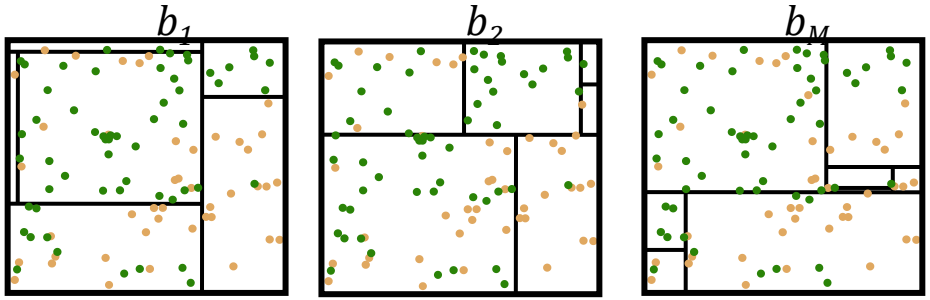
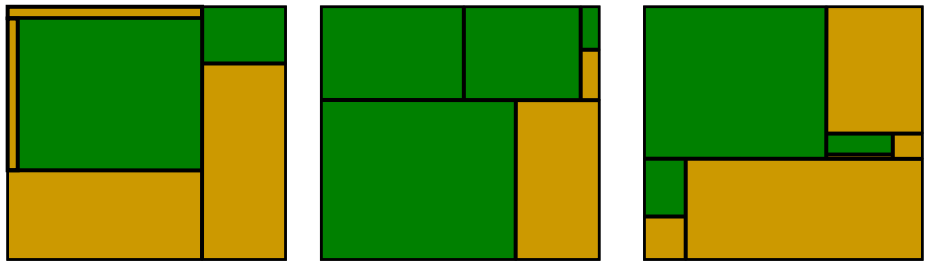
Ансамбли моделей

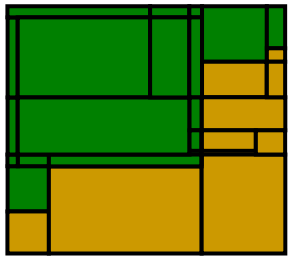
Общая идея ансамблей

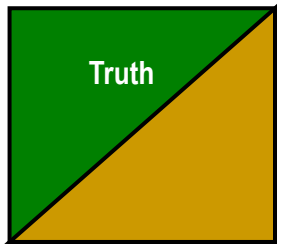
- Ансамбль:
  - Строим **базовые** (слабые) **алгоритмы** (модели) $\{b_i(x)|b_i: X \to R\}_{i=1}^{M}$, хотелось бы независимые, но хотя бы существенно отличающиеся
  - Агрегируем их прогнозы в **ансамбль** $a(x) = F(b_1(x), \dots, b_M(x))$, где $F: R^M \to Y$ – функция агрегации или **мета-алгоритм**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



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







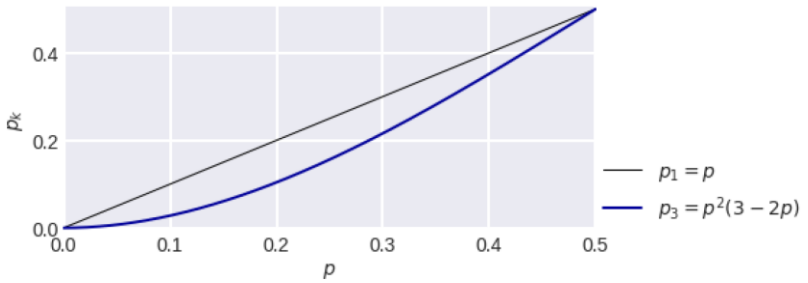
Примеры агрегаций

- Голосование:
  - простое $a(x) = \text{argmax}_i[b_i(x)]$
  - взвешенное $a(x) = \text{argmax}_i[\alpha_i b_i(x)], \sum\alpha_i = 1, \alpha_i \geq 0$
  - с регуляризацией, например, $a(x) = \text{argmax}_i[\alpha_i b_i(x)], \ \sum|\alpha_i| \leq C$
- 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 $\min\limits_{1\leq i\leq M} b_i \leq f(b_1, \dots, b_M) \leq \max\limits_{1\leq i\leq M} b_i$, $f(.)$ – непрерывная, монотонная, …
- Смесь экспертов
  - $a(x) = \sum g_i(x) b_i(x)$, где $g_i: X \to \mathbb{R}^+$ - функция компетентности, строится (обучается) отдельно и зависит от x

Проблема разнообразия и независимости базовых алгоритмов

- Оценка непрерывной с.в. ξ по ее **независимым** измерениям $\{\xi_i\}$:
  - $\mathrm{E}(\xi) = \mathrm{E}\left(\frac{1}{M}\sum\xi_i\right) = \mathrm{E}\xi_i$, $\mathrm{D}\xi = \frac{1}{M^2}\sum\mathrm{D}\xi_i = \frac{1}{M}\mathrm{D}\xi_i \to 0$, при $M \to \infty$
- Голосование в комитете (демо-пример) пусть вероятность ошибки p, тогда при трех **независимых** базовых алгоритмах и верном ответе 0 получаем варианты:
  - верные $\mathrm{P}(1,0,0) = \mathrm{P}(0,1,0) = \mathrm{P}(0,0,1) = (1-p)^2 p$, $\mathrm{P}(0,0,0) = (1-p)^3$
  - неверные $\mathrm{P}(1,1,1) = p^3$, $\mathrm{P}(1,1,0) = \mathrm{P}(0,1,1) = \mathrm{P}(1,0,1) = (1-p)p^2$
  - 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?

Основные типы ансамблей

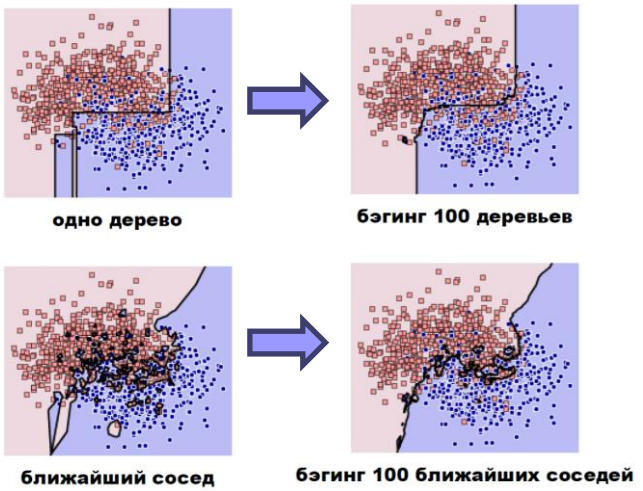
- Комитеты (голосование/усреднение) – простые агрегации, базовые алгоритмы однотипные, обычно варьируем выборку:
  - Pasting - случайные выборки (Bagging - с возвращением)
  - Random subspaces – случайные подмножества признаков
  - Random patches = Pasting/Bagging + Random subspaces
  - Cross-validation 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

Основные типы ансамблей

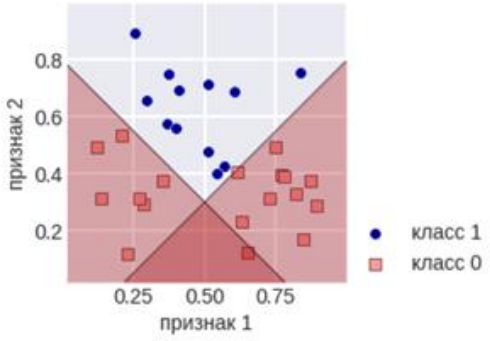
- Boosting («усиление слабых моделей») – каждый следующий базовый алгоритм пытается исправить ошибку предыдущих:
  - аддитивный (не совсем бустинг) – каждый следующий базовый алгоритм обучается на остатках от предыдущего ансамбля (например, FSAM)
  - каждый следующий базовый алгоритм с взвешенной функцией 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 (в курсе не рассматриваем)
- ECOC - кодирование отклика (в курсе не рассматриваем)

Чем хороши ансамбли?

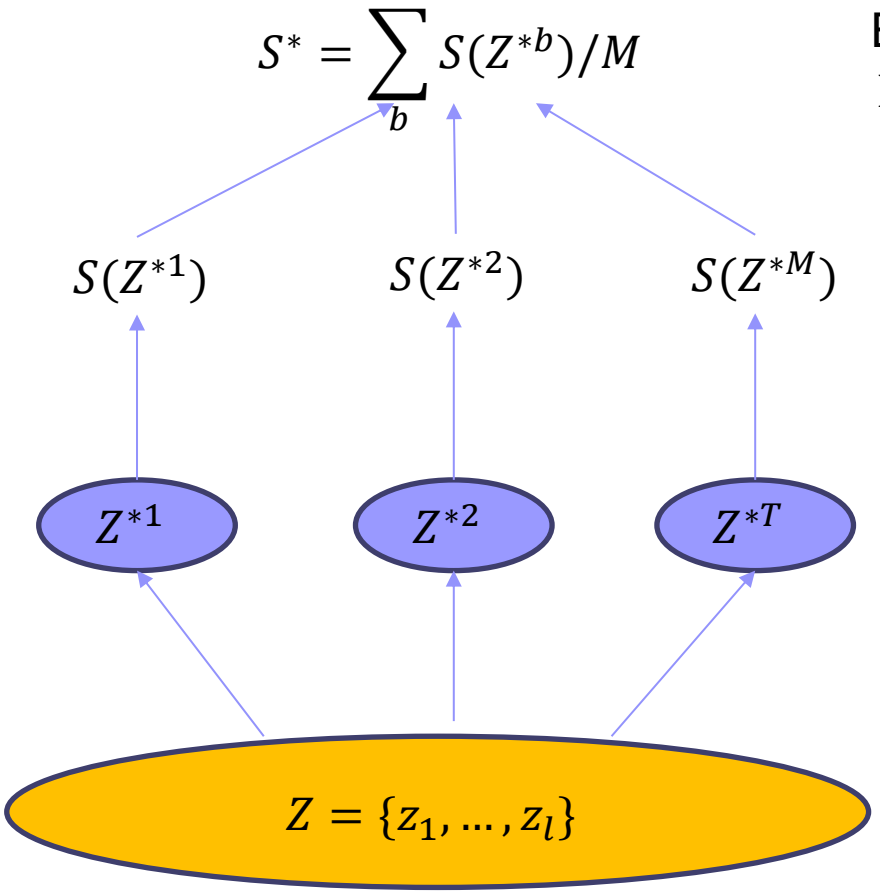
- Статистическое обоснование:
  - Борьба с недообучением
  - Борьба с переобучением



- 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)



Бутстреп оценка с дисперсией

$\mathrm{D}S^* = \sum_b (S(Z^{*b}) - S^*)/(M-1)$

Оценки на бутстреп-выборках

Бутстреп-выборки (случайные с возвращением),

$$P(x \in Z^{*b}) = 1 - \left(1 - \frac{1}{l}\right)^l \approx 0.632$$

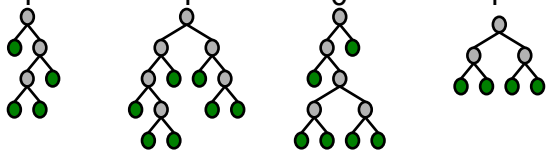
выборка

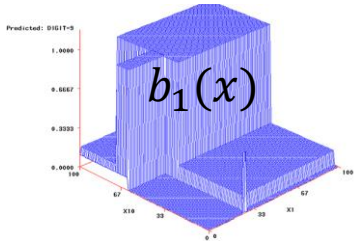
Важно: в отличии от методов макс. правдоподобия бутсреппинг позволяет строить не точечную оценку, а **распределение оценки** (в том числе прогноза, или параметра модели), даже в ситуациях, где ее теоретически не оценить

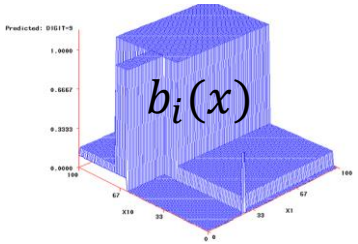
B(ootstrap)AG(gregation)ing

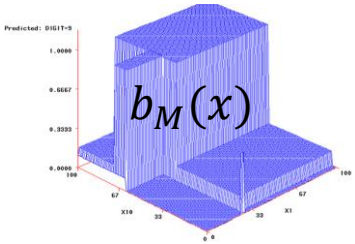
- Алгоритм:
  - Обучение: генерируем M выборок с возвращением, независимо подгоняем на них базовые классификаторы
  - Применение: применяем каждый базовый, результат усредняем

	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

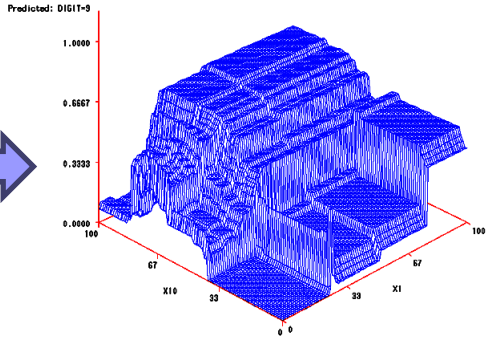








$$a_{bag}(x) = \frac{1}{M}\sum b_i(x)$$



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

В идеале при $M \to \infty$:

$a_{bag}(x) \to a_{opt}(x)$,

$\mathrm{Var}[a_{bag}(x)] \to 0$

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

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

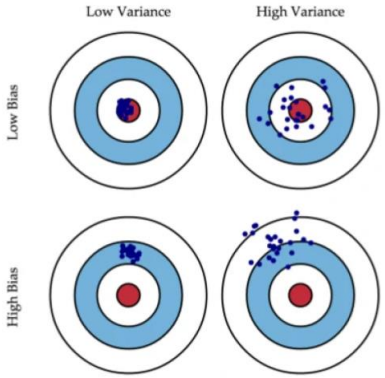
- Out-of-bag прогноз x:

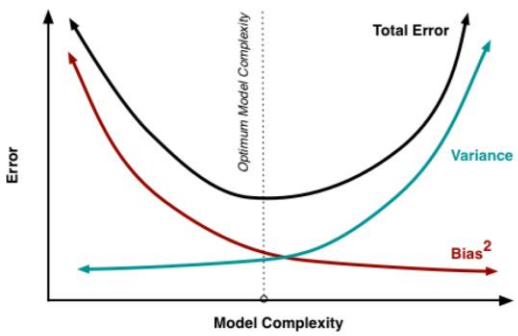
$$a_{OOB}(x) = \frac{1}{|\{i: x \in OOB_i\}|} \sum_{i:x \in OOB_i} b_i(x)$$

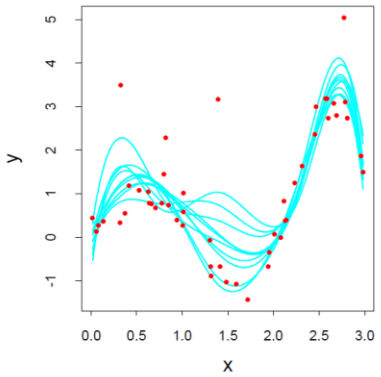
- Out-of-bag оценка (несмещенная):
  - с функцией потерь $L(b(x), y)$ для всего ансамбля $a(x)$ на обучающей выборке Z: $OOB = \frac{1}{l} \sum_{i:x_i \in Z} L(\, a_{OOB}(x_i), y_i)$
- Основное достоинство:
  - можно оценивать качество модели не исключая примеры из тренировочной выборки

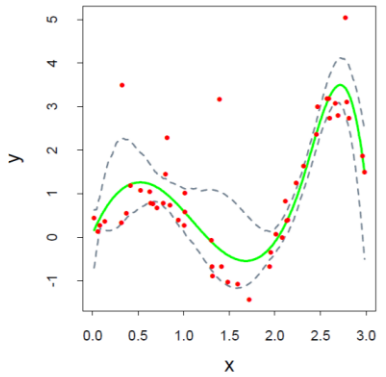
Какие модели хороши для Bagging

- 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



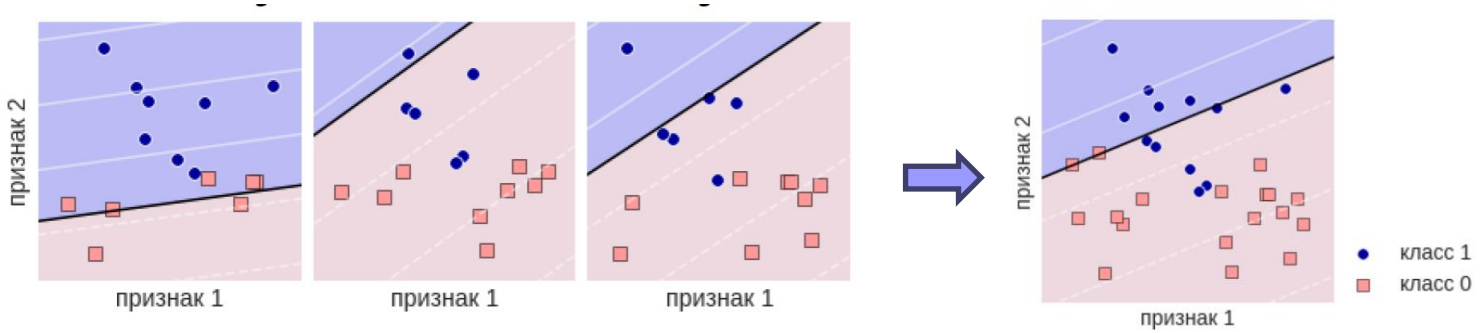




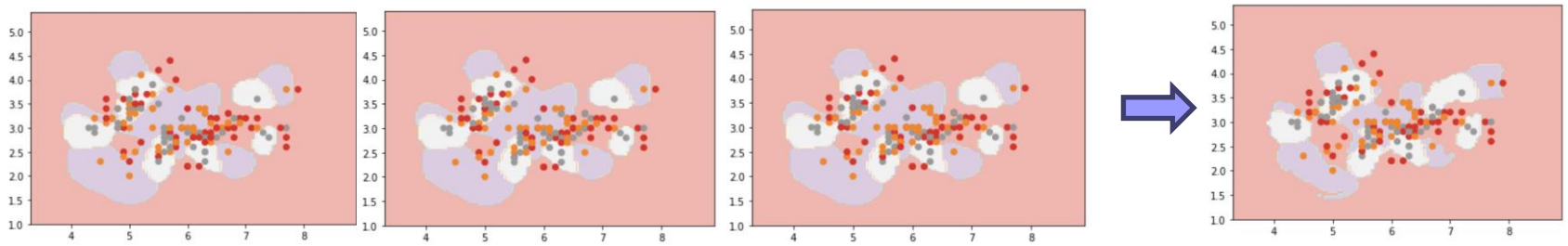


Какие модели плохи для Bagging

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



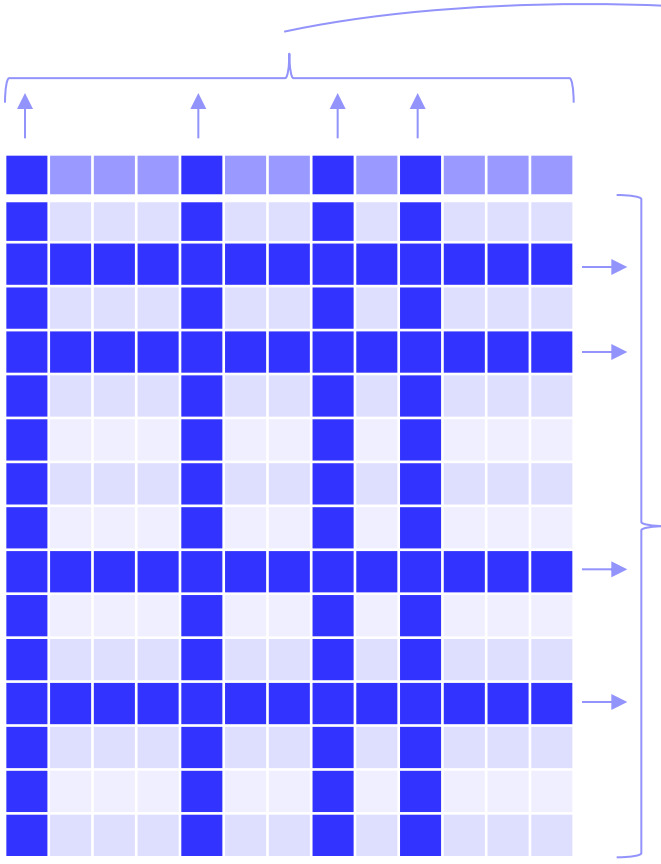
  - сложные нелинейные, но стабильные модели, например, SVM



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). $\sqrt{\#\ inputs}$ 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



In bag sample

Out of Bag sample

Prune sample (иногда)

Ищем разбиения, Строим дерево

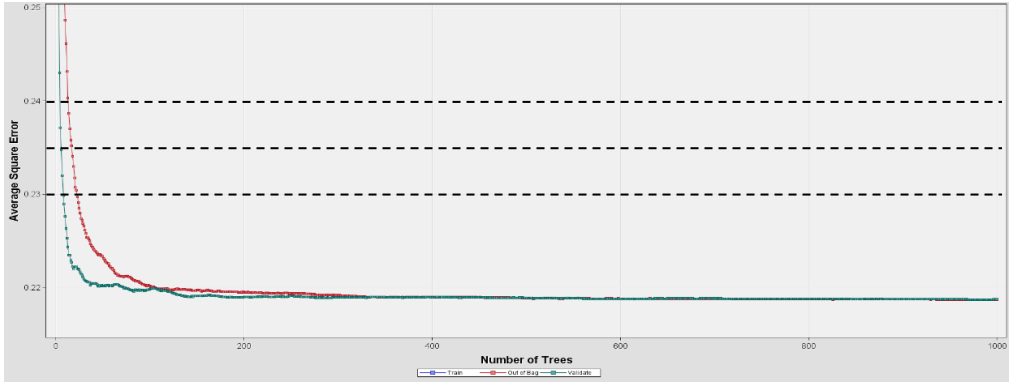
Оцениваем качество модели дерево

Оцениваем качество разбиения

instant pruning

Повторяем N раз

Результат модели – среднее по всем N деревьям

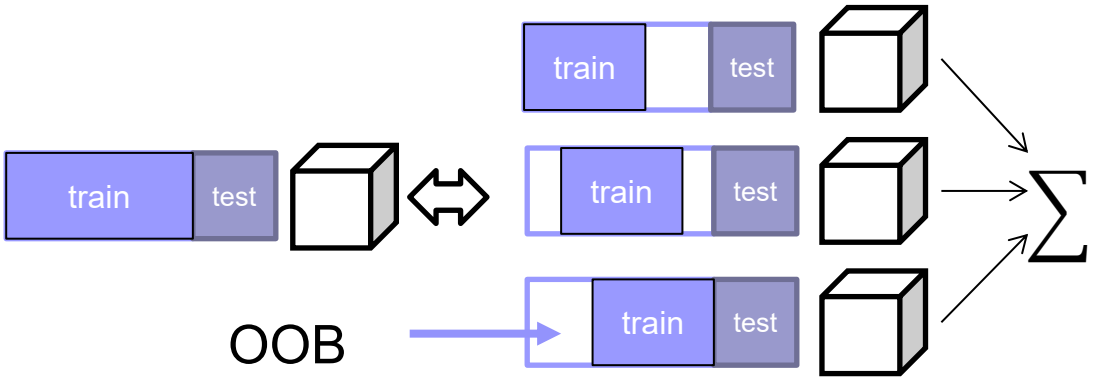


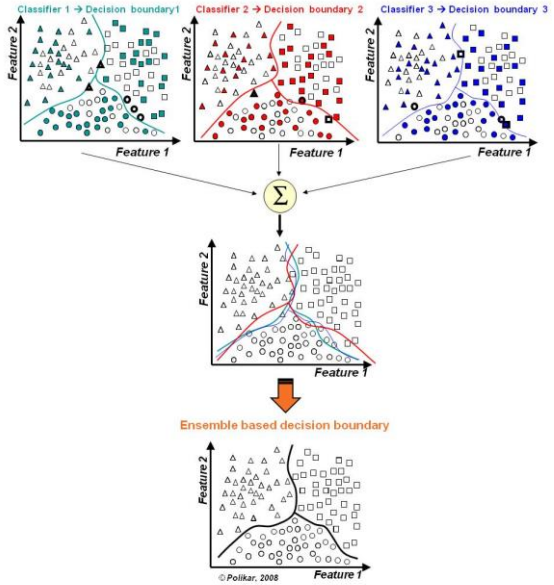
Ключевые параметры

- 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

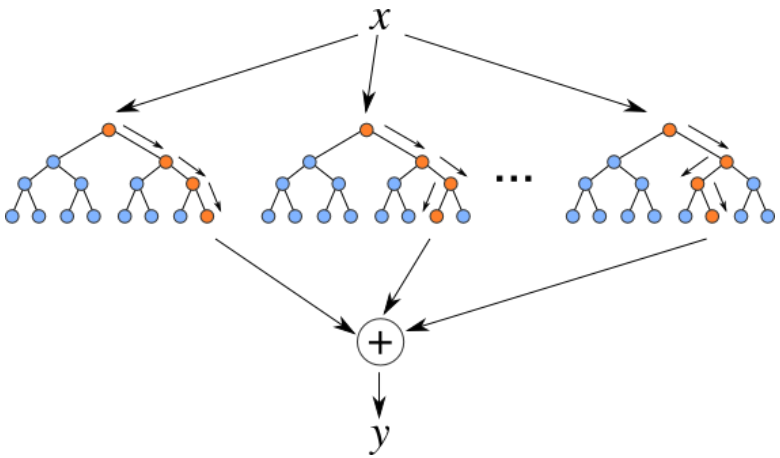
Иллюстрация работы случайного леса

Bagging с пропорцией:

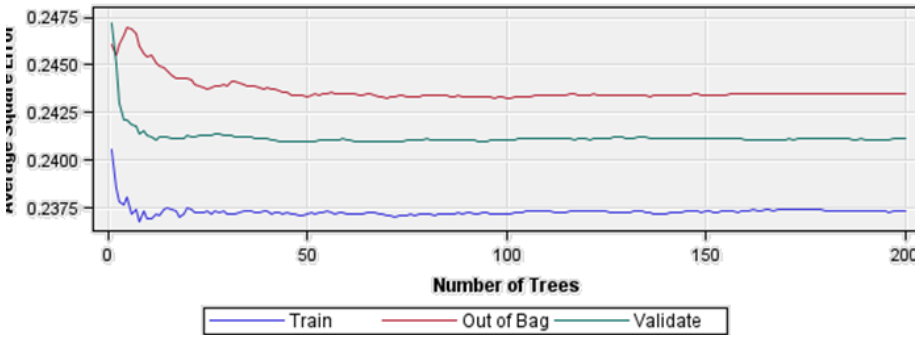




Применение ансамбля:



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 covertypes
-----------------

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 covertypes, making this a multiclass classification problem.
Each sample has 54 features, described on the
`dataset's homepage <https://archive.ics.uci.edu/ml/datasets/Covertype>`__.
Some of the features are boolean indicators,
while others are discrete or continuous measurements.
```

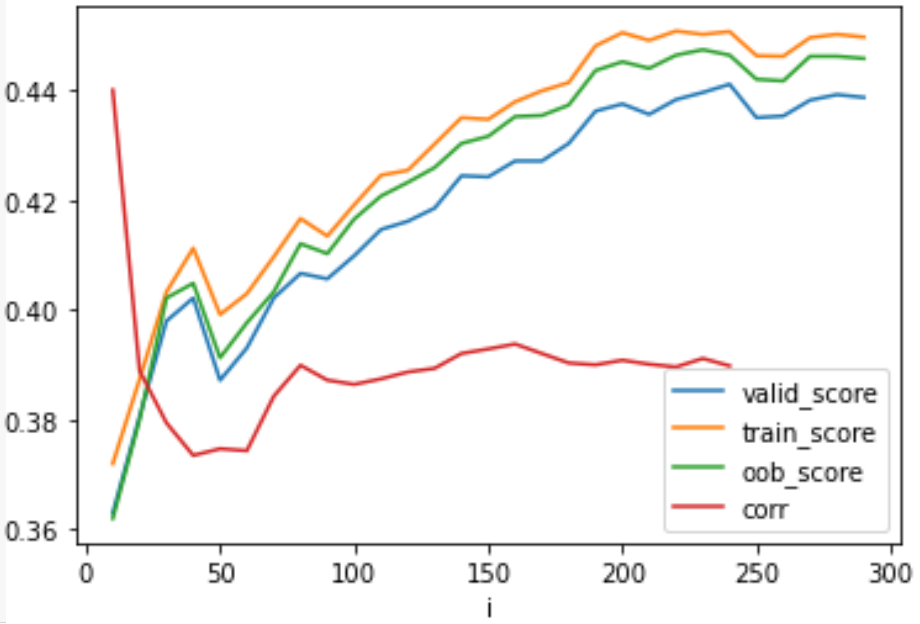
```
covtype_split = train_test_split(X, y, train_size=10000, test_size=10000, stratify=y, random_state=0)
X_train, X_test, y_train, y_test = covtype_split
```

```
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)
```

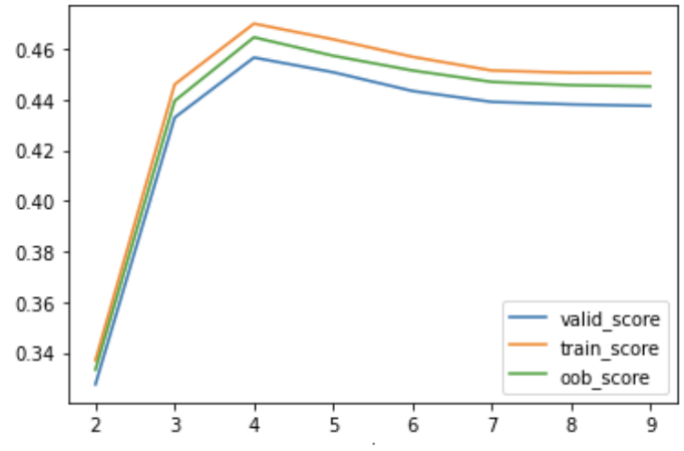
```
def sklearn_fit_history(model, n_estimators, X_train, y_train, valid=None):
    result = []
    warm_start=False
    for i in range(10, n_estimators, 10):
        model.set_params(n_estimators=i, warm_start=warm_start)
        model.fit(X_train, y_train)
        d = {}
        d["i"]=i
        if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

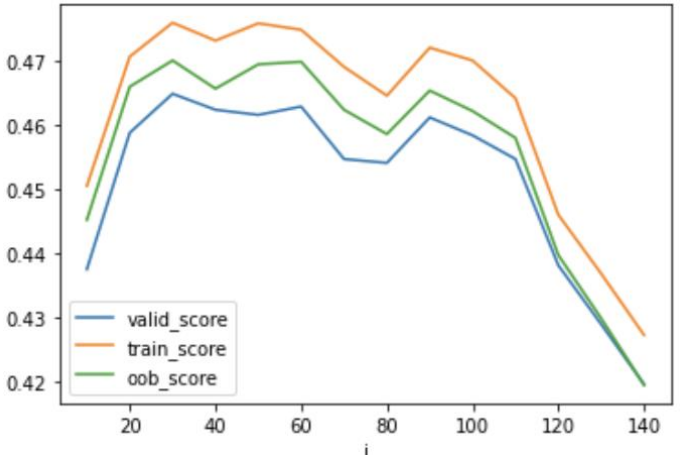
- Глубина дерева:

```
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:

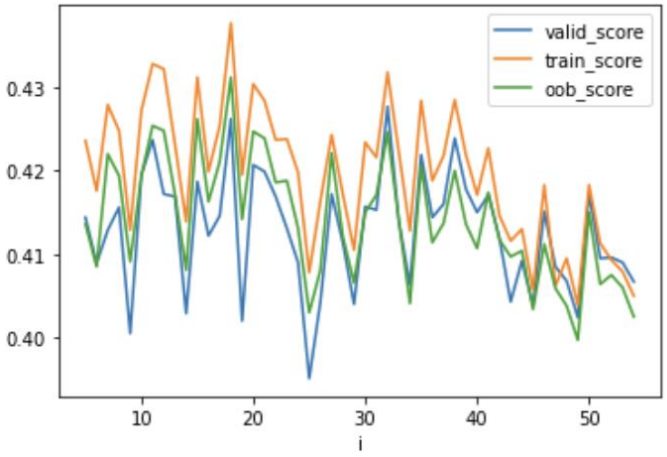
```
def sklearn_fit_history2(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(10, 150, 10):
        model.set_params(min_samples_leaf=i) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        d["i"]=i
        if valid is not None: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

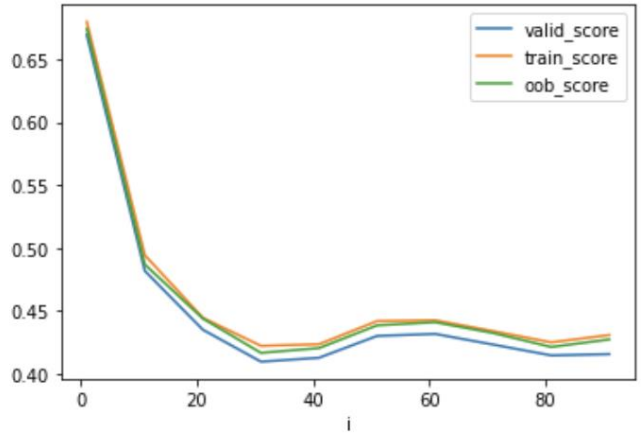
- Число признаков:

```
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

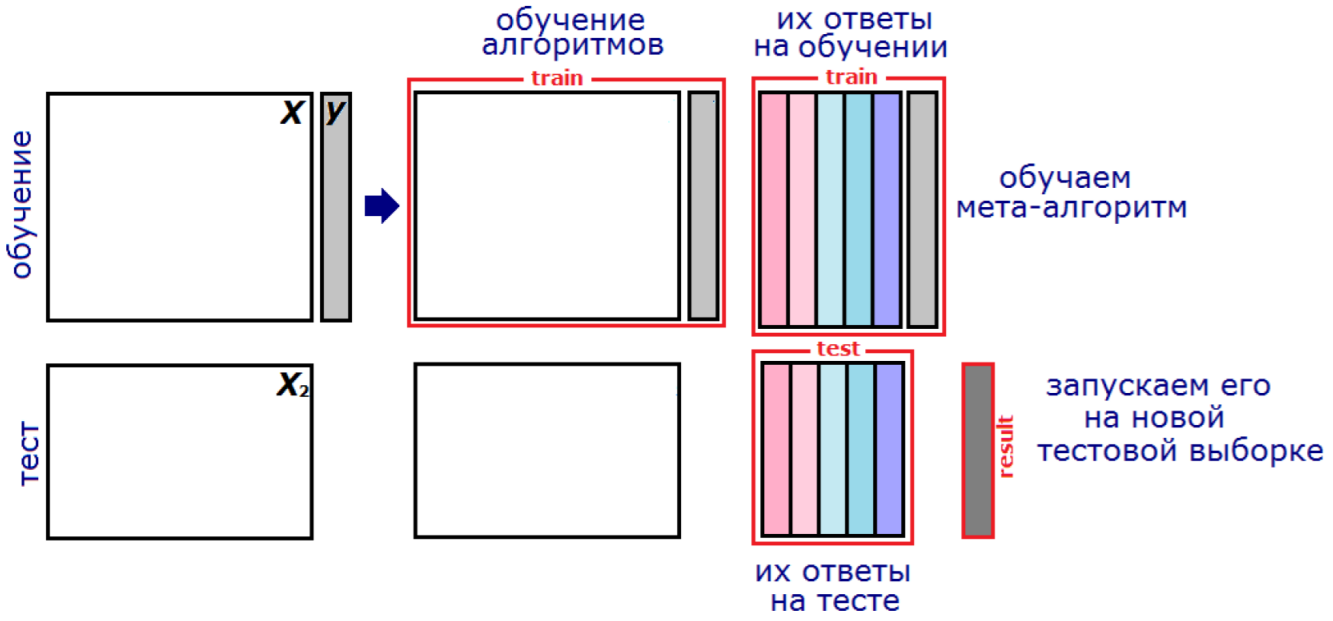
- Достоинства:
  - Большее изменение наборов чем в обычном bagging, а значит **большая вариация** и **меньшая корреляция** отклика моделей ведет к **несмещенному прогнозу с малой дисперсией**.
  - **Сложность** – можно оценивать по OOB, не нужно CV и HO набор.
  - Случайный лес **не склонен к переобучению** 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 диксретизация 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?

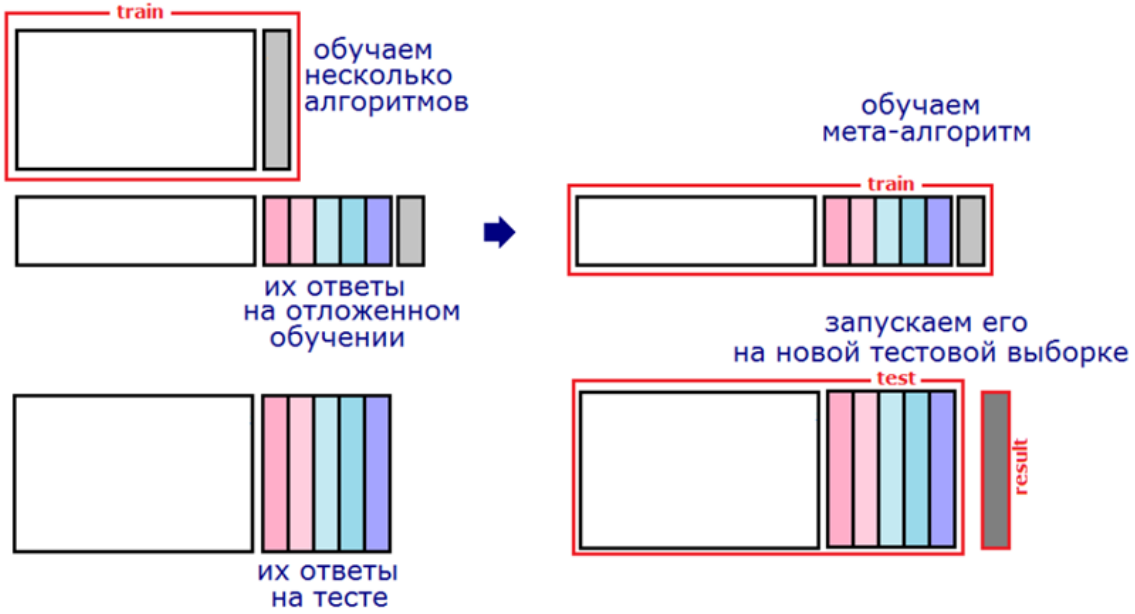
Варианты стекинга: простой

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



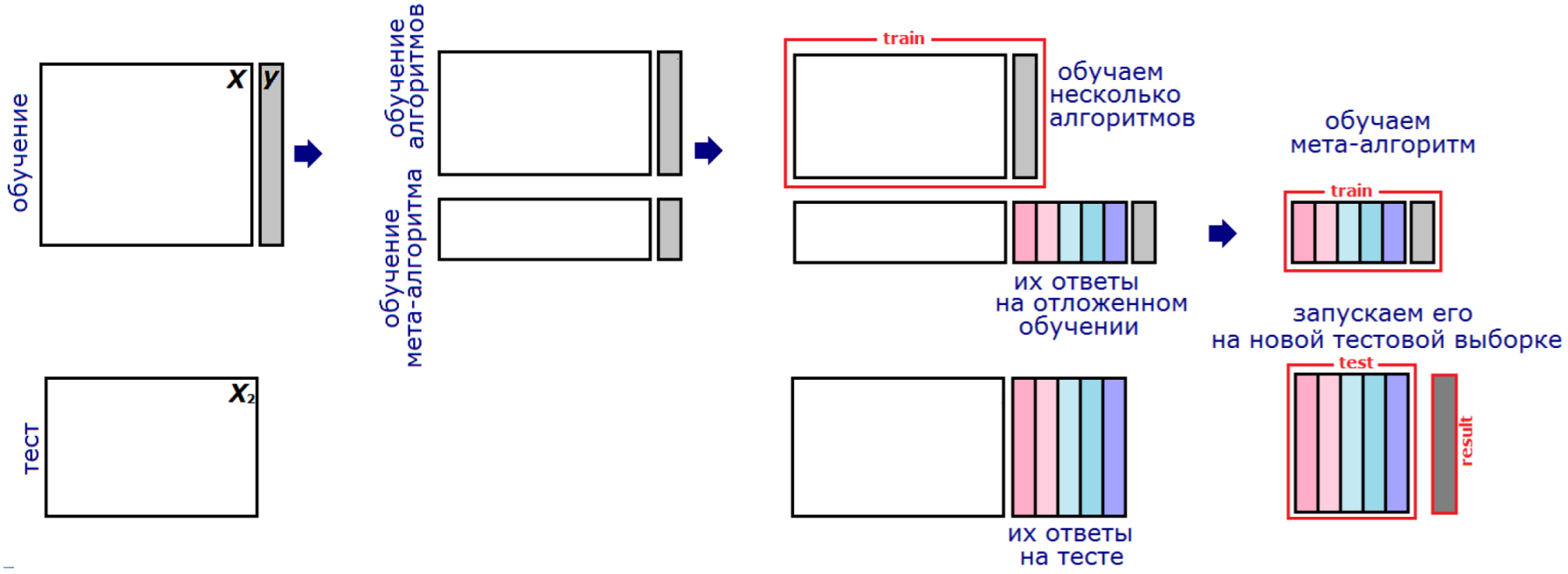
Варианты стекинга: объединение метапризнаков и обычных признаков

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



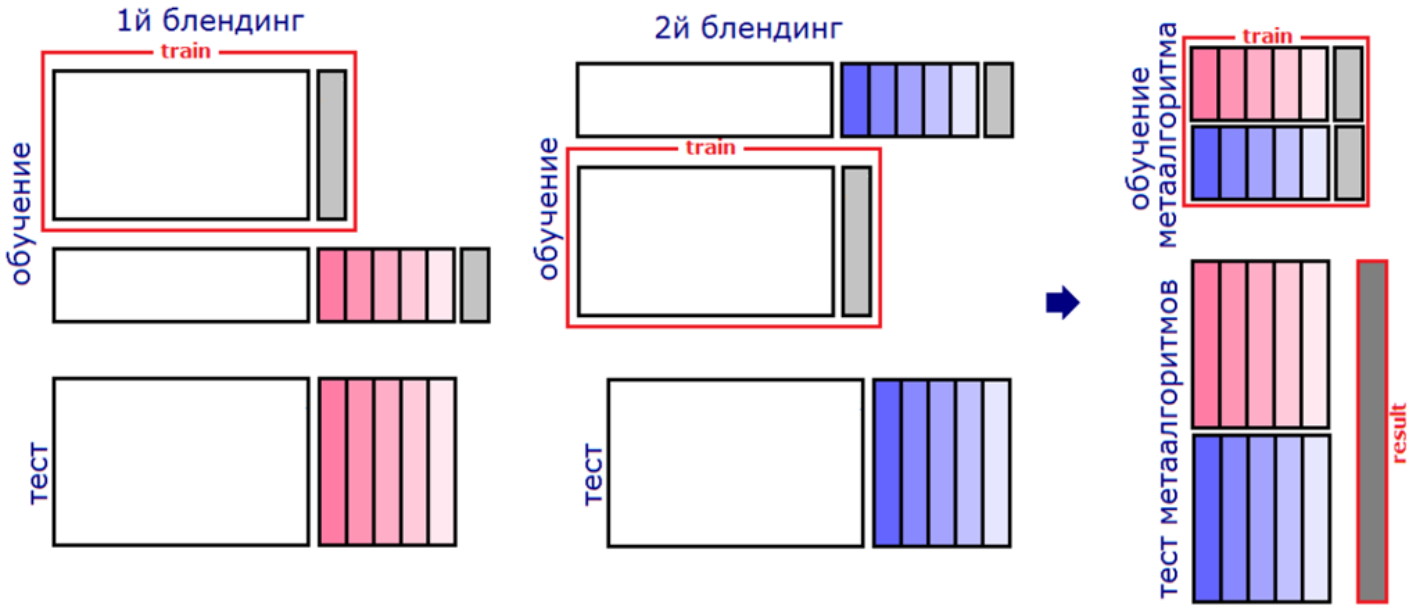
Варианты стекинга: блендинг

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



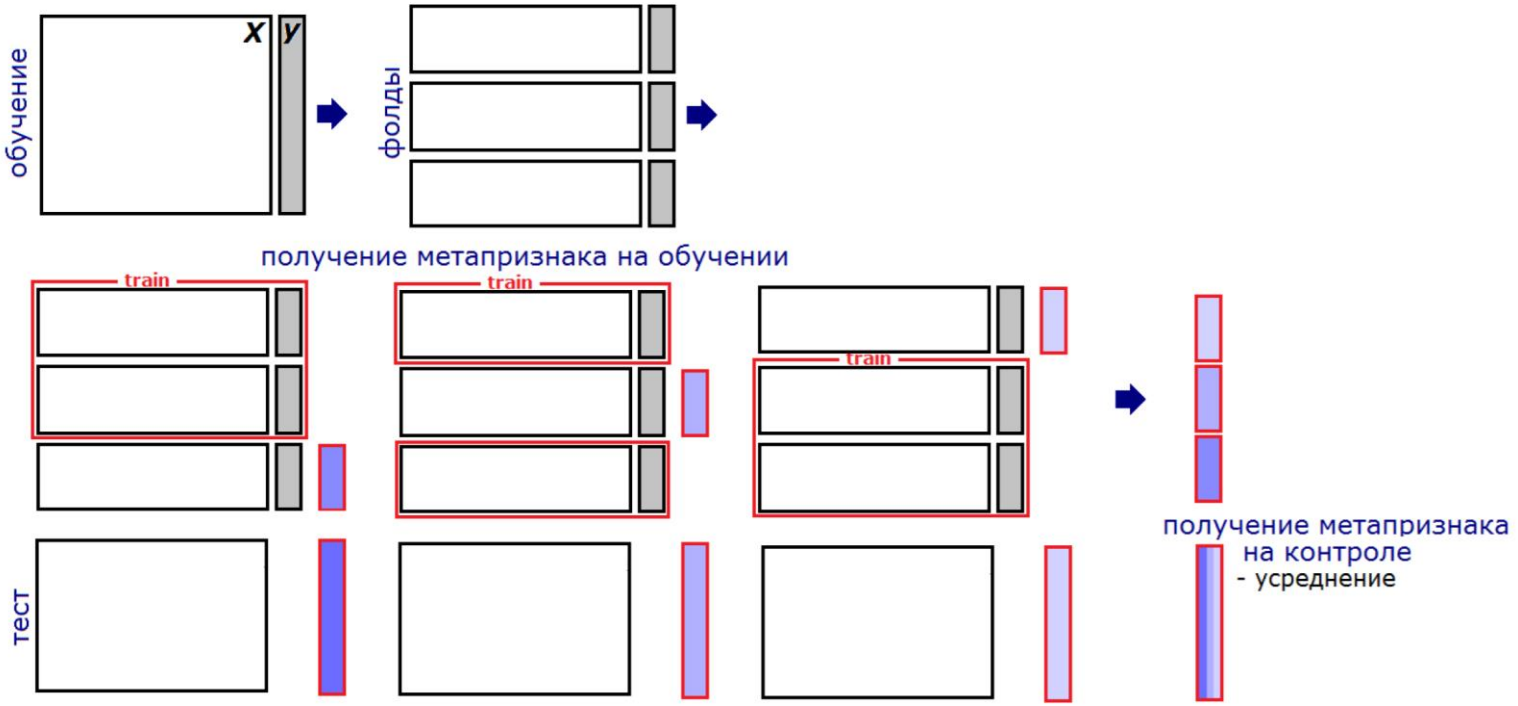
Варианты стекинга: объединение таблиц

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



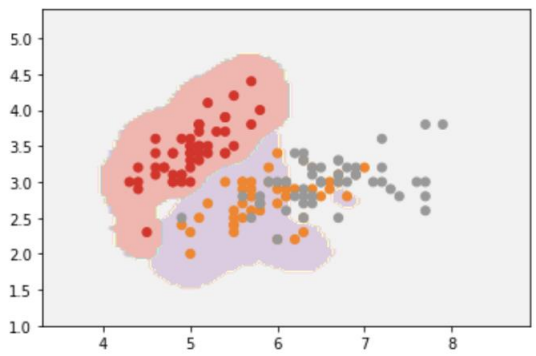
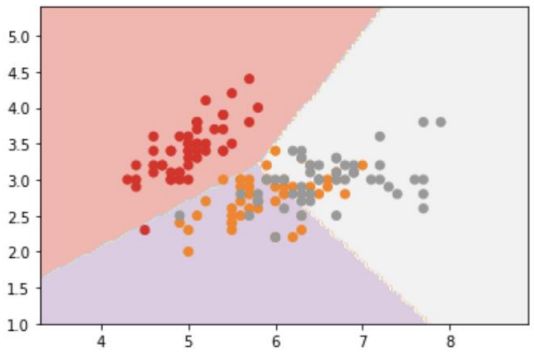
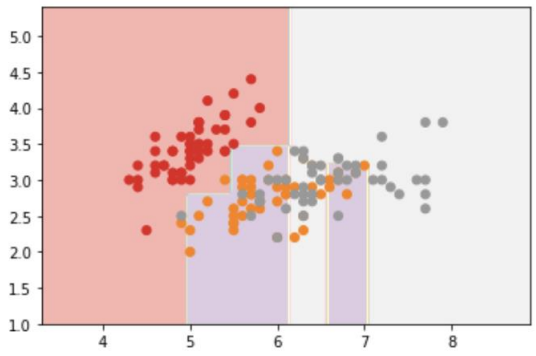
Варианты стекинга: объединение метапризнаков

- 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

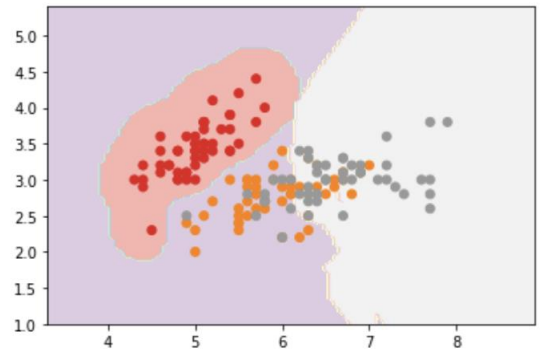


Пример стекинга

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

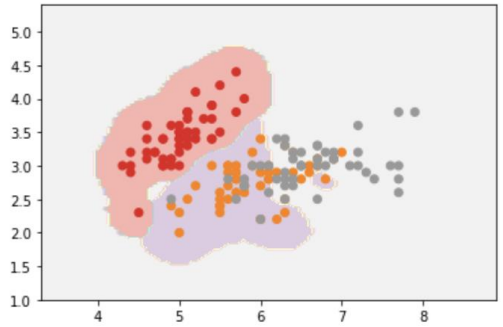
- Мета-алгоритм – линейная логистическая регрессия:

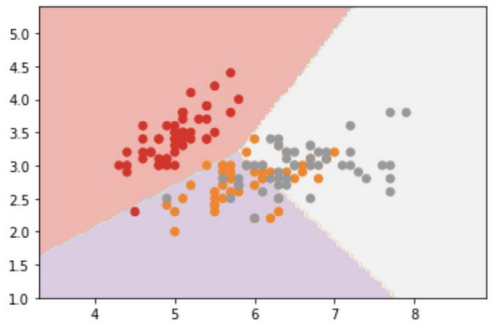


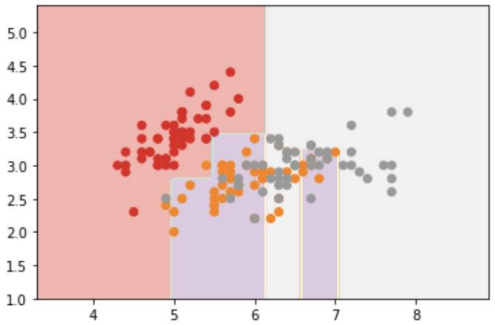
```
from sklearn.ensemble import StackingClassifier
estimators = [
    ('tree',  DecisionTreeClassifier(criterion="gini", max_depth=5,
            min_samples_split=5, min_samples_leaf=3, ccp_alpha=0.0)),
    ('svc', SVC(kernel="rbf", gamma=10)), ('logreg', LogisticRegression())
]
clf = StackingClassifier(estimators=estimators,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

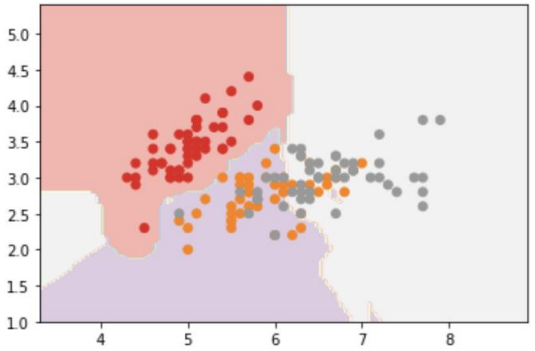
- Базовые алгоритмы:







- Мета-алгоритм – 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 коррелированны и появляются дополнительные мета-параметры настройки
- Достоинства ☺
  - Нет необходимости в глубоком тьюнинге 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)

«Самый древний» дискретный бустинг

- Усиление за счет фильтрации (Schapire 1989):
  - Случайно выбрать $n_1 < n$ примеров из выборки D без удаления
  - Получить D_1 и построить «слабую» модель $b_1(x)$
  - Выбрать $n_2 < n$ примеров из D с фильтрацией по $b_1(x)$ - применить к примерам $b_1(x)$, и с вероятностью ½ добавлять ошибки в D_2 и с вероятностью ½ не ошибки
  - На D_2, где половина – ошибки $b_1(x)$, построить «слабую» модель $b_2(x)$
  - Выбрать все D_3 из D где $b_1(x)$ и $b_2(x)$ дают разный прогноз и построить «слабую» модель $b_3(x)$
  - Финальный «усиленный» классификатор – голосование $b_1(x), b_2(x), b_3(x)$
- Почему работает?
  - в голосовании должно быть 2+ голосов
  - $b_1(x)$ и $b_2(x)$ - максимально не похожи
  - если $b_1(x) \neq b_2(x)$, то решает $b_3(x)$, который для этого учился
  - если $b_1(x) = b_2(x)$, то мнение $b_3(x)$ не интересно

Идея адаптивного бустинга

- Развивает идею «старого» бустинга с фильтрацией:
  - Вместо случайной выборки – перевзвешивание, т.е. оценка **«важности»** или **«сложности» примеров** (зависит от ошибки ансамбля на примере)
  - Построение «слабых» классификаторов (точность хотя бы >50%)
  - Результирующий «сильный» классификатор может иметь много слабых классификаторов и пользуется взвешенным голосованием, здесь **вес базового классификатора – его «сила»**
- Заимствует постановку задачи и подход у Forward Stagewise Additive Modeling (FSAM), решающего задачу прогнозирования:
  - Выборка $Z = \{(x_i, y_i)\}_{i=1}^{l} \in X \times Y$
  - Модель – взвешенный с весами α_i ансамбль *M* базовых моделей $a(x) = \sum_m \alpha_m \, b_m(x)$, который строится последовательно $a_m(x) = a_{m-1}(x) + \alpha_m b_m(x)$, где ищутся α_m, b_m при фиксированном a_{m-1}
  - Функция потерь $L: Y \times Y \to \mathbb{R}^+$

Forward Stagewise Additive Modeling

- FSAM
  - строит модель **последовательно**, добавляя к текущей модели a_m на шаге m «лучший» b_m с «лучшим» весом α_m :
  - Инициализация $a_0(x) = 0$
  - Цикл *M* итераций по m :

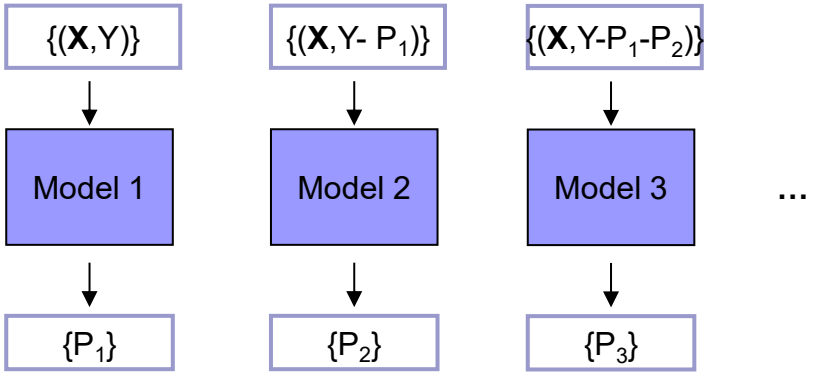
$$(b_m, \alpha_m) = \underset{b,\alpha}{\operatorname{argmin}} \sum_{i=1}^{l} L\big(y_i, a_{m-1}(x_i) + \boldsymbol{\alpha b}(\boldsymbol{x_i})\big)$$

$$a_m(x) = a_{m-1}(x) + \alpha_m b_m(x)$$

- Для некоторых L и b_i - простое аналитическое решение, примеры:
  - **AdaBoost:** L – экспоненциальная функция потерь, b_i - простое дерево (или даже «пень»), a_i вычисляются аналитически
  - **Additive Groves**: L – квадратичная, а b_i - дерево, $a_i = 1$, получаем ансамбль на остатках, для борьбы с переобучением используется bagging и специальный алгоритм, ищущий «оптимальное» сочетание сложности ансамбля и индивидуальных моделей

Additive (Tree) Groves

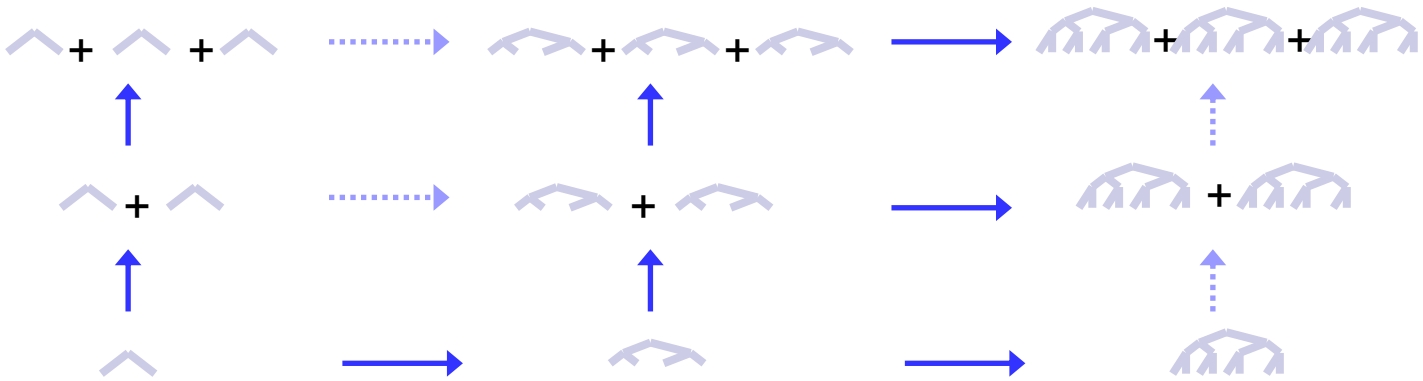
- Особенности метода:
  - Аддитивный ансамбль (без весов) $a(x) = \sum_m b_m(x)$, деревьев решений $b_m(x)$, построенных на $Z_m = \{(x_i, y_i - a_{m-1}(x_i))\}_i^l$, где вместо отклика используются остатки от предыдущего ансамбля.



  - Поскольку обучение на остатках, то на сложных b_m быстро переобучается при увеличении, исчерпываются остатки $\{(x_i, 0)\}_i^{n<l}$
  - Использует бутсрепинг подвыборку меньшего размера Z_m^* (как в random forest)
  - специальный алгоритм для контроля сложности на основе роста дерева «по слоям»

Контроль сложности Additive Groves

- Переборный алгоритм контроля сложности:
  - От простого (одно маленькое дерево) делаются шаги в сторону увеличения ансамбля без изменения сложности деревьев (увеличивается параметр M – размер ансамбля) и в сторону увеличения глубины деревьев без увеличения ансамбля (меняется параметр обрубания или число листьев D)
  - Если качество по OOB улучшается, то шаг принимается
  - Осуществляется поиск «оптимальной» точки гиперпарамтеров (N,D), без повторных вычислений (если пришли в точку разными путями)

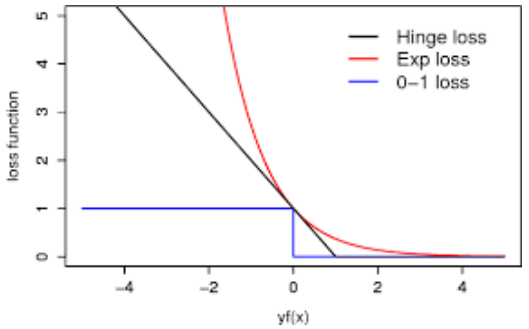


Адаптивный дискретный бустинг

- Основные допущения:
  - Выборка $Z = \{(x_i, y_i)\}_{i=1}^{l} \in X \times \{-1, +1\}$, где у каждого наблюдения (x_i, y_i) свой вес $w_i \geq 0$, причем $\sum_i w_i = 1$ (распределение)
  - Модель-ансамбль с взв. голосованием $a(x) = sign[\sum_m \alpha_m b_m(x)]$
  - Введем понятие взвешенной (по распределению w_i) ошибки классификации:

$$Err_w\big(a(x)\big) = \sum_{i=1}^{l} w_i \, I[y_i \neq a(x_i)]$$

  - Будем использовать экспоненциальную функцию потерь $L_{exp}\big(y, a(x)\big) = \exp(-y \sum_m \alpha_m b_m(x))$
- Идея алгоритма обучения (цикл):
  1. считаем ошибки для всех примеров
  2. перевзвешиваем все примеры
  3. обучаем базовый классификатор
  4. добавляем его в ансамбль с новым весом



AdaBoost (алгоритм)

- Инициализация:
  - $a_0(x) = 0, \forall i: w_i^{(1)} = 1/l$
- Итераций по m от 1 до M:
  - Строим **слабый классификатор** $b_m(x)$, минимизирующий и допускающий ошибку $Err_{w^{(m)}}(b_m) < 0.5$

$$b_m(x) = \underset{b}{\operatorname{argmin}} \sum_{i=1}^{l} w_i^{(m)} I[b(x_i) \neq y_i]$$

  - Добавляем b_m в ансамбль $a_m(x) = a_{m-1}(x) + \alpha_m b_m(x)$, где **силу классификатора** вычисляем как:

$$\alpha_m = \frac{1}{2} \log \left(\frac{\left(1 - Err_{w^{(m)}}(b_m)\right)}{Err_{w^{(m)}}(b_m)} \right)$$

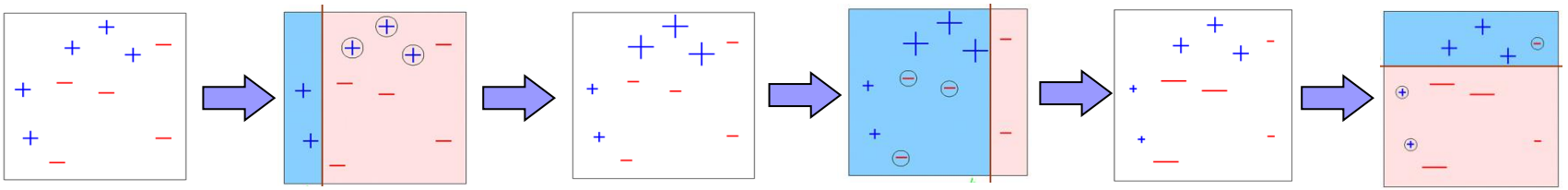
  - Пересчитываем и нормируем **веса важности** примеров:

$$\tilde{w}_i = w_i^{(m)} \exp\left(-y_i \alpha_m b_m(x_i)\right), \; w_i^{(m+1)} = \tilde{w}_i \; / \sum_j \tilde{w}_j$$

- Откуда взялись формулы для b_m, α_m и $w^{(m)}$?

Adaboost (демо)

- Простой пример Adaboost (длины 3) на «пнях» (деревьях глубины 1):

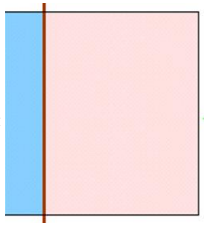
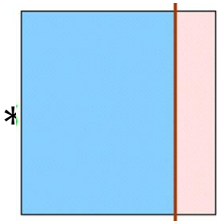
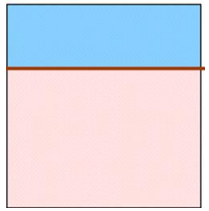


$Err_{w^{(1)}} = 0.3$
$\alpha_1 = 0.42$

$Err_{w^{(2)}} = 0.21$
$\alpha_2 = 0.65$

$Err_{w^{(3)}} = 0.14$
$\alpha_3 = 0.92$

- Финальный классификатор:

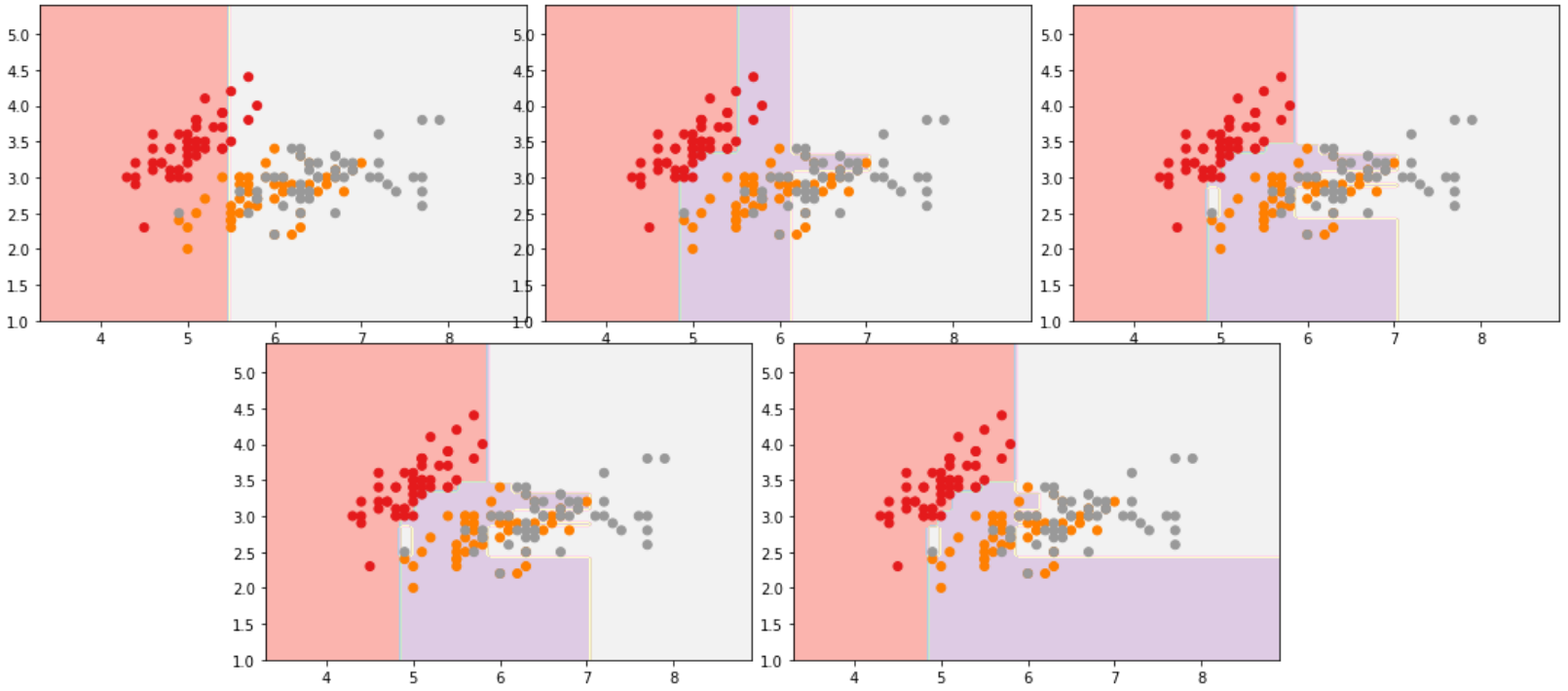
$$a(x) = \text{sign}\left(0.42 * \right.$$  $$+ 0.65 *$$  $$+ 0.92 *$$  $$\left. \right)$$

Пример

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

weak_learner = DecisionTreeClassifier(max_leaf_nodes=2)

for i in range(1,50,10):
    adaboost_clf = AdaBoostClassifier(estimator=weak_learner, n_estimators=i,
        algorithm="SAMME",random_state=42).fit(X, y)
    DecisionBoundaryDisplay.from_estimator(adaboost_clf, X, cmap="Pastel1")
    plt.scatter(*X.T, c=y, cmap="Set1")
```



Пересчет весов классификаторов

- Рассмотри на итерации m алгоритма эмпирический риск с функцией потерь на основе ошибки классификации, он ограничен сверху риском с экспоненциальной функцией потерь:

$$Q_{perc}^{(m)} = \sum_{i=1}^{l} I[y_i \neq a_m(x_i)] \leq Q_{exp}^{(m)} = \sum_{i=1}^{l} \exp[-y_i a_m(x_i)] =$$

$$= \sum_{i=1}^{l} \exp\left[-y_i\left(\sum_{j=1}^{m} \alpha_j b_j(x_i)\right)\right] = \sum_{i=1}^{l} \mathbf{exp}\left[-\boldsymbol{y_i}\left(\sum_{\boldsymbol{j=1}}^{\boldsymbol{m-1}} \boldsymbol{\alpha_j b_j(x_i)}\right)\right] \exp[-y_i \alpha_m b_m(x_i)] \sim$$

$\sim \boldsymbol{w_i^{(m)}}$ – не зависит от α_m, b_m

$$\sim \sum_{i=1}^{l} w_i^{(m)} \exp[-y_i \alpha_m b_m(x_i)] = e^{-\alpha_m} \sum_{\boldsymbol{i:y_i=b_m(x_i)}} \boldsymbol{w_i^{(m)}} + e^{\alpha_m} \sum_{\boldsymbol{i:y_i \neq b_m(x_i)}} w_i^{(m)} =$$

= …

$$e^{-\alpha b} = e^{-\alpha} I[b = 1] + e^{\alpha} I[b = -1]$$

доля верных прогнозов с весами $(1 - Err_{w^{(m)}})$

доля ошибок с весами $Err_{w^{(m)}}$

Пересчет весов классификаторов

- Продолжение:

$$\dots = \left(1 - Err_{w^{(m)}}\right)e^{-\alpha_m} + \left(Err_{w^{(m)}}\right)e^{\alpha_m} = Q_{Err_{w^{(m)}}}(\alpha_m) \to min$$

- Найдем $\min\limits_{\alpha_m}[Q_{Err_{w^{(m)}}}(\alpha_m)]$:

$$\frac{\partial Q_{Err_{w^{(m)}}}}{\partial \alpha_m} = 0 \Rightarrow \alpha_m = \frac{1}{2}\log\left(\frac{1 - Err_{w^{(m)}}}{Err_{w^{(m)}}}\right)$$

- Подставляем α_m в $Q_{Err_{w^{(m)}}}$ и получаем, что верхняя оценка эмпирического риска экспоненциально уменьшается с уменьшением взвешенной ошибки $Err_{w^{(m)}}$:

$$\left(1 - Err_{w^{(m)}}\right)e^{-\frac{1}{2}\log\left(\frac{1-Err_{w^{(m)}}}{Err_{w^{(m)}}}\right)} + \left(Err_{w^{(m)}}\right)e^{\frac{1}{2}\log\left(\frac{1-Err_{w^{(m)}}}{Err_{w^{(m)}}}\right)} =$$

$$= 2\sqrt{Err_{w^{(m)}}\left(1 - Err_{w^{(m)}}\right)} \le \exp\left(-2\left(0.5 - Err_{w^{(m)}}\right)^2\right)$$

Обучение слабых классификаторов и пересчет весов

- При фиксированном $\alpha_m > 0$ выразим наилучший слабый b_m :

$$e^{-\alpha_m} \sum_{i:y_i=b(x_i)} w_i^{(m)} + e^{\alpha_m} \sum_{i:y_i \neq b(x_i)} w_i^{(m)} =$$

$$= [(e^{\alpha_m} - e^{-\alpha_m}) \sum_i w_i^{(m)} I[y_i \neq b(x_i)] + e^{-\alpha_m} \sum_i w_i^{(m)}] \to \min_{b_m}$$

$$b_m(x) = \operatorname{argmin}_b \sum_{i=1}^{l} w_i^{(m)} I[b(x_i) \neq y_i]$$

Не зависит от b_m и >0 при $\alpha_m > 0$

- Поскольку $a_m(x) = a_{m-1}(x) + \alpha_m b_m(x)$ потери выразим рекурсивно:

$$\sum_{i=1}^{l} \exp[-y_i a_m(x_i)] = \sum_{i=1}^{l} \exp\left[-y_i \left(\sum_{j=1}^{m} \alpha_j b_j(x_i)\right)\right] =$$

$$= \sum_{i=1}^{l} w_i^{(m)} \exp[-y_i \alpha_m b_m(x_i)] = \sum_{i=1}^{l} w_i^{(m-1)} \exp[-y_i (\alpha_{m-1} b_{m-1}(x_i) + \alpha_m b_m(x_i))] =$$

$$= \sum_{i=1}^{l} w_i^{(1)} \exp[-y_i \alpha_1 b_1(x_i)] \exp[-y_i \alpha_2 b_2(x_i)] \dots \exp[-y_i \alpha_m b_m(x_i)] \Rightarrow$$

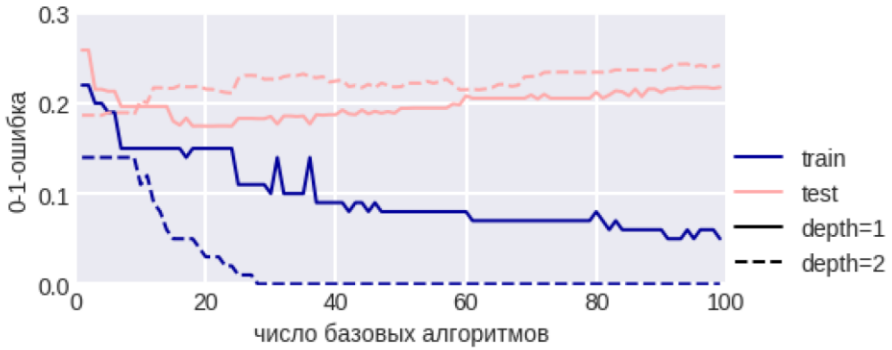
$$\Rightarrow w_i^{(m+1)} = w_i^{(m)} \exp(-y_i \alpha_m b_m(x_i))$$

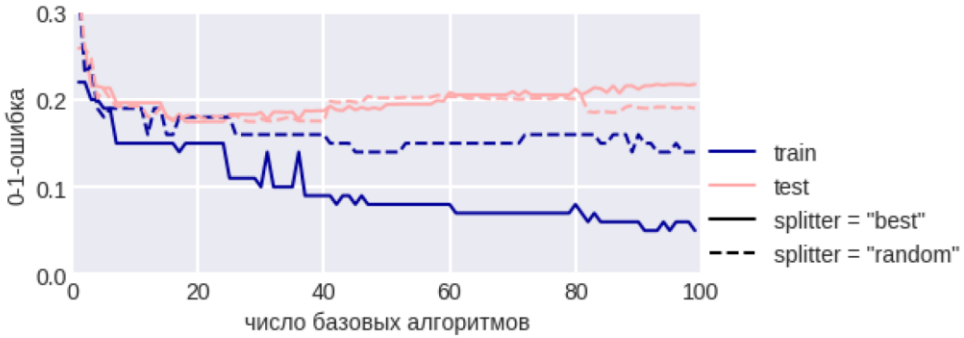
Теоретические результаты

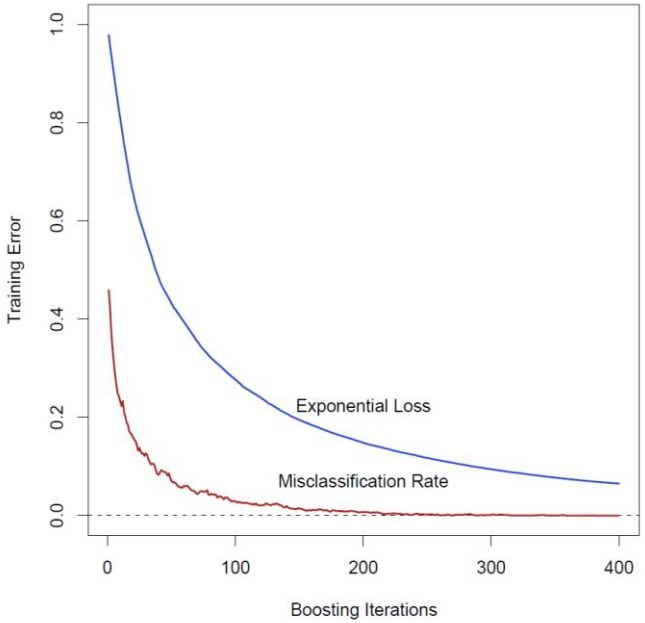
- Условия применимости:
  - «достаточно богатое» семейство слабых классификаторов
  - качество слабого классификатора лучше случайного прогноза
  - «не слишком высокая» зашумленность данных
- Формально доказан ряд важных свойств алгоритма Adaboost:
  - приводит к минимизации эмпирического риска с экспоненциальной функцией потерь при использовании предложенной процедуры пошагового усложнения ансамбля и полученных формул пересчета нормированных весов примеров и весов слабых классификаторов
  - сходится за конечное число шагов
  - с увеличением числа итераций увеличивает зазор между классами и, значит, уменьшает ошибку классификации – полезно строить график зазора от шага итерации и распределение отступов на каждом шаге

Сложность ансамбля Adaboost и борьба с переобучением

- Сложность ансамбля:
  - Определяется размером ансамбля (чем больше тем сложнее)
  - Определяется сложностью базового классификатора
  - За счет экспоненциальной функции потерь может улучшаться на проверочной выборке даже когда ошибка на тренировочной ушла в 0

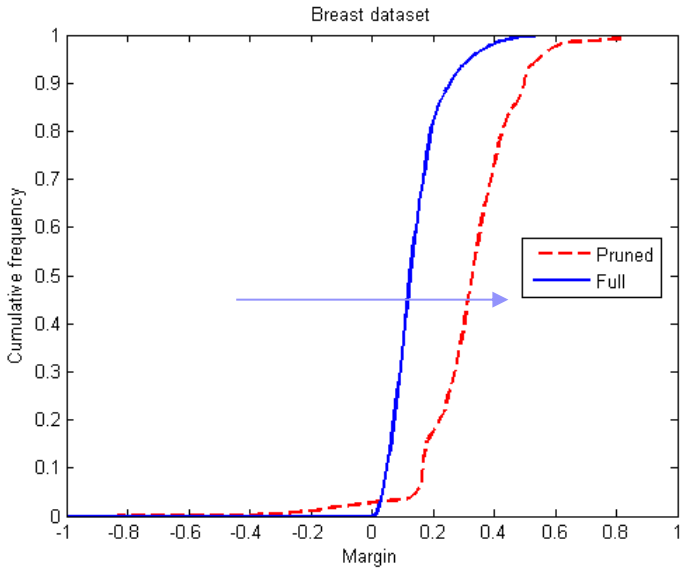


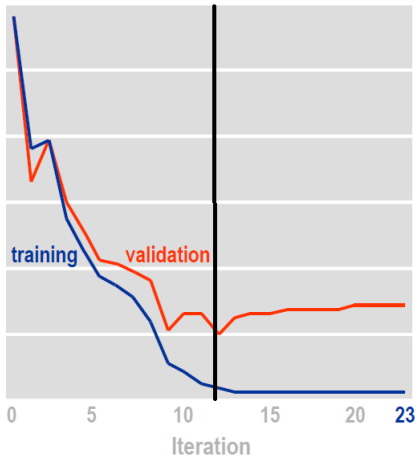




Борьба с переобучением

- Методы борьбы с переобучением:
  - Управлять размером ансамбля
  - Ограничивать сложность или упрощать (например pruning) слабые классификаторы
  - Можно пытаться контролировать (менять вес) или удалять выбросы «на лету», анализируя веса или отступы и их распределение (оно с каждой итерацией должно «смещаться вправо», «раздвигая» классы)
  - В некоторых реализациях можно контролировать learning rate или использовать регуляризацию
  - Т.к. бустинг ищет зависимости от простых к сложным, то можно использовать early stopping

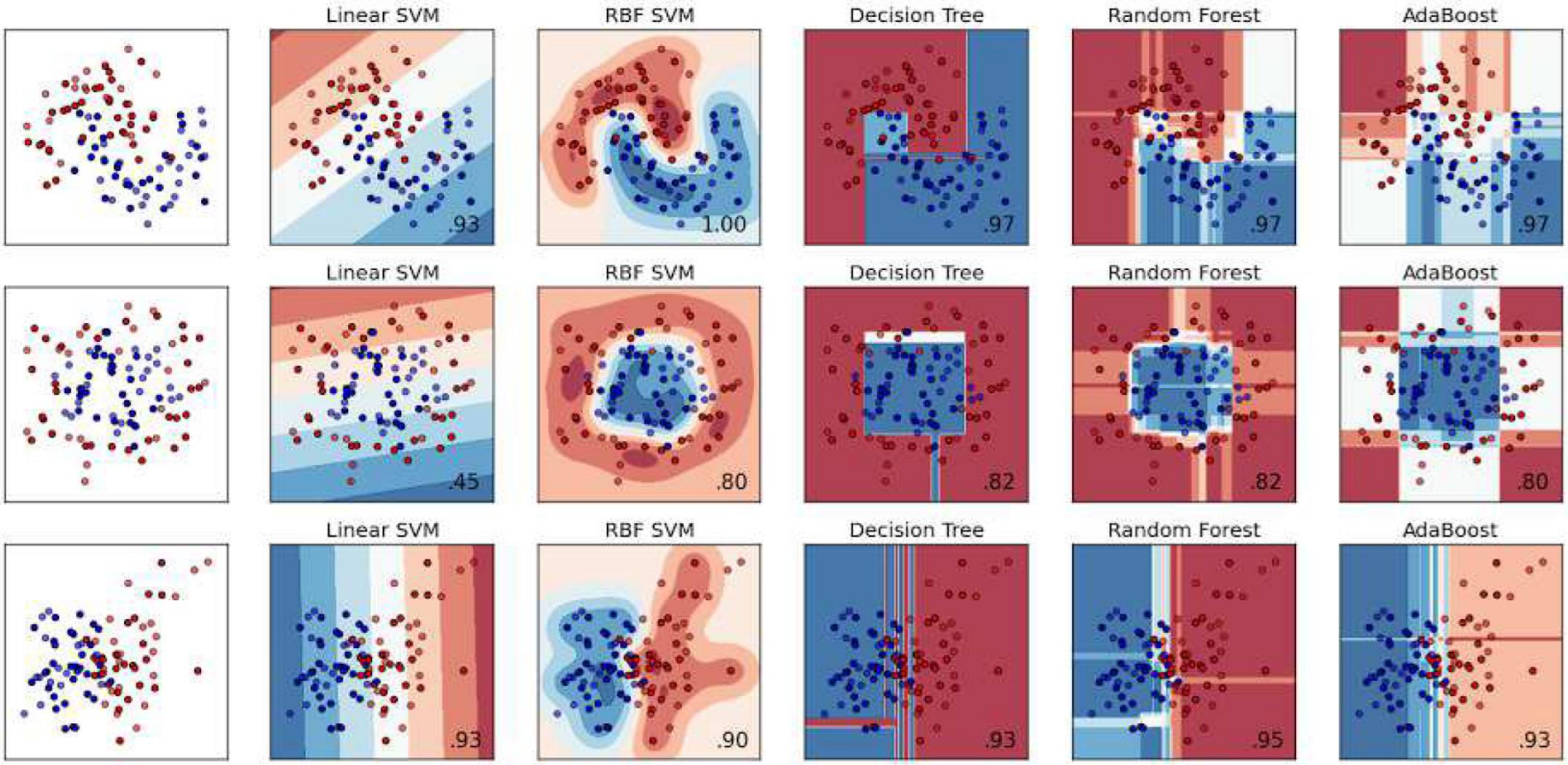




Недостатки Adaboost

- Чувствительность к выбросам:
  - из-за экспоненциальной функции потерь может быстро переобучаться при наличии шума
  - можно пытаться контролировать выбросы «на лету»
  - нужно использовать простые слабые модели, в результате не позволяет строить маленькие ансамбли сложных моделей, только большие ансамбли простых моделей
- Проблемы применения:
  - Вычислительно сложная модель (если много базовых)
  - Не интерпретируемая модель, даже если базовые модели интерпретируется
- Проблемы обучения:
  - Склонен к переобучению в зашумленных задачах
  - Требуются большие выборки
  - Плохо распаралелливается (только «внутри» обучения базовой модели)

Сравнение Adaboost



Бустинг с перевыборкой (пример)

- Arc-x4 (Breiman, 1996) как в AdaBoost последовательно строит модель $a_m(x) = a_{m-1}(x) + \alpha_m b_m(x)$, но:
  - В классической версии **нет весов классификаторов** – голосующий комитет или усреднение (хотя есть версии с эвристическим или FSAM пересчетом весов)
  - слабый классификатор **не учитывает веса примеров**, а строится на случайной выборке, где у наблюдения вероятность попасть в нее пропорциональная весу
  - Не требуется взвешенная функция ошибки или функция потерь – **может «бустить» любые базовые модели**
  - Вес имеет семантику как и в AdaBoost – **«сложность» примера**, и пересчитывается в зависимости от числа ошибок на нем слабых классификаторов ансамбля, классическая формула для классификации (эвристика):

$$w_i^{(m)} = \frac{(1 + \sum_{s=1}^{m} I[b_s(x_i) \neq y_i])^4}{\sum_{j=1}^{l} w_j^{(m)}}$$

Алгоритм Arc-X4

- Инициализация: $a_0(x) = 0, \forall i: w_i^{(1)} = 1/l$
- Цикл алгоритма (M итераций):
  - Формируется из набора Z **случайная выборка** $Z^{(m)}$ размера n, с вероятностью выбора примера пропорциональной распределению весов примеров ($n < l$ – параметр): $P(x_i \in Z^{(m)}) \propto w_i^{(m)}$
  - Обучается **слабая модель** $b_m(x)$ со своей функцией потерь
  - Добавляется b_m в ансамбль $a_m(x)$, где **силу классификатора** не меняют (хотя есть версии с весами и регуляризацией):

    для классификации: $a(x) = \text{argmax}_i[b_i(x)]$

    для регрессии: $a(x) = \frac{1}{M}\sum b_i(x)$
  - Пересчитываются и нормируются **веса важности** примеров в зависимости от числа ошибшихся слабых классификаторов:

$$w_i^{(m)} = \frac{(1 + \sum_{s=1}^{m} L[b_s(x_i), y_i])^{\mathbf{4}}}{\sum_{j=1}^{l} w_j^{(m)}}$$

Arc-x4

- Достоинства:
  - Простота и эффективность (сопоставим с AdaBoost по качеству)
  - Не накладывает требования на тип моделей и функции потерь
  - Можно использовать свои эвристики весов
  - Легко адаптируется к разным задачам прогнозирования
- Основной недостаток – нет теоретического обоснования
- Демо-пример:

$$P(x_i \in Z^{(m)}) \propto (1 + Err_m)^{\mathbf{4}}$$

$$Err_m = \sum_{s=1}^{m} I[b_s(x_i) \neq y_i]$$

	m=1		m=2		m=3		m=4	…
x	w	Err	w	Err	w	Err	w	
1	1	1	1.5	1	.5	2	.97	
2	1	0	.75	0	.25	0	.06	
3	1	1	1.5	2	4.25	3	4.69	
4	1	0	.75	1	.5	1	.11	
5	1	0	.75	0	.25	0	.06	
6	1	0	.75	0	.25	1	.11