

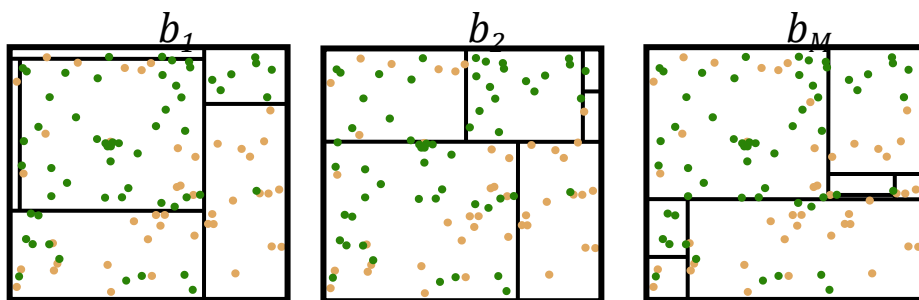


Лекция 12: Ансамбли моделей

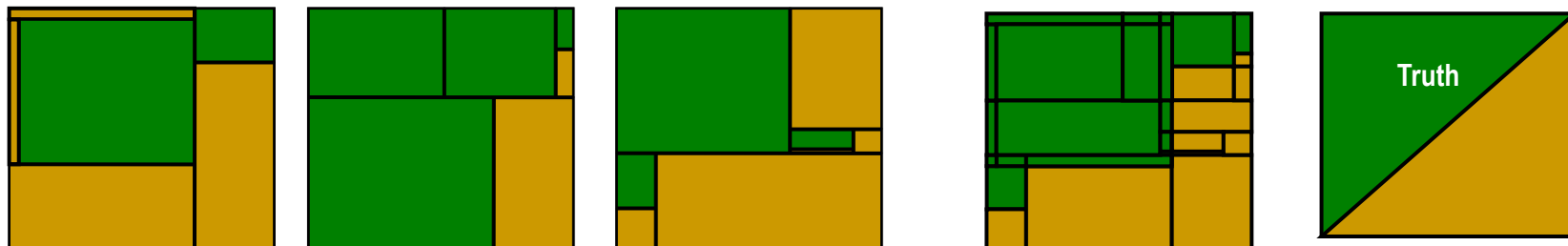
Общая идея ансамблей

■ 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 \gg 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. ξ 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:

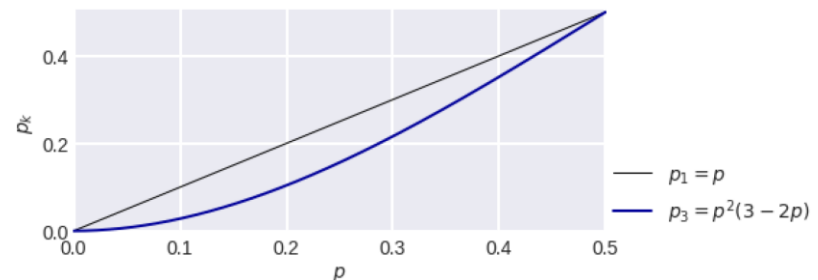
- верные $P(1,0,0) = P(0,1,0) = P(0,0,1) = (1-p)^2p$, $P(0,0,0) = (1-p)^3$

- 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:

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



- Но базовые алгоритмы не независимы ... как их разнообразить?

Типы ансамблей

- ЕСОС - 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

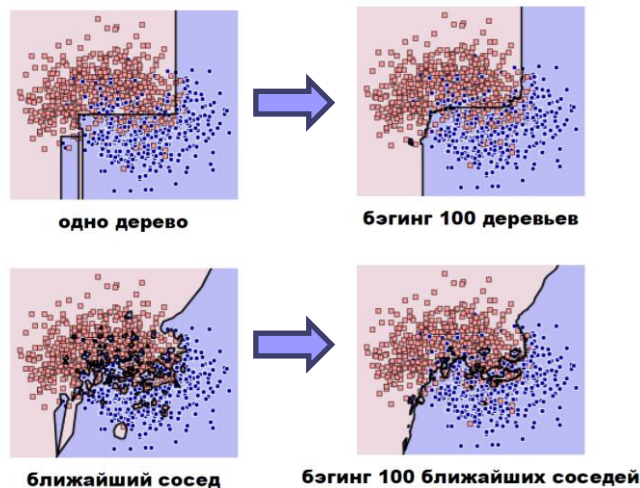
Типы ансамблей

- 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

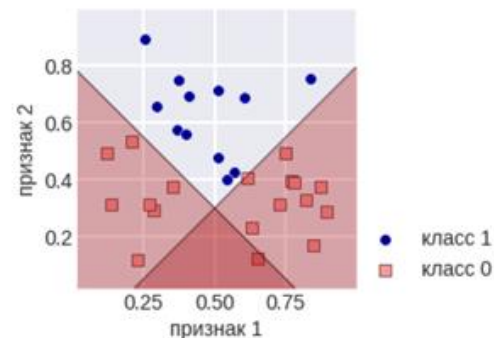


- Вычислительное обоснование:

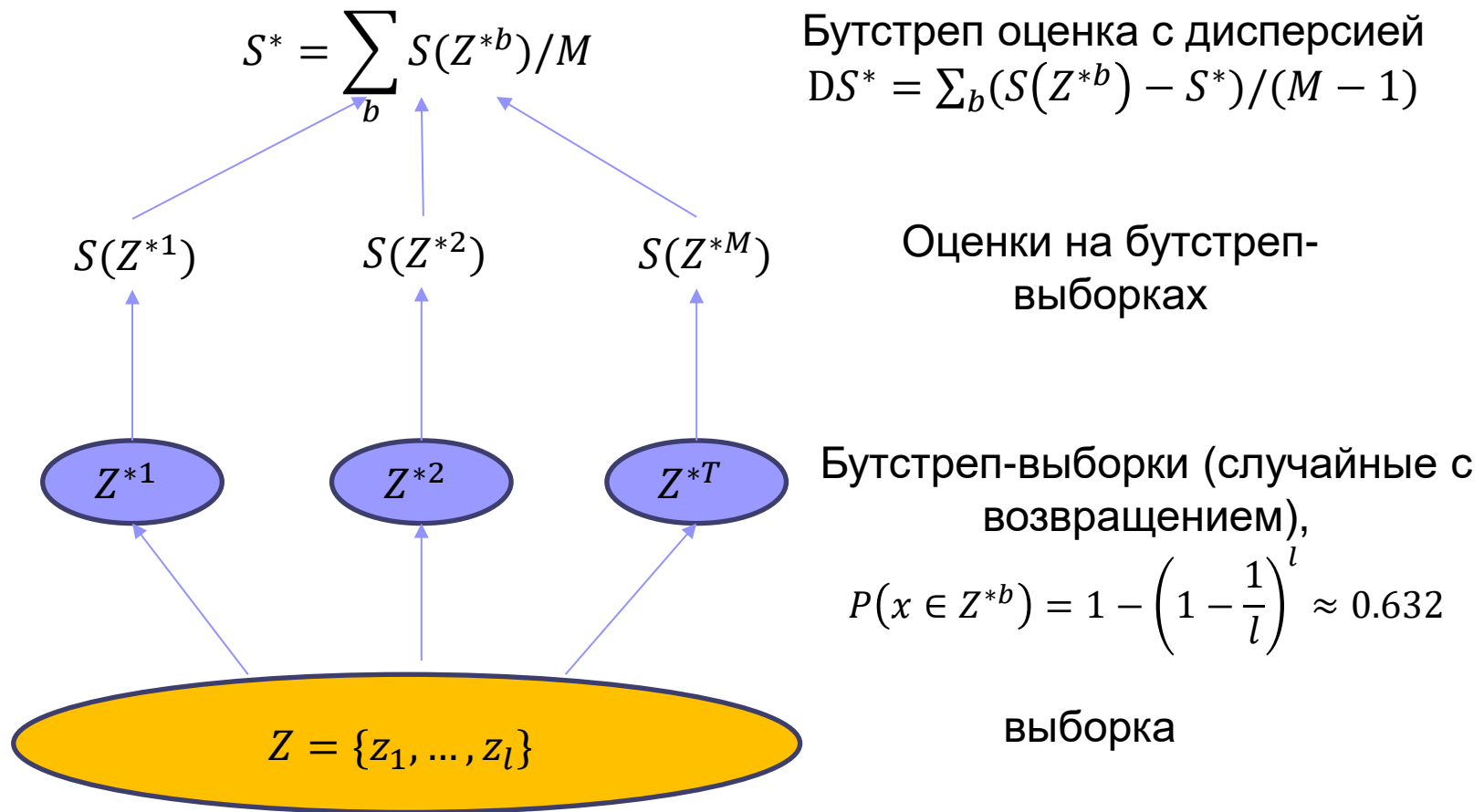
- ☐ Обучение многих типов ансамблей распараллеливается
- ☐ Зачастую ансамбль простых моделей обучать быстрее чем одну сложную модель

- Функциональное обоснование:

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



Бутсреппинг (вспоминаем)



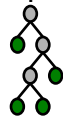
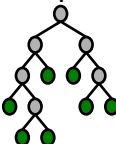
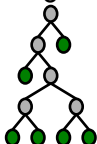

Важно: в отличие от методов макс. правдоподобия бутсреппинг позволяет строить не точечную оценку, а **распределение оценки** (в том числе прогноза, или параметра модели), даже в ситуациях, где ее теоретически не оценить

B(ootstrap)AG(gregation)ing

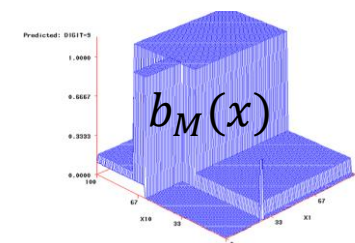
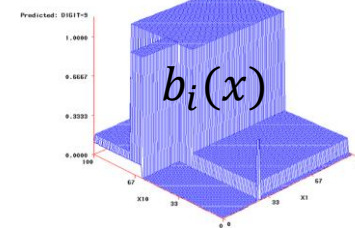
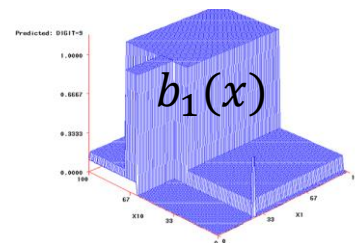
■ Алгоритм:

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

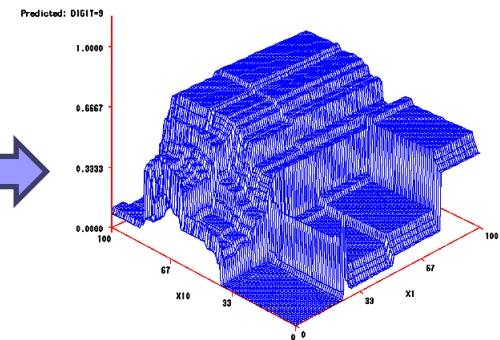
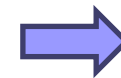
	b=1	b=2	b=...	b=M
case	freq	freq	freq	freq
1	1	0	3	1
2	0	1	1	1
3	2	0	0	2
4	0	2	2	0
5	2	2	0	1
6	1	1	0	1

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



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



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

ООВ 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 (несмещенная):

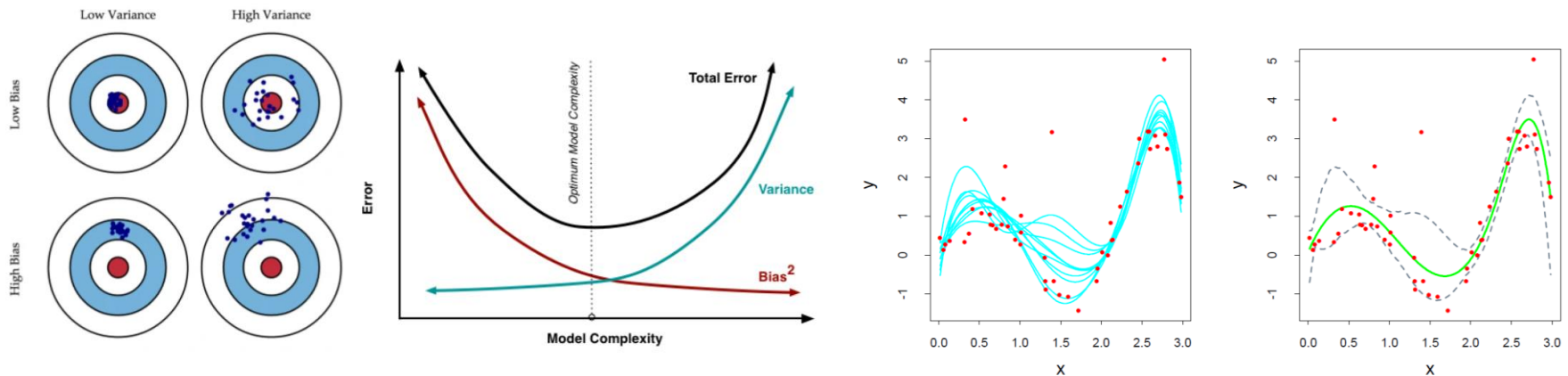
- с функцией потерь $L(b(x), y)$ для всего ансамбля $a(x)$ на обучающей выборке Z : $OOB = \frac{1}{l} \sum_{i: x_i \in Z} L(a_{OOB}(x_i), y_i)$

- Основное достоинство:

- можно оценивать качество модели не исключая примеры из тренировочной выборки

Какие модели хороши для Bagging

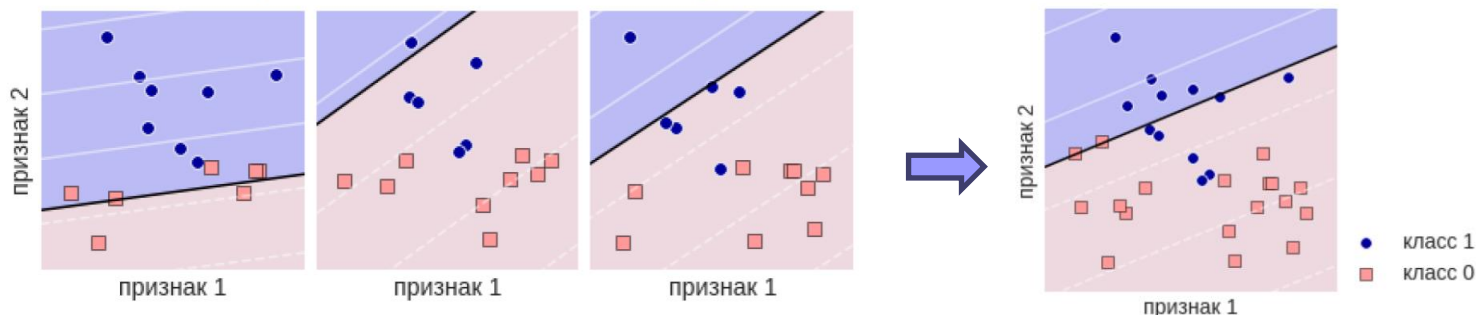
- Хотелось бы добиться «независимости» прогнозов в ансамбле:
 - Формально это невозможно, но можно «сымитировать» за счет использования **нестабильных** моделей, чтобы минимизировать корреляцию откликов базовых алгоритмов
 - Желательно **маленькое смещение + большая дисперсия** => хороши сложные переобученные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



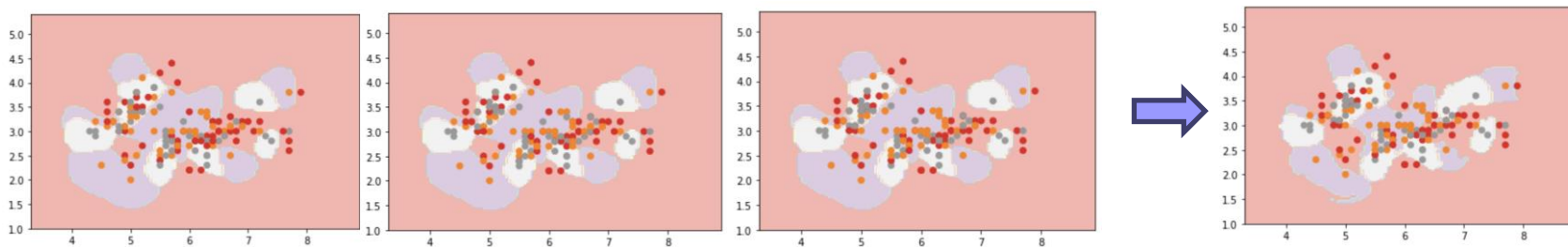
Какие модели плохи для Bagging

- 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

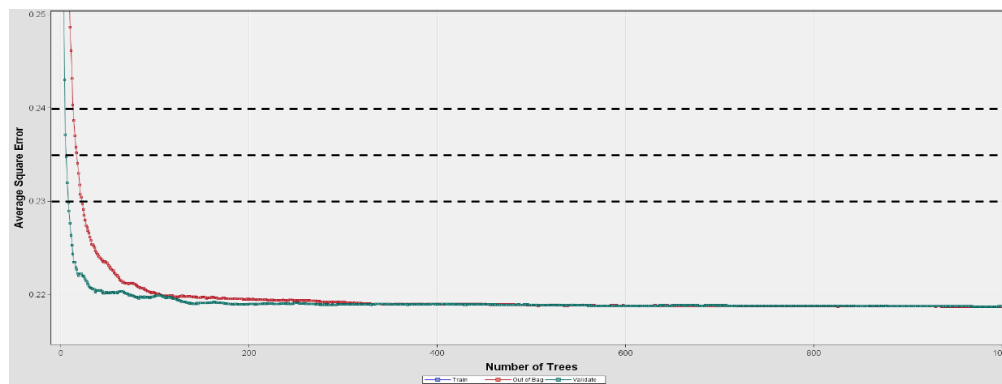
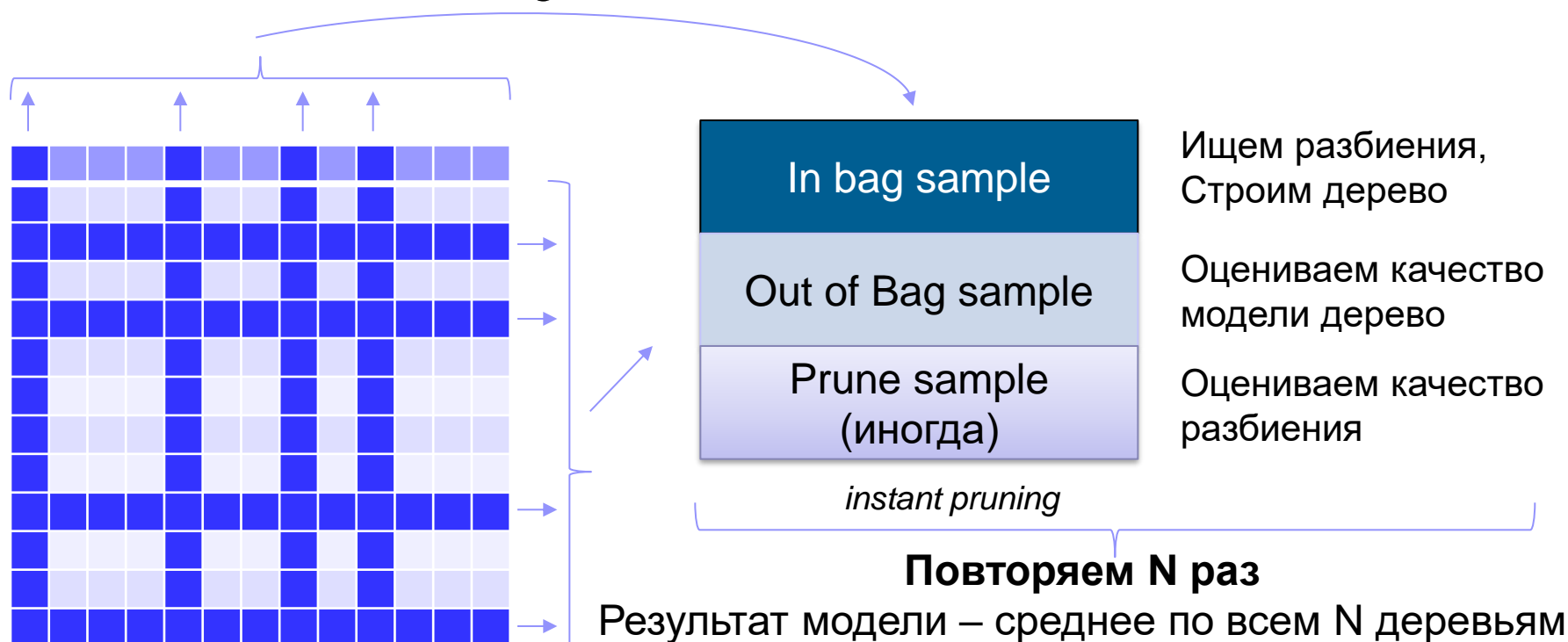
■ Основные особенности:

- *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

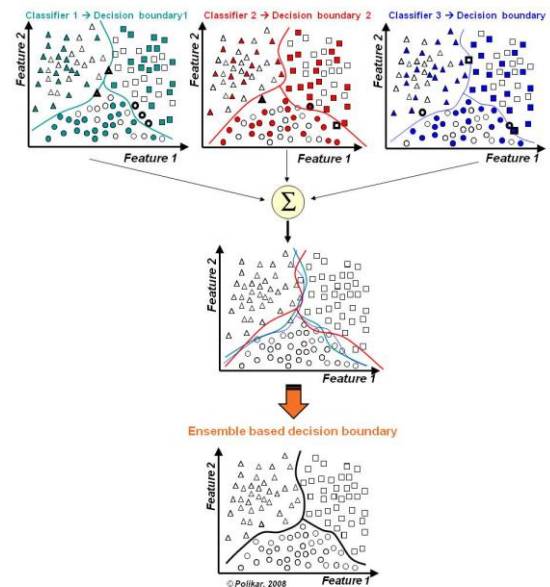
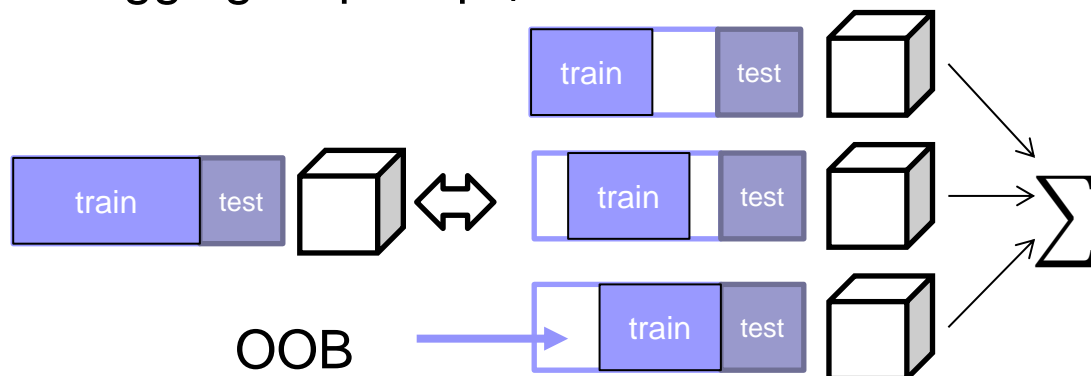


Ключевые параметры

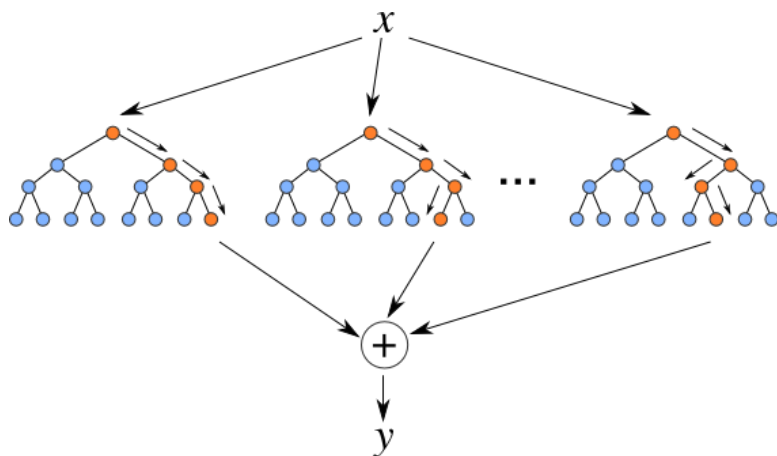
- Контроль сложности ансамбля:
 - размер ансамбля, чем больше тем сложнее, но не склонен переобучаться даже на больших ансамблях и выборках
- Контроль случайности базовой модели:
 - Число случайных признаков для поиска разбиения – чем меньше тем случайнее
 - Пропорция для sampling – чем меньше выборка 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

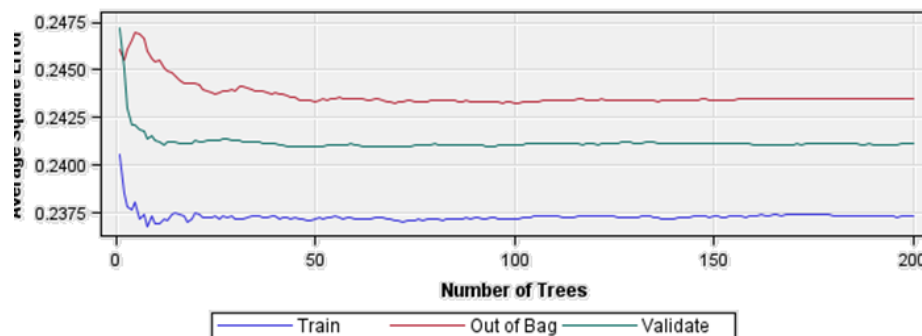
Bagging с пропорцией:



Применение ансамбля:



Оценка качества по OOB:



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.

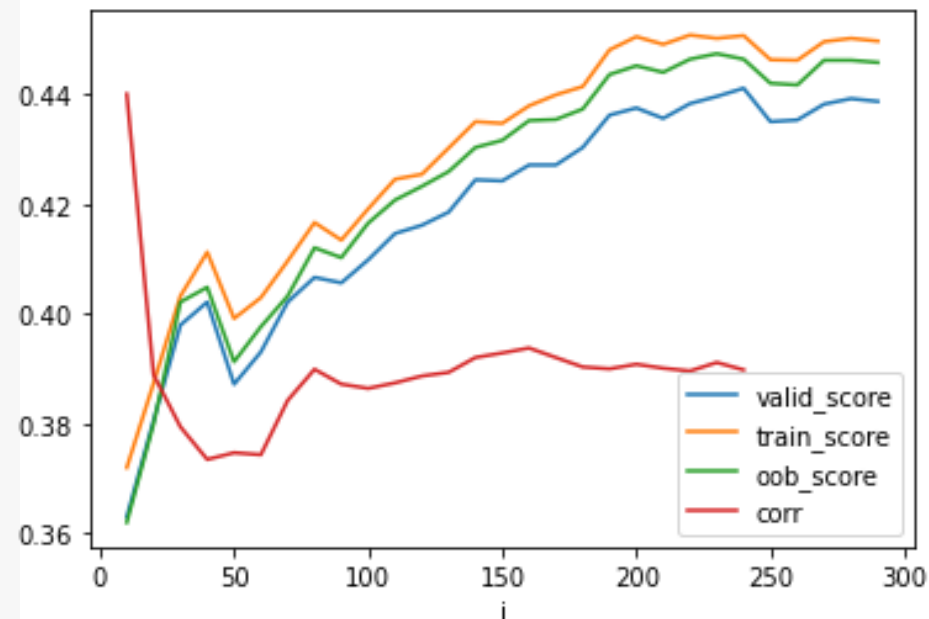
```
covtype_split = train_test_split(X, y, train_size=10000, test_size=10000, stratify=y, random_state=0)
X_train, X_test, y_train, y_test = covtype_split
```

```
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)
```

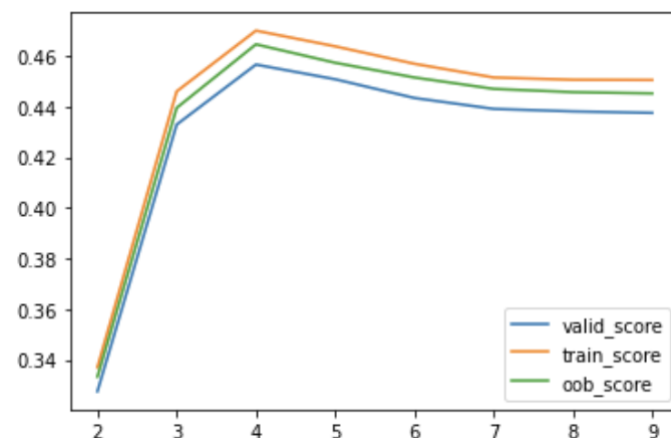
```
def sklearn_fit_history(model, n_estimators, X_train, y_train, valid=None):
    result = []
    warm_start=False
    for i in range(10, n_estimators, 10):
        model.set_params(n_estimators=i, warm_start=warm_start)
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        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)

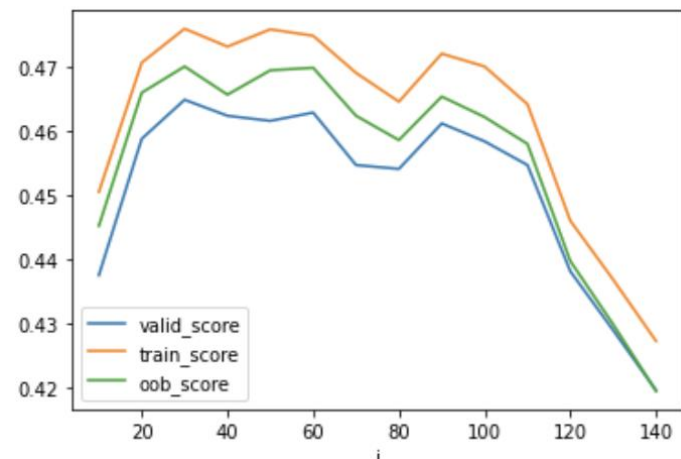
■ Глубина дерева:

```
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:

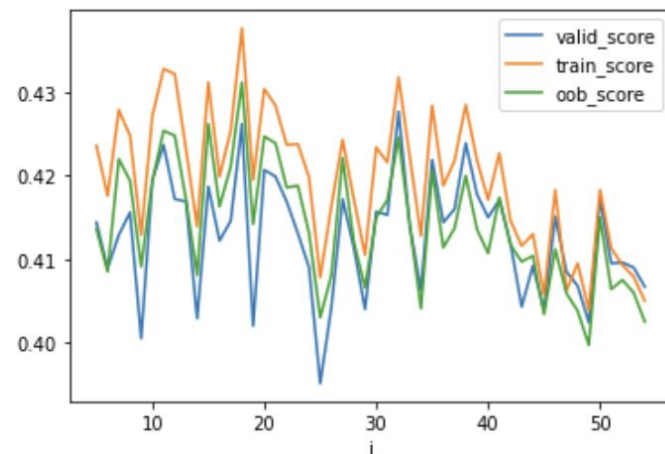
```
def sklearn_fit_history2(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(10, 150, 10):
        model.set_params(min_samples_leaf=i) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        d["i"] = i
        if valid is not None:
            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)

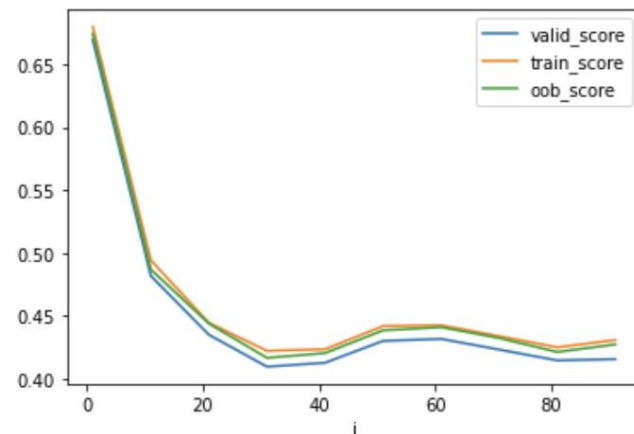
■ Число признаков:

```
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:

- Более **большое 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 ООВ, 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:

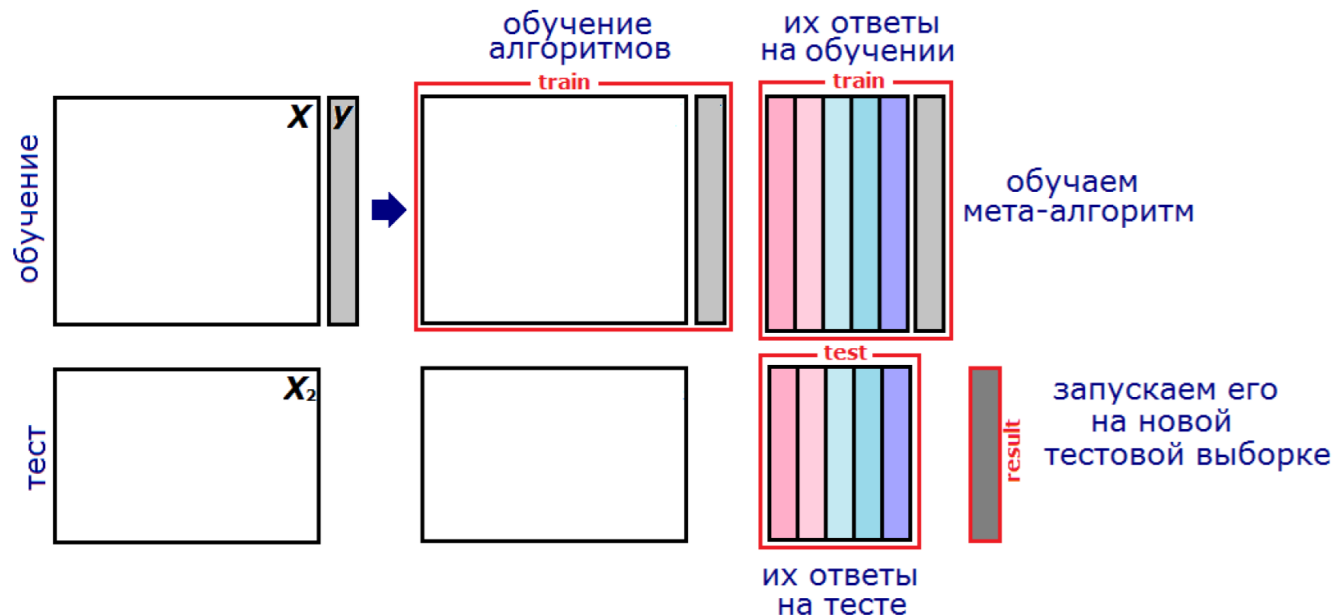
- Преобразование пространства признаков (feature engineering), использующее информацию об отклике, например, WOE, группировка или дискретизация на основе прогнозных моделей
- Некоторые привычные алгоритмы можно рассматривать как стекинг, например, SVM – стекинг базовых функций $b_i(x) = y_i K(x_i, x)$

- 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

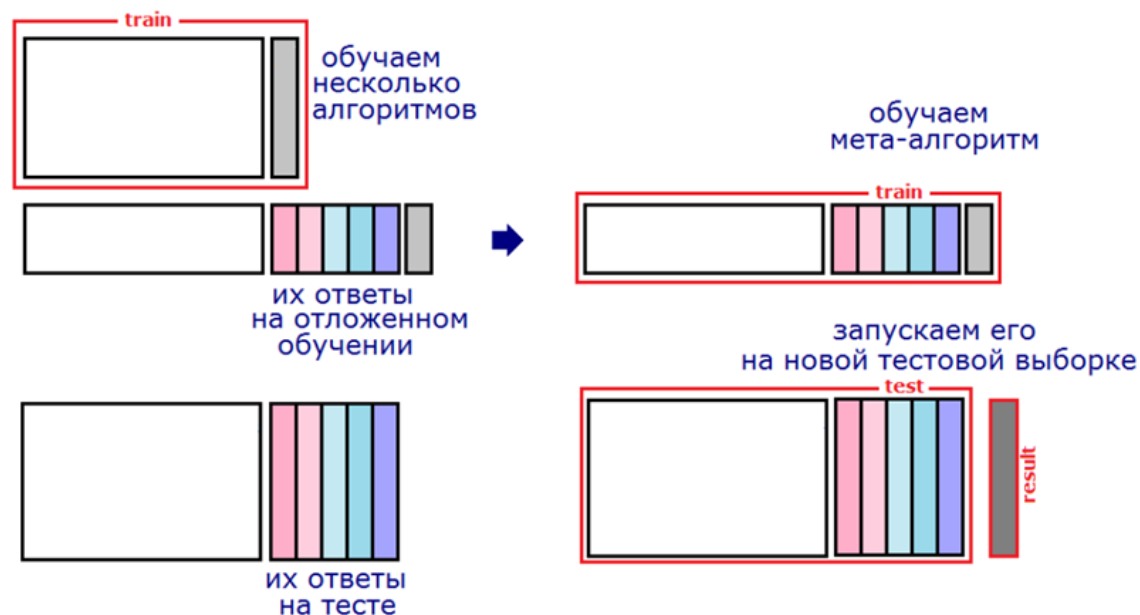
- Основные особенности:
 - обучаем мета-алгоритм на тех же данных, что и базовые алгоритмы
 - получаем переобученный прогноз



Варианты стекинга: объединение метапризнаков и обычных признаков

■ Основные особенности:

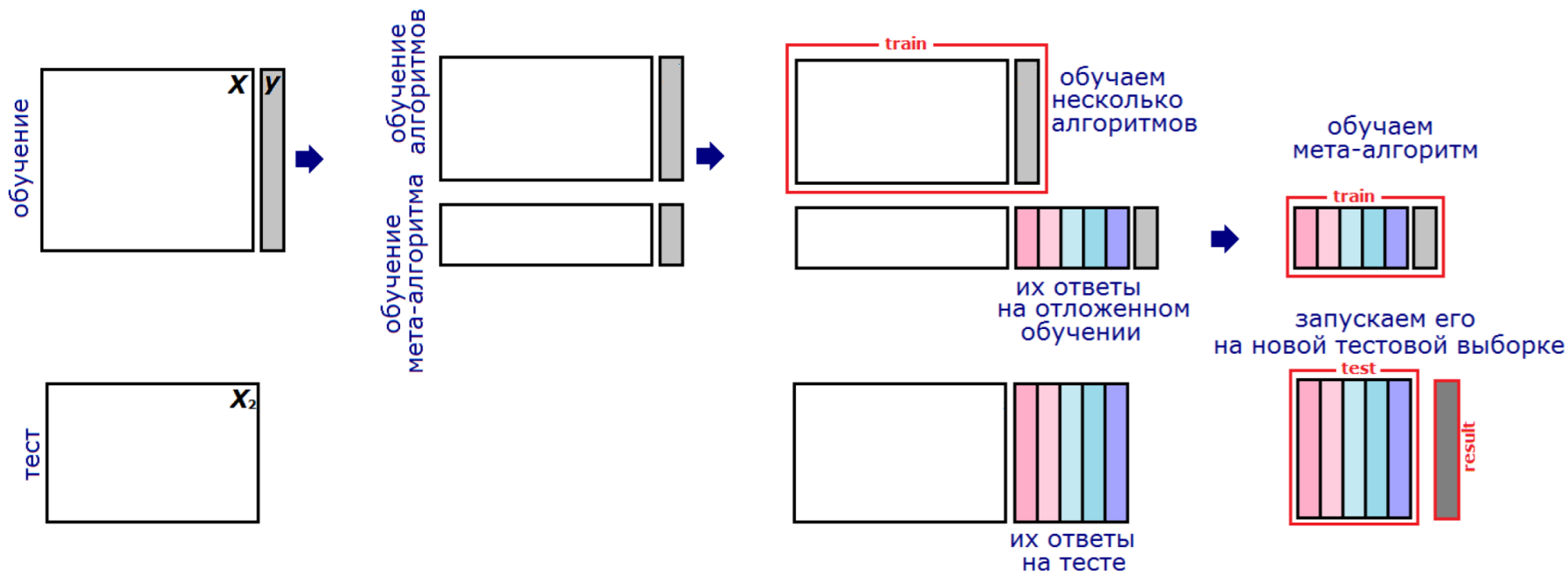
- Выделяем набор для контроля
- Обучаем базовые алгоритмы на тренировочном и применяем на контроле
- На комбинации признаков обучаем мета-алгоритм



Варианты стекинга: блендинг

■ Основные особенности:

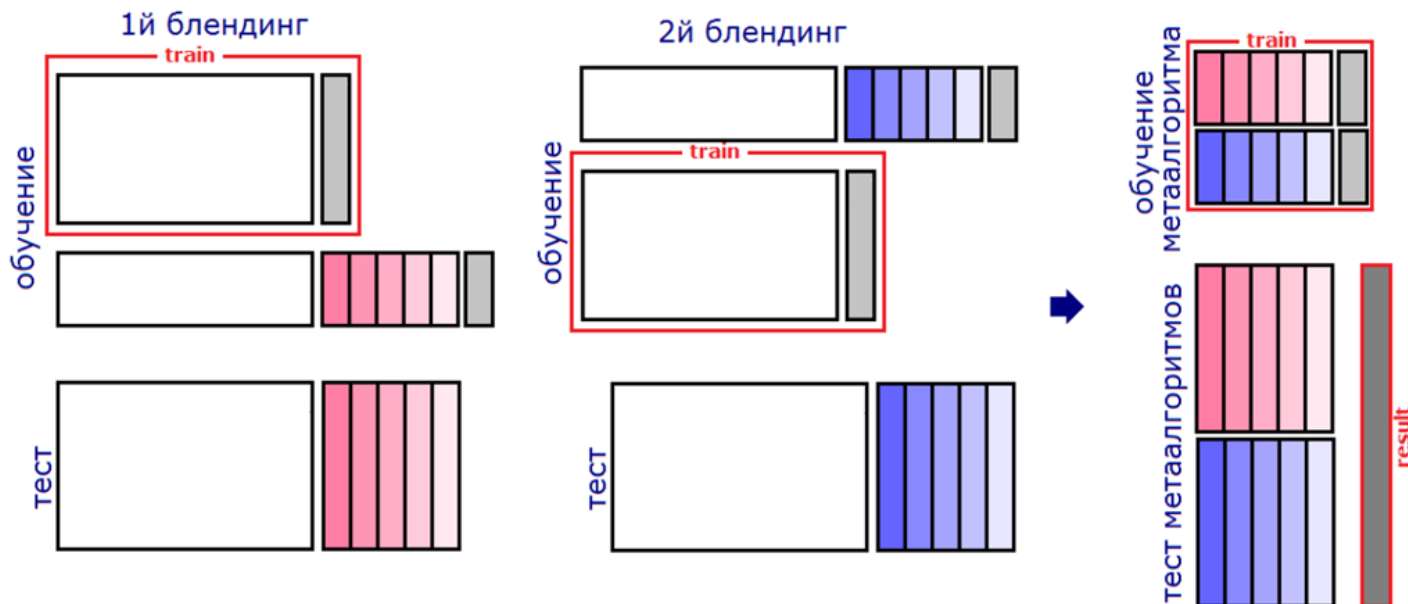
- обучаем мета-алгоритм на отложенных данных
- нет переобучения, но учим базовые и мета-алгоритмы не на всей выборке



Варианты стекинга: объединение таблиц

■ Основные особенности:

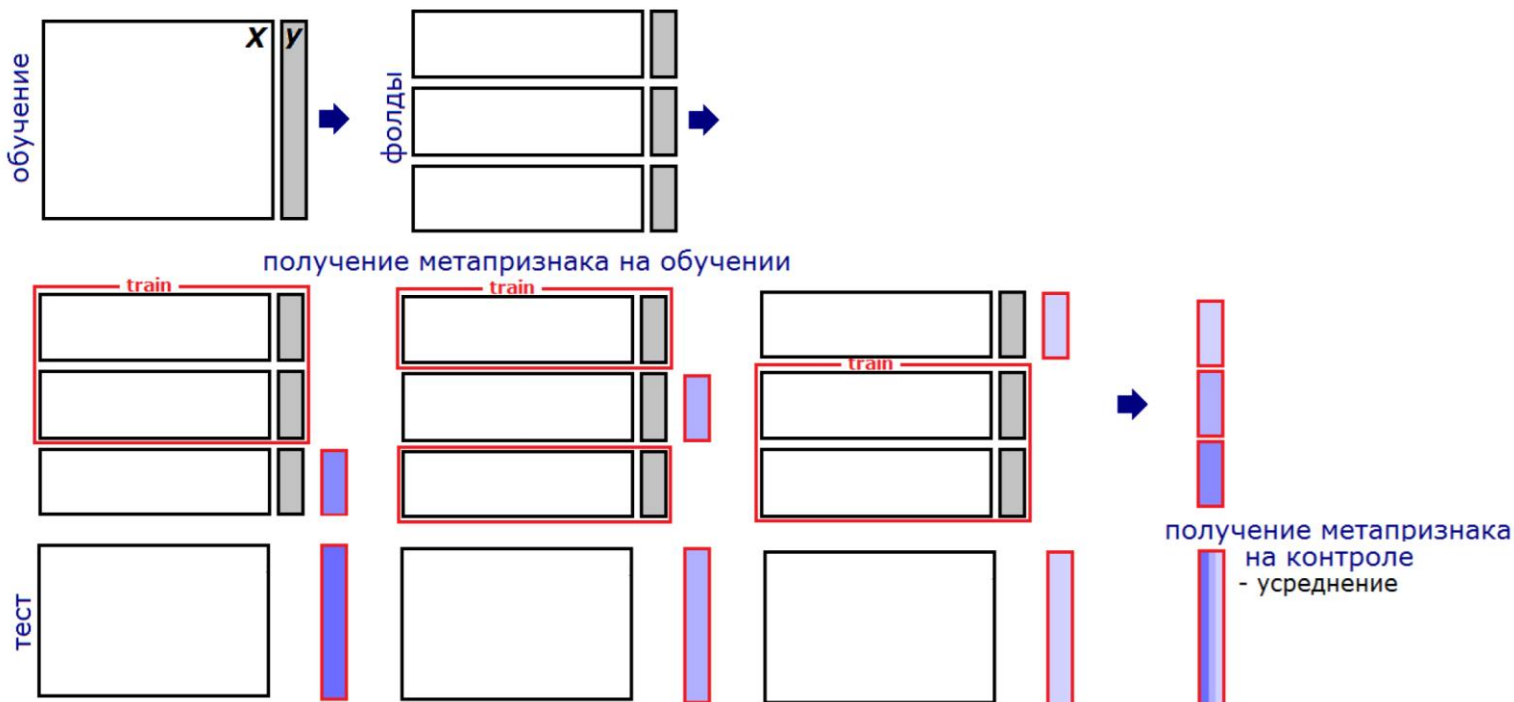
- разбиваем выборку на блоки, на части учим базовые алгоритмы, на оставшихся применяем
- «склеиваем» мета-признаки, мета-признаки строятся разными базовыми моделями



Варианты стекинга: объединение метапризнаков

■ Основные особенности:

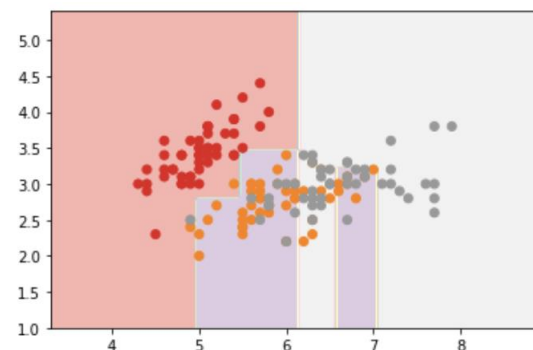
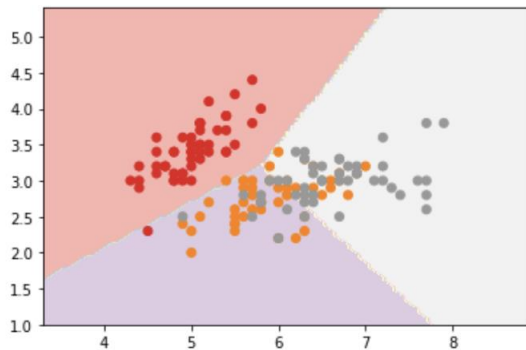
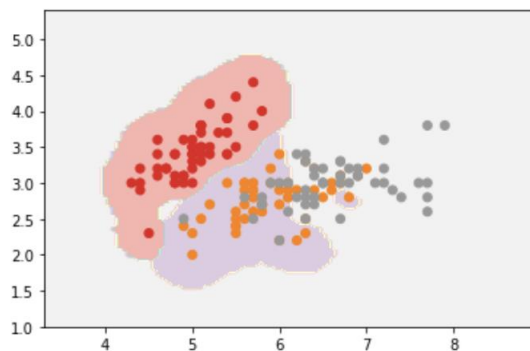
- разбиваем выборку на блоки, на части учим базовые алгоритмы, на оставшихся применяем (как в CV)
- «перемешанные» мета-признаки, одни и те же мета-признаки строятся разными базовыми моделями, надо агрегировать



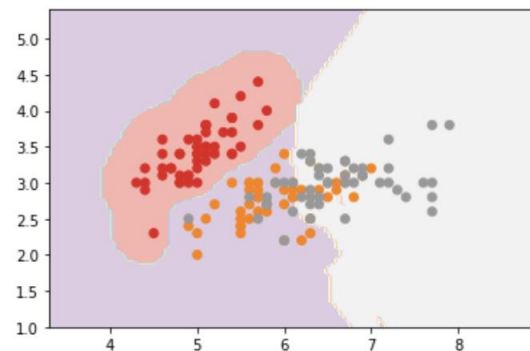
Пример стекинга

- Базовые алгоритмы (слева направо):

- RBF SVM, логистическая регрессия, дерево решений с gini:



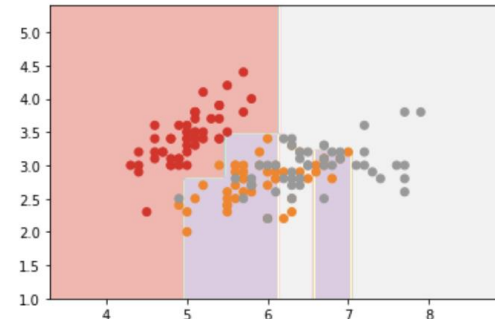
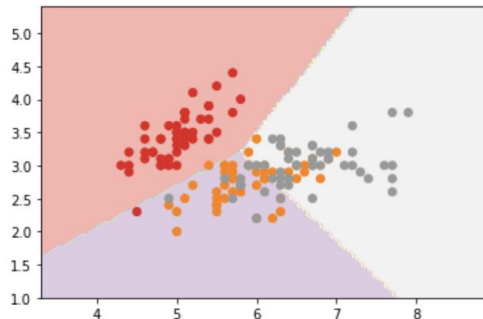
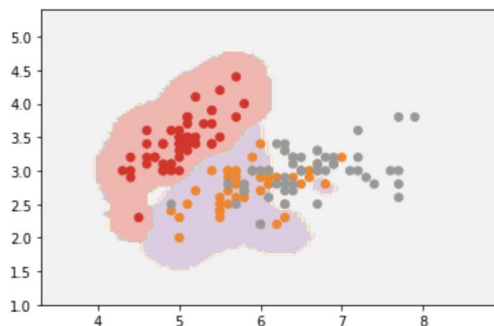
- Мета-алгоритм – линейная логистическая регрессия:



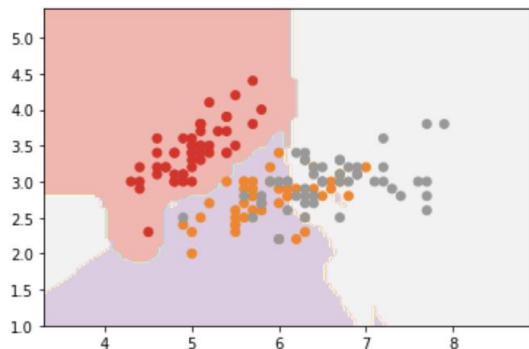
```
from sklearn.ensemble import StackingClassifier
estimators = [
    ('tree', DecisionTreeClassifier(criterion="gini", max_depth=5,
                                   min_samples_split=5, min_samples_leaf=3, ccp_alpha=0.0)),
    ('svc', SVC(kernel="rbf", gamma=10)), ('logreg', LogisticRegression())
]
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 – регуляризованный 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)