курс «Машинное обучение»

# Ансамбли алгоритмов машинного обучения

Александр Дьяконов

# План

**Ансамбли алгоритмов: примеры и обоснование**

**Повышение разнообразия в ансамблях**

**Комитеты (голосование, Voting Ensembles), усреднение**
**Бэгинг (Bagging)**
**Пэстинг (Pasting)**
**Случайные подпространства (Random Subspaces)**
**Случайные патчи (Random Patches)**
**Cross-Validated Committees**
**Стекинг (Stacking)**
**Блендинг (Blending)**

**Случайные леса (Random Forests)**

**Бустинг (Boosting)**

# Ансамбль алгоритмов (Ensemble / Multiple Classifier System)

**– алгоритм, который состоит из нескольких алгоритмов машинного обучения (базовых алгоритмов – base learners)**

**простой ансамбль в регрессии:**

$$a(x) = \frac{1}{n}\left(b_1(x) + \ldots + b_n(x)\right)$$

**простой ансамбль в классификации:**

$$a(x) = \text{mode}\left(b_1(x), \ldots, b_n(x)\right)$$

**комитет большинства**

**В чём может быть усложнение?**

# Ансамбль алгоритмов

$$a(x) = b\big(b_1(x),\ldots,b_n(x)\big)$$

$b$ – **мета-алгоритм (meta-estimator),**

$b_i$ – **базовые алгоритмы (base learners)**

**в бустинге – слабые (weak)**

# Реализация в scikit-learn

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier

clf1 = LogisticRegression(multi_class='multinomial', random_state=1)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = GaussianNB()

eclf1 = VotingClassifier(estimators=[
     ('lr', clf1), ('rf', clf2), ('gnb', clf3)],
     voting='hard', # по большинству или soft - сумме вероятностей
     weights=None, # веса алгоритмов
     flatten_transform=True) # для мягкого голосования - форма ответа

eclf1 = eclf1.fit(X, y)
print(eclf1.predict(X))
```

# Ошибка суммы регрессоров: теоретическое обоснование

**Если ответы регрессоров на объекте – независимые случайные величины с одинаковым матожиданием и дисперсией**

$$\xi = \frac{1}{n}\left(\xi_1 + \ldots + \xi_n\right)$$

$$\mathrm{E}\xi = \frac{1}{n}\left(\mathrm{E}\xi_1 + \ldots + \mathrm{E}\xi_n\right) = \mathrm{E}\xi_i$$

$$\mathbf{D}\xi = \frac{1}{n^2}\left(\mathbf{D}\xi_1 + \ldots + \mathbf{D}\xi_n\right) = \frac{\mathbf{D}\xi_i}{n}$$

**А если есть корреляция между базовыми алгоритмами?**

# Ошибка комитета большинства: теоретическое обоснование

**Пусть три (независимых) классификатора на два класса с вероятностью ошибки** $p$

**Пусть верный ответ – 0**

$$\left.\begin{array}{ll}(0,0,0) & (1-p)(1-p)(1-p)\\ (1,0,0) & p(1-p)(1-p)\\ (0,1,0) & (1-p)p(1-p)\\ (0,0,1) & (1-p)(1-p)p\end{array}\right\}$$

**верный ответ**

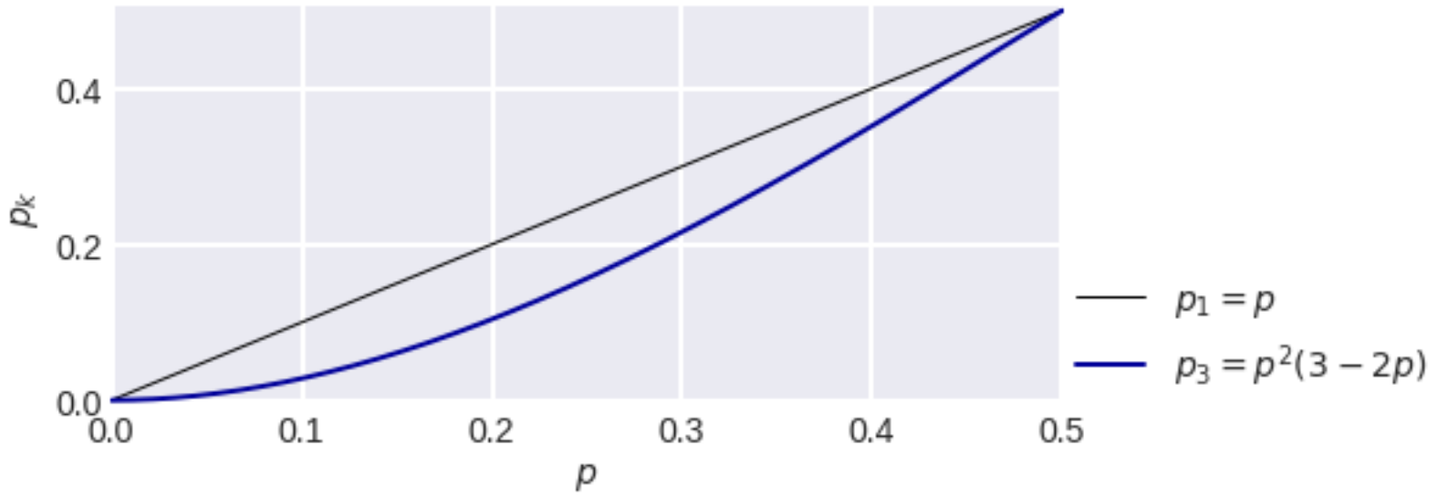
$$\left.\begin{array}{ll}(1,1,1) & ppp\\ (1,1,0) & pp(1-p)\\ (0,1,1) & (1-p)pp\\ (1,0,1) & p(1-p)p\end{array}\right\}$$

**ошибка**

**вероятность ошибки**

$$p^3+3(1-p)p^2=p^2(3-2p)$$

# Ошибка комитета большинства




**При малых $p$ ошибка комитета очень мала!**

**При $p = 0.2$ – почти в два раза меньше**

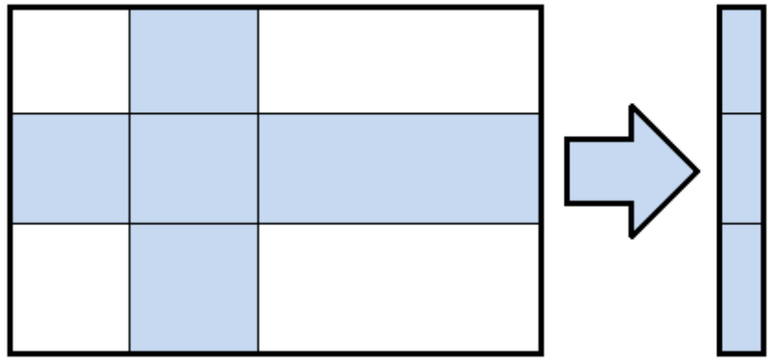
# На практике

**Классификаторы / регрессоры точно не являются независимыми**

**Почему?**

- **Решают одну задачу**
- **Настраиваются на один целевой вектор**
- **Могут быть из одной модели (ну, 2-3 разных)!**

# Повышения разнообразия – что «варьируют»



- **обучающую выборку**
  (бэгинг)
- **признаки**
  (Random Subspaces)
- **целевой вектор**
  (ECOC, f(y))
- **модели**
  (стекинг)
- **алгоритмы в модели**

(разные гиперпараметры, инициализации, snapshot, разные random seed в RF, …)

**И результаты «усредняют» — тоже есть разные способы**

# Способы усреднения: комитеты (голосование, Voting Ensembles)

**голосование по большинству (Majority vote)**

$$a(x) = \text{mode}\left(b_1(x),\ldots,b_n(x)\right)$$

**комитеты единогласия**

**в бинарной задаче классификации** – $a(x) = \min\left(b_1(x),\ldots,b_n(x)\right)$

**обнаружение аномалий** – $a(x) = \max\left(b_1(x),\ldots,b_n(x)\right)$

# Способы усреднения: обобщения среднего арифметического

**«среднее арифметическое»**

$$a(x) = \frac{1}{n}\left(b_1(x) + \ldots + b_n(x)\right)$$

**+ любые другие средние (ex: по Колмогорову)**

$$a(x) = \frac{1}{n} f^{-1}\left( f(b_1(x)) + \ldots + f(b_n(x))\right)$$

**Ранговое усреднение (Rank Averaging)**

$$a(x) = \frac{1}{n}\left(\text{rank}(b_1(x)) + \ldots + \text{rank}(b_n(x))\right)$$

**ориентировано на конкретный AUC ROC**

# Способы усреднения: усреднение с весами (weighted averaging)

## Усреднение (регрессия)

$$a(x) = \frac{1}{w_1 + \ldots + w_n}\left(w_1 \cdot b_1(x) + \ldots + w_n \cdot b_n(x)\right)$$

## Голосование (классификация)

$$a(x) = \arg\max_j \left[ \sum_{t:b_t(x)=j} w_t \right]$$

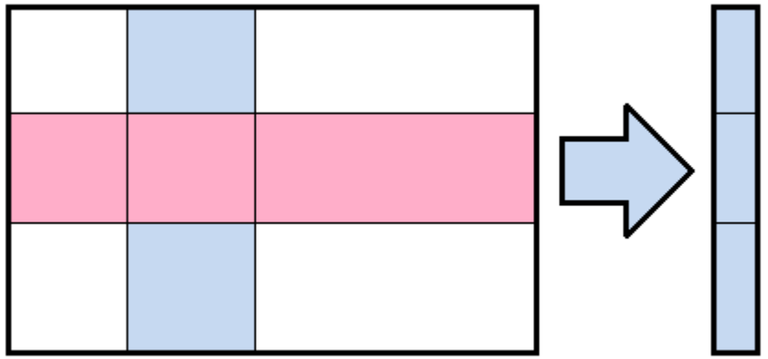
## Feature-Weighted Linear Stacking

### Области компетентности алгоритмов – линейные регрессии

$$a(x) = w_1(x) \cdot b_1(x) + \ldots + w_n(x) \cdot b_n(x) =$$

$$= \sum_t \left( \sum_i w_{ti} x_{[i]} \right) b_t(x) = \sum_{t,i} w_{ti} x_{[i]} b_t(x)$$

# Варьирование выборки: Бэгинг (Bagging)



**bootstrap aggregating**

**каждый базовый алгоритм настраивается на случайной подвыборке обучения**

# Бэгинг (Bagging)

**1. Цикл по** $t$ **(номер базового алгоритма)**

**1.1. Взять подвыборку** $[X', y']$ **обучающей выборки** $[X, y]$

**1.2. Обучить** $t$**-й базовый алгоритм на этой подвыборке:**

$$b_t = \text{fit}(X', y')$$

**2. Ансамбль**

$$a(x) = \frac{1}{n}\left(b_1(x) + \ldots + b_n(x)\right)$$

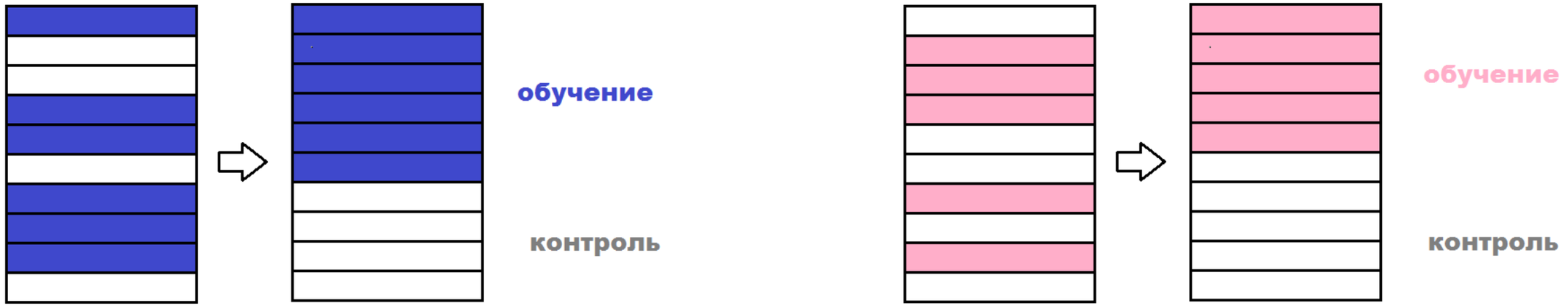
**(для задач регрессии).**

**Каждый базовый алгоритм обучается ~ на 63% данных, остальные называются – out-of-bag-наблюдениями (OOB)**

$$1 - \frac{1}{e} \approx 0.632$$

**~ процедура снижения variance в статистическом обучении**

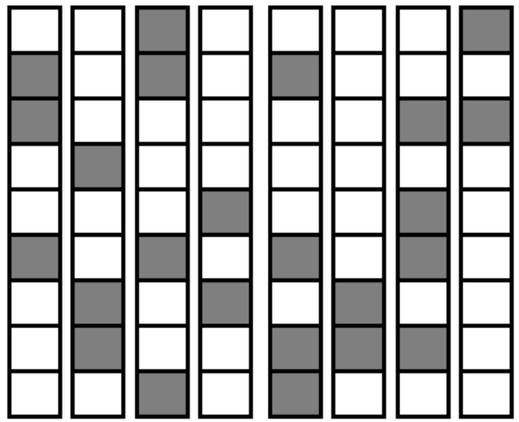
# Бэггинг и OOB (out of bag)




**Выбор объектов для обучения (с помощью бутстрепа), остальные – локальный контроль…**

**OOB-ответы бэгинга (OOB-prediction)**

$$a_{\text{OOB}}(x_j) = \frac{1}{|\{i : x_j \in \text{OOB}_i\}|} \sum_{i: x_j \in \text{OOB}_i} b_i(x_j)$$



**Ответы разных деревьев – можно усреднить и вычислить качество**

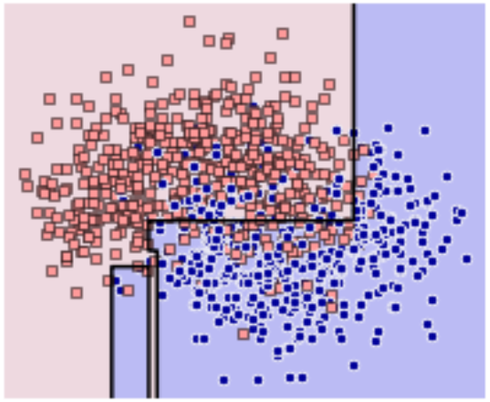
# Реализация в scikit-learn

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier

clf = BaggingClassifier(estimator=SVC(),   # Базовая модель (раньше: base_estimator)
                        n_estimators=10,   # Число алгоритмов в ансамбле
                        max_samples=1.0,   # Размер подобучения (доля или число)
                        max_features=1.0,  # Число / доля признаков для подобучения
                        bootstrap=True,    # Выбирать ли подобучение с возвращением
                        bootstrap_features=False, # Аналогичная опция для признаков
                        oob_score=False,   # Вычислять ли OOB-ошибку
                        warm_start=False)  # Использовать ли в качестве начальных
                                           # приближений старые веса

clf.fit(X, y)
```

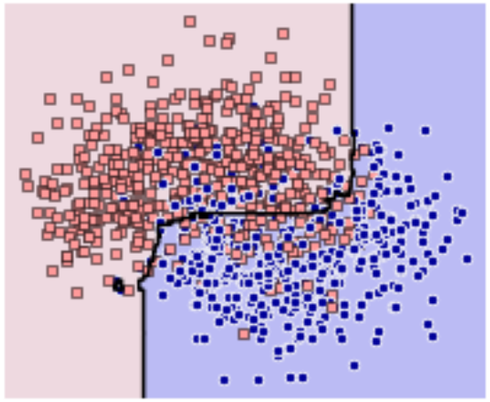
**есть ещё**

**ensemble.BaggingRegressor**

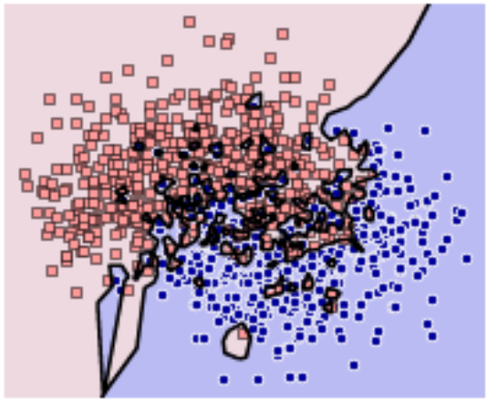
# Примеры бэгинга



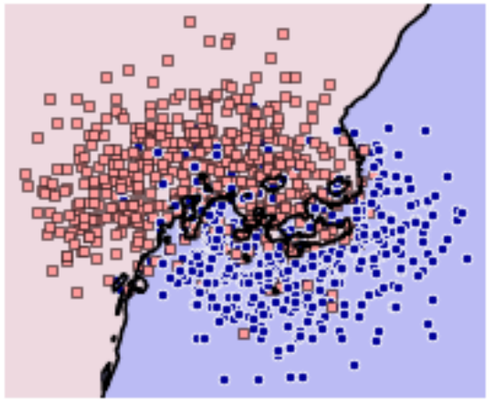
одно дерево



бэгинг 100 деревьев

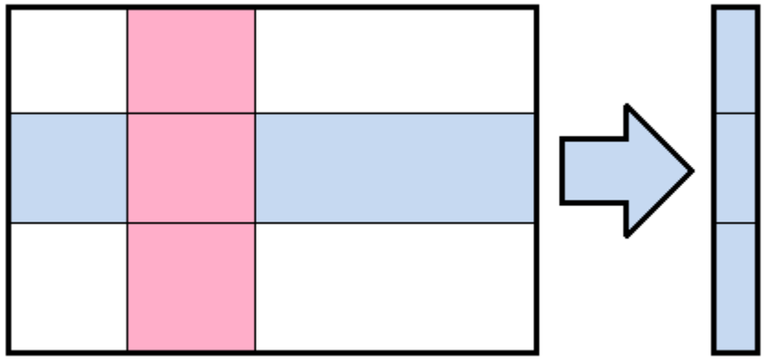


ближайший сосед



бэгинг 100 ближайших соседей

# Варьирование признаков



# Варьирование признаков

| | |
|---|---|
| **Случайные подпространства** (Random Subspaces) | **случайное подмножество признаков** |
| **Бэгинг** (Bagging) | **подвыборка обучающей выборки берётся с помощью бутстрепа** |
| **Пэстинг** (Pasting) | **случайная обучающая подвыборка** |
| **Cross-Validated Committees** | **k обучений на (k-1)-м фолде** |
| **Случайные патчи** (Random Patches) | **одновременно берём случайное подмножество объектов и признаков** |

# Построение случайного леса

1. Выбирается подвыборка `max_samples` (м.б. с повторением) – на ней строится дерево
   чаще всего используется bootstrap

2. Строим дерево
   2.1. Для построения каждого расщепления просматриваем `max_features` случайных признаков
   2.2. Как правило, дерево строится до исчерпания выборки (без прунинга)

**Ответ леса в задачах классификации:**

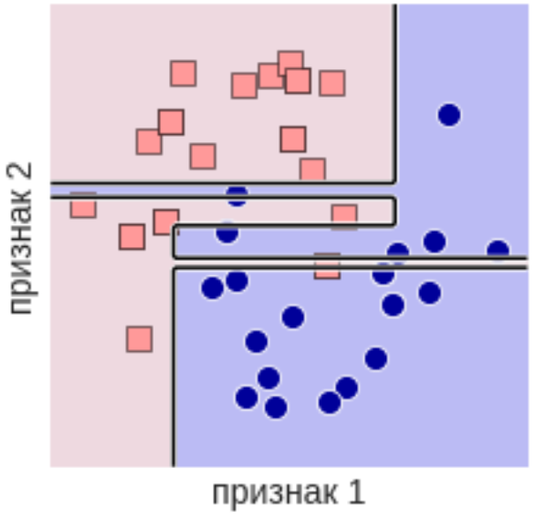
по большинству, вероятность = процент деревьев (R)

сравниваем вероятность с порогом / по максимальной вероятности, вероятность = среднее арифметическое вероятностей в листьях деревьев ансамбля (sklearn)

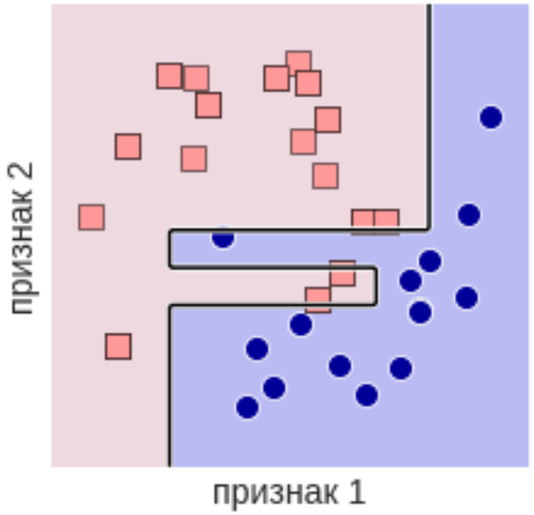
**Ответ леса в задачах регрессии:**

среднее арифметическое

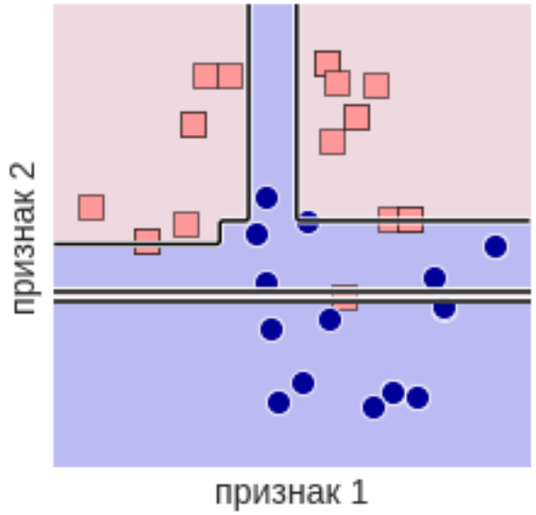
# Случайный лес (Random Forest)



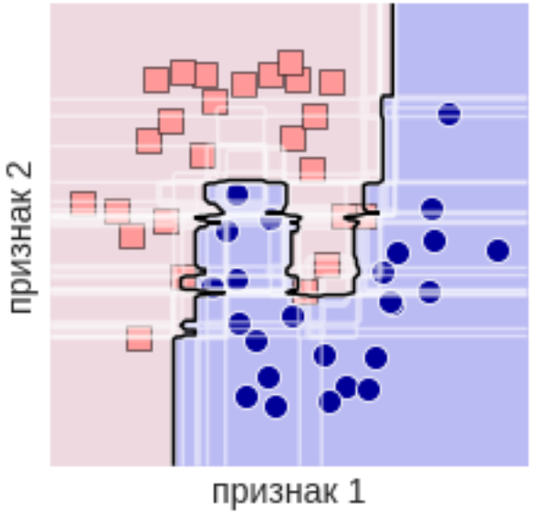

дерево № 1



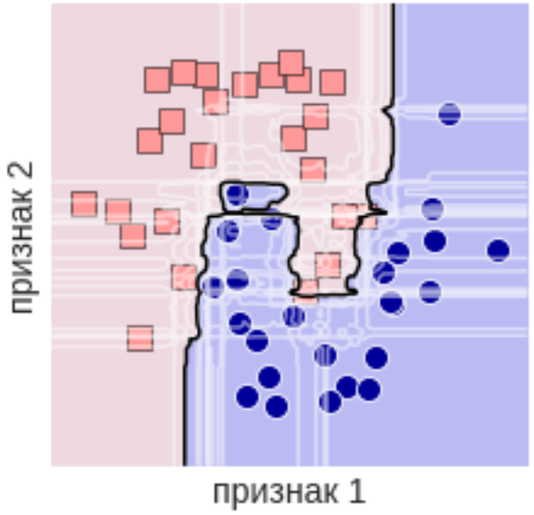

дерево № 2



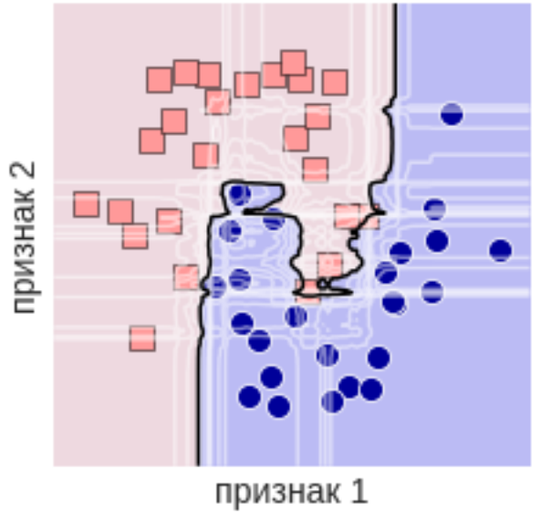

дерево № 3




RF, число деревьев=10




RF, число деревьев=100

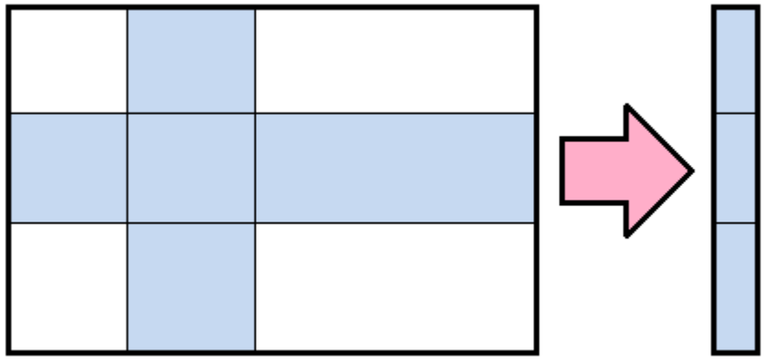


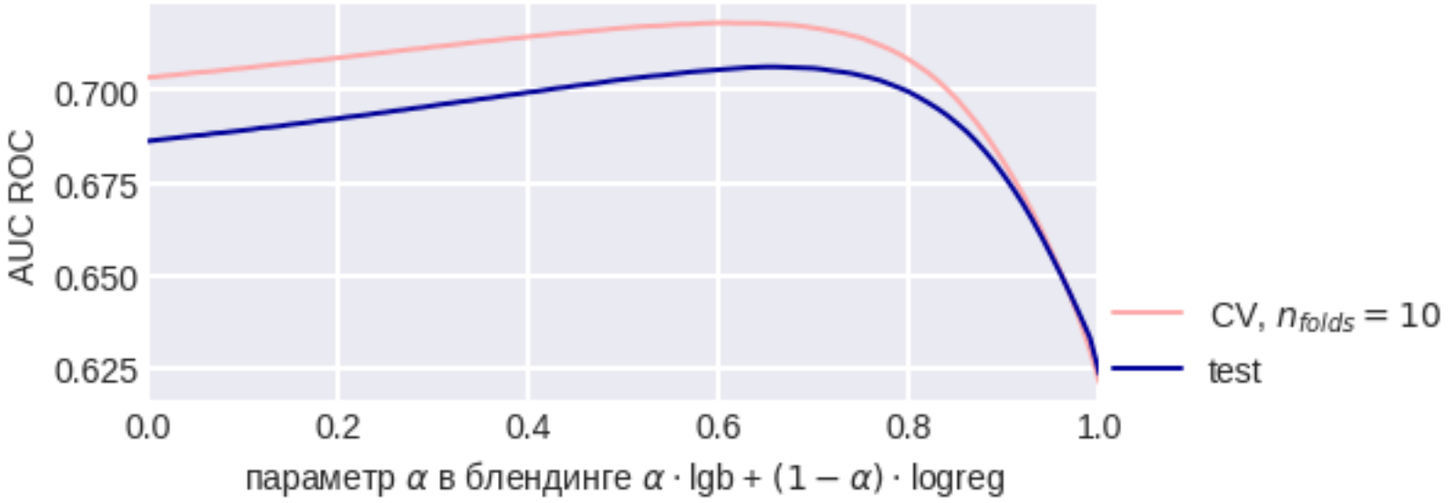

RF, число деревьев=1000

# Реализация случайного леса

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, # число деревьев
                             criterion='gini', # было
                             max_depth=None, # было
                             min_samples_split=2, # было
                             min_samples_leaf=1, # было
                             min_weight_fraction_leaf=0.0, # было
                             max_features='sqrt', # было
                             max_leaf_nodes=None, # было
                             min_impurity_decrease=0.0, # было
                             bootstrap=True, # делать ли бутстреп
                             oob_score=False, # OOB-оценка качества
                             n_jobs=None,
                             random_state=None,
                             verbose=0,
                             warm_start=False, # дополнять ли существующий лес
                             class_weight=None, # было
                             ccp_alpha=0.0, # было
                             max_samples=None) # объём подвыборки (при bootstrap=True)
clf.fit(X, y)
```

# Варьирование моделей: Блендинг (Blending)





# Варьирование моделей + обобщение усреднения: Стекинг (stacking)

**Идея: хорошо усреднять алгоритмы, но почему именно усреднять?**

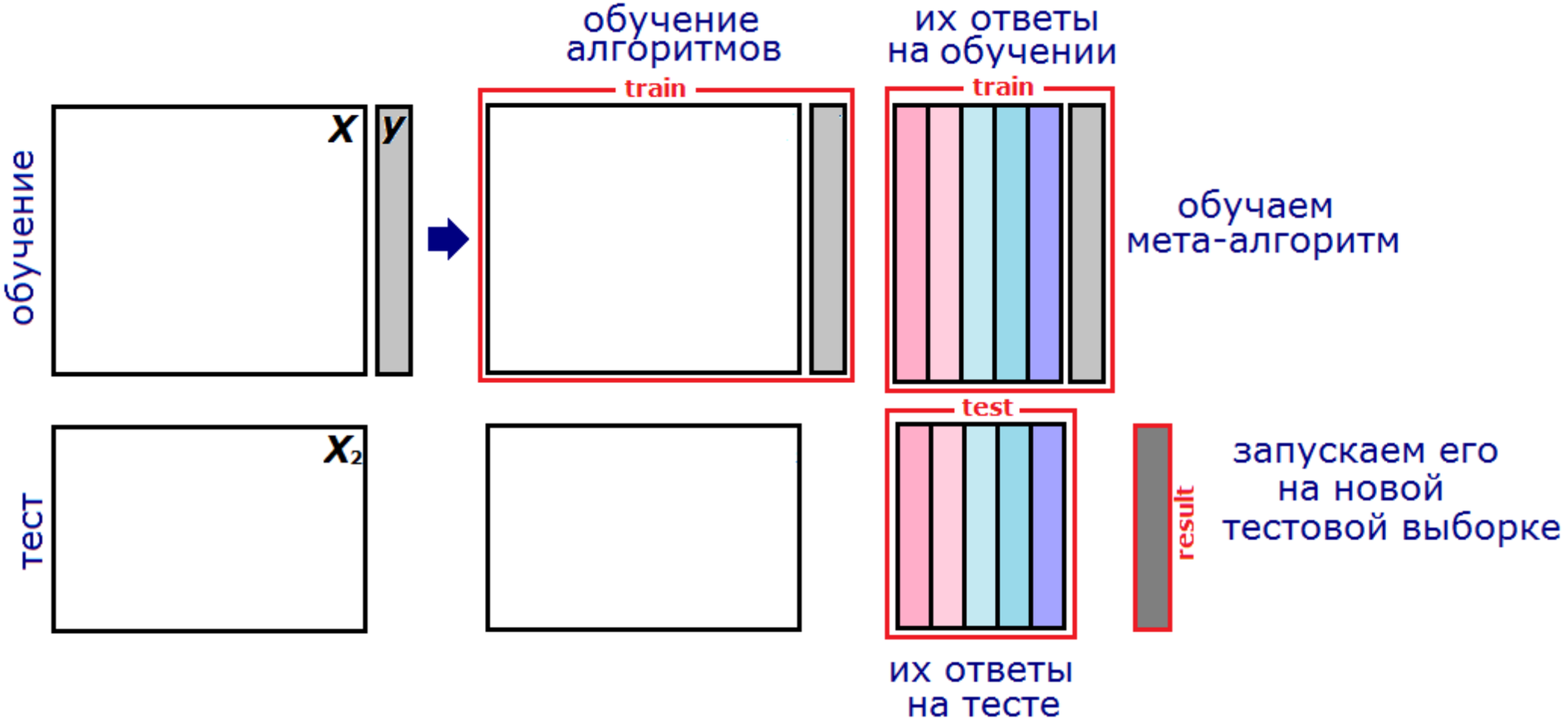
**приходит в голову всем…**

$$a(x) = b\big(b_1(x),\ldots,b_n(x)\big)$$

$b$ **– мета-алгоритм,**
**который нужно отдельно настроить!**

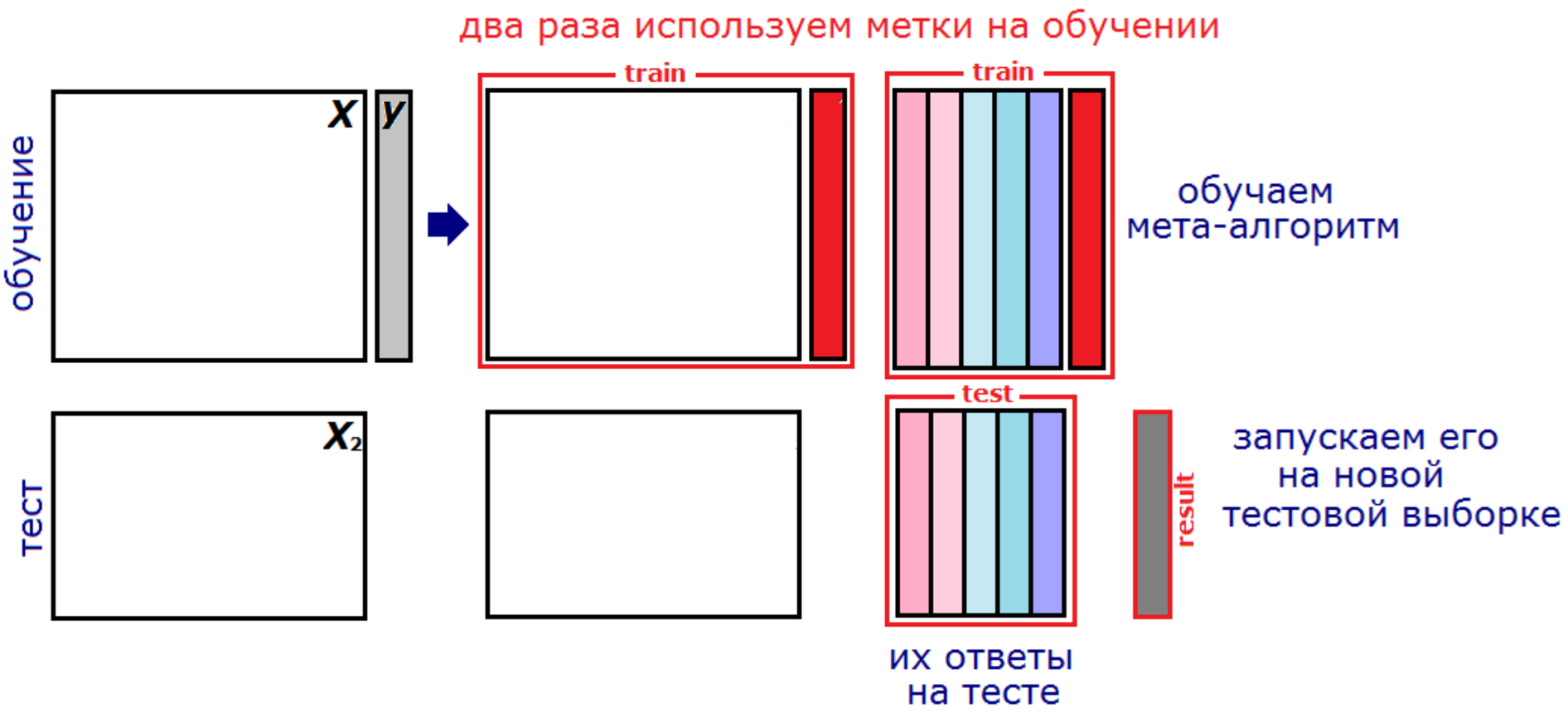
**Д. Волпертом, автором серии теорем «No free lunch…» в 1992 году**

# Наивная форма стекинга

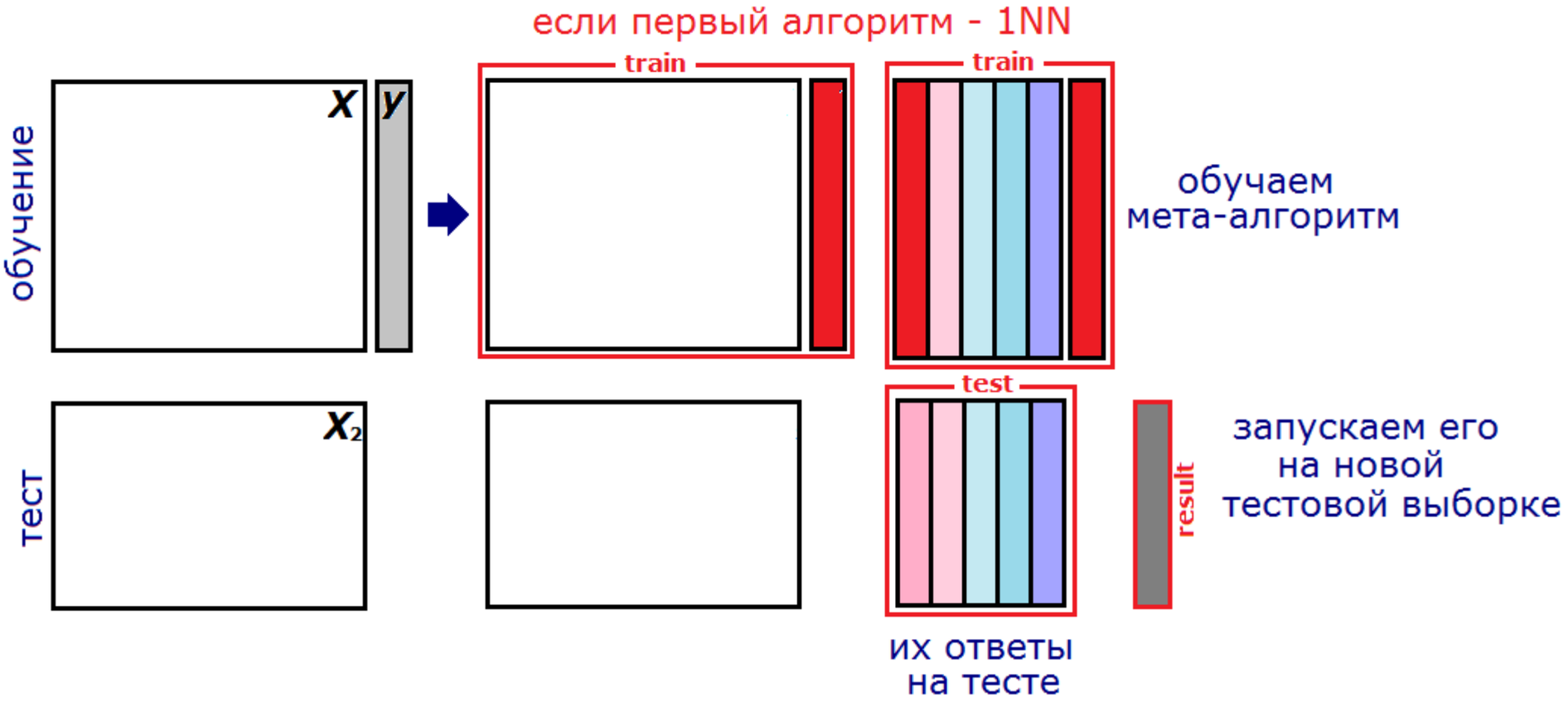



**что здесь неправильно?**

# Наивная форма стекинга



# Наивная форма стекинга

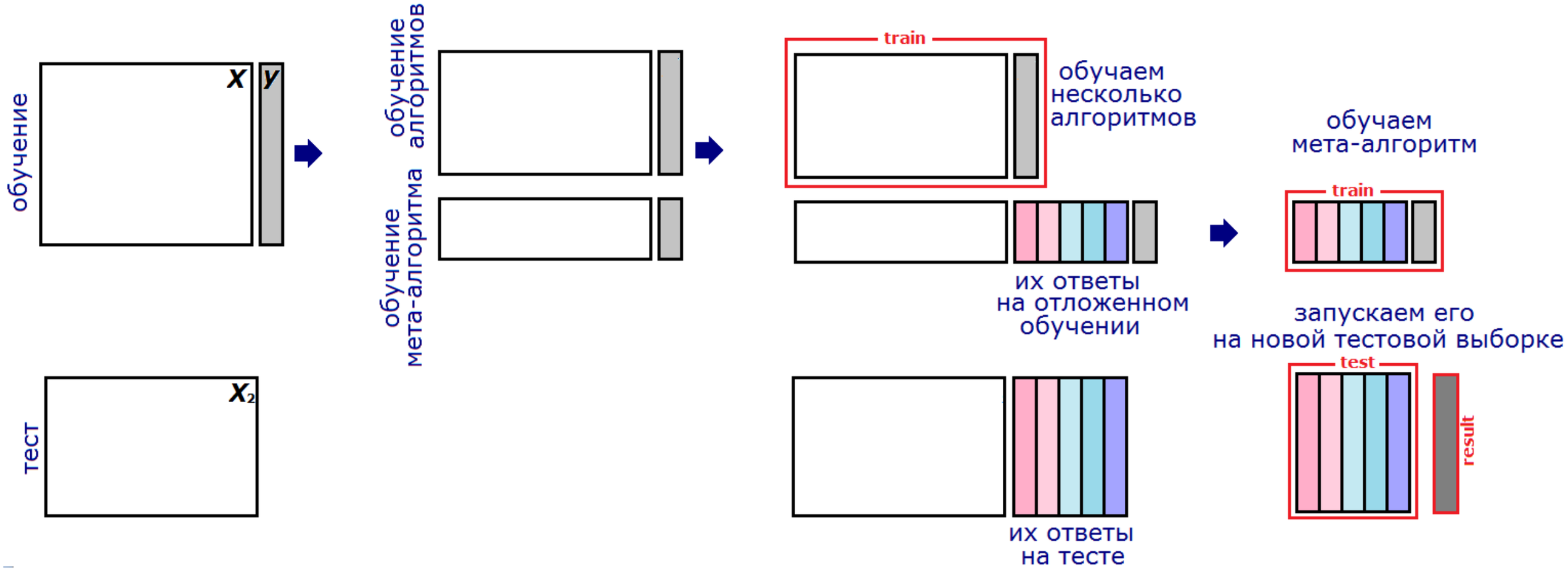



**происходит переобучение**

**базовый алгоритм на обучении воспроизводит истинные метки,**

**метаалгоритм ему доверяет… но на тесте он уже не знает правильных меток**

# Блендинг (Blending) – простейшая форма стекинга



## Блендинг

**– термин введён победителями конкурса Netflix**

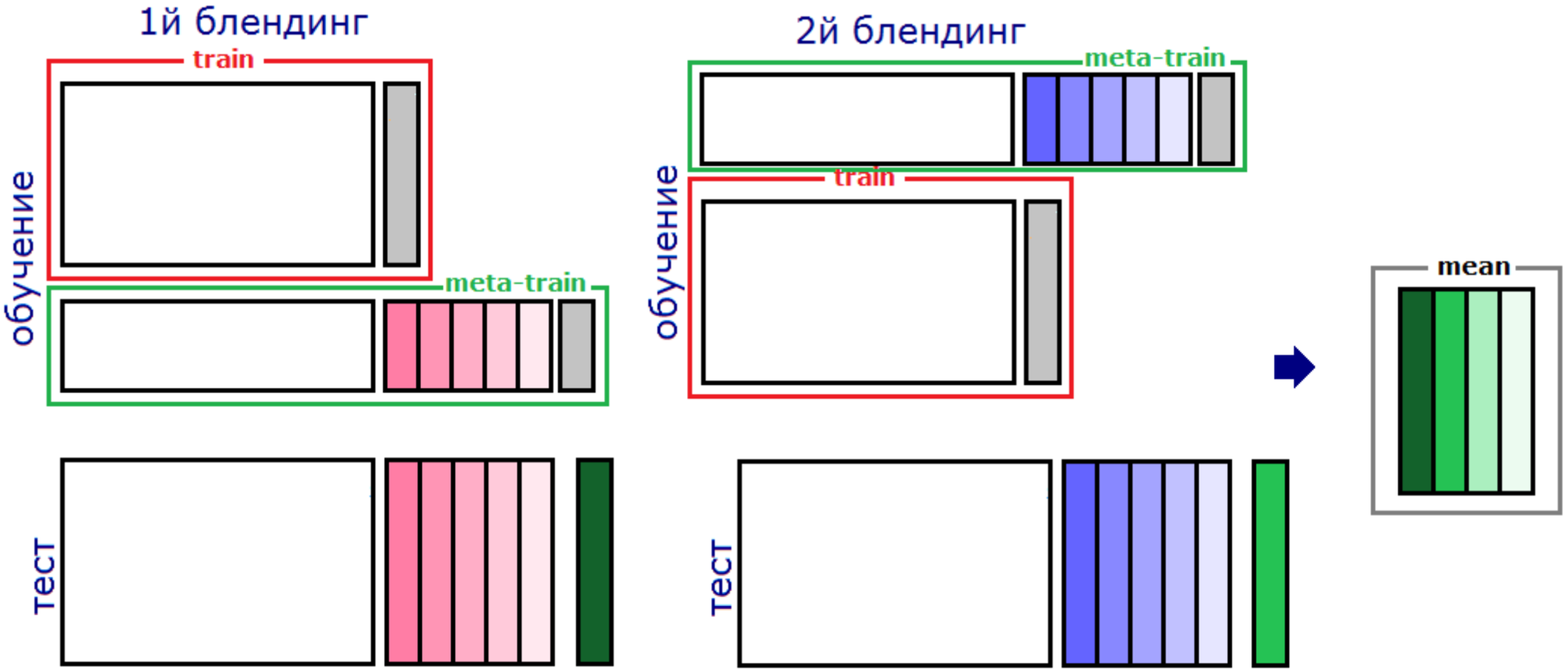
**Сейчас блендингом называются простейшие формы стекинга, например, выпуклую комбинацию алгоримтов**

## Недостатки

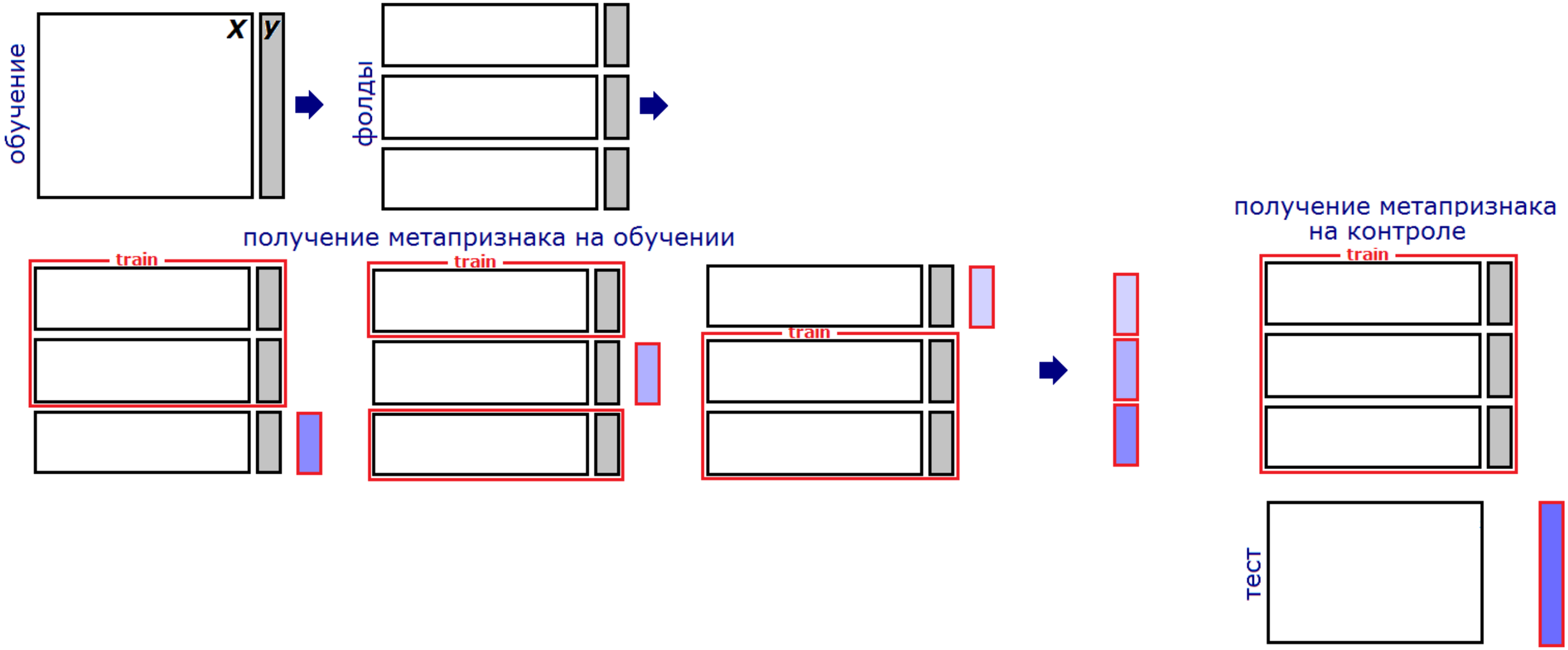
**Используется не вся обучающая выборка**

- **можно усреднить несколько блендингов**
- **можно «состыковать»**
  - **долго и не всегда лучше по качеству**
  - **ответы всё равно надо будет усреднить**

# Блендинг: усреднение ответов

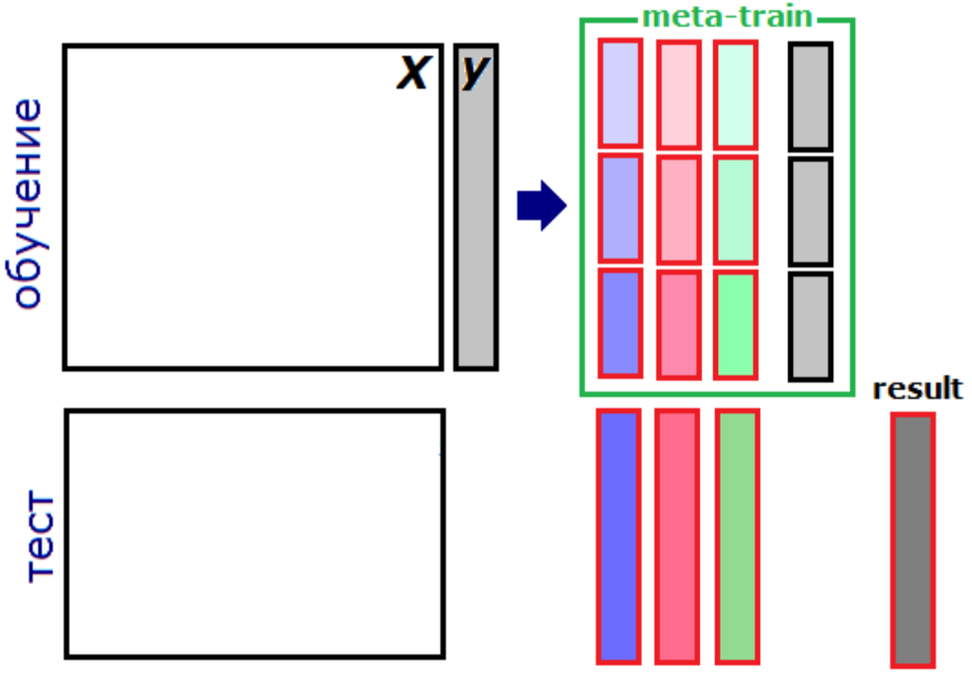


# Стекинг – хотим использовать всю обучающую выборку



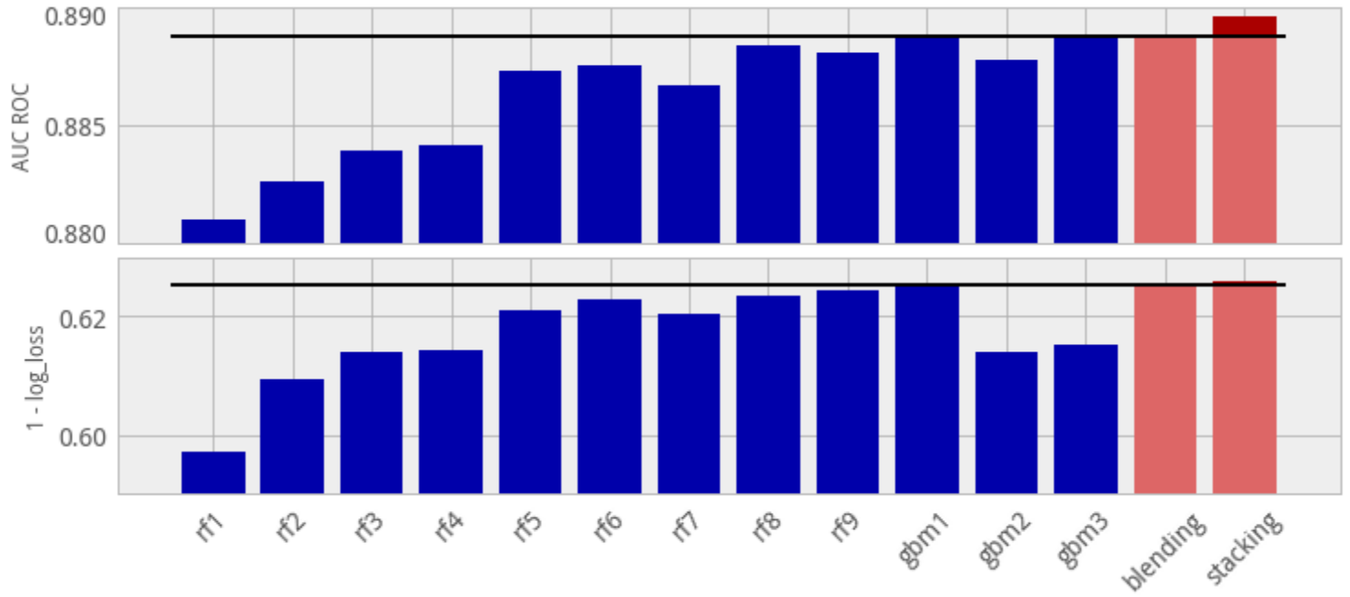

**м.б. разные разбиения на фолды и усреднить ответы базовых алгоритмов или стекингов**

# Стекинг – хотим использовать всю обучающую выборку



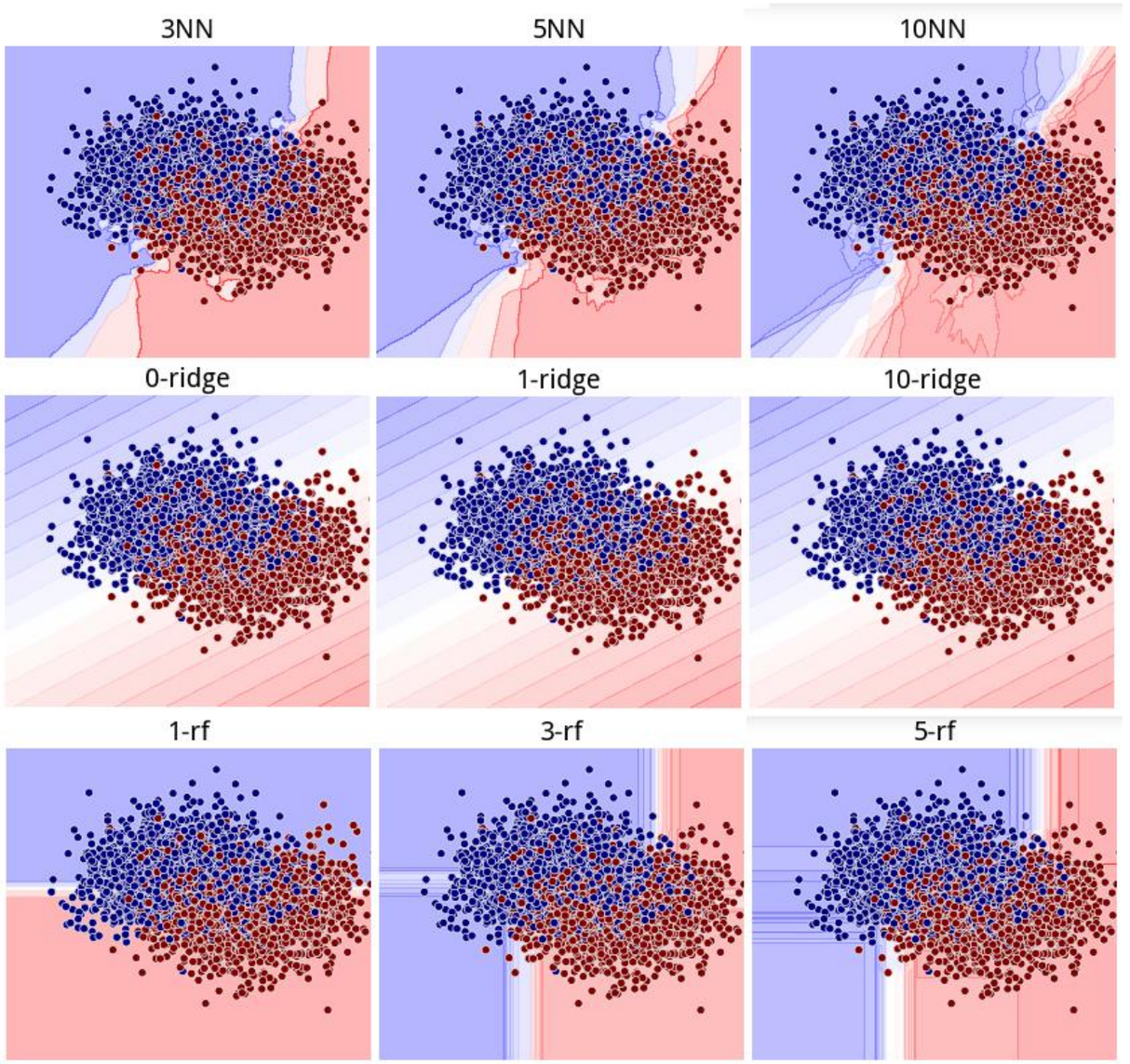

**получаем k-Fold-методом все метапризнаки**
**(используем все базовые алгоритмы)**
**обучаем мета-алгоритм**

# Стекинг

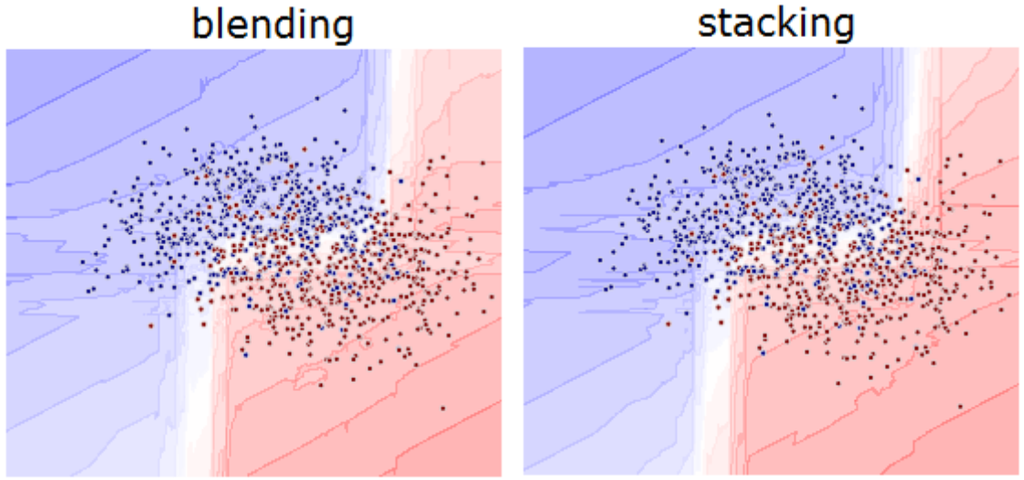



**На данных реальной задачи mlbootcamp**

# Геометрия стекинга



# Геометрия стекинга



# Стекинг

- **Нужны достаточно большие выборки**

- **Заточен на работу алгоритмов разной природы**

Но для каждого м.б. своё признаковое пространство

- **Хорош на практике (бизнес-задачи)**

Пример: регрессоры + RF =

устойчивость к аномальным значениям признаков

- **Многоуровневый стекинг**

Оправдан только в спортивном анализе данных

- **Появляются дополнительные гиперпараметры**

количество фолдов, уровень шума

# Минутка кода: стекинг

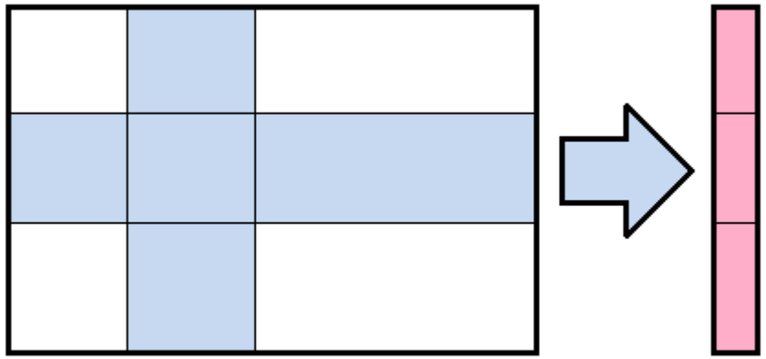
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict

class DjStacking(BaseEstimator, ClassifierMixin):
    """Стэкинг моделей scikit-learn"""

    def __init__(self, models, ens_model):
        """
        Инициализация
        models - базовые модели для стекинга
        ens_model - мета-модель
        """
        self.models = models
        self.ens_model = ens_model
        self.n = len(models)
        self.valid = None
```

```
    def predict(self, X, y=None):
        """
        Работа стэкинга
        """
        # заполение матрицы для мета-классификатора
        X_meta = np.zeros((X.shape[0], self.n))

        for t, clf in enumerate(self.models):
            X_meta[:, t] = clf.predict(X)

        a = self.ens_model.predict(X_meta)

        return (a)
```

https://github.com/Dyakonov/ml_hacks/blob/master/dj_stacking.ipynb

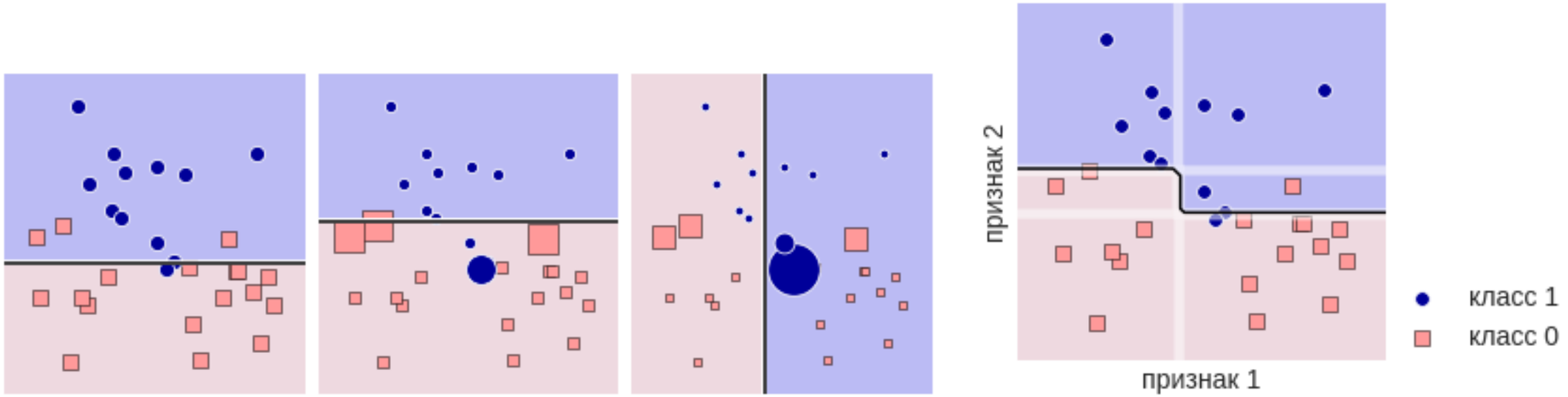
```
def fit(self, X, y=None, p=0.25, cv=3, err=0.001, random_state=None):
        """
        Обучение стекинга                                    cv  (при p=0) - сколько фолдов использовать
        p - в каком отношении делить на обучение / тест  err (при p=0) - случайная добавка к метапризнакам
            если p = 0 - используем всё обучение!        random_state - инициализация генератора
        """
        if (p > 0): # делим на обучение и тест
            # разбиение на обучение моделей и метамодели
            train, valid, y_train, y_valid = train_test_split(X, y, test_size=p, random_state=random_state)

            self.valid = np.zeros((valid.shape[0], self.n)) # заполнение матрицы для обучения метамодели

            for t, clf in enumerate(self.models):
                clf.fit(train, y_train)
                self.valid[:, t] = clf.predict(valid)

            self.ens_model.fit(self.valid, y_valid)         # обучение метамодели

        else: # используем всё обучение

            self.valid = err*np.random.randn(X.shape[0], self.n) # для регуляризации - берём случ. добавки

            for t, clf in enumerate(self.models):
                # это oob-ответы алгоритмов
                self.valid[:, t] += cross_val_predict(clf, X, y, cv=cv, n_jobs=-1, method='predict')
                clf.fit(X, y) # но сам алгоритм надо настроить

            self.ens_model.fit(self.valid, y)    # обучение метамодели
        return self
```

# Варьирование целевого вектора: искусственные, бустинг



# Бустинг

**Главная идея – базовые алгоритмы строятся не независимо, каждый следующий мы строим так, чтобы он исправлял ошибки предыдущих и повышал качество всего ансамбля**



# Идея градиентного бустинга

## FSAM + минимизация в случае дифференцируемой функции ошибки

**Задача регрессии с выборкой** $(x_i, y_i)_{i=1}^m$,
**дифференцируемая функция ошибки** $L(y,a)$,
**уже есть алгоритм** $a(x)$ **– строим** $b(x)$:

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

**т.е. настраиваемся на невязку**

$$b(x_i) \approx y_i - a(x_i)$$

**формально надо:**

$$\sum_{i=1}^m L(y_i, a(x_i) + b(x_i)) \to \min$$

**а не**

$$\sum_{i=1}^m L(y_i - a(x_i), b(x_i)) \to \min$$

**хотя часто они эквивалентны**

# Проблема

**Задача**

$$\sum_{i=1}^{m} L(y_i, a(x_i) + b(x_i)) \to \min$$

**может не решаться аналитически**

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

**Функция** $F(b_1,\ldots,b_m)$ **убывает в направлении антиградиента, поэтому выгодно считать**

$$b_i = -L'(y_i, a(x_i)),\ \ i \in \{1,2,\ldots,m\}.$$

**новая задача для настройки второго алгоритма:**

$$(x_i, -L'(y_i, a(x_i)))_{i=1}^{m}.$$

# Алгоритм градиентного бустинга (примитивный вариант)

- **Строим алгоритм в виде**

$$a_n(x) = \sum_{t=1}^{n} b_t(x),$$

**для удобства можно даже считать, что** $a_0(x) \equiv 0.$

- **Пусть построен** $a_t(x)$**, тогда обучаем алгоритм** $b_{t+1}(x)$ **на выборке**

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^{m}$$

- $a_{t+1}(x) = a_t(x) + b_{t+1}(x).$

**Итерационно получаем сумму алгоритмов…**

**Вот почему называется градиентный бустинг**

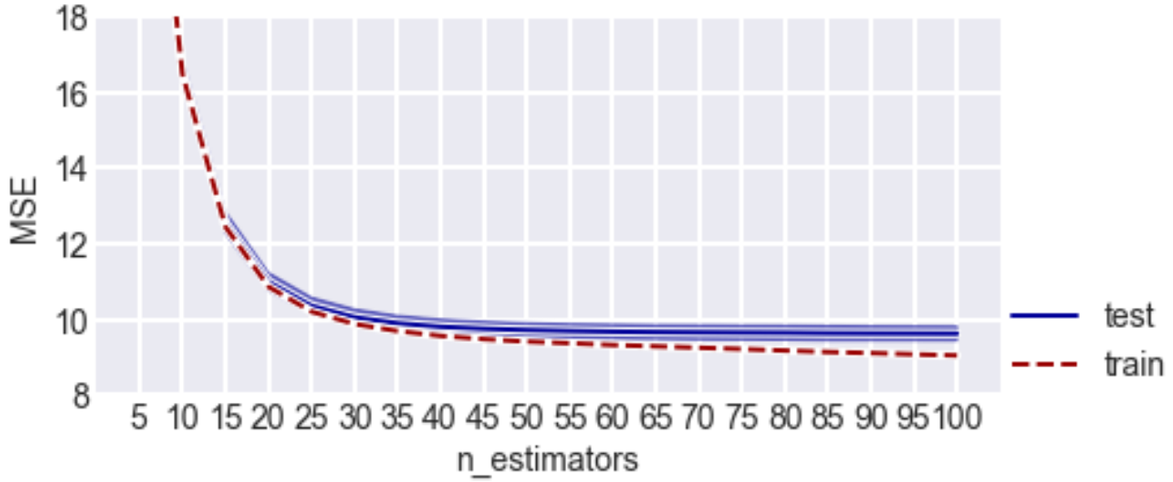
# Частный случай: регрессия с СКО

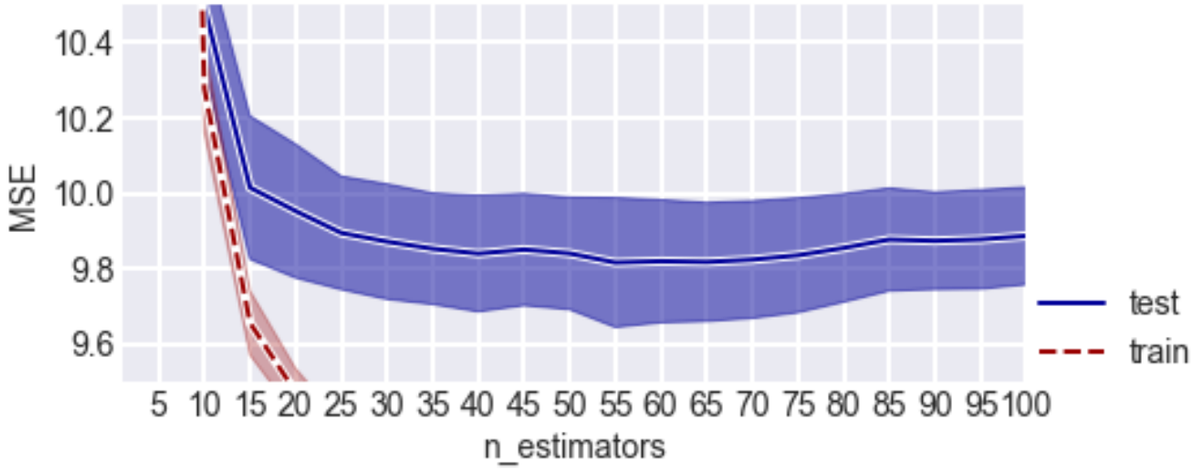
$$L(y,a)=\frac{1}{2}(y-a)^2, \quad L'(y,a)=-(y-a)$$

**Задача для настройки следующего алгоритма**

$$(x_i,\ y_i-a_t(x_i))_{i=1}^m$$

**т.е. очень логично: настраиваемся на невязку!**

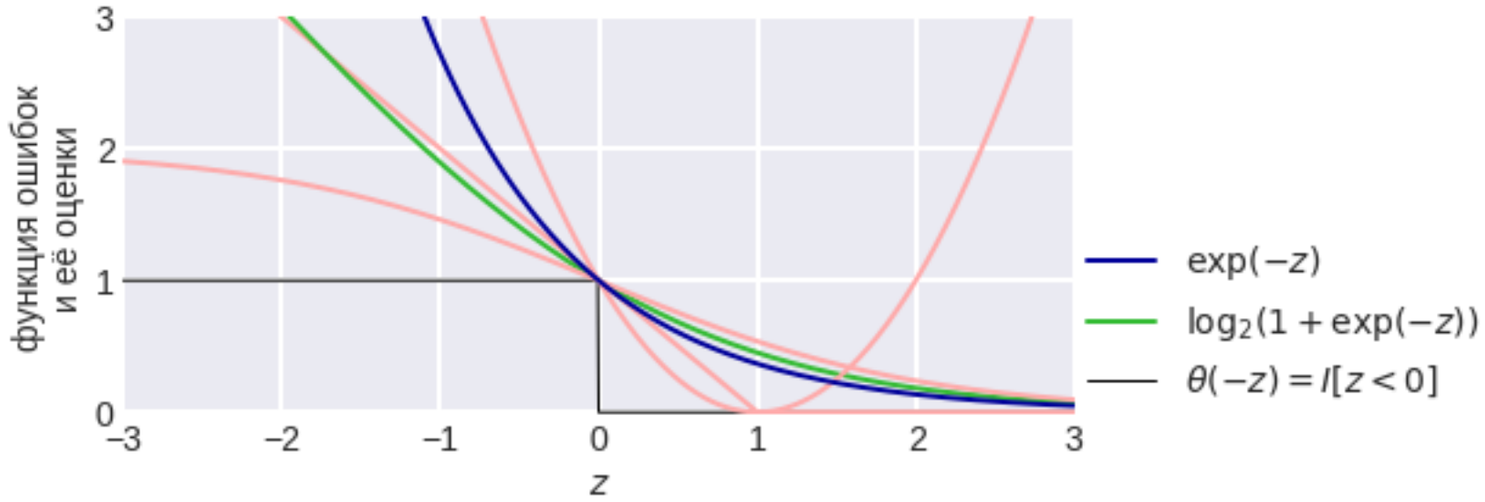


# Частный случай: классификация на два класса

**нужна дифференцируемая функция ошибки…**

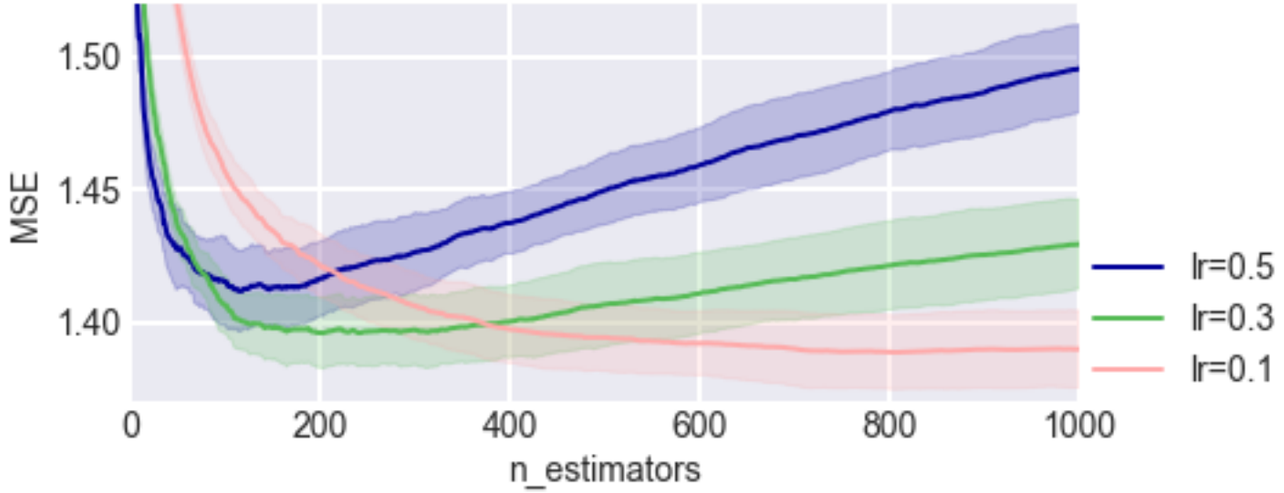
- **предполагаем, что алгоритм выдаёт вещественные значения**
- **нам подходят суррогатные функции ошибки**



# Эвристика сокращения – Shrinkage

$$a_{t+1}(x) = a_t(x) + \eta \cdot b_t(x),$$

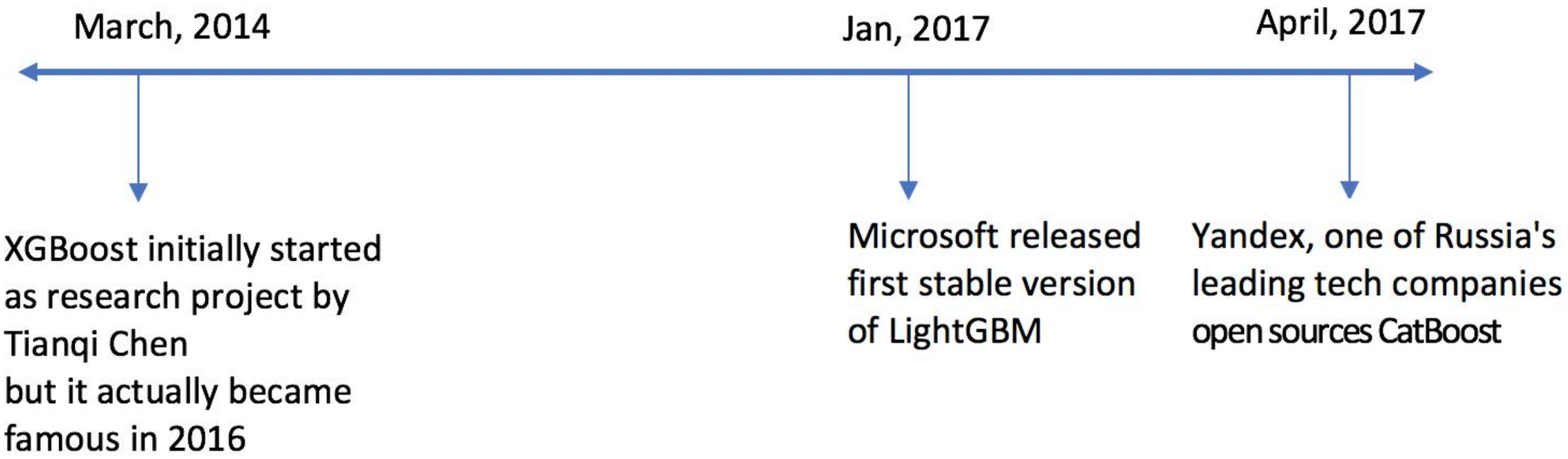
$\eta \in (0, 1]$ – **скорость (темп) обучения (learning rate)**




**Видно, что число слагаемых (базовых алгоритмов) – шагов бустинга – надо контролировать (при увеличении можем переобучиться)**

**Чем меньше скорость, тем больше итераций надо**

# История продвинутых методов / современные реализации




| sklearn.ensemble. | GradientBoostingRegressor<br>GradientBoostingClassifier |
|---|---|
| XGBoost (eXtreme Gradient Boosting) | https://github.com/dmlc/xgboost |
| LightGBM, Light Gradient Boosting Machine | https://github.com/Microsoft/LightGBM |
| CatBoost | https://github.com/catboost/catboost |

https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

# Итог: ключевые идеи ансамблирования

**1. Объединение ответов разных алгоритмов**

усреднение / голосование / стекинг …

**2. Повышения разнообразия / независимости базовых алгоритмов**

«варьирование» признаков, объектов, моделей, в модели и т.п.

**Использование подвыборок / весов**

**3. Ансамблирование: параллельное и последовательное**

**Parallel ensembles** – все алгоритмы строятся независимо

Идея: усреднить (high complexity, low bias)-модели, для снижения variance

**Sequential ensembles** – алгоритмы строятся последовательно

# Некоторые библиотеки

**ML-Ensemble** http://ml-ensemble.com/ General ensemble learning

**mlxtend** http://rasbt.github.io/mlxtend/ Regression and Classification ensembles

**H20** http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html
Distributed stacked ensemble learning. Limited to estimators in the H20 library

# Литература

**Статья про ансамбли**

Dietterich, T. G. (2000). «Ensemble Methods in Machine Learning» // First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science (pp. 1-15). New York: Springer Verlag.

**Предложен Feature-Weighted Linear Stacking**

Sill, J.; Takacs, G.; Mackey, L.; Lin, D. (2009). «Feature-Weighted Linear Stacking». arXiv:0911.0460.

**Бэгинг и аналогичные идеи:**

L. Breiman, Pasting small votes for classification in large databases and on-line, Machine Learning, 36(1), 85-103, 1999.

L. Breiman, Bagging predictors, Machine Learning, 24(2), 123-140, 1996.

T. Ho, The random subspace method for constructing decision forests, Pattern Analysis and Machine Intelligence, 20(8), 832-844, 1998.

G. Louppe and P. Geurts, Ensembles on Random Patches, Machine Learning and Knowledge Discovery in Databases, 346-361, 2012.

**Ансамбли в машинном обучении**

https://dyakonov.org/2019/04/19/ансамбли-в-машинном-обучении/

**Стекинг (Stacking) и блендинг (Blending)**

https://dyakonov.org/2017/03/10/стекинг-stacking-и-блендинг-blending/

# Литература

A. Liaw, M. Wiener Classification and Regression by randomForest // R News (2002) Vol. 2/3 p. 18.
http://www.bios.unc.edu/~dzeng/BIOS740/randomforest.pdf

A. Natekin, A. Knoll Gradient boosting machines, a tutorial // Front Neurorobot. 2013; 7: 21.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/

**все статьи по XGBoost, LightGBM, CatBoost**

**Сравнения**

https://www.kaggle.com/nholloway/catboost-v-xgboost-v-lightgbm
https://medium.com/riskified-technology/xgboost-lightgbm-or-catboost-which-boosting-algorithm-should-i-use-e7fda7bb36bc

**Про параметры**

https://neptune.ai/blog/lightgbm-parameters-guide