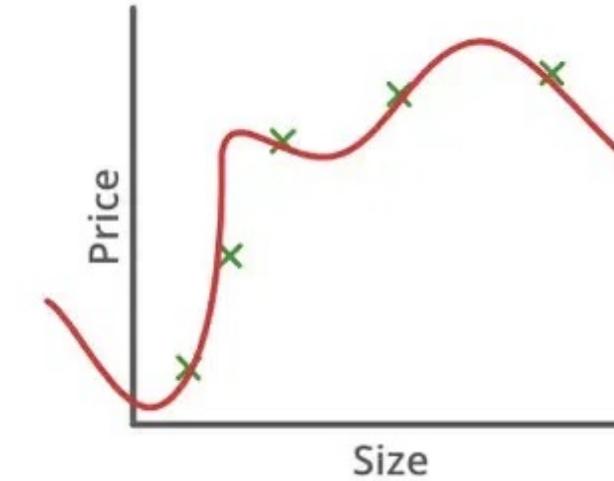
Разброс / дисперсия (variance) – это ошибка, возникающая из-за настройки алгоритма на малые отклонения (шум) в тренировочном наборе (**переобучение / overfitting**).



High Bias
(Underfitting)



Low Bias, Low Variance
(Goodfitting)



High Variance
(Overfitting)



<https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>

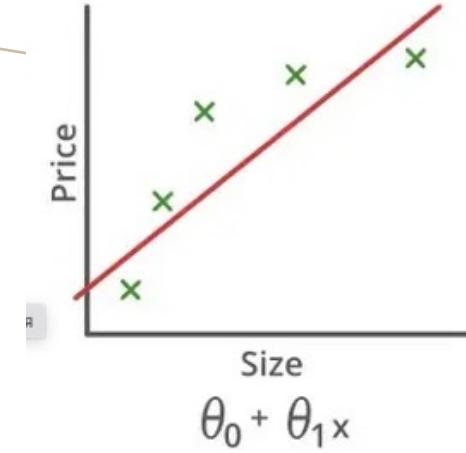
UNDERFITTING

ПРИЧИНЫ

- Модель слишком проста, поэтому она не может отразить сложную структуру данных.
- Входные признаки, используемые для обучения модели, не являются адекватным представлением базовых факторов, влияющих на целевую переменную.
- Размер используемого набора обучающих данных недостаточен.
- Для предотвращения избыточной подгонки используется чрезмерная регуляризация, которая не позволяет модели хорошо отражать данные.
- Признаки не масштабируются.

ТЕХНИКИ СНИЖЕНИЯ

- Повышение сложности модели.
- Увеличение продолжительности обучения.
- Увеличение числа признаков, выполнив инженерию признаков.
- Удаление шума из данных.



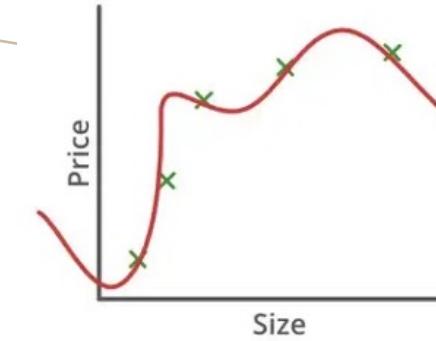
OVERFITTING

ПРИЧИНЫ

- Высокая дисперсия и низкая погрешность обучающих данных.
- Малый размер обучающих данных.
- Модель слишком сложная.

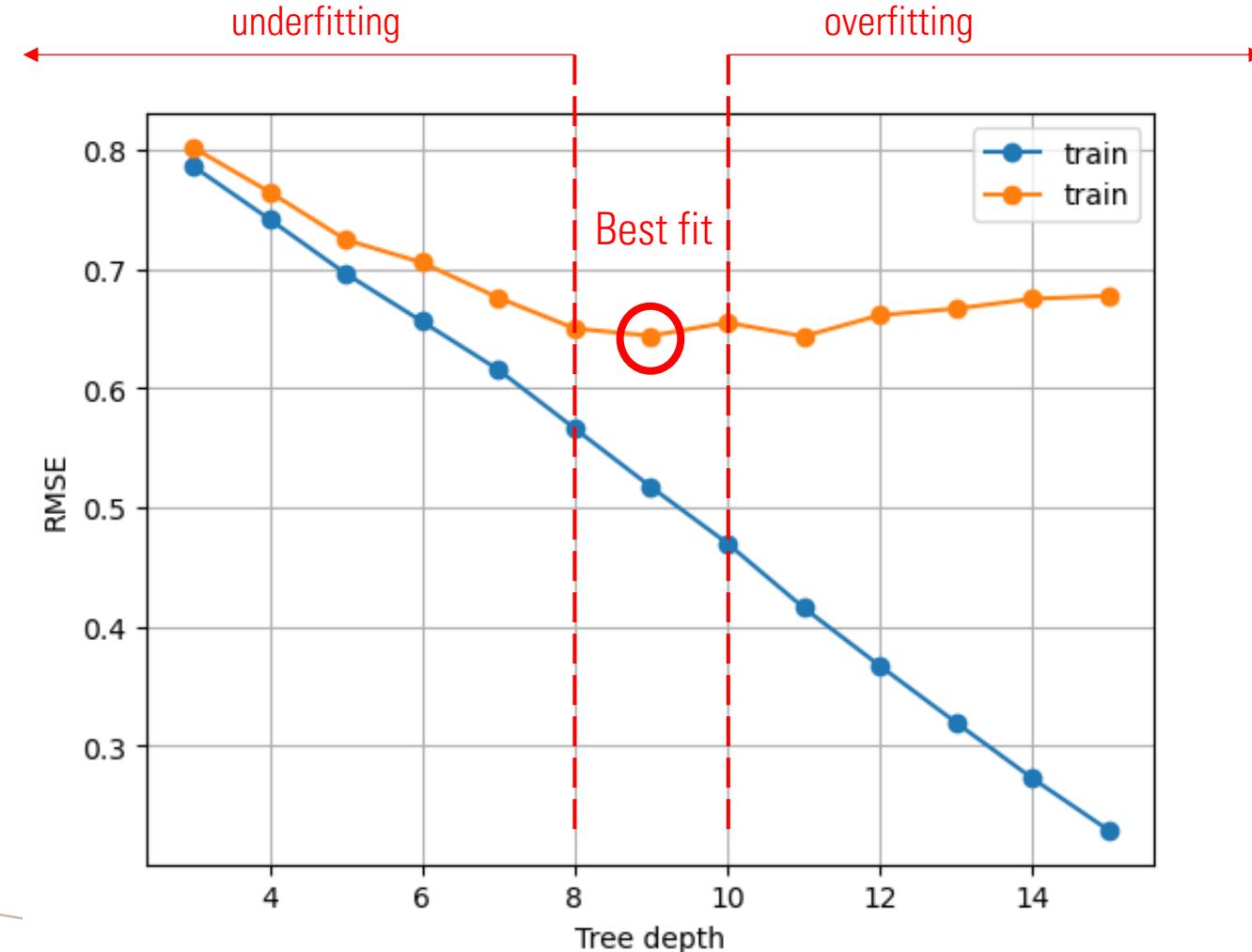
ТЕХНИКИ СНИЖЕНИЯ

- Увеличение объема обучающих данных.
- Снижение размерности / отбор признаков
- Уменьшение сложность модели.
- Ранняя остановка на этапе обучения.
- Регуляризация – штраф за сложность модели.



КОНТРОЛЬ КАЧЕСТВА МОДЕЛИ

Модель недообучена.
При увеличении
сложности модели
ошибка на тестовых
данных снижается.



Модель переобучена.
При увеличении
сложности модели
ошибка на тестовых
данных увеличивается.

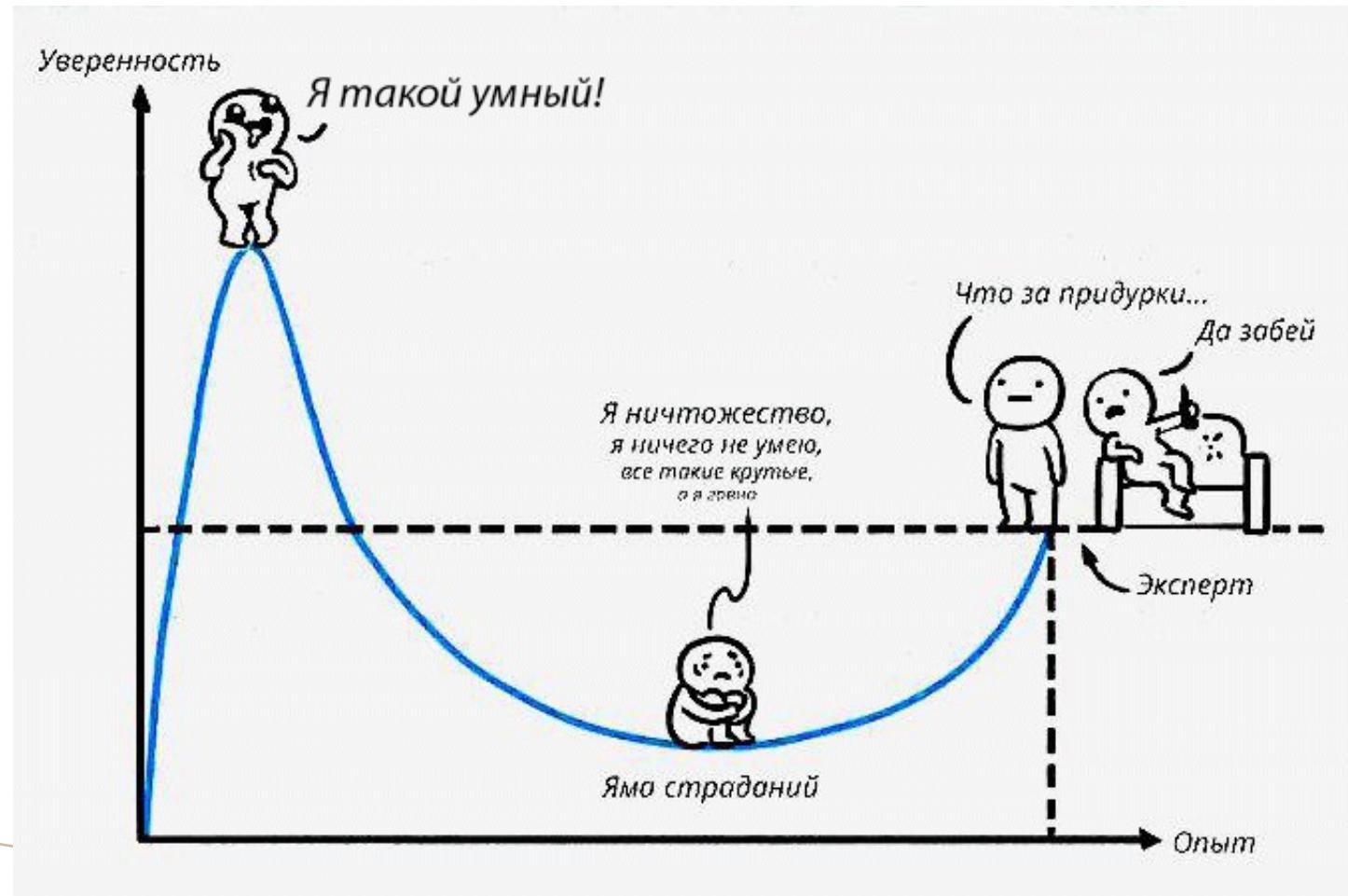
КОГНИТИВНЫЕ СПОСОБНОСТИ ЧЕЛОВЕКА



Герд Гигеренцер

- Реальные когнитивные модели не рассчитывают наиболее рациональное решение из совокупности имеющейся в распоряжении информации (см. процесс принятия решений Г. Саймона).
- Решения прежде всего принимаются интуитивно на основе эмпирических закономерностей, которым рациональные стратегии принятия решений подчиняются в качестве поздних вспомогательных средств.
- Take the best: возьмите лучший критерий и решите — если нет большой разницы, возьмите второй критерий и так далее (~ дерево принятия решений). Такой подход ранее обычно объяснялся как нерациональное поведение.
- Человеческий мозг решает дилемму в случае разреженных плохо описанных тренировочных наборов, полученных в результате личного опыта, путём использования эвристики высокого смещения/низкой дисперсия. Это отражает факт, что подход с нулевым смещением плохо обобщается на новые ситуации, а также беспричинно предполагает точное знание состояния мира

ЭФФЕКТ ДАННИНГА-КРЮГЕРА



РАЗЛОЖЕНИЕ ОШИБКИ

$$\mathbb{E}[(y - \hat{f}(x))^2] = (\text{Bias}[\hat{f}(x)])^2 + \text{Var}[\hat{f}(x)] + \sigma^2$$

$\text{Bias}[\hat{f}(x)] = \mathbb{E}[\hat{f}(x) - y]$

ошибка, вызванная упрощением модели

Неустранимая ошибка

$$\text{Var}[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)^2] - (\mathbb{E}[\hat{f}(x)])^2$$

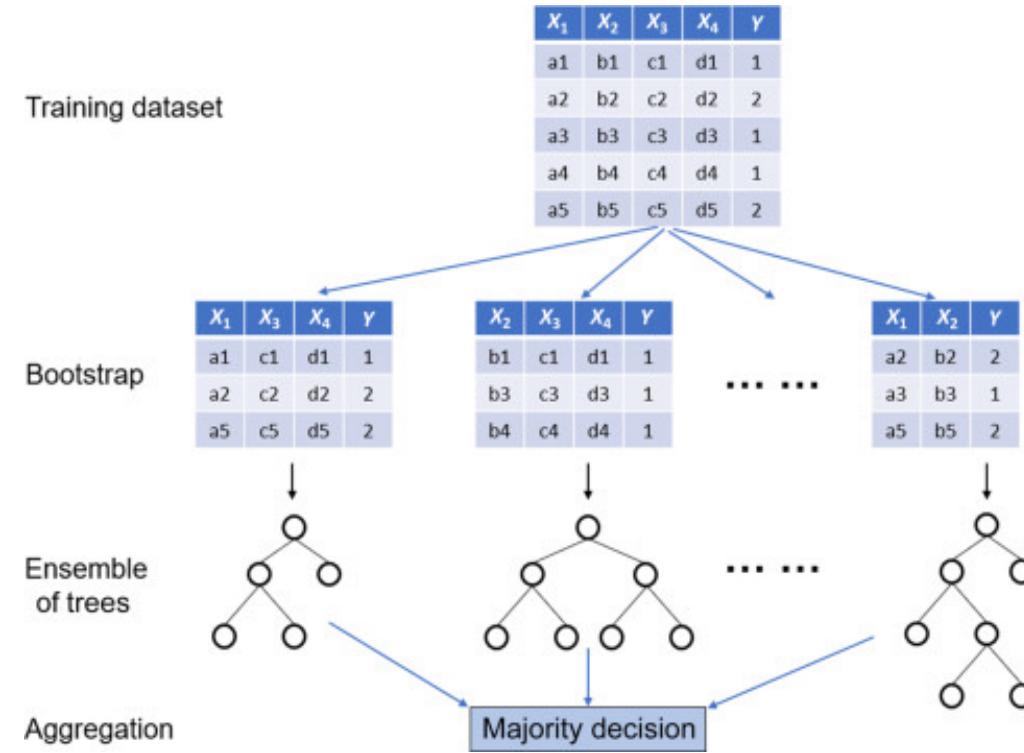
Как далеко $\hat{f}(x)$ уходит от среднего значения

*КОМПОЗИЦИИ (АНСАМБЛИ)
МОДЕЛЕЙ*

BAGGING / RANDOM FOREST



Leo Breiman
(1928 – 2005)



Идея в том, что усреднение моделей должно снизить $\text{Var}[\hat{f}(x)]$, не изменяя $\text{Bias}[\hat{f}(x)]$
(но только тогда, когда деревья максимально "независимы")

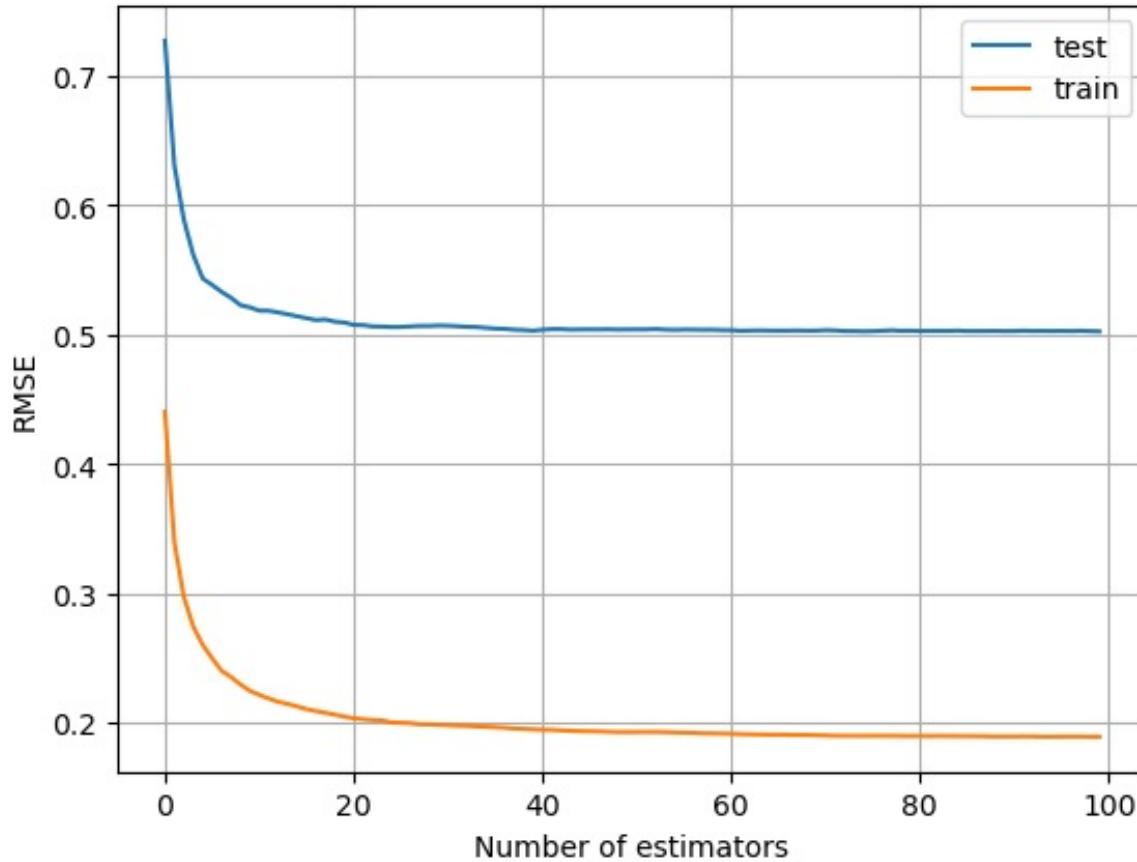
ДЕМО

		Decision Tree (max_depth = d)		Random Forest	
		Train	Test	Train	Test
Регрессия (стоимость жилья)	MSE	0.321	0.422	0.036	0.252
	RMSE	0.566	0.650	0.189	0.502
Классификация (банкроты)	Accuracy	0.860	0.826	1.000	0.839
	AUC ROC	0.693	0.670	1.000	0.634

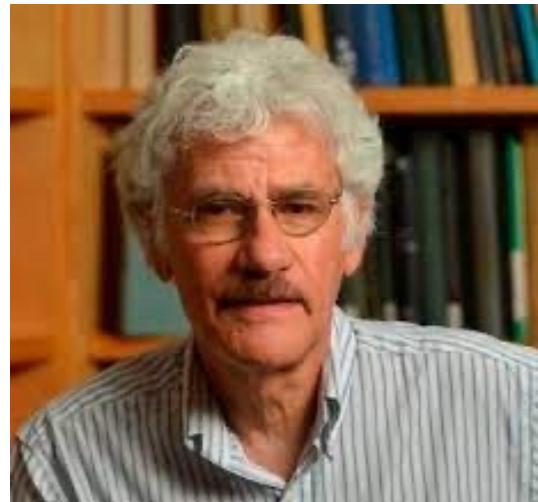
Переобучение.
Необходимо подбирать гиперпараметры модели.

```
rfr = RandomForestRegressor(n_estimators=100,  
                           criterion='squared_error',  
                           max_depth=None,  
                           max_features=1.0,  
                           bootstrap = True,  
                           max_samples = None,  
                           random_state = 1)  
# кол-во деревьев  
# критерий информативности  
# max глубина дерева  
# кол-во признаков при формировании bootstrap выборки  
# используем bootstrap  
# размер bootstrap выборки  
# фиксация генератора случ. чисел для повторяемости
```

ЗАВИСИМОСТЬ ОШИБКИ ОТ ЧИСЛА ДЕРЕВЬЕВ

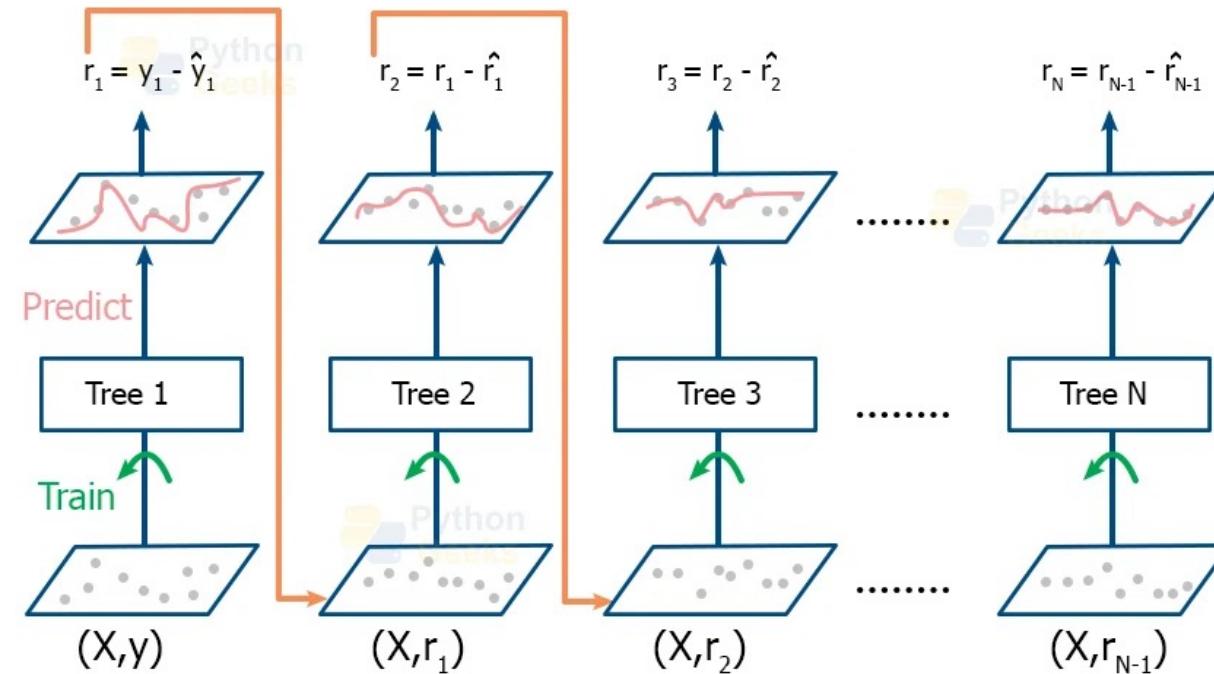


GRADIENT BOOSTING



Jerome H. Friedman

Working of Gradient Boosting Algorithm



<https://pythongeeks.org/gradient-boosting-algorithm-in-machine-learning/>

Идея: пусть каждое следующее дерево исправляет ошибки ансамбля, построенного на предыдущем шаге.
Обучаясь на величину ошибки, модель на самом деле обучается на градиенте функции потерь на предыдущем шаге.

GRADIENT BOOSTING

Пусть мы имеем ансамбль m деревьев

$$F_m = \sum_{j=1}^m f_j$$

Функция потерь (MSE):

$$L(y, F_m) = \sum_{i=1}^n (y_i - F_m(x_i))^2$$

Градиент:

$$\nabla L(y, F_m) = 2(y_i - F_m(x_i))$$

Целевая переменная, на которой обучается
 $m + 1$ дерево:

$$y_i - F_m(x_i)$$

Функция потерь может быть любая (единственное условие – дифференцируемость). Таким образом, GB – это универсальный фреймворк, позволяющий построить модели для разных задач / функций потерь.

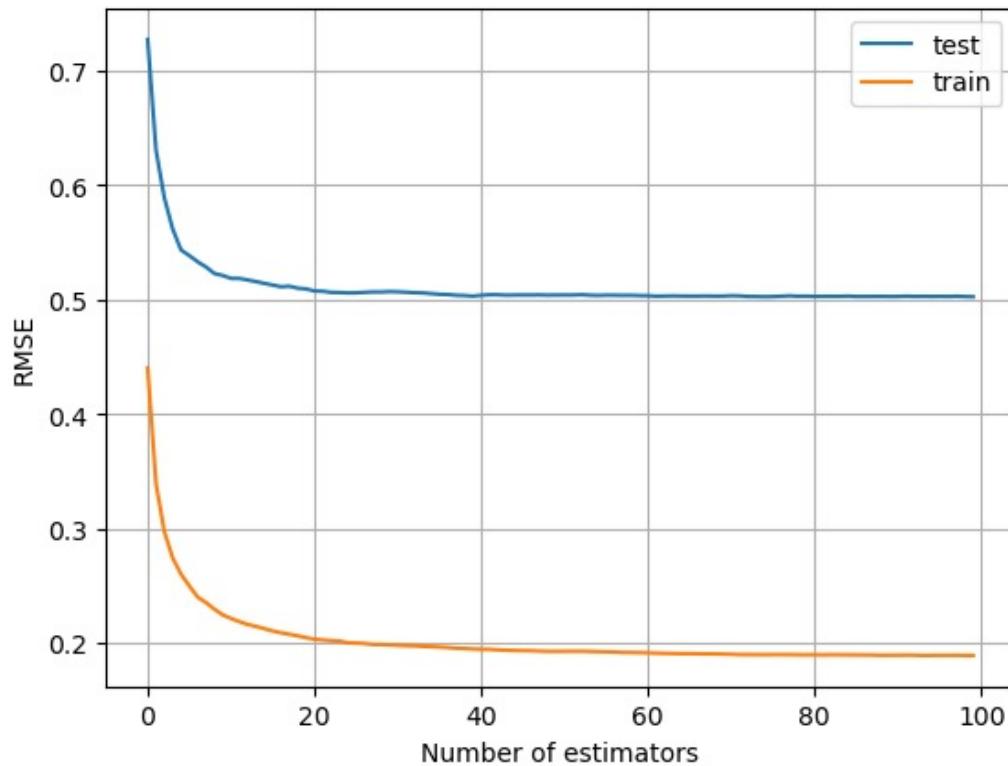
ДЕМО

		Decision Tree (max_depth = d)		Random Forest		Gradient Boosting	
		Train	Test	Train	Test	Train	Test
Регрессия (стоимость жилья)	MSE	0.321	0.422	0.036	0.252	0.174	0.246
	RMSE	0.566	0.650	0.189	0.502	0.418	0.496
Классификация (банкроты)	Accuracy	0.860	0.826	1.000	0.839	0.971	0.824
	AUC ROC	0.693	0.670	1.000	0.634	0.923	0.648

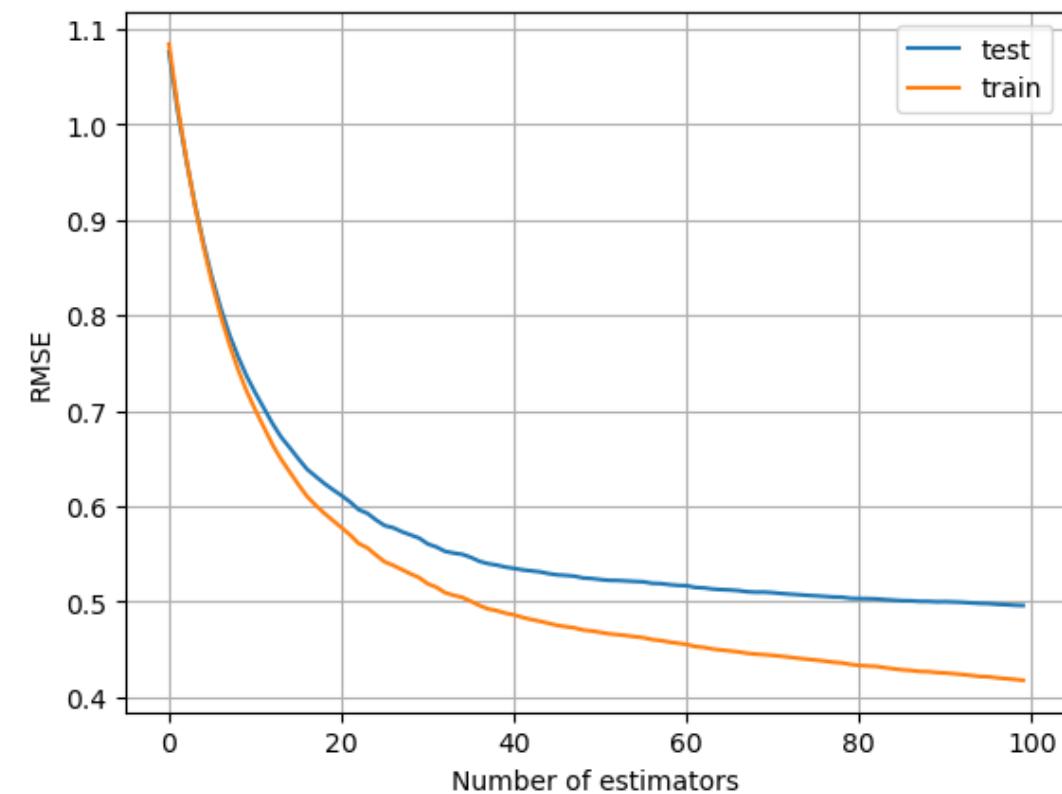
```
gbr = GradientBoostingRegressor(loss='squared_error',  
                                 learning_rate = 0.1,  
                                 n_estimators = 100,  
                                 criterion = 'friedman_mse',  
                                 max_depth = 5,  
                                 random_state = 1) # loss функция  
# шаг обучения  
# количество итераций / деревьев  
# критерий информативности  
# максимальная глубина дерева  
# фиксация генератора случ. чисел для повторяемости
```

ЗАВИСИМОСТЬ ОТ ЧИСЛА ДЕРЕВЬЕВ

Random Forest



Gradient Boosting



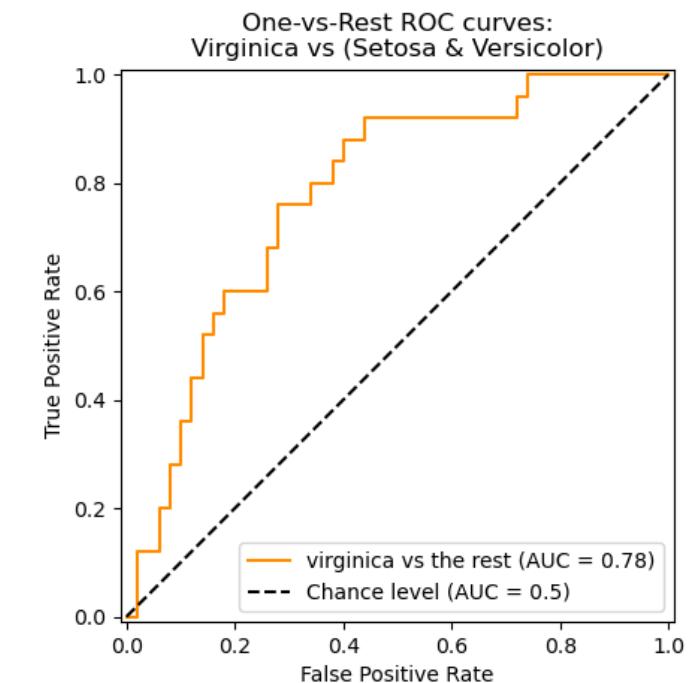
РЕАЛИЗАЦИИ GRADIENT BOOSTING

- XGBoost (DMLC)
 - <https://xgboost.readthedocs.io/en/stable/>
 - Python, R, Java, Ruby, Swift, Julia, C, C++, CLI
 - GPU
 - boosting type = {'gbtree', 'dart', 'gblinear'}
- LightGBM (Microsoft)
 - <https://lightgbm.readthedocs.io/en/stable/>
 - C, Python, R
 - GPU
 - boosting type = {'gbdt', 'dart', 'rf'}
- CatBoost (Yandex)
 - <https://catboost.ai>
 - Python, R
 - GPU
 - Categorical features, quantile regression

МЕТРИКИ КАЧЕСТВА И ВАЛИДАЦИЯ МОДЕЛЕЙ

МЕТРИКИ КАЧЕСТВА КЛАССИФИКАЦИИ

		Predicted Class		Sensitivity $\frac{TP}{(TP + FN)}$	Specificity $\frac{TN}{(TN + FP)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$
		Positive	Negative			
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error			
	Negative	False Positive (FP) Type I Error	True Negative (TN)			
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$			

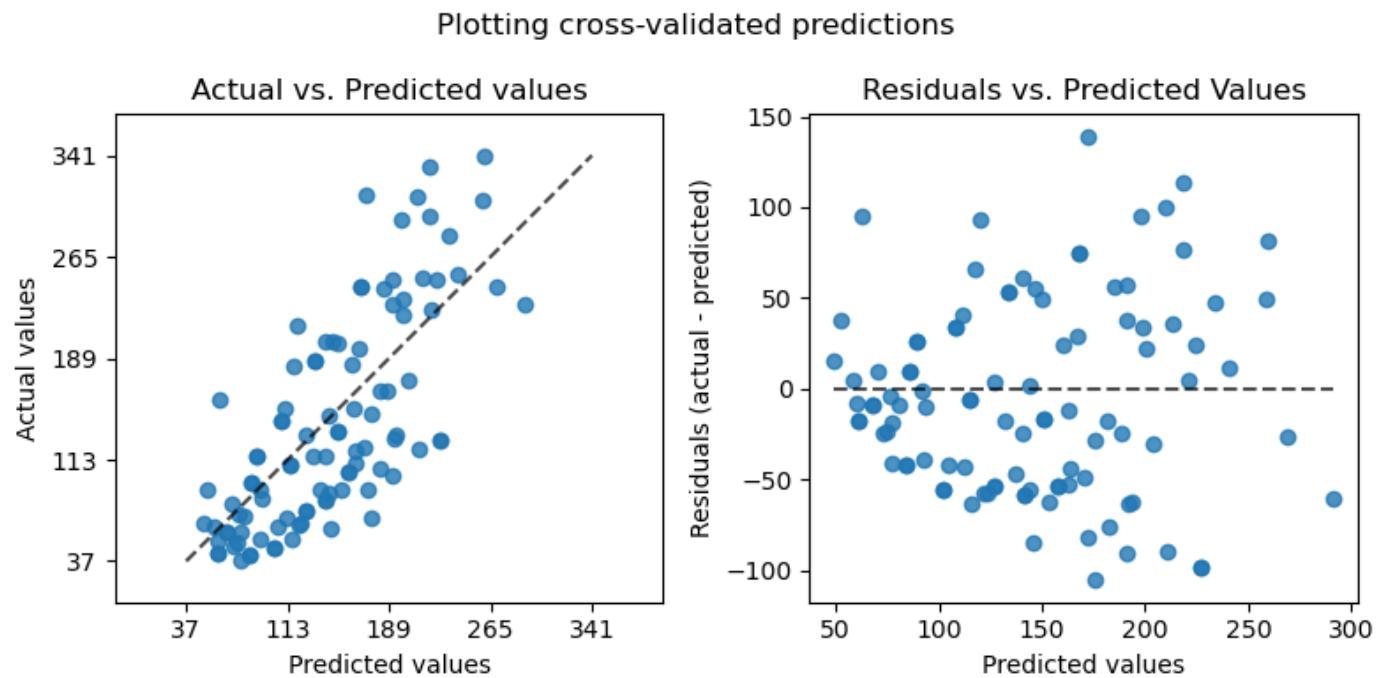


ROC AUC

МЕТРИКИ КАЧЕСТВА РЕГРЕССИИ

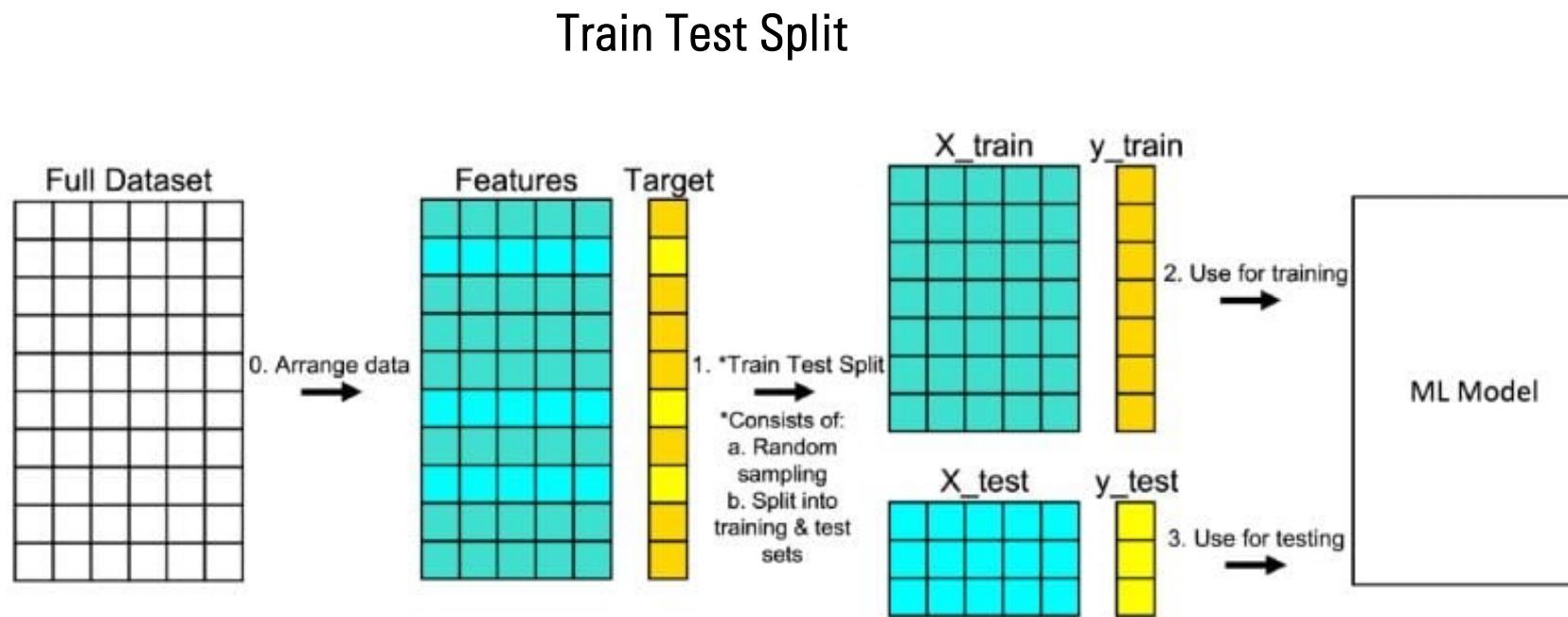
$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

$$\text{MAPE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)}$$

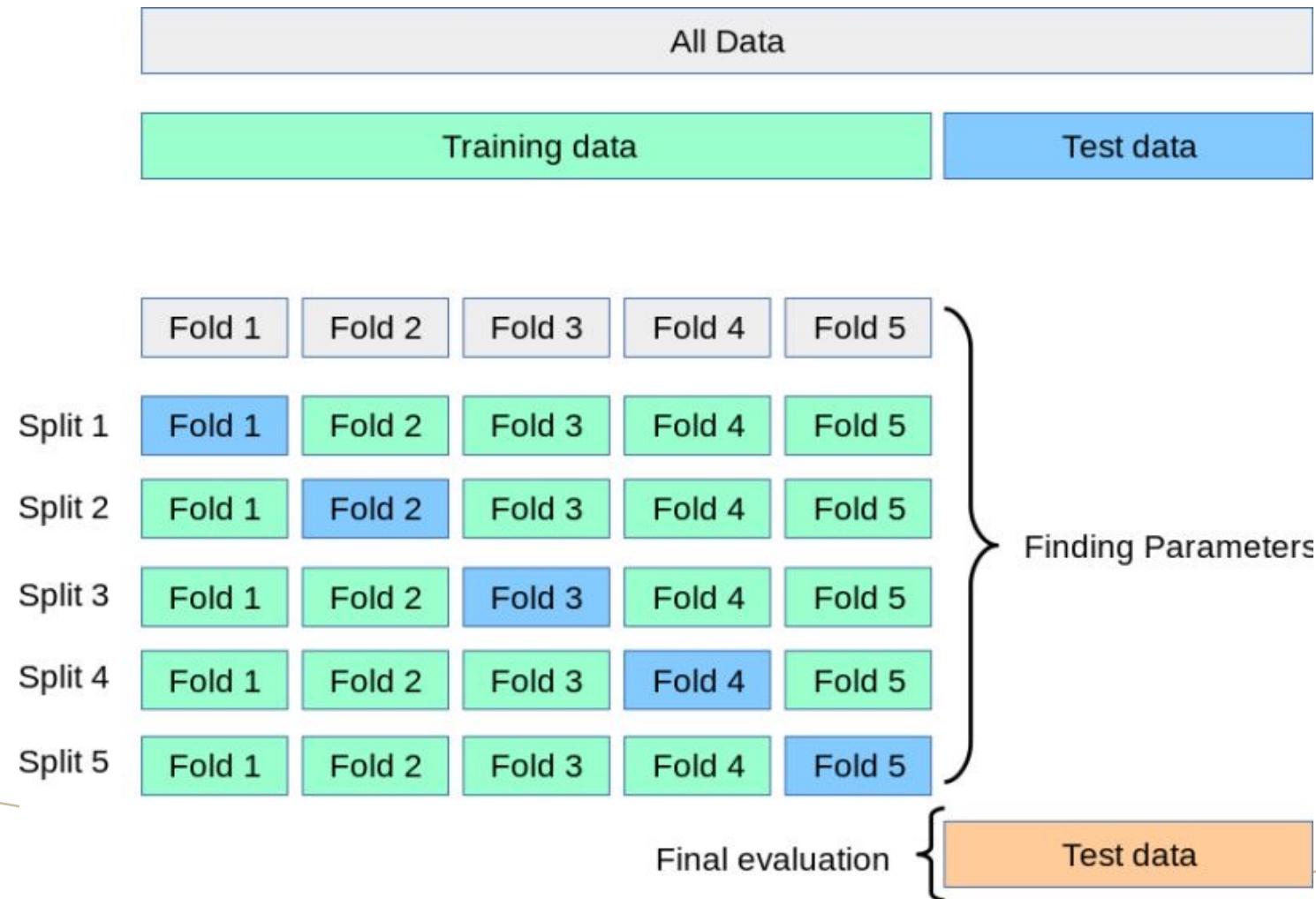


Визуализация качества модели

ОЦЕНКА КАЧЕСТВА МОДЕЛИ



КРОССВАЛИДАЦИЯ



ДЕМО (CV = 5)

		Decision Tree	Random Forest	Gradient Boosting
Регрессия (стоимость жилья)	MSE	0.609 (0.064)	0.432 (0.068)	0.439 (0.113)
Классификация (банкроты)	AUC ROC	0.735 (0.016)	0.772 (0.032)	0.767 (0.031)

ВОПРОСЫ?

