

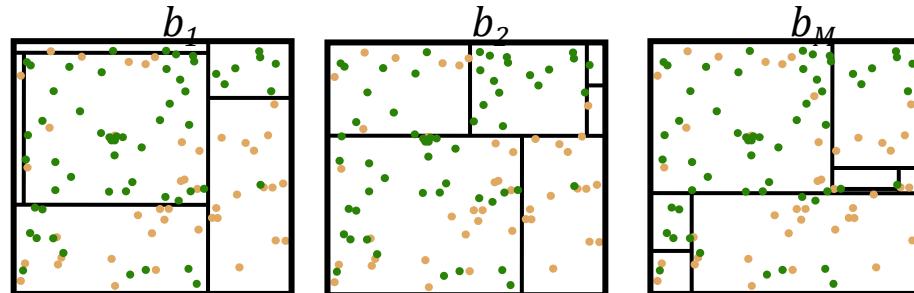


# Лекция 12: Ансамбли моделей

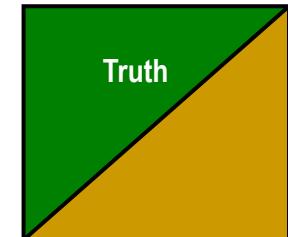
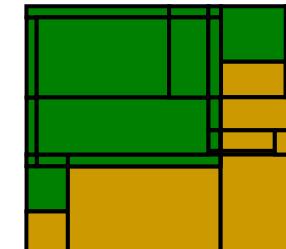
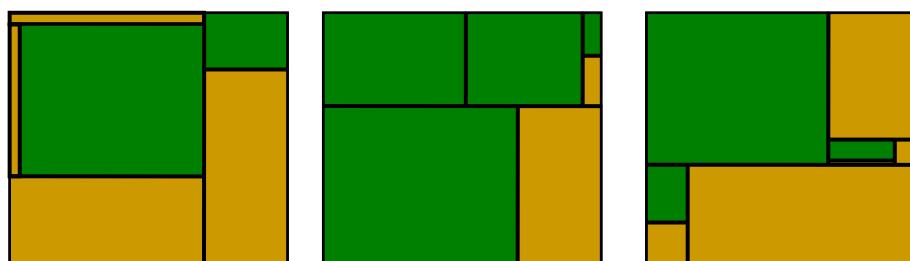
# Общая идея ансамблей

## ■ Ансамбль:

- Строим **базовые** (слабые) **алгоритмы** (модели)  $\{b_i(x) | b_i: X \rightarrow R\}_{i=1}^M$ ,  
хотелось бы независимые, но хотя бы существенно отличающиеся
- Агрегируем их прогнозы в **ансамбль**  $a(x) = F(b_1(x), \dots, b_M(x))$ , где  
 $F: R^M \rightarrow Y$  – функция агрегации или **мета-алгоритм**
- $R$  - порядковая или числовая шкала оценок, новое признаковое  
пространство для мета-алгоритма
- Ожидаем качество ансамбля  $>>$  качества любого базового алгоритма



$$a(x) = F(b_1(x), \dots, b_M(x)) \rightarrow y(x)$$



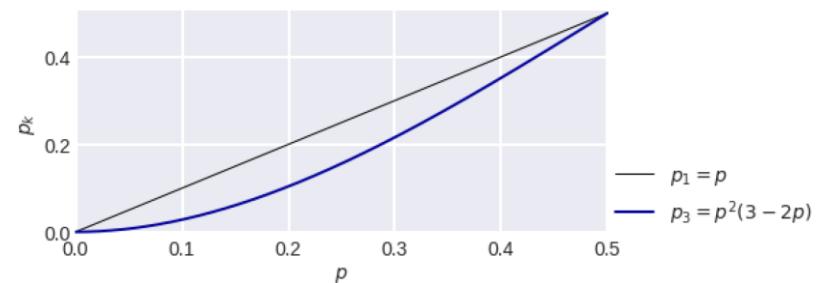
# Примеры агрегаций

- Голосование:
  - простое  $a(x) = \operatorname{argmax}_i [b_i(x)]$
  - взвешенное  $a(x) = \operatorname{argmax}_i [\alpha_i b_i(x)], \sum \alpha_i = 1, \alpha_i \geq 0$
  - с регуляризацией, например,  $a(x) = \operatorname{argmax}_i [\alpha_i b_i(x)], \sum |\alpha_i| \leq C$
- Усреднение:
  - простое  $a(x) = \frac{1}{M} \sum b_i(x)$
  - взвешенное  $a(x) = \sum \alpha_i b_i(x), \sum \alpha_i = 1, \alpha_i \geq 0$
  - с регуляризацией, например,  $a(x) = \sum \alpha_i b_i(x), \sum |\alpha_i| \leq C$
- Обобщённое усреднение (по Колмогорову):
  - $a(x) = \frac{1}{M} f^{-1} \sum f(b_i(x)),$  где  $\min_{1 \leq i \leq M} b_i \leq f(b_1, \dots, b_M) \leq \max_{1 \leq i \leq M} b_i,$   $f(\cdot)$  – непрерывная, монотонная, ...
- Смесь экспертов
  - $a(x) = \sum g_i(x) b_i(x),$  где  $g_i: X \rightarrow \mathbb{R}^+$  – функция компетентности, строится (обучается) отдельно и зависит от  $x$

# Проблема разнообразия и независимости базовых алгоритмов

- Оценка непрерывной с.в.  $\xi$  по ее **независимым** измерениям  $\{\xi_i\}$ :
  - $E(\xi) = E\left(\frac{1}{M} \sum \xi_i\right) = E\xi_i, D\xi = \frac{1}{M^2} \sum D\xi_i = \frac{1}{M} D\xi_i \rightarrow 0$ , при  $M \rightarrow \infty$
- Голосование в комитете (демо-пример) пусть вероятность ошибки  $p$ , тогда при трех **независимых** базовых алгоритмах и верном ответе 0 получаем варианты:
  - верные  $P(1,0,0) = P(0,1,0) = P(0,0,1) = (1-p)^2 p$ ,  $P(0,0,0) = (1-p)^3$
  - неверные  $P(1,1,1) = p^3$ ,  $P(1,1,0) = P(0,1,1) = P(1,0,1) = (1-p)p^2$
  - вероятность ошибки комитета  $p_k = p^2(3 - 2p) \ll p$
  - Общий случай:
- Но базовые алгоритмы не независимы ... как их разнообразить?

$$p_k = \sum_{t=1}^{k/2} C_k^t p^t (1-p)^{k-t}$$



# Типы ансамблей

- ECOC - кодирование отклика (уже разбирали)
- Комитеты (голосование/усреднение) – простые агрегации, базовые алгоритмы однотипные, обычно варьируем выборку:
  - Pasting - случайные выборки (Bagging - с возвращением)
  - Random subspaces – случайные подмножества признаков
  - Random patches = Pasting/Bagging + Random subspaces
  - Cross-validation комитет/усреднение – ансамбль из  $k$  базовых моделей, каждая обучена на  $(k-1)$  блоках кросс-разбиения
- Stacking/Blending:
  - простой (или сильно регуляризированный) обучаемый мета-алгоритм на комбинации откликов базовых алгоритмов из одного или разных семейств
  - иногда вместе с признаками из исходного пространства или с их комбинациями

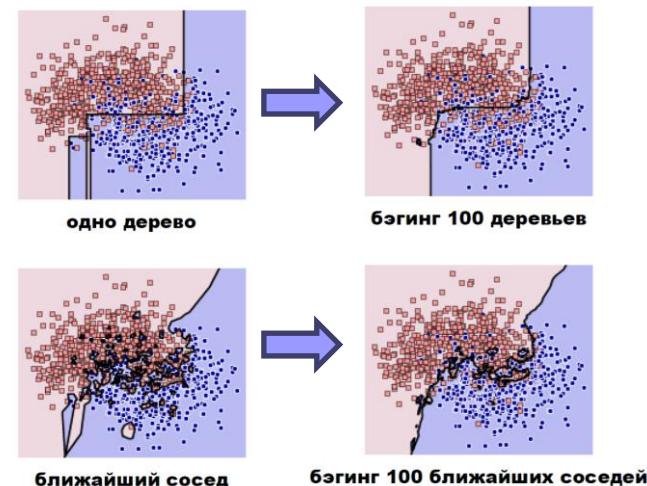
# Типы ансамблей

- Boosting («усиление слабых моделей») – каждый следующий базовый алгоритм пытается исправить ошибку предыдущих:
  - аддитивный (не совсем бустинг) – каждый следующий базовый алгоритм обучается на остатках от предыдущего ансамбля (например, FSAM)
  - каждый следующий базовый алгоритм с взвешенной функцией потерь, вес зависит от ошибки предыдущего ансамбля (Adaboost)
  - с перевыбором (вероятность pasting как функция от ошибки) – каждый следующий базовый алгоритм обучается на случайной подвыборке, где вероятность попасть в нее для наблюдения зависит от ошибки на нем предыдущего ансамбля
  - градиентный - взвешенный ансамбль с обучением на псевдоостатках, на каждом шаге «градиентно» минимизируется некоторая общая функция потерь всего ансамбля
- Байесовские ансамбли (поговорим в разделе методов Байеса)
- Комбинации всех или части перечисленных подходов

# Чем хороши ансамбли?

## ■ Статистическое обоснование:

- Борьба с недообучением
- Борьба с переобучением

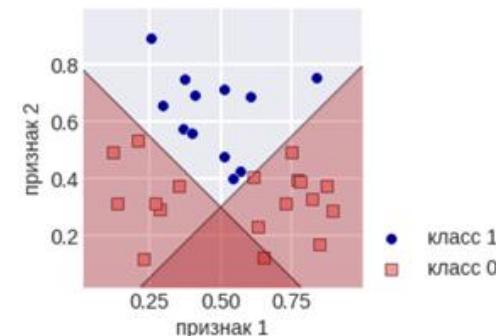


## ■ Вычислительное обоснование:

- Обучение многих типов ансамблей распараллеливается
- Зачастую ансамбль простых моделей обучать быстрее чем одну сложную модель

## ■ Функциональное обоснование:

- Комбинация моделей может описывать зависимость, которую нельзя описать отдельной моделью данного типа

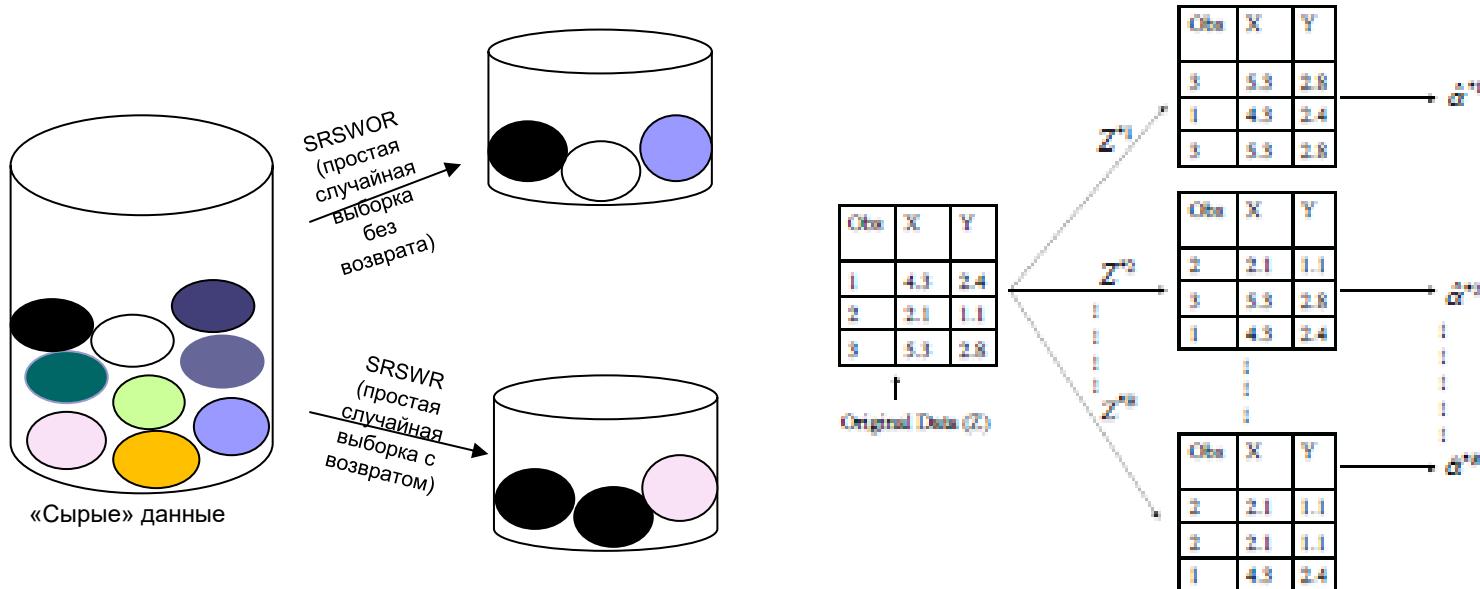


# Бутстрэппинг

- *Бутстрэппинг* статистический инструмент для:
  - количественной оценки неопределенности и имитации процесса получения новых случайных наборов данных из общей генеральной совокупности, но вместо получения независимых наборов данных, мы получаем различные наборы путем многократной выборки наблюдений из исходного набора с замещением (или с возвращением), в результате некоторые наблюдения могут появляться более одного раза в наборе данных бутстреппинга, а некоторые нет вообще.
- Использование термина **бутстрэппинг** происходит от фразы, чтобы *to pull oneself up by one's bootstraps*:

*Барон упал на дно глубокого озера. Когда казалось, что все было потеряно, он решил вытащить себя своими собственными силами*  
*(«Удивительные приключения барона Мюнхгаузена»)*

# Демонстрационный пример



- Графическая иллюстрация бутстреппингового подхода на маленькой выборке
- Каждый бутстреп набор данных содержит наблюдения, отобранные с заменой из исходного набора.
- Каждый такой набор данных начальной используется для получения оценки

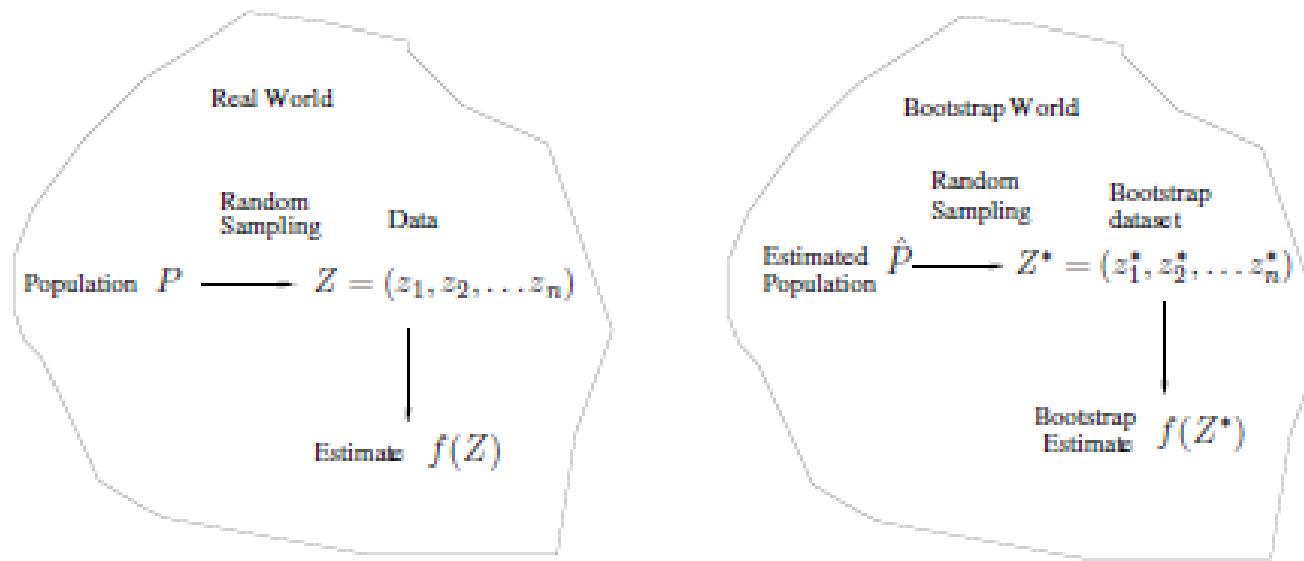
# Бутстрэппинг для оценки параметров

- Обозначая  $k$ -й набор данных бутстреппинга как  $Z^{*k}$ , мы используем его, чтобы выполнить новую оценку для  $a^{*k}$
- Эта процедура повторяется  $B$  раз для некоторого большого значения  $B$  (например, 100 или 1000), чтобы получить  $B$  различных наборов данных бутстреппинга  $Z^{*1}, Z^{*2}, \dots, Z^{*B}$ , и  $B$  соответствующих оценок  $a^{*1}, a^{*2}, \dots, a^{*B}$
- Оценим среднее и стандартную ошибку этих оценок бутстреппинга:

$$\bar{a} = \frac{1}{B} \sum_{r=1}^B (a^{*r}), SE_B(a) = \sqrt{\frac{1}{B-1} \sum_{r=1}^B (\bar{a} - a^{*r})^2}$$

- Они служат в качестве оценки, полученной на тестовом наборе данных.

# Общая схема бутстрепинга



- В более сложных ситуациях, определение подходящего способа для получения выборок бутстрепинга может потребовать значительных усилий.
- Например, если данные представляют собой временные ряды, мы не можем просто выбирать наблюдения с замещением

# Пример (Python)

```
ITER = 100
SAMPLES = 100
frame = []
for i in range(ITER):
    sample = resample(X, replace=True, n_samples=SAMPLES, stratify=None)
    stat = sample["HouseAge"].mean()
    frame.append(stat)
frame = np.array(frame).flatten()
frame = pd.Series(frame).sort_values()
```

Доверительные интервалы 90% для среднего возраста жилища:

```
frame.quantile(0.05), frame.quantile(0.95)
```

(26.248, 30.6515)

# Бутстреп-регрессия (Python)

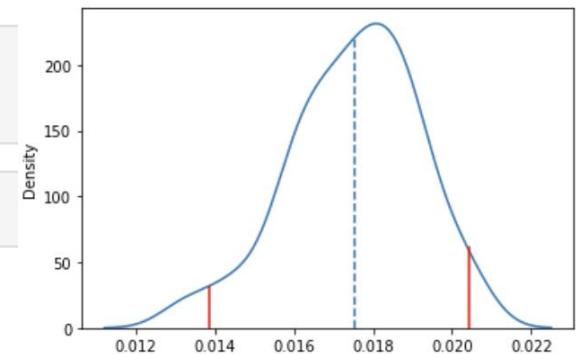
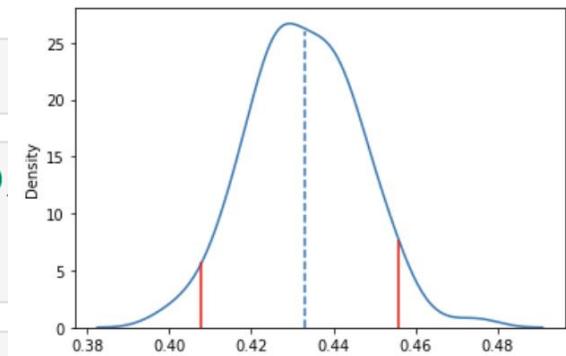
```
from sklearn.ensemble import BaggingRegressor  
  
estimator = BaggingRegressor(LinearRegression(), n_estimators=100  
                             bootstrap=True, max_samples=0.1,  
                             random_state=42)  
  
features = X[["MedInc", "HouseAge"]]
```

```
estimator.fit(features, y)  
pass
```

```
coefs = np.array([x.coef_ for x in estimator.estimators_])
```

```
sns.kdeplot(data=coefs[:, 0])  
plt.axvline(coefs[:, 0].mean(), 0, 0.93, ls="--")  
plt.axvline(np.quantile(coefs[:, 0], 0.025), 0, 0.20, c="red")  
plt.axvline(np.quantile(coefs[:, 0], 0.975), 0, 0.27, c="red")
```

```
sns.kdeplot(data=coefs[:, 1])  
plt.axvline(coefs[:, 1].mean(), 0, 0.91, ls="--")  
plt.axvline(np.quantile(coefs[:, 1], 0.025), 0, 0.13, c="red")  
plt.axvline(np.quantile(coefs[:, 1], 0.975), 0, 0.25, c="red")  
....
```

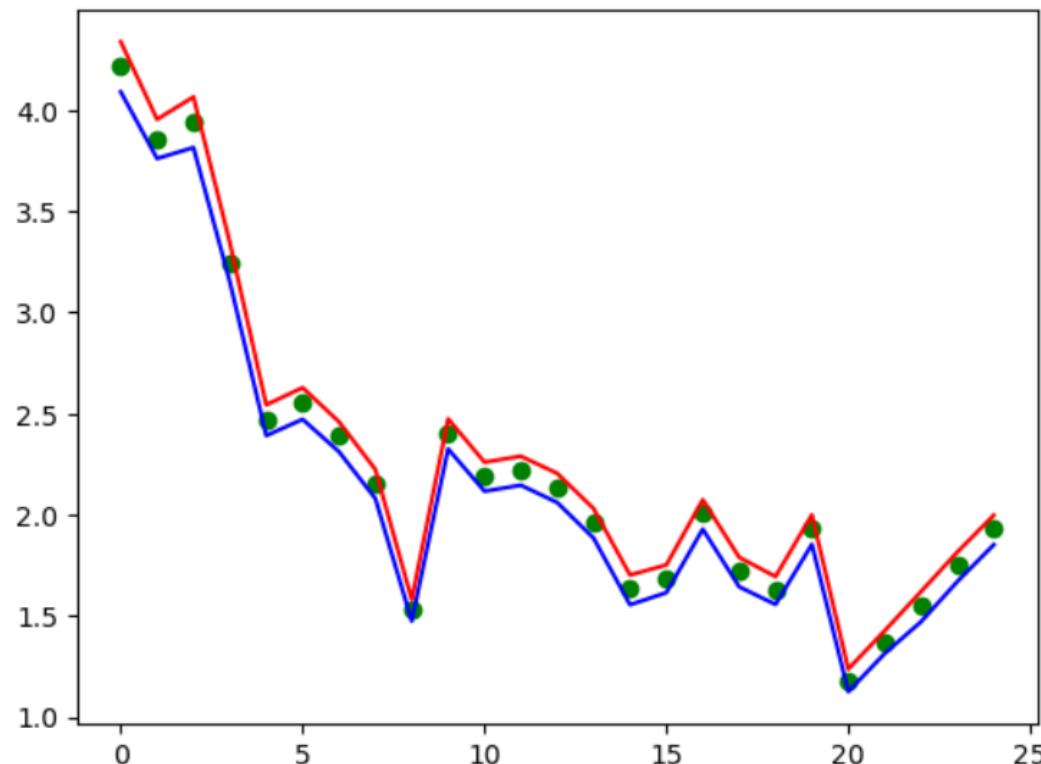


# Бутстреп-регрессия (Python)

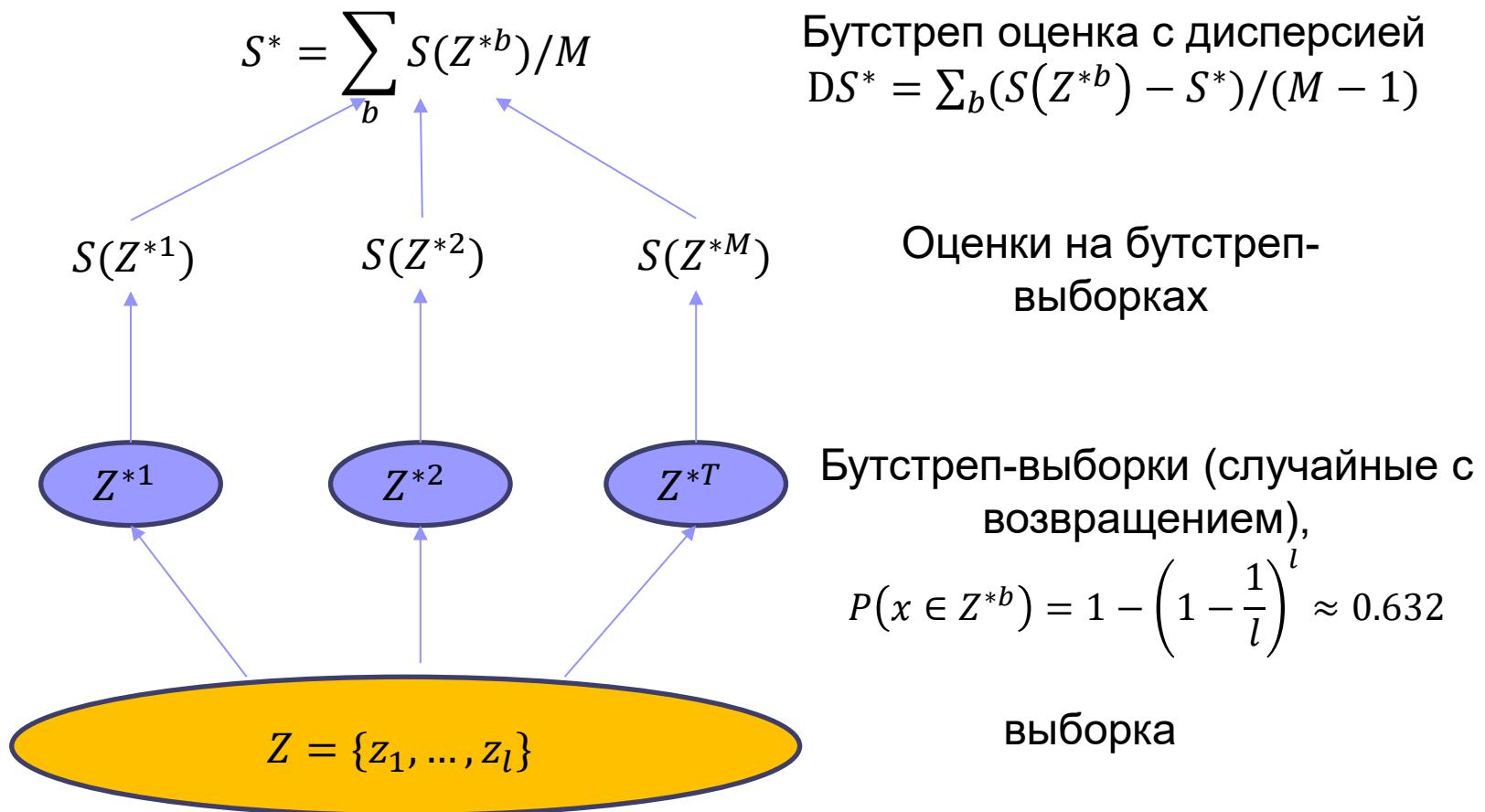
```
pred = np.array([x.predict(features.values) for x in estimator.estimators_]).T
```

```
(20640, 100)
```

```
plt.plot(np.percentile(pred, q=5, axis=1)[:25], c="blue")
plt.plot(np.percentile(pred, q=95, axis=1)[:25], c="red")
plt.scatter(range(25), estimator.predict(features)[:25], c="green")
pass
```



# Бутстреппинг



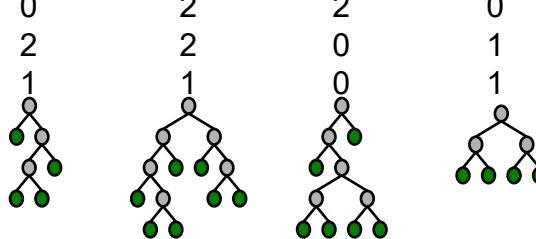
**Важно:** в отличии от методов макс. правдоподобия бутстреппинг позволяет строить не точечную оценку, а **распределение оценки** (в том числе прогноза, или параметра модели), даже в ситуациях, где ее теоретически не оценить

# B(ootstrap)AG(gregation)ing

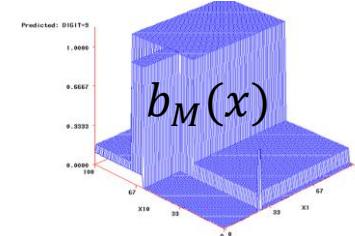
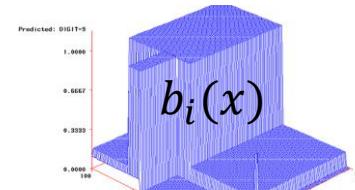
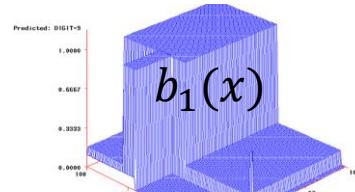
## ■ Алгоритм:

- Обучение: генерируем  $M$  выборок с возвращением, независимо подгоняем на них базовые классификаторы
- Применение: применяем каждый базовый, результат усредняем

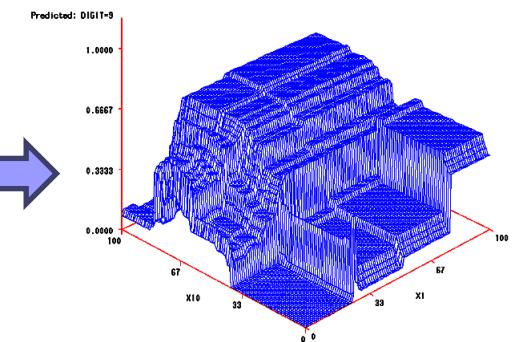
	b=1	b=2	b=...	b=M
case	freq	freq	freq	freq
1	1	0	3	1
2	0	1	1	1
3	2	0	0	2
4	0	2	2	0
5	2	2	0	1
6	1	1	0	1



Каждый  $b_i(x)$  строится независимо на бутстреп выборке  $Z^{*i}$



$$a_{bag}(x) = \frac{1}{M} \sum b_i(x)$$



В идеале при  $M \rightarrow \infty$  :  
 $a_{bag}(x) \rightarrow a_{opt}(x)$ ,  
 $\text{Var}[a_{bag}(x)] \rightarrow 0$

# OOB оценка качества ансамбля

- Out-of-bag ( $OOB_i$ ):
  - часть выборки тренировочного набора, не попавшая в обучающую выборку  $i$ -го базового алгоритма, вероятность попасть для  $x$

$$P(x \in OOB_i) = \left(1 - \frac{1}{l}\right)^l \approx 0.368$$

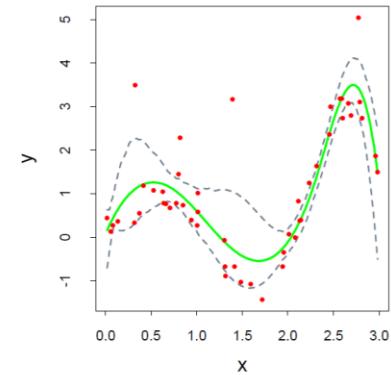
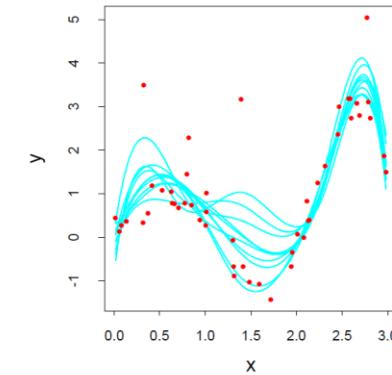
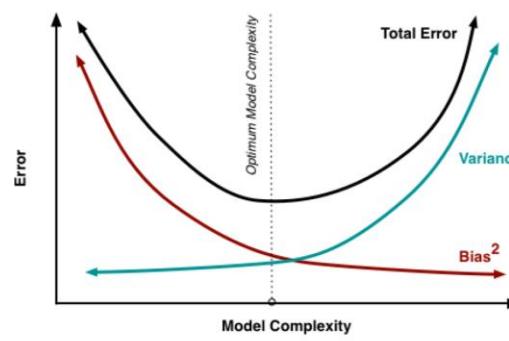
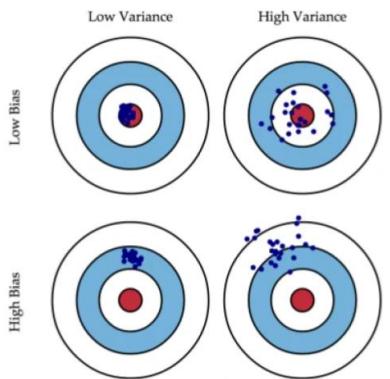
- Out-of-bag прогноз  $x$ :

$$a_{OOB}(x) = \frac{1}{|\{i: x \in OOB_i\}|} \sum_{i:x \in OOB_i} b_i(x)$$

- Out-of-bag оценка (несмешенная):
  - с функцией потерь  $L(b(x), y)$  для всего ансамбля  $a(x)$  на обучающей выборке  $Z$ :  $OOB = \frac{1}{l} \sum_{i:x_i \in Z} L(a_{OOB}(x_i), y_i)$
- Основное достоинство:
  - можно оценивать качество модели не исключая примеры из тренировочной выборки

# Какие модели хороши для Bagging

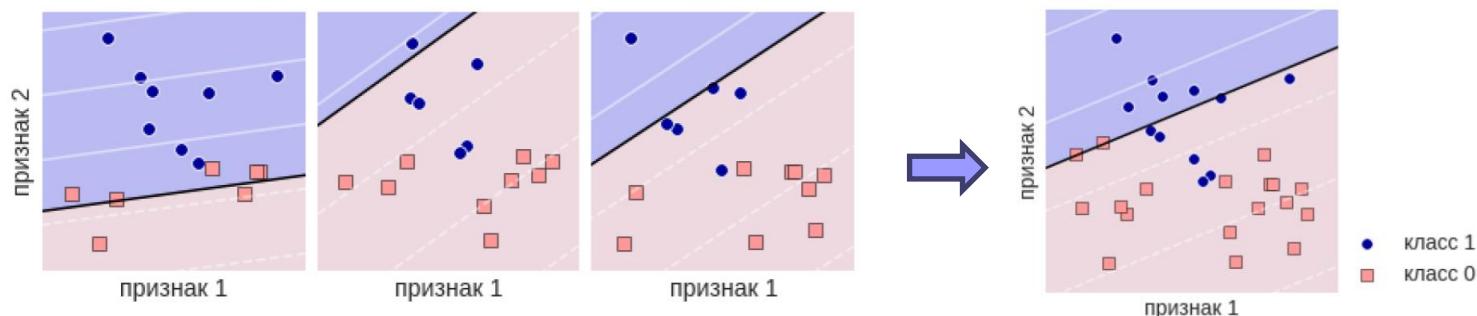
- Хотелось бы добиться «независимости» прогнозов в ансамбле:
  - Формально это невозможно, но можно «сымитировать» за счет использования **нестабильных** моделей, чтобы минимизировать корреляцию откликов базовых алгоритмов
  - Желательно **маленькое смещение + большая дисперсия** => хороши сложные переобученные нелинейные модели, например, деревья решений, KNN (к-мало), нейросети (но их долго учить), непараметрические сплайны и локальные взвешенные регрессии



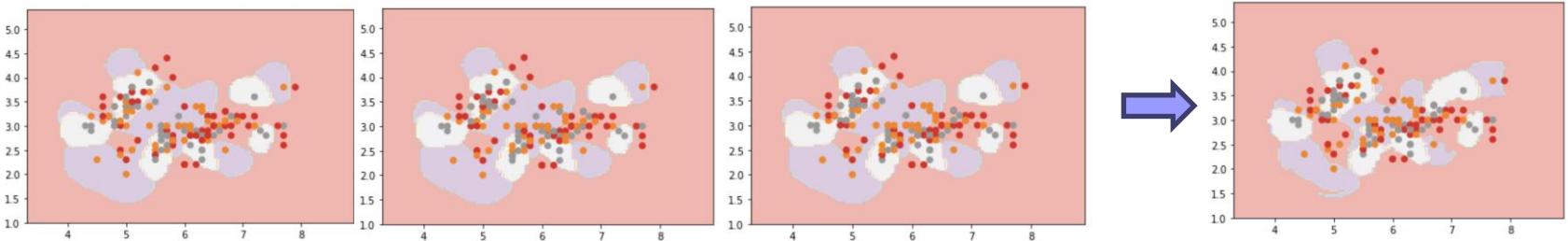
# Какие модели плохи для Bagging

- Плохо подходят:

- простые модели (большое смещение и маленькая дисперсия), например, простые линейные регрессии, KNN ( $k$  – велико)



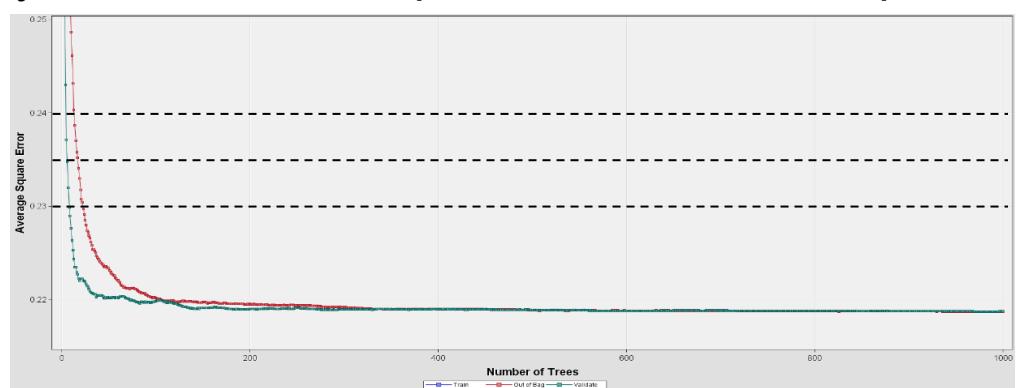
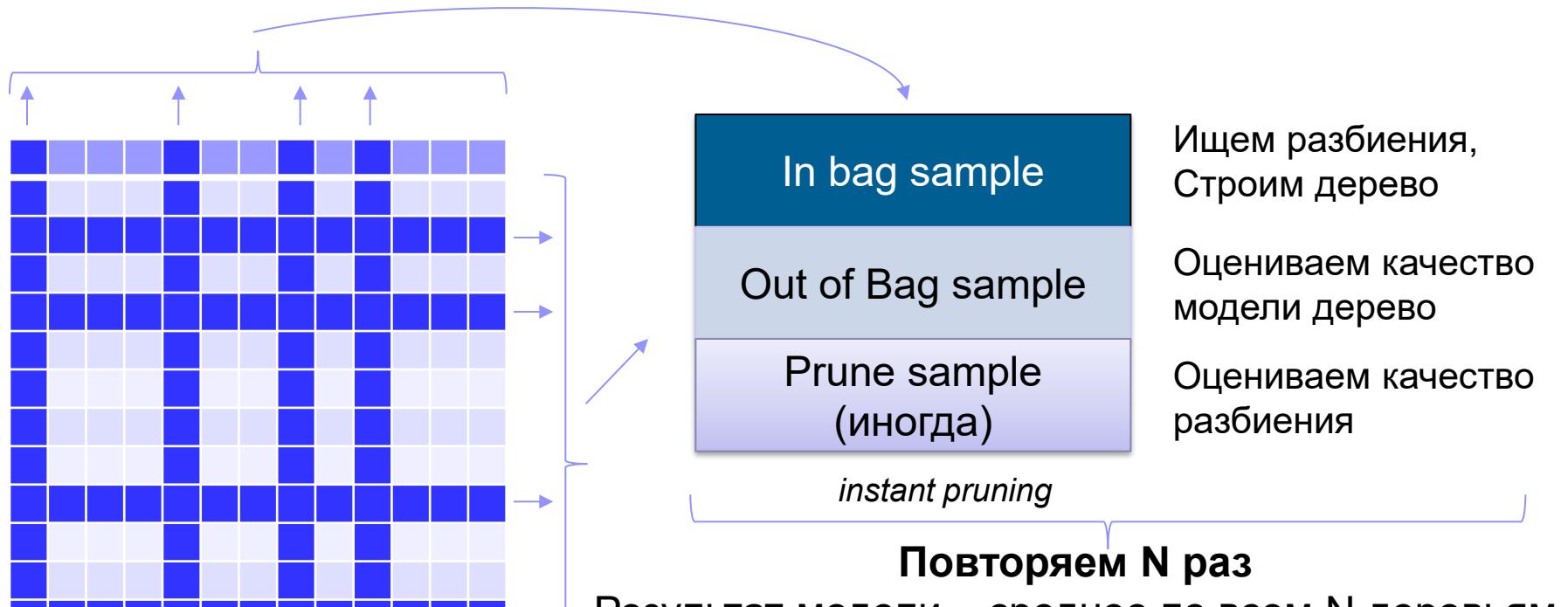
- сложные нелинейные, но стабильные модели, например, SVM



# Случайный лес

- Основные особенности:
  - *Bagging* (с пропорцией от выборки) ансамбль, есть дополнительный sampling – выборка с возвращением набора меньшего размера.
  - Случайные подпространства признаков на каждом шаге (sampling признаков).  $\sqrt{\# \text{inputs}}$  или явно задано число предикторов.
  - *Out-of-bag* для контроля сложности.
  - Медленно работает, но хорошо *распараллеливается*.
- Помимо прогнозирования можно использовать для:
  - оценки важности предикторов (как в одиночном дереве, но сумма по всему ансамблю)
  - для поиска аномалий (наблюдения в узле, близком к корню) с учителем и без (случайные разбиения)
  - для оценки близости наблюдений (по частоте попадания в общий лист или по пути «внутри дерева»)

# Случайный лес

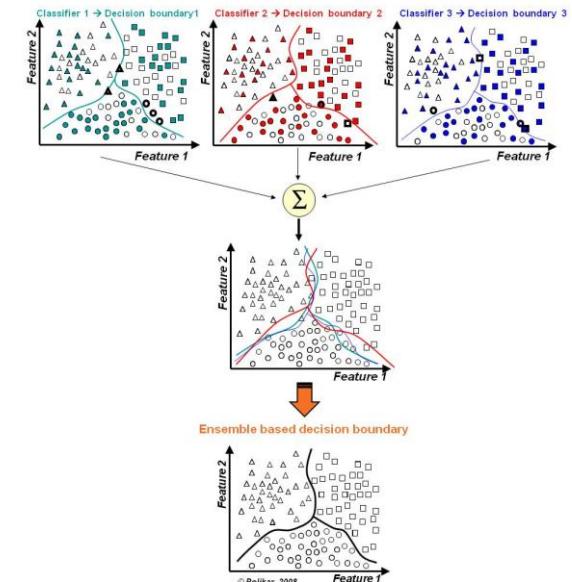
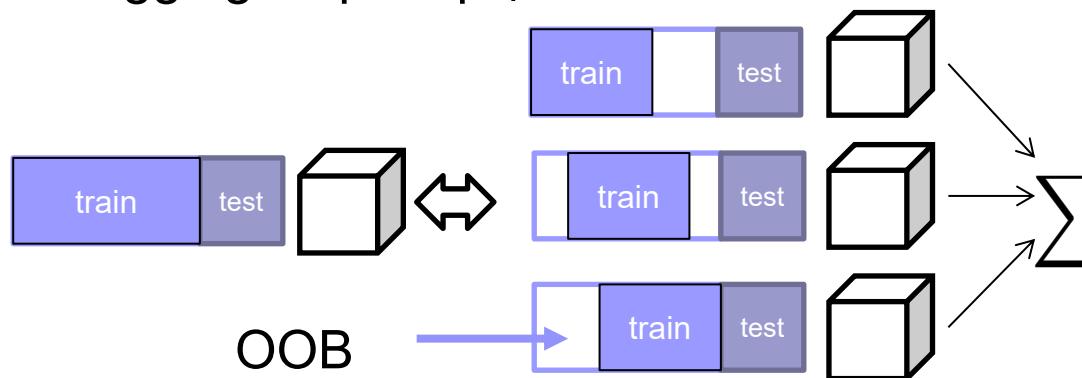


# Ключевые параметры

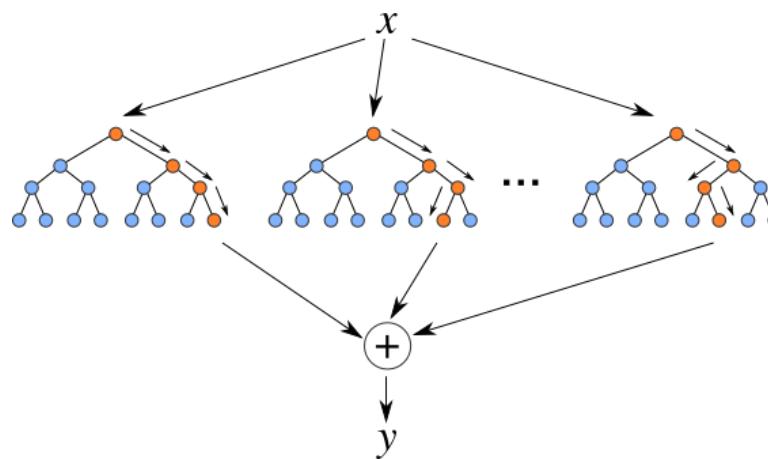
- Контроль сложности ансамбля:
  - размер ансамбля, чем больше тем сложнее, но не склонен переобучаться даже на больших ансамблях и выборках
- Контроль случайности базовой модели:
  - Число случайных признаков для поиска разбиения – чем меньше тем случайнее
  - Пропорция для *sampling* – чем меньше выборка тем случайнее
  - Можно контролировать случайность, анализируя попарные корреляций откликов, чем меньше тем лучше
  - Чем случайнее каждая модель, тем больше ансамбль нужен
- Контроль сложности базовой модели:
  - Глубина дерева, число ветвей, минимальный размер листа, пороги на разнородность или *p-value*, и др. – если мало выбросов, то можно строить сложные базовые модели
  - Чем проще каждая модель, тем больше ансамбль нужен
  - Остальные параметры (типа критерия разбиения) не очень важны

# Иллюстрация работы случайного леса

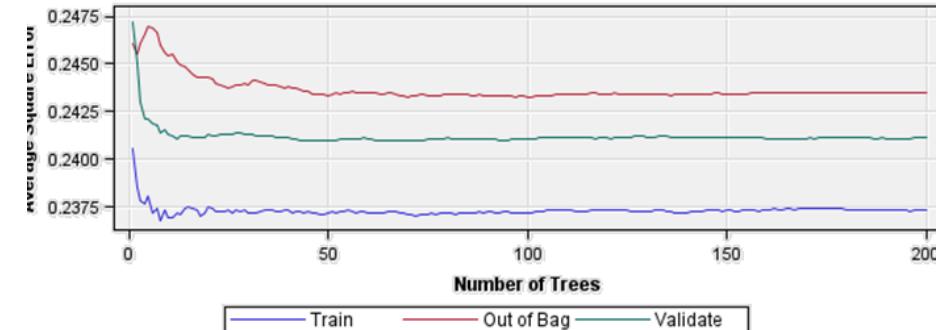
Bagging с пропорцией:



Применение ансамбля:



Оценка качества по ОВ:



# Random Forest (Python)

```
from sklearn.datasets import fetch_covtype
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
```

```
covtype = fetch_covtype()
X, y = covtype.data, covtype.target
labels = np.unique(y)
X.shape, y.shape, labels

((581012, 54), (581012,), array([1, 2, 3, 4, 5, 6, 7], dtype=int32))
```

```
print(covtype.DESCR)
```

```
.. _covtype_dataset:
```

Forest covtypes

The samples in this dataset correspond to 30×30m patches of forest in the US, collected for the task of predicting each patch's cover type, i.e. the dominant species of tree.

There are seven covtypes, making this a multiclass classification problem.

Each sample has 54 features, described on the

`dataset's homepage <<https://archive.ics.uci.edu/ml/datasets/Covtype>>`\_\_.

Some of the features are boolean indicators, while others are discrete or continuous measurements.

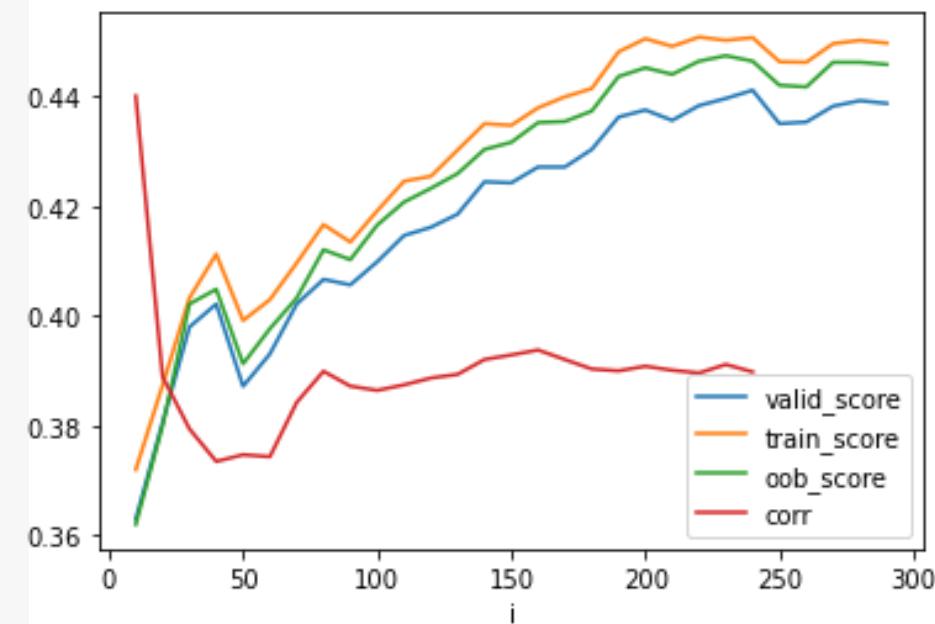
```
covtype_split = train_test_split(X, y, train_size=10000, test_size=10000, stratify=y, random_state=0)
X_train, X_test, y_train, y_test = covtype_split
```

```
class_weight = compute_class_weight("balanced", y=y_train, classes=labels)
covtype_class_weight = dict(zip(labels, class_weight))
```

# Random Forest (размер ансамбля)

```
n_estimators = 150
forest = RandomForestClassifier(n_estimators=n_estimators,
                                criterion="entropy",
                                min_samples_split=10,
                                min_samples_leaf=10,
                                max_features=10, # features to consider for each split
                                max_depth=10,
                                max_leaf_nodes=10,
                                class_weight=covtype_class_weight,
                                bootstrap=True, max_samples=0.15, # Samples % for each tree
                                ccp_alpha=0.0, # pruning
                                oob_score=True, # compute out-of-bag
                                warm_start=True, # add trees to the existing forest
                                random_state=0)
```

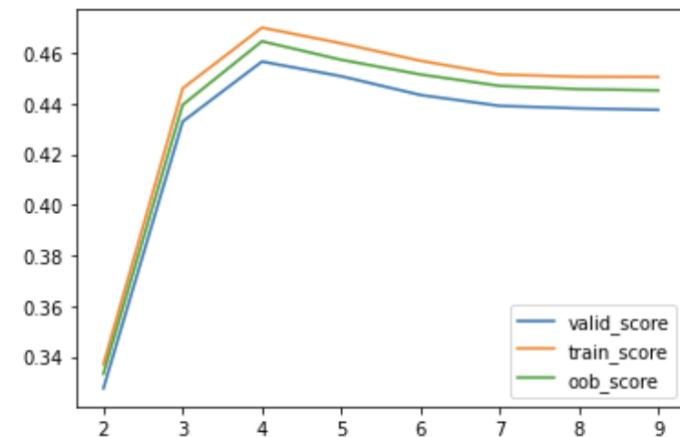
```
def sklearn_fit_history(model, n_estimators, X_train, y_train, valid=None):
    result = []
    warm_start=False
    for i in range(10, n_estimators, 10):
        model.set_params(n_estimators=i, warm_start=warm_start)
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        if valid is not None:
            d["valid_score"] = model.score(*valid)
        d["train_score"] = model.score(X_train, y_train)
        if hasattr(model, "oob_score_"):
            d["oob_score"] = model.oob_score_
        cc = []
        for c in range(0, 7, 1):
            dd = pd.DataFrame()
            for j in range(0, i, 1):
                res = forest.estimators_[j].predict_proba(X_train)[:, [c]]
                dd[str(j)] = pd.DataFrame(res).copy()
            cc.append(dd.corr().values.mean())
        d["corr"] = np.mean(cc)
        result.append(d)
        warm_start=True
    return pd.DataFrame(data=result).set_index("i")
```



# Random Forest (размера базовой модели)

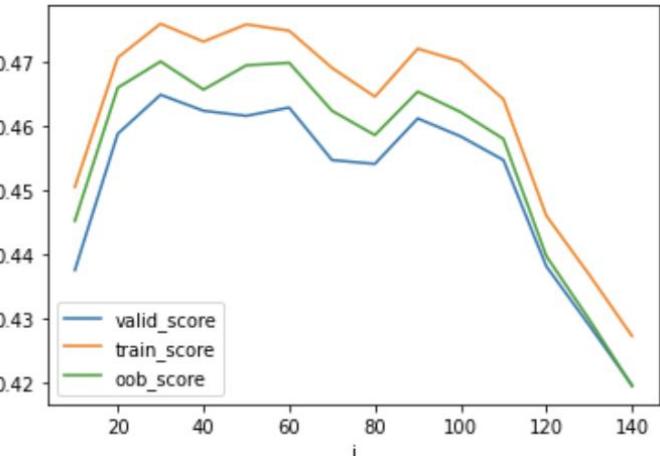
## ■ Глубина дерева:

```
def sklearn_fit_history1(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(2, 10, 1):
        model.set_params(max_depth=i) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        if valid is not None:
            d["valid_score"] = model.score(*valid)
        d["train_score"] = model.score(X_train, y_train)
        if hasattr(model, "oob_score_"):
            d["oob_score"] = model.oob_score_
        result.append(d)
    return pd.DataFrame(data=result).set_index("i")
history = sklearn_fit_history1(forest, 200, X_train, y_train, (X_test, y_test))
history.plot()
```



## ■ Размер листа:

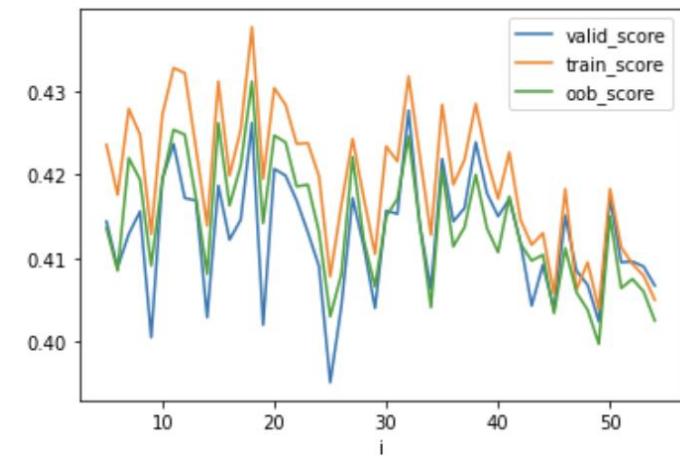
```
def sklearn_fit_history2(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(10, 150, 10):
        model.set_params(min_samples_leaf=i) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        if valid is not None:
            d["valid_score"] = model.score(*valid)
        d["train_score"] = model.score(X_train, y_train)
        if hasattr(model, "oob_score_"):
            d["oob_score"] = model.oob_score_
        result.append(d)
    return pd.DataFrame(data=result).set_index("i")
history = sklearn_fit_history2(forest, 200, X_train, y_train, (X_test, y_test))
history.plot()
```



# Random Forest (уровень случайности)

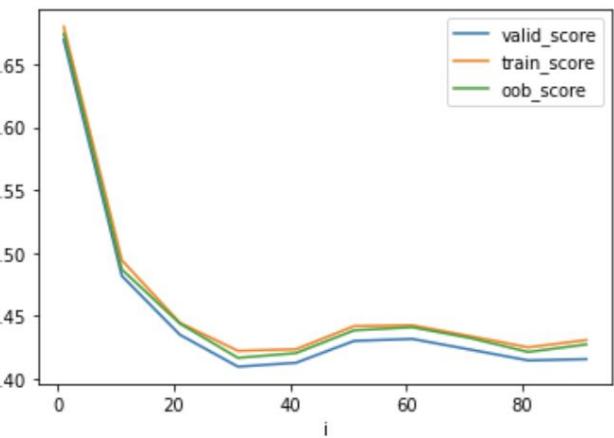
## ■ Число признаков:

```
def sklearn_fit_history3(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(5, 55, 1):
        model.set_params(max_features=i) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        if valid is not None:
            d["valid_score"] = model.score(*valid)
        d["train_score"] = model.score(X_train, y_train)
        if hasattr(model, "oob_score_"):
            d["oob_score"] = model.oob_score_
        result.append(d)
    return pd.DataFrame(data=result).set_index("i")
history = sklearn_fit_history3(forest, 200, X_train, y_train, (X_test, y_test))
history.plot()
```



## ■ Размер подвыборки:

```
def sklearn_fit_history4(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(1, 100, 10):
        model.set_params(max_samples=i*0.01) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        if valid is not None:
            d["valid_score"] = model.score(*valid)
        d["train_score"] = model.score(X_train, y_train)
        if hasattr(model, "oob_score_"):
            d["oob_score"] = model.oob_score_
        result.append(d)
    return pd.DataFrame(data=result).set_index("i")
history = sklearn_fit_history4(forest, 200, X_train, y_train, (X_test, y_test))
history.plot()
```



# Ключевые особенности

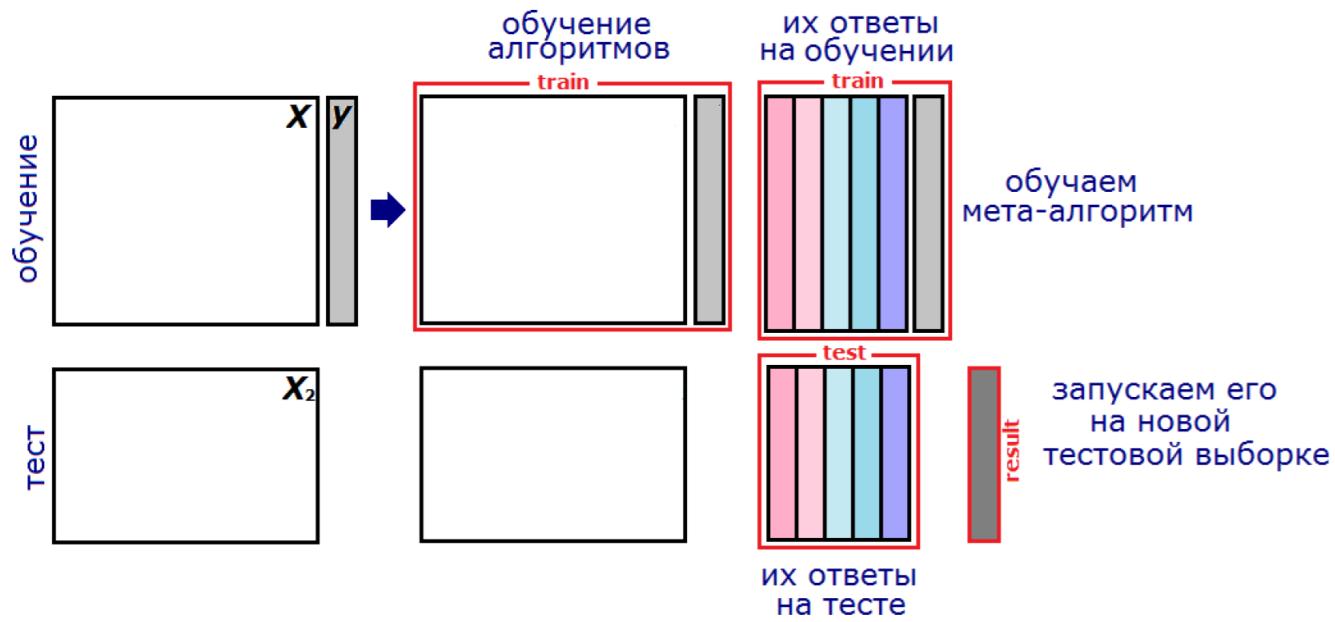
- Достоинства:
  - Большее изменение наборов чем в обычном bagging, а значит **большая вариация и меньшая корреляция** отклика моделей ведет к **несмещенному прогнозу с малой дисперсией**.
  - **Сложность** – можно оценивать по ОOB, не нужно CV и НО набор.
  - Случайный лес **не склонен к переобучению** даже на сложных деревьях (не нужно обрубать) и больших ансамблях.
  - Модель «из коробки» – мало гиперпараметров, любые входные данные, но при этом высокое качество
  - Хорошо **распараллеливается** и не требует всю выборку в памяти
- Недостатки:
  - Теряется интерпретируемость
  - Вычислительная сложность
  - Неочевидные метапараметры, которые нужно подбирать

# Стекинг

- Основные особенности:
  - Обучаемый мета-алгоритм  $F$  для  $a(x) = F(b_1(x), \dots, b_T(x))$
  - Расширение или замена признакового пространства за счет оценок базовых алгоритмов (обычно разных типов) как новых признаков для мета-алгоритма
- Неожиданные примеры стекинга:
  - Преобразование пространства признаков (feature engineering), использующее информацию об отклике, например, WOE, группировка или дикретизация на основе прогнозных моделей
  - Некоторые привычные алгоритмы можно рассматривать как стекинг, например, SVM – стекинг базовых функций  $b_i(x) = y_i K(x_i, x)$
- Ключевые вопросы:
  - Какие возможны базовые и мета алгоритмы? Как их обучать и комбинировать?
  - Требования к стекингу: желательно использовать всю выборку, при этом не обучая базовые и мета- алгоритмы на одних примерах
  - Есть ли теоретическое обоснование стекинга?

# Варианты стекинга: простой

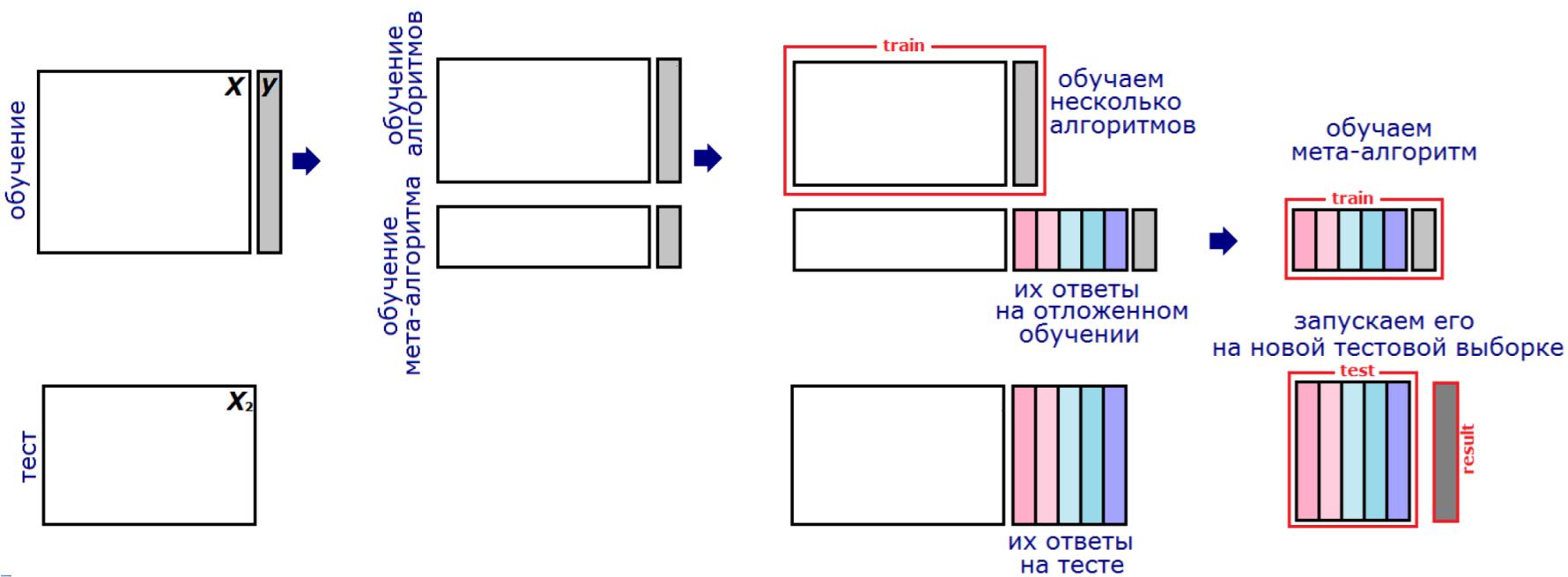
- Основные особенности:
  - обучаем мета-алгоритм на тех же данных, что и базовые алгоритмы
  - получаем переобученный прогноз



# Варианты стекинга: блендинг

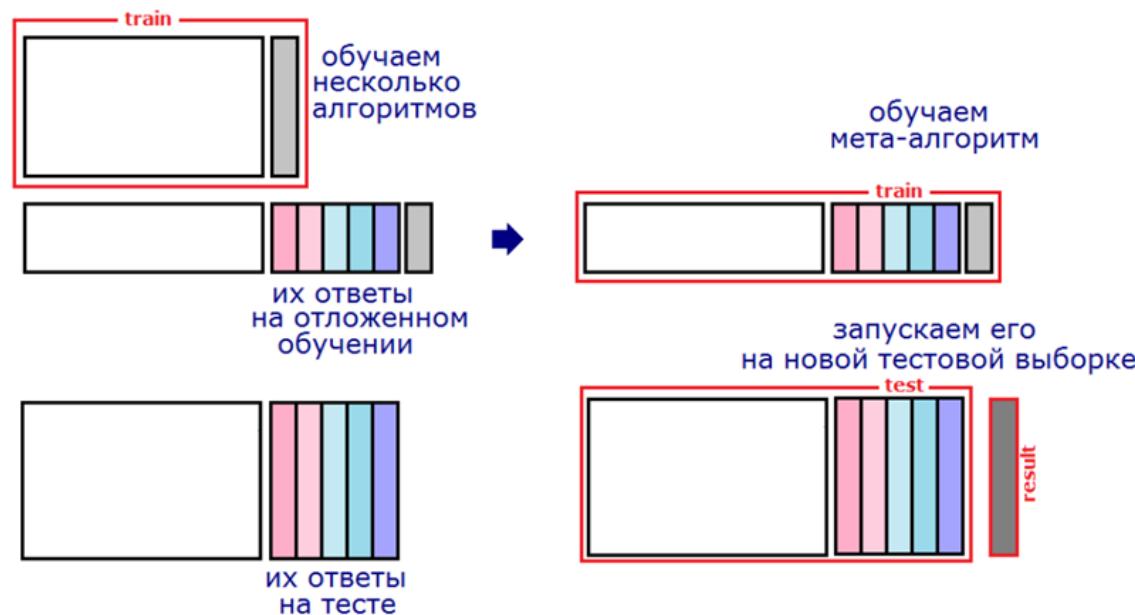
## ■ Основные особенности:

- обучаем мета-алгоритм на отложенных данных
- нет переобучения, но учим базовые и мета-алгоритмы не на всей выборке



# Варианты стекинга: объединение метапризнаков и обычных признаков

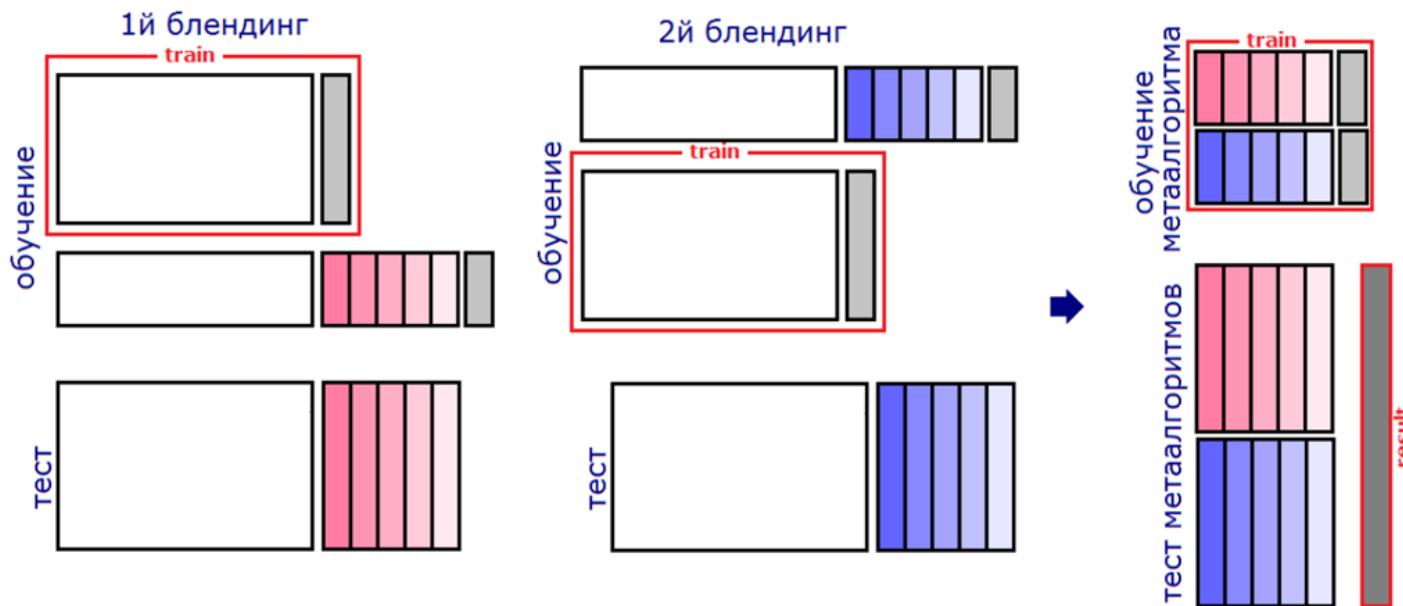
- Основные особенности:
  - Выделяем набор для контроля
  - Обучаем базовые алгоритмы на тренировочном и применяем на контроле
  - На комбинации признаков обучаем мета-алгоритм



# Варианты стекинга: объединение таблиц

## ■ Основные особенности:

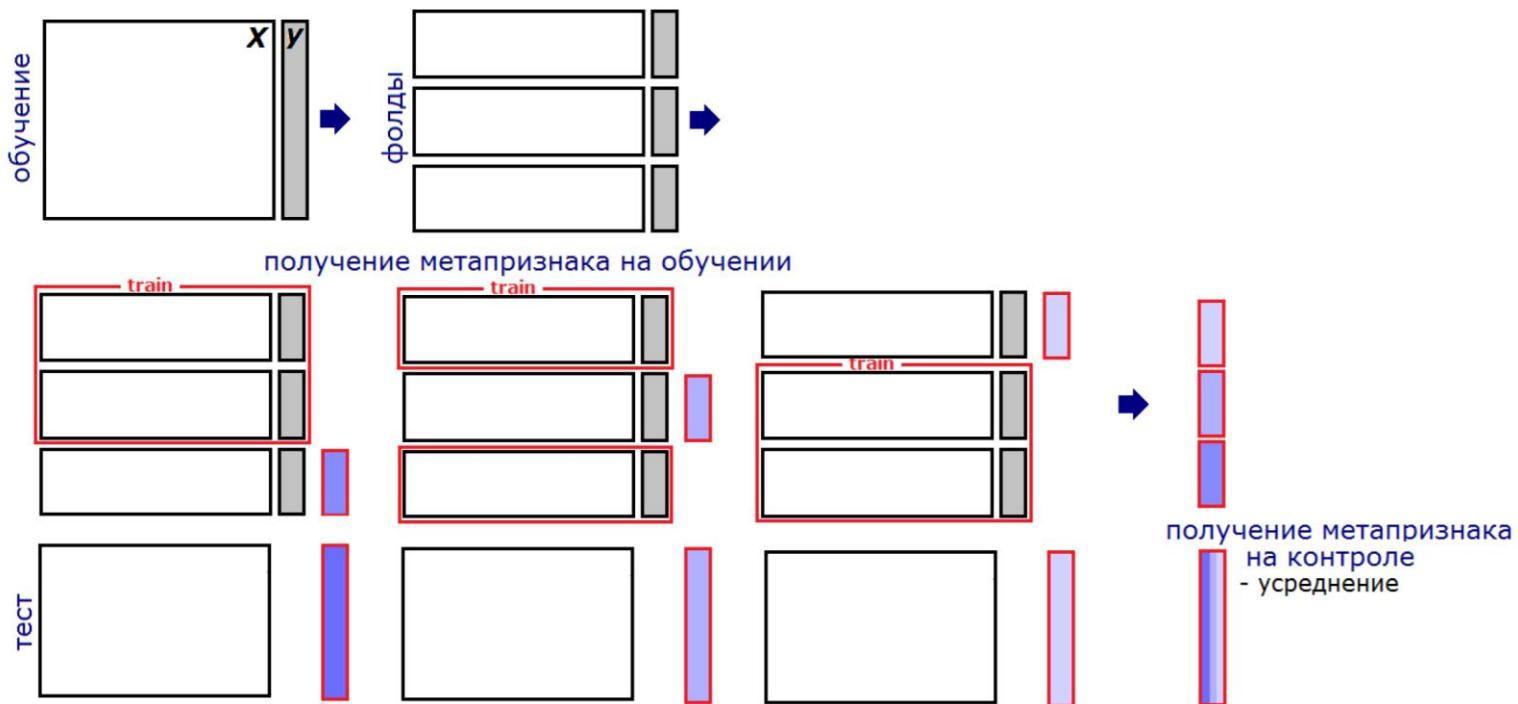
- разбиваем выборку на блоки, на части учим базовые алгоритмы, на оставшихся применяем
- «склеиваем» мета-признаки, мета-признаки строятся разными базовыми моделями



# Варианты стекинга: объединение метапризнаков

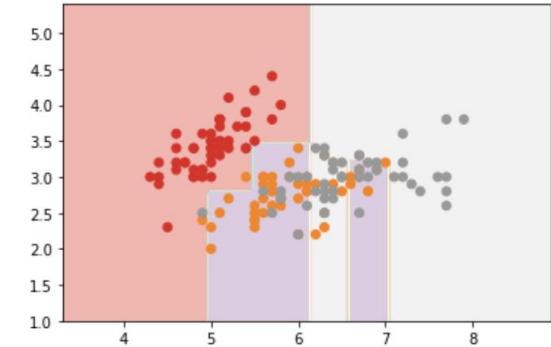
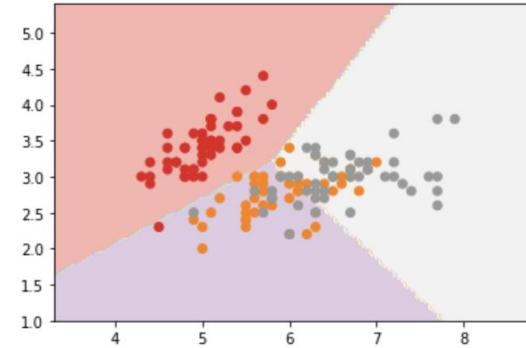
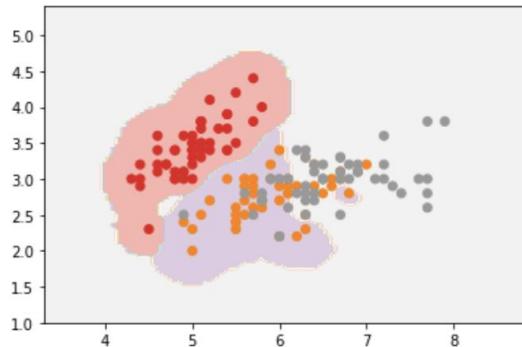
## ■ Основные особенности:

- разбиваем выборку на блоки, на части учим базовые алгоритмы, на оставшихся применяем (как в CV)
- «перемешанные» мета-признаки, одни и те же мета-признаки строятся разными базовыми моделями, надо агрегировать

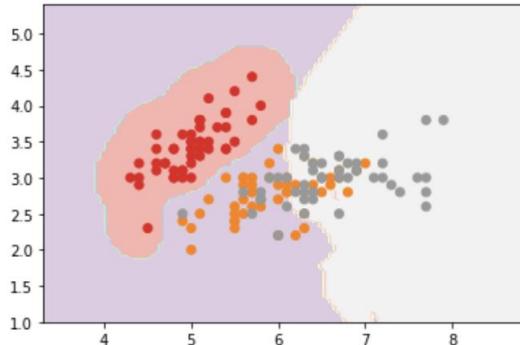


# Пример стекинга

- Базовые алгоритмы (слева направо):
  - RBF SVM, логистическая регрессия, дерево решений с gini:



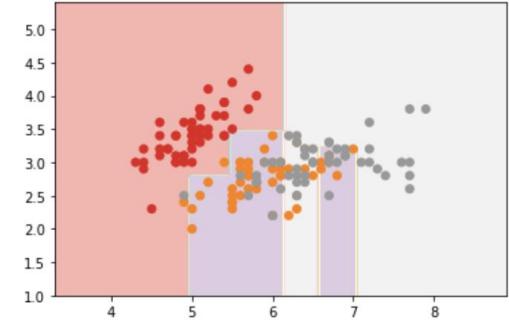
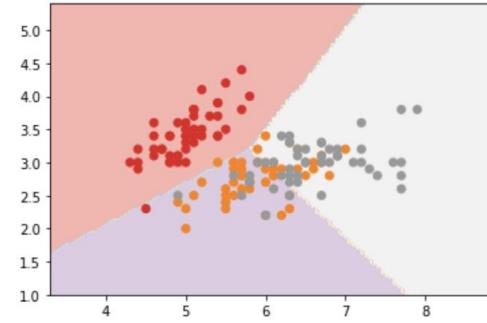
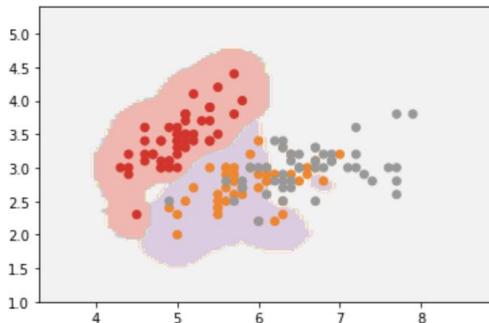
- Мета-алгоритм – линейная логистическая регрессия:



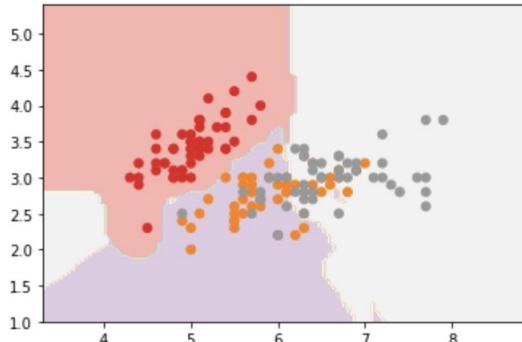
```
from sklearn.ensemble import StackingClassifier
estimators = [
    ('tree', DecisionTreeClassifier(criterion="gini", max_depth=5,
                                    min_samples_split=5, min_samples_leaf=3, ccp_alpha=0.0)),
    ('svc', SVC(kernel="rbf", gamma=10)), ('logreg', LogisticRegression())
]
clf = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
clf.fit(X, y)
DecisionBoundaryDisplay.from_estimator(clf, X, cmap="Pastel1")
plt.scatter(*X.T, c=y, cmap="Set1")
```

# Пример стекинга (продолжение)

- Базовые алгоритмы:



- Мета-алгоритм – регуляризированный однослойный персептрон:



```
from sklearn.neural_network import MLPClassifier
clf = StackingClassifier(estimators=estimators, final_estimator =
    MLPClassifier(
        alpha=0.1,
        hidden_layer_sizes=[3],
        max_iter=1000))
clf.fit(X, y)
DecisionBoundaryDisplay.from_estimator(clf, X, cmap="Pastel1")
plt.scatter(*X.T, c=y, cmap="Set1")
```

# Особенности стекинга

- Недостатки ☹
  - Нет теоретического обоснования
  - Нужно много данных
  - Метапризнаки коррелированы и появляются дополнительные мета-параметры настройки
- Достоинства ☺
  - Нет необходимости в глубоком тюнинге базовых алгоритмов
  - Позволяет объединять разнотипные модели, в том числе для каждого могут быть свои признаки, и даже свой отклик или вообще без него
  - Высокое качество на практических задачах и в соревнованиях
  - Пространство метапризнаков удобнее признакового (метапризнаки как правило числовые или порядковые)

# Смесь экспертов

- Постановка:
  - $a(x) = \sum g_m(x)b_m(x)$
  - где  $g_m: X \rightarrow \mathbb{R}^+$  - функция **компетентности**, строится (обучается) отдельно и зависит от  $x$ , нормирована  $\sum g_m(x) = 1$
  - можно нормировать через параметрическую softmax (получим голосование при  $\gamma \rightarrow \infty$ ):
$$g_m(x) = \text{softmax}(\tilde{g}_1(x), \dots, \tilde{g}_M(x); \gamma) = \frac{e^{\gamma \tilde{g}_m(x)}}{\sum_{j=1}^M e^{\gamma \tilde{g}_j(x)}}$$
- Виды функций компетентности (похоже на функции активации в нейросетях), экспертная или параметрическая привязка к:
  - $j$ -му  $f_j(x)$  признаку  $g(x) = 1/(1 + \exp(-\alpha f_j(x) - \beta))$
  - направлению  $\alpha$  – вектор  $g(x) = 1/(1 + \exp(-x^T \alpha - \beta))$
  - наблюдению  $\alpha$  – вектор (наблюдение)  $g(x) = \exp(-\beta(x - \alpha)^2))$
  - $\alpha, \beta$  – могут обучаться вместе с ансамблем, обучаться заранее и фиксироваться или задаваться экспертом и фиксироваться

# Выпуклая функция потерь

- Для выпуклых  $L(a(x), y)$  и  $\sum g_m(x) = 1$ :

- выполняется неравенство Йенсена для эмпирического риска:

$$\begin{aligned} Q(a) &= \sum_{i=1}^l L\left(\left[\sum_{m=1}^M g_m(x_i) b_m(x_i)\right], y_i\right) \leq \\ &\leq \sum_{m=1}^M \left( \sum_{i=1}^l g_m(x_i) L(b_m(x_i), y_i) \right) = \sum_{m=1}^M (Q_m(g_m, b_m)) \rightarrow \min \end{aligned}$$

- Итерационный ЕМ алгоритм (в цикле до сходимости):

- Начальное приближение (случайные, равные, подобранные)  $g_1, \dots, g_M$
  - **М-шаг:** для всех  $m = 1, \dots, M$  фиксируем  $g_m$  и находим

$$b_m = \operatorname{argmin}_b \sum_{i=1}^l g_m(x_i) L(b(x_i), y_i)$$

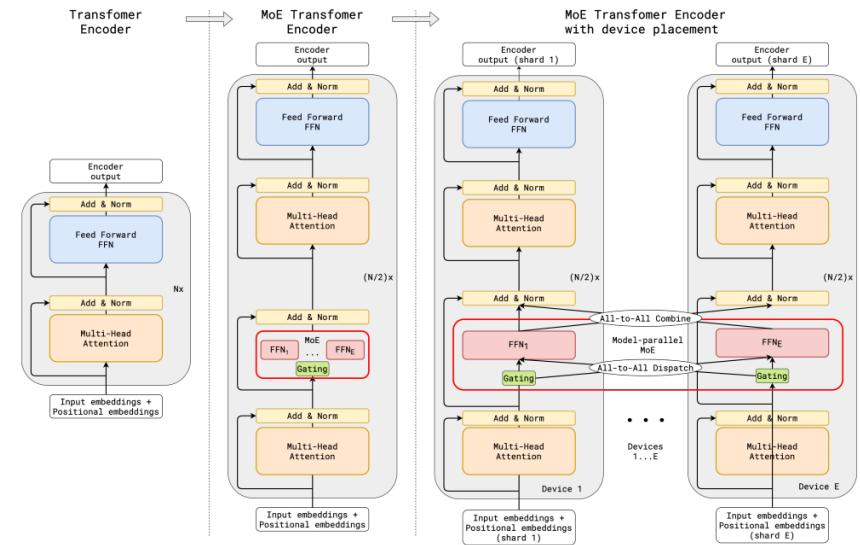
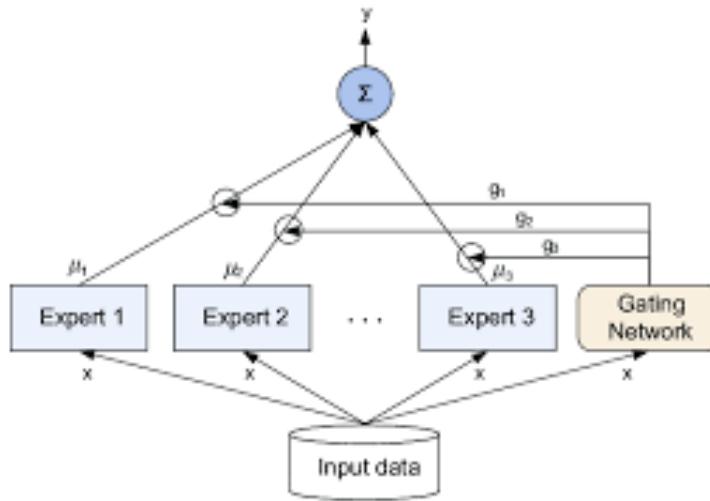
- **Е-шаг:** для всех  $m = 1, \dots, M$  фиксируем  $b_m$  и находим

$$(\tilde{g}_1, \dots, \tilde{g}_M) = \operatorname{argmin}_{g_1, \dots, g_M} \sum_{i=1}^l L\left(\left[\sum_{m=1}^M g_m(x_i) b_m(x_i)\right], y_i\right)$$

- Нормировка для всех  $m = 1, \dots, M$ :  $g_m(x) = \text{softmax}(\tilde{g}_1(x), \dots, \tilde{g}_M(x); \gamma)$
  - Если не стабилизировались  $g_m$  и  $b_m$  то переход на М-шаг

# Продвинутые смеси экспертов

- Предложено много расширений и подходов к обучению:
  - Иерархические:  $g_{i|j}(x)g_j(x)$
  - Байесовские:  $g_m = P(b_m|Z) \sim P(b_m) \cdot P(Z|b_m)$  – «байесовский вес» базовой модели или «условная компетентность» на наборе  $Z$ ,  $P(Z|b_m)$  – правдоподобие модели компетентности
  - Нейросетевые (в том числе с глубоким обучением и трансформерами)

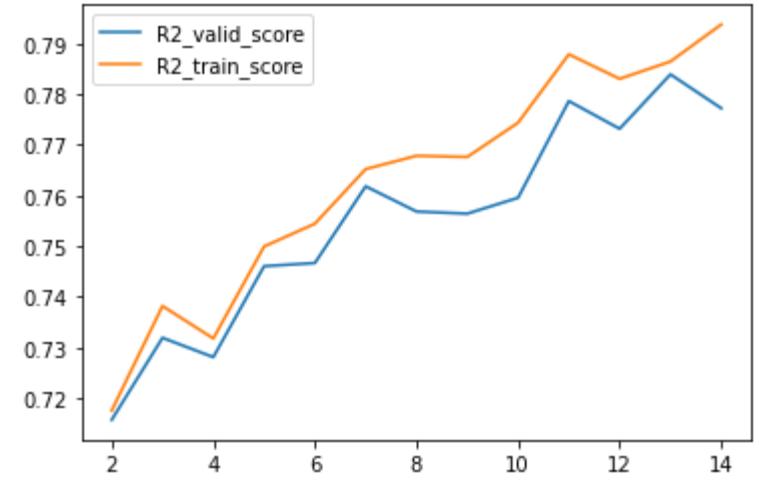


# Пример МОЕ

- Смесь экспертов - МНК линейных и полиномиальных регрессий на наборе California Housing, оценка качества по  $R^2$

```
import numpy as np
from smt.applications import MOE
housing = fetch_california_housing()
X, y = housing.data, housing.target
X.shape, y.shape, housing.target_names
N = 15000
X_train, y_train = X[:N], y[:N]
X_test, y_test = X[N:], y[N:]
def sklearn_fit_history_moe(X_train, y_train, X_valid, y_valid):
    result = []
    for i in range(2, 15, 1):
        print(f"Number of experts={i}")
        moe = MOE(n_clusters=i, allow=["LS", "QP"])
        moe.set_training_values(X_train, y_train)
        moe.train()
        d = {}
        d["i"] = i
        y_moe = moe.predict_values(X_valid)
        d["R2_valid_score"] = r2_score(y_valid, y_moe)
        y_moe = moe.predict_values(X_train)
        d["R2_train_score"] = r2_score(y_train, y_moe)
        result.append(d)
    return pd.DataFrame(data=result).set_index("i")

history=sklearn_fit_history_moe(X,y,X_test, y_test)
history.plot()
```



Number of experts=10	LS 3.1637648277425874	QP 2.9549016444130336
LS 10.29898080711618	QP 9.687474569666911	Best expert = QP
LS 10.528101535305112	QP 9.575876793498624	Best expert = QP
LS 5.501784342614732	QP 11.215110336591684	Best expert = QP
LS 5.224169041577111	QP 4.791085016027925	Best expert = LS
LS 7.296109359634265	QP 6.289615840759818	Best expert = QP
LS 6.771505099293365	QP 6.377682524464259	Best expert = QP
LS 7.732637050336106	QP 8.768475340330385	Best expert = LS
LS 7.939421703962029	QP 68.86759267628352	Best expert = LS