Практические задачи анализа данных

Лекция 4. Ансамбли

Платонов Е.Н.

Московский авиационный институт
«МАИ»

22 сентября 2021 г.

Комбинация алгоритмов

n — число признаков

m — длина выборки

$\mathcal{X}$ — множество объектов

$\mathcal{Y}$ — множество откликов

$\{\mathcal{X}, \mathcal{Y}\} = \{x_i, y_i\}_{i=1}^{m}$ — выборка

$x^i = \{x_1^i, \ldots, x_m^i\}^T$ — значение i-го признака на $\mathcal{X}$

$x_j = \{x_j^1, \ldots, x_j^m\}^T$ — вектор признаков j-го объекта

k — количество классов

t, $t = 1, \ldots, k$ — метки классов

Комбинация алгоритмов

Рассмотрим задачу классификации на k классов

$$\mathcal{Y} = \{1,2,\ldots,k\}$$

Пусть имеется M базовых алгоритмов (base learners) $b_1, b_2, \ldots, b_M$

$$b_s : \mathcal{X} \to \mathcal{Y},\ s = 1,\ldots,M.$$

Построим новый классификатор: простое голосование:

$$f(x) = \arg\max_{t=1,\ldots,k} \sum_{s=1}^{M} \mathrm{I}(b_s(x) = t),$$

взвешенное голосование:

$$f(x) = \arg\max_{t=1,\ldots,k} \sum_{s=1}^{M} \alpha_s \,\mathrm{I}(b_s(x) = t), \quad \alpha_s \geq 0, \quad \sum_{s=1}^{M} \alpha_s = 1$$

или

$$f(x) = \arg\max_{t=1,\ldots,k} \sum_{s=1}^{M} \alpha_s(x) \,\mathrm{I}(b_s(x) = t), \quad \alpha_s(x) \geq 0, \quad \sum_{s=1}^{M} \alpha_s(x) = 1$$

В задаче регрессии

простое голосование:

$$f(x) = \frac{1}{M}\sum_{s=1}^{M} b_s(x),$$

взвешенное голосование:

$$f(x) = \frac{1}{M}\sum_{s=1}^{M} \alpha_s(x) b_s(x), \quad \alpha_s \geq 0, \quad \sum_{s=1}^{M} \alpha_s = 1$$

Замечание. Если мы строим регрессию по критерию MAE, то вместо усреднения нужно брать медиану $\mathrm{median}\{b_1, \ldots, b_M\}$

Комитеты (голосование, Voting Ensembles)

голосование по большинству (Majority vote)

$$f(x) = mode(b_1(x), \ldots, b_M(x))$$

комитеты единогласия в бинарной задаче классификации

$$f(x) = \min(b_1(x), \ldots, b_M(x))$$

обнаружение аномалий: «тревога при малейшем подозрении»

$$f(x) = \max(b_1(x), \ldots, b_M(x))$$

Реализация в scikit-learn

`sklearn.ensemble.VotingClassifier`

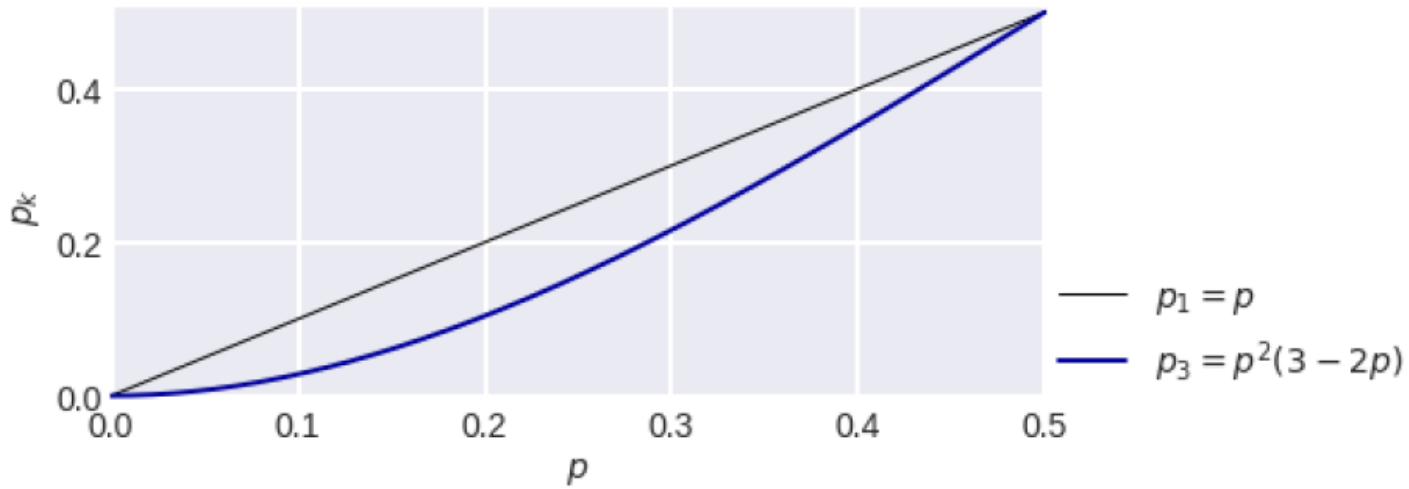
estimators	Список базовых алгоритмов
voting = «hard»	Голосование по меткам или усреднение вероятностей
weights = None	Веса
n_jobs = None	«number of jobs»
flatten_transform=True	Формат ответа (для soft-ансамбля)

есть ещё
`ensemble.VotingRegressor`

Ошибка комитета большинства

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

Тогда для комитета большинства ошибка равна $p_3 = p^2(3 - 2p)$



При малых p ошибка комитета очень мала!

При $p = 0.2$ — почти в два раза меньше

Комбинация алгоритмов

В общем случае для ошибки комитета большинства верно неравенство Хёфдинга (Hoeffding):

$$\sum_{t=0}^{[n/2]} C_n^t (1-p)^t p^{n-t} \leq \exp\left\{-\frac{1}{2} n(2p-1)^2\right\}$$

Ошибка экспоненциально снижается с увеличением числа базовых алгоритмов, но это верно только при выполнении условия о независимости базовых алгоритмов

На практике выполнения этого условия достичь невозможно

Проблема разнообразия базовых алгоритмов

Базовые алгоритмы

- решают одну и ту же задачу
- настраиваются на один целевой вектор
- часто выбираются из одной и той же модели

Пусть алгоритмы ошибаются, но по-разному:
ошибки одних компенсируются правильными ответами других

Способы повышения разнообразия базовых алгоритмов:

- обучение по различным (случайным) подвыборкам
- обучение по различным (случайным) наборам признаков
- обучение из разных параметрических моделей
- обучение с использованием кросс-валидации
- варьирование целевого вектора $f(y)$
- обучение по зашумленным данным

Обоснование применения ансамблей

Статистическое (Statistical)	— ошибка может быть меньше
Вычислительное (Computational)	— обучение = оптимизация функции, а ансамбль «распараллеливает» процесс
Функциональное (Representational)	— можно представить функции, которые было нельзя с помощью базовых алгоритмов

Комбинация алгоритмов

- **комитеты (голосование) / усреднение**

  в том числе, усреднение по Коши, калибровка + усреднение сюда же

  бэггинг (bagging) – усреднение моделей, обученных на бутстреп-подвыборках + обобщения (RF)
- **бустинг (boosting)**

  построение суммы нескольких алгоритмов, каждое следующее слагаемое строится с учётом ошибок предыдущих
- **стекинг (stacking)**

  построение метапризнаков — ответы алгоритмов на объектах выборки, обучение на них мета-алгоритма
- **перекодировки ответа**

  кодирование целевого вектора ECOC (error-correcting output coding)
- **«ручные методы»**

  Эвристические способы комбинирования ответов базовых алгоритмов
- **однородные ансамбли**

  рекурсия в формуле мета-алгоритм(базовые) + общая схема оптимизации (пример: нейросети)

Бэггинг (Bagging)

Bootstrap aggregation [Breiman, 1994]

Базовый алгоритм b_s обучается на bootstrap–выборке (повторной выборке)

Финальный алгоритм — функция голосования

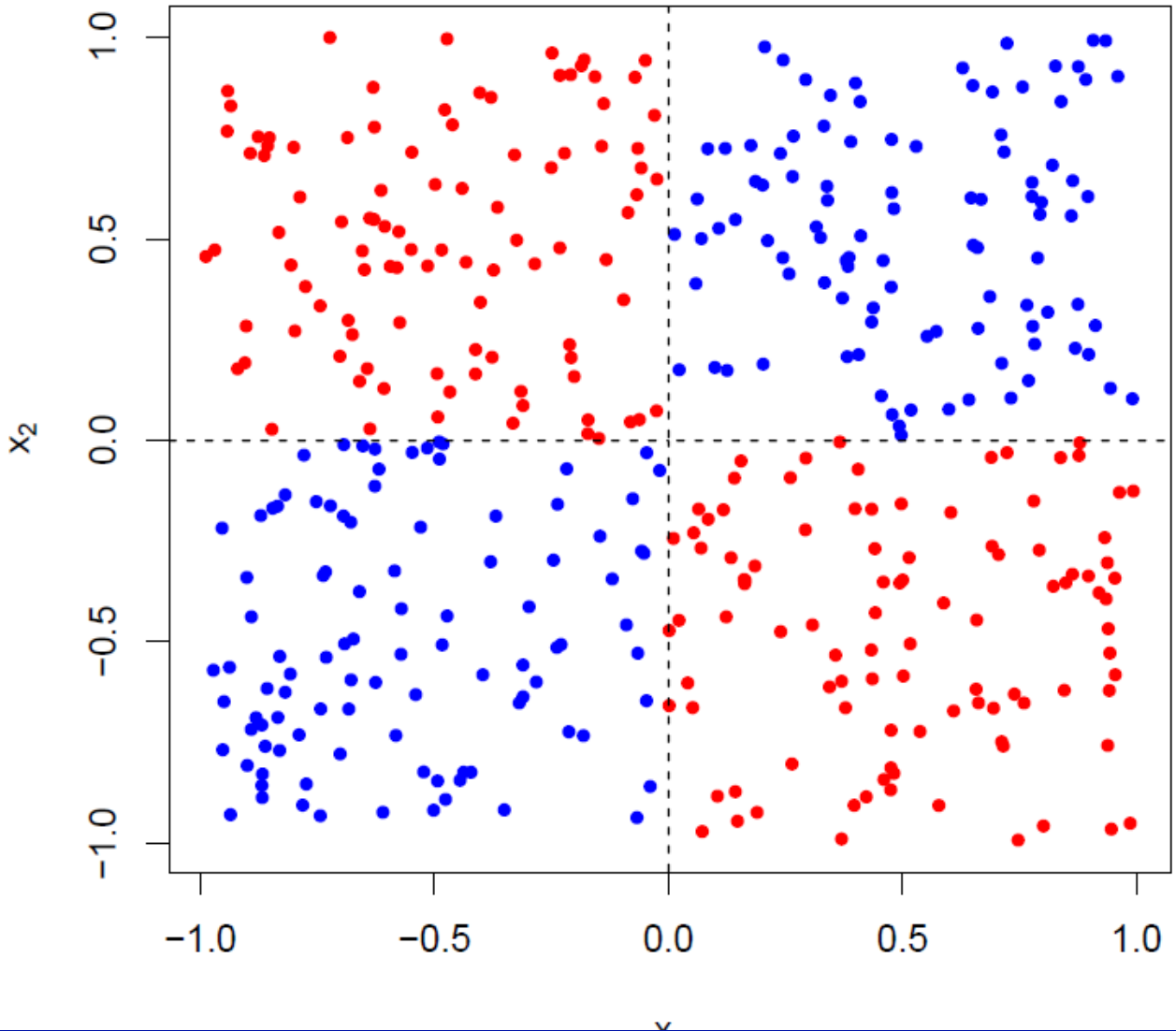
для задачи классификации:

$$f(x) = \arg \max_{t=1,\ldots,k} \sum_{s=1}^{M} \mathrm{I}(b_s(x) = t),$$

для задачи регрессии:

$$f(x) = \frac{1}{M} \sum_{s=1}^{M} b_s(x),$$

Рассмотрим задачу Xor

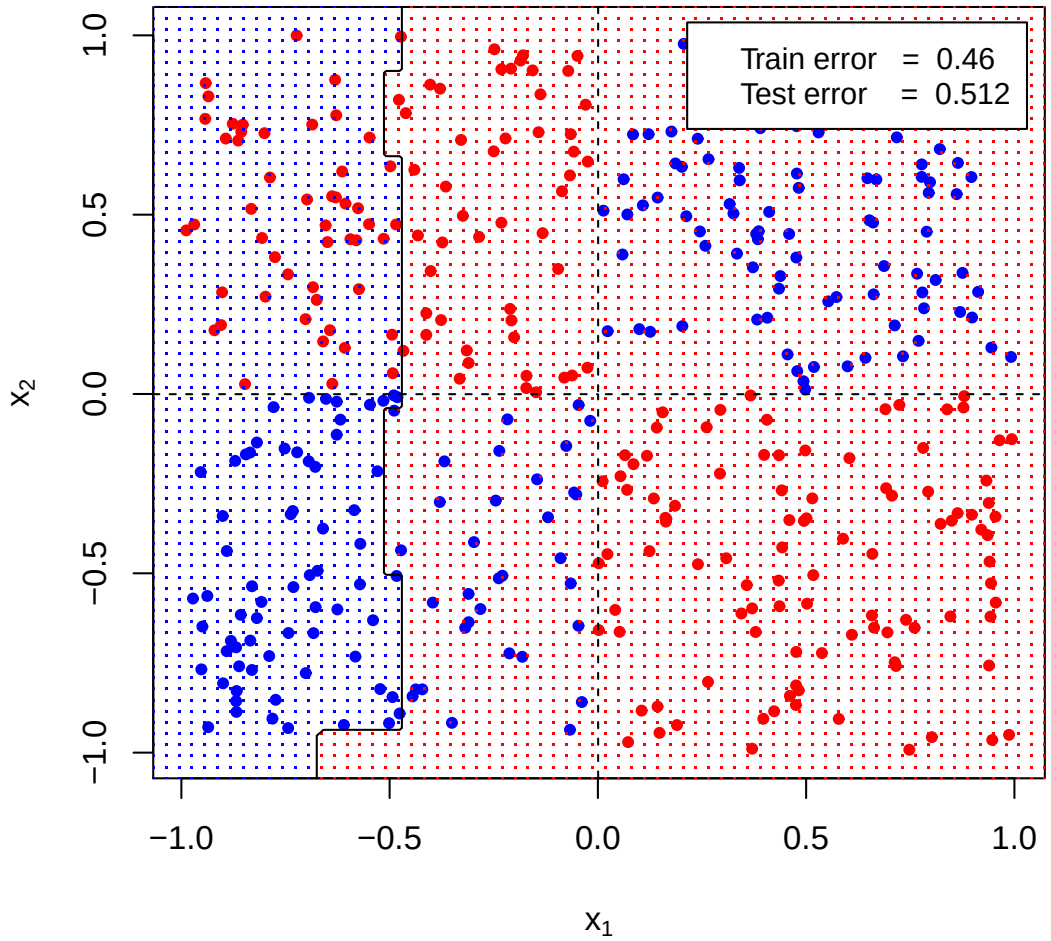


Бэггинг

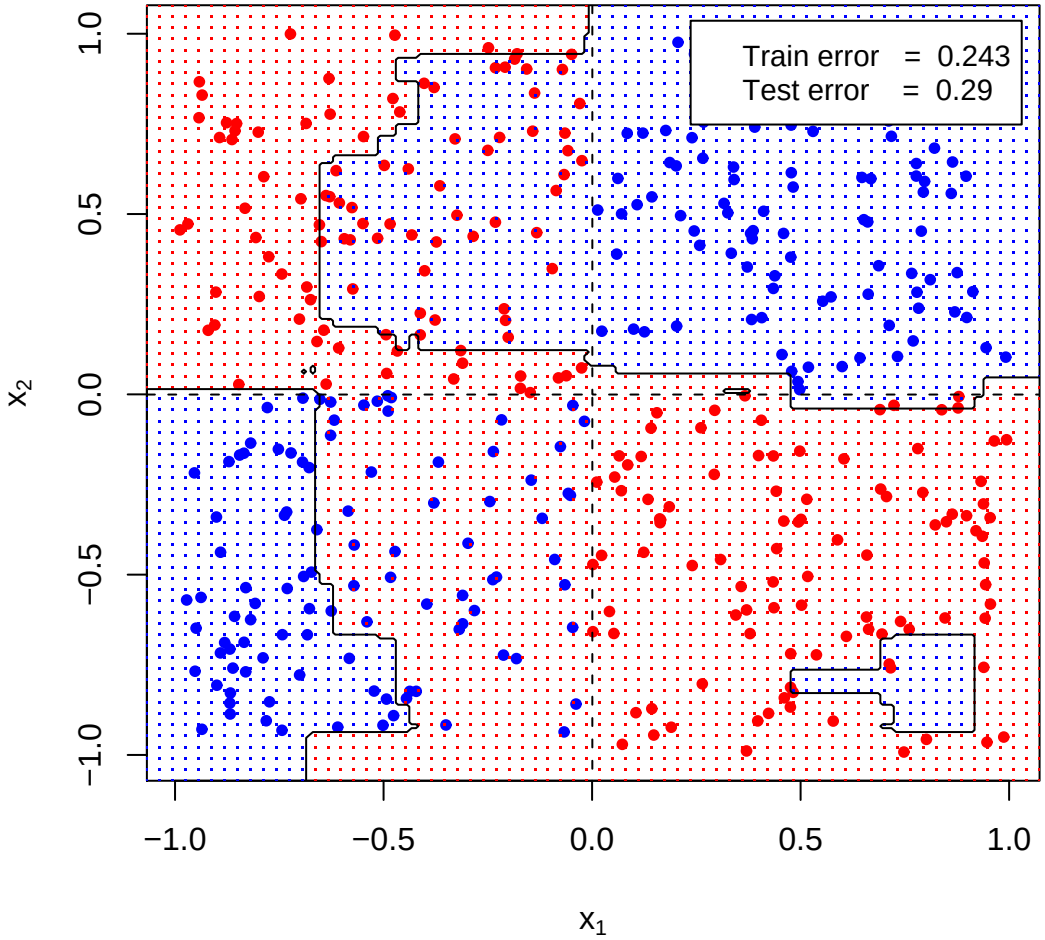
xor: низких деревьев (высоты 1 или 2) должно быть много.

Еще при 1000 деревьях картина не удовлетворительная.

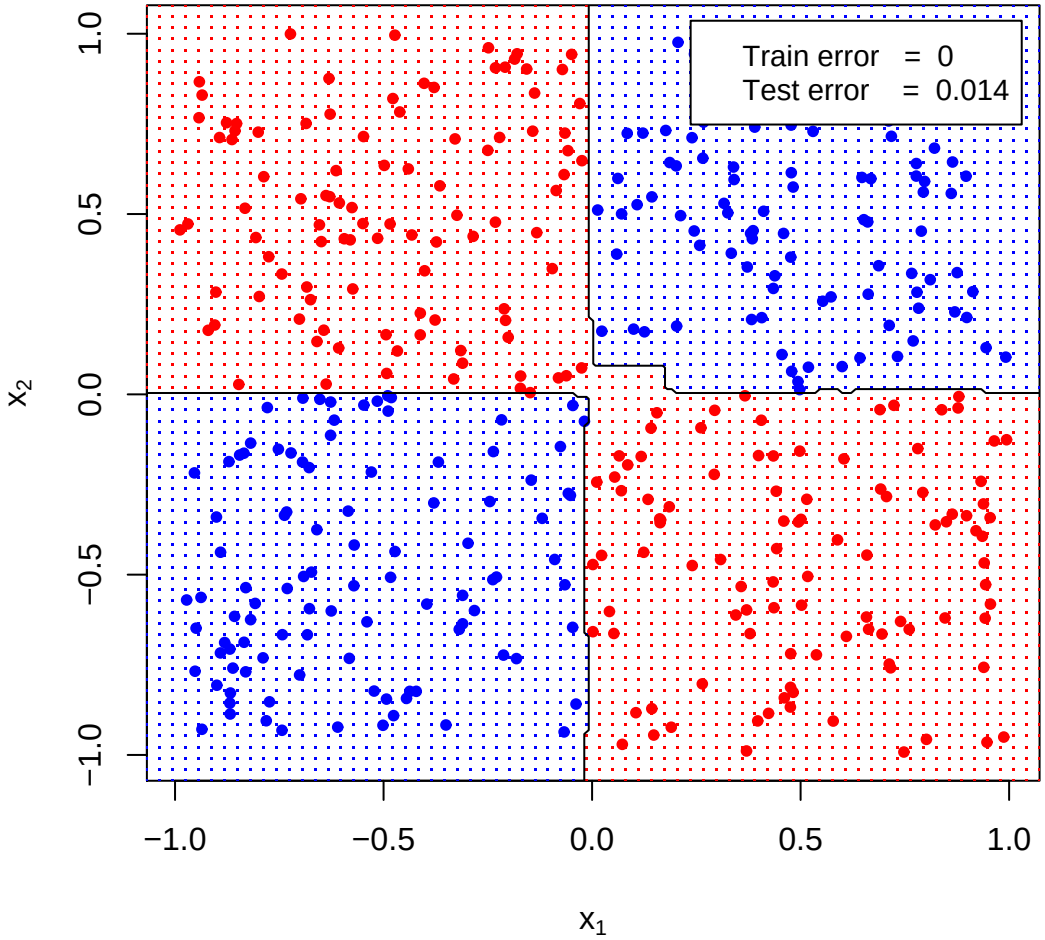
На рис.: деревья решений высоты 1 (100 деревьев).



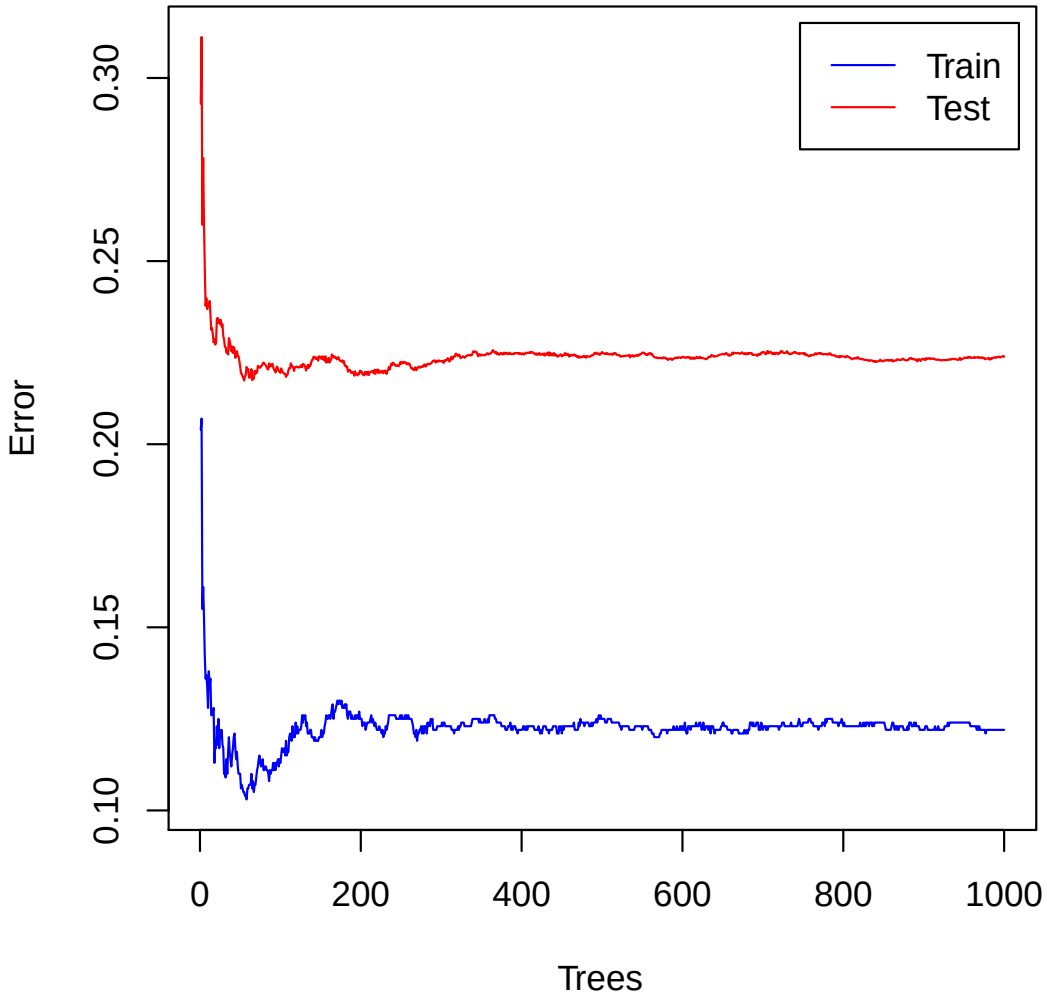
Bagging — деревья решений высоты 2 (100 деревьев)



Bagging — деревья решений высоты 3 (100 деревьев)



Бэггинг



Переобучается ли баггинг?

Бэггинг

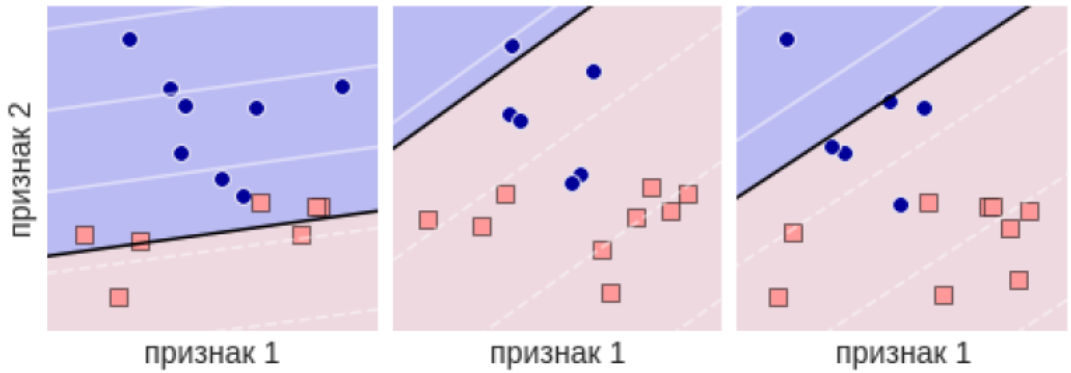
Каждый базовый алгоритм настраивается на случайной подвыборке обучения

Бэггинг	Подвыборка обучающей выборки берётся с помощью бутстрепа
Пэстинг (Pasting)	Случайная обучающая подвыборка
Случайные подпространства	Случайное подмножество признаков
Случайные патчи (Random Patches)	Одновременно берём случайное подмножество объектов и признаков
cross-validated committees	k обучений на (k-1)-м фолде

Бэггинг

Не всегда получается «как было задумано»...

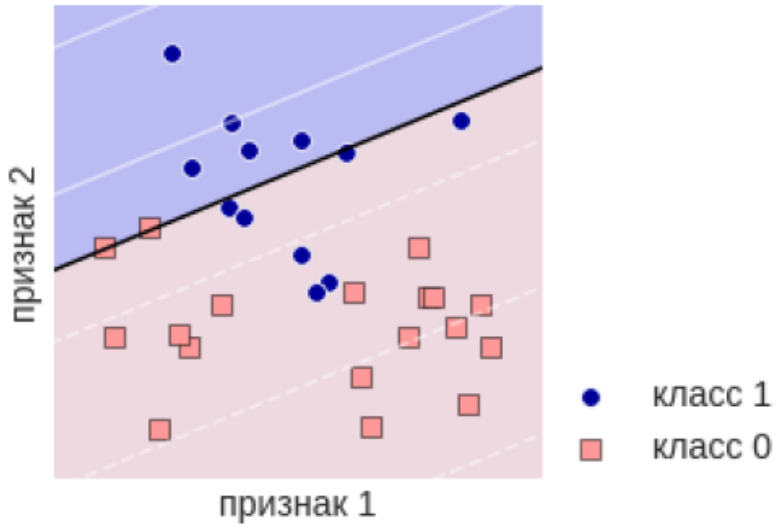
Построим 100 базовых логистических регрессий




```
model = BaggingClassifier(base_estimator=LogisticRegression(),
                          n_estimators=100,
                          max_samples=1.0,
                          max_features=1.0,
                          bootstrap=True,
                          bootstrap_features=False,
                          oob_score=False,
                          warm_start=False,
                          n_jobs=None,
                          random_state=None,
                          verbose=0)
```

```
model.fit(X, y)
```

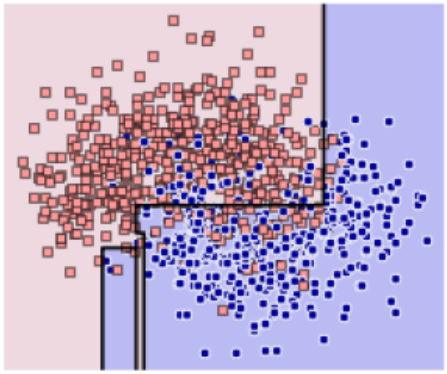
Результат



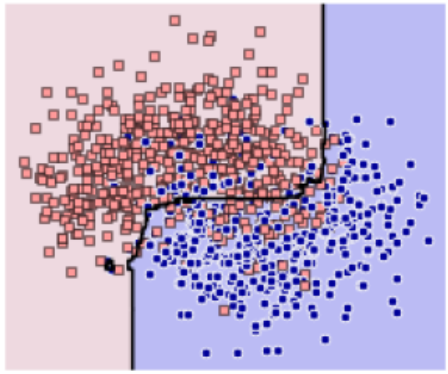
Это пример бэггинга над стабильными классификаторами, т.е. алгоритмами, решение которых изменяется незначительно при варьировании выборки

Деревья решений — нестабильные алгоритмы $\Rightarrow$ случайный лес

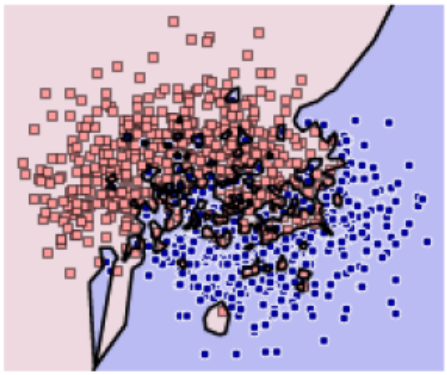
Примеры бэггинга



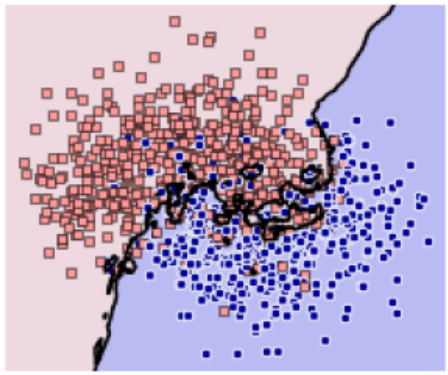
Одно дерево



Бэгинг 100 деревьев



Ближайший сосед



Бэгинг 100 ближайших соседей

Случайный лес (Random Forest)

Развитие идеи бэггинга [Breiman,2001]

Ансамбль параллельно обучаемых «независимых» деревьев решений

Независимое построение определённого количества деревьев:

- генерация случайной bootstrap-выборки из обучающей выборки(50–70% от размера всей обучающей выборки)
- построение дерева решений по данной подвыборке: в каждом новом узле дерева переменная для разбиения выбирается не из всех признаков, а из случайно выбранного их подмножества небольшой мощности

Бэггинг

procedure Random Forest

for $s = 1,2,\ldots,M$

begin

Из обучающей выборки построить бутстрэп-выборку

Построить дерево b_s, рекурсивно применяя следующую процедуру, пока не будет достигнут минимальный размер sz:

begin

Построить случайный набор из p признаков

Выбрать из него лучший признак и построить 2 дочерних узла

end

end

Для задачи регрессии **return** $f(x) = \frac{1}{M}\sum_{s=1}^{M} b_s(x)$

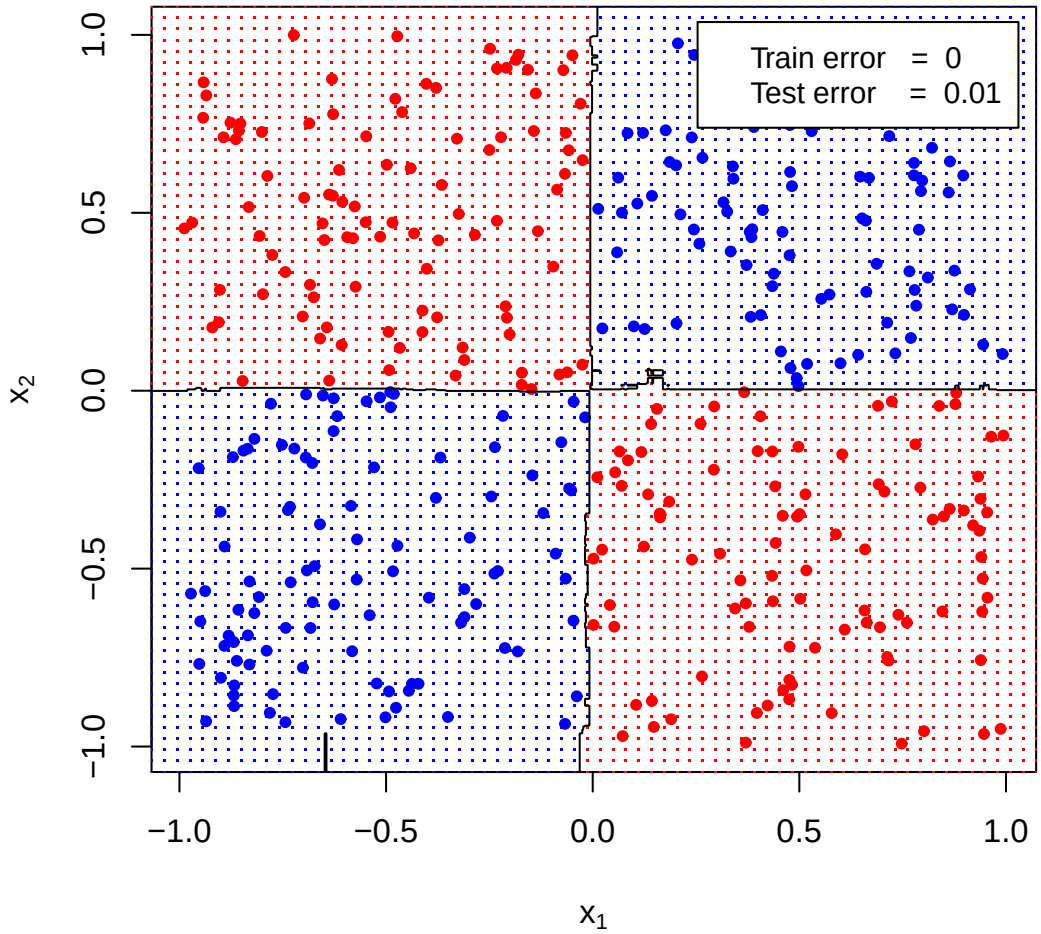
Для задачи классификации **return** $f(x) = \arg\max_{t=1,\ldots,k} \sum_{s=1}^{M} \mathrm{I}(b_s(x) = t)$

end

Для задачи регрессии, например, $p = \sqrt{n}$, $sz = 3$

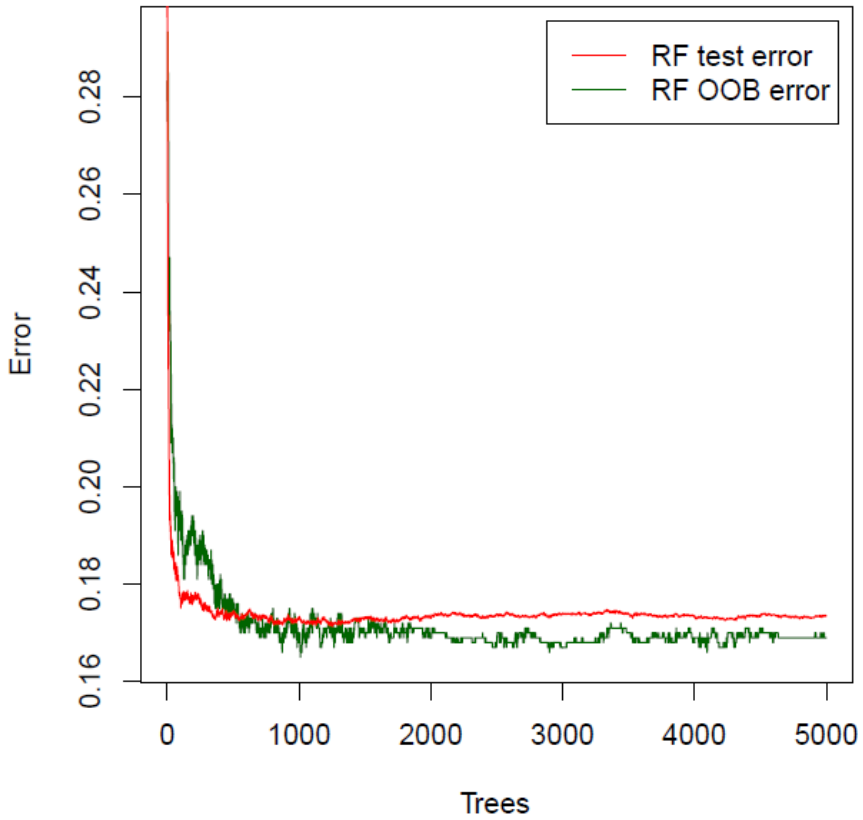
Для задачи классификации, например, $p = n/3$, $sz = 1$

Random Forest (50 деревьев)



Out of bag error (OOB)

Средняя ошибка классификатора (RF) по отдельным объектам, усреднённая по всем слабым классификаторам (деревьям), при обучении которых данный объект не вошёл в бутстрэп-выборку



Ошибка ансамбля на единичном объекте:

$$OOB(x_i) = \frac{1}{|T_i|}\sum_{t\in T} b_t(x_i), \quad T_i = \{t : x_i \notin U_t\}$$

Оценка ошибки ансамбля на обучающей выборке:

$$OOB(\mathcal{X}) = \sum_{i=1}^{m} L(OOB(x_i), y_i),$$

где $L(\cdot)$ — функция потерь.

Оценивание важности признаков:

$$\text{importance}_j = \frac{OOB^j(\mathcal{X}) - OOB(\mathcal{X})}{OOB(\mathcal{X})} \cdot 100\%.$$

При вычислении $b_t(x_i)$ для $OOB^j(\mathcal{X})$ признак с номером j случайным образом перемешивается на всех объектах не принадлежащим U_t.

Параметры, которые можно настраивать (в частности, по OOB):

- число деревьев (n_estimators)
- размер подвыборки (sampsize)
- число p случайно выбираемых признаков (mtry / max_features)
- максимальная глубина дерева (max_depth)
- минимальное число объектов в расщепляемой подвыборке (min_samples_split)
- минимальное число объектов в листьях (min_samples_leaf)
- критерий расщепления, например, для классификации энтропийный или Джини (criterion)

Бэггинг

Самый серьёзный параметр **mtry** / **max_features**

- Настраивается в первую очередь
- Зависимость унимодальная
- Зависит от числа шумовых признаков
- Надо перенастраивать при добавлении новых признаков
- Чем больше — тем однотипнее деревья
- Чем больше — тем медленнее настройка!
- Часто ансамблируют алгоритмы с разными mtry.

Бэггинг

Преимущества:

- метод-обёртка над базовым методом обучения
- простая реализация и простое распараллеливание
- универсальный метод, применяемый к широкому кругу задач

Недостатки:

- требуется очень много базовых алгоритмов
- трудно агрегировать устойчивые базовые методы обучения

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Бустинг

To boost — улучшать, повышать, рекламировать.
Попробуем строить последовательность алгоритмов, каждый из которых осведомлен об ошибках предыдущих.
Рассмотрим задачу бинарной классификации: $\mathcal{Y} = \{-1, +1\}$

Простая схема [Schapire,1990]
3 классификатора

b_1 обучается на m_1 объектах

b_2 обучается на m_2, объектах таких, что b_1 ровно на половине дает верный ответ

b_3 обучается на m_3 объектах, на которых $b_1(x) \neq b_2(x)$

return $f(x) = \operatorname{sgn}\left(\sum_{t=1}^{3} b_t(x)\right)$

Откуда брать данные для новых обучающих выборок?
Например, из исходной выборки путем изъятия с возвращением

Идея бустинга в задаче регрессии

Функция ошибки: $L(y, f)$,

уже есть алгоритм f(x), строим b(x):

$$f(x_i) + b(x_i) = a_i, \quad i = 1, \ldots, m$$

Надо:

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

Общая схема бустинга для задачи бинарной классификации

Взвешенное голосование:

$$f(x) = \operatorname{sgn}\left(\sum_{s=1}^{M} \alpha_s b_s(x)\right).$$

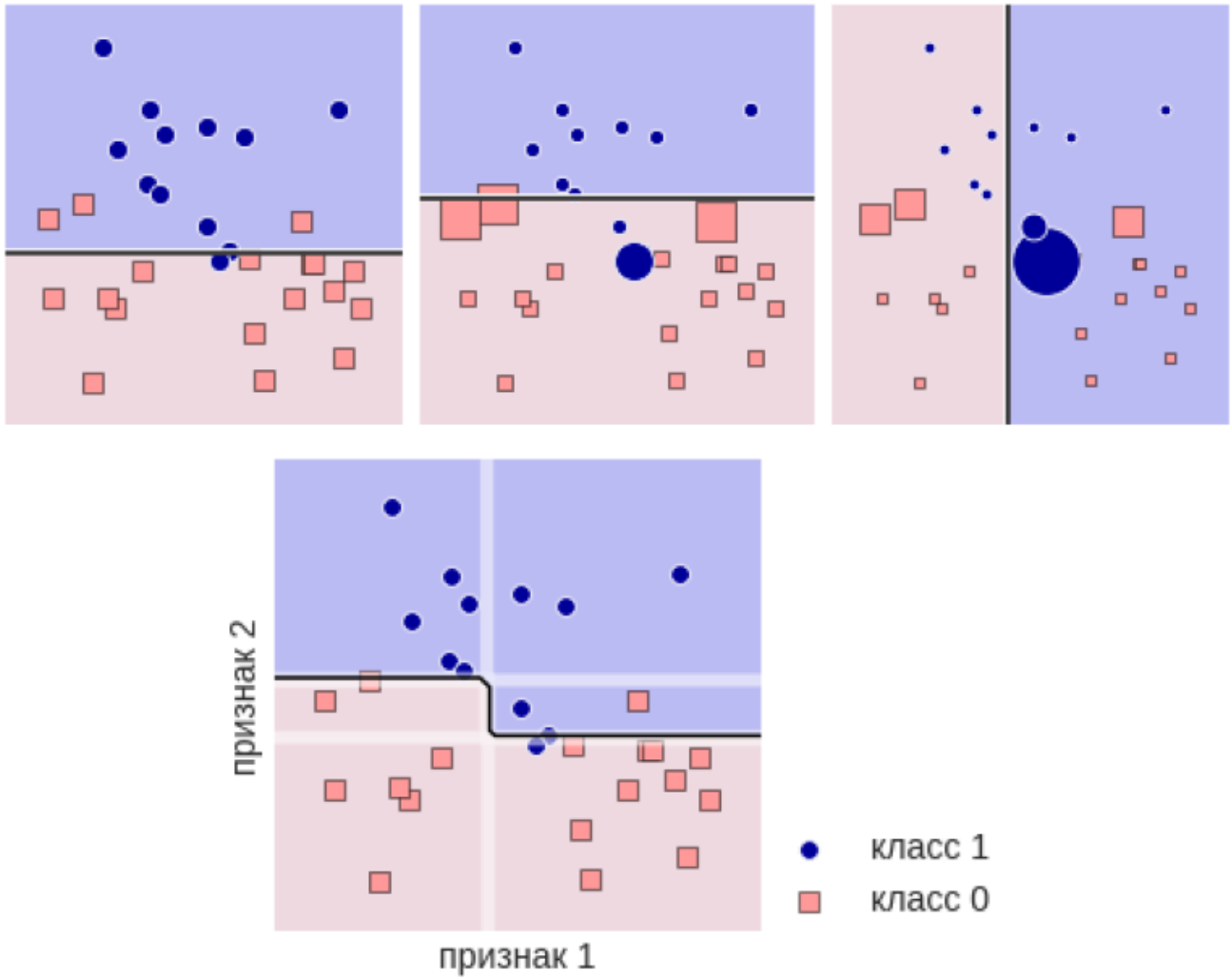
Функционал качества композиции — число ошибок на $\mathcal{X}$:

$$Q_M = \sum_{i=1}^{m} \mathrm{I}\left(y_i \sum_{s=1}^{M} \alpha_s b_s(x) < 0\right)$$

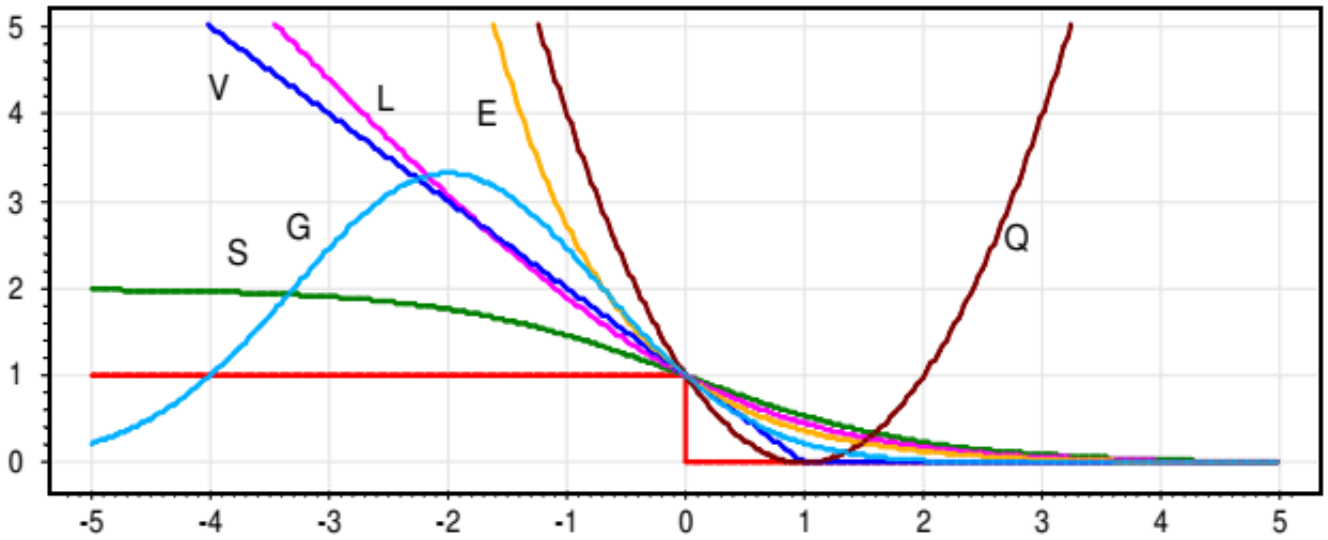
Две основных идеи бустинга:

- фиксируем $\alpha_1 b_1(x), \ldots, \alpha_{s-1} b_{s-1}(x)$ при добавлении $\alpha_s b_s(x)$
- гладкая аппроксимация пороговой функции потерь $\mathrm{I}(z < 0)$

Иллюстрация идеи бустинга



Гладкие аппроксимации пороговой функции потерь



$E(x) = \exp(x)$ — экспоненциальная (AdaBoost)

$L(x) = log_2(1 + e^{-x})$ — логарифимическая (LogitBoost)

$Q(x) = (1 - x)^2$ — квадратичная (GentleBoost)

$G(x) = \exp(-cx(x + s))$ — гауссовская (BrownBoost)

$S(x) = 2(1 + e^x)^{-1}$ — сигмоидная

$V(x) = (1 - x)_+$ — кусочно-линейная

Экспоненциальная аппроксимация пороговой функции потерь (AdaBoost)

$$Q_M \leq \widetilde{Q}_M = \sum_{i=1}^{m} \underbrace{\exp\left\{-y_i \sum_{s=1}^{M-1} \beta_s b_s(x_i)\right\}}_{w_i} \exp\left\{-y_i \beta_M b_M(x_i)\right\}$$

Чем больше вес w_i, тем сильнее текущий ансамбль ошибается на объекте x_i

Далее веса нормируются: $\widetilde{w}_i = w_i / \sum w_t$

Взвешенное число ошибочных классификаций при векторе весов $w = (w_1, \ldots, w_m)^T$:

$$\mathrm{Err}(f, w) = \sum_{i=1}^{m} w_i \, \mathrm{I}(f(x_i) = -y_i)$$

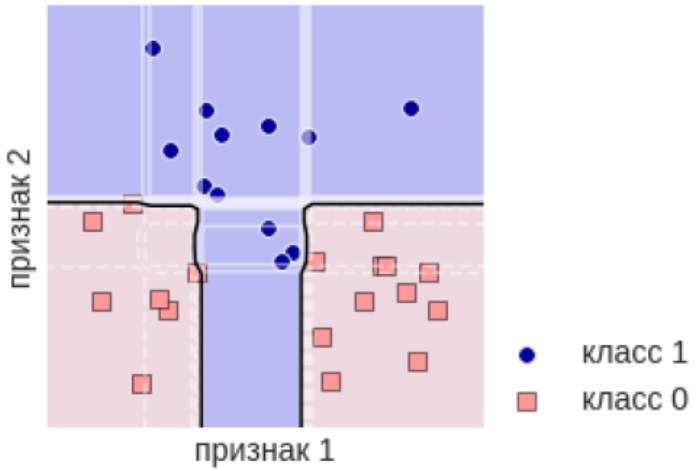
Теорема (1995)

Пусть $\forall \widetilde{w}$ существует алгоритм $f(\cdot)$, классифицирующий выборку хотя бы немного лучше, чем наугад: $\mathrm{Err}(f, w) < \frac{1}{2}$.
Тогда минимум функционала $\widetilde{Q}_M$ достигается при

$$f_M = \arg\min_f \mathrm{Err}(f, \widetilde{w}), \quad \beta_M = \frac{1}{2} \ln \frac{1 - \mathrm{Err}(f_M, \widetilde{w})}{\mathrm{Err}(f_M, \widetilde{w})}$$

AdaBoost: минутка кода

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
model =
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(
                   max_depth=1), base_estimators=20,
                   learning_rate=1.0, algorithm='SAMME.R',
                   random_state=1)
model.fit(X, y)
```



Недостатки AdaBoost

- чрезмерная чувствительность к выбросам
- неинтерпретируемое нагромождение из сотен алгоритмов
- не удается строить короткие композиции из «сильных» алгоритмов типа SVM (только длинные из слабых)
- требуются достаточно большие обучающие выборки (бэггинг обходится более короткими)

Способы преодоления:

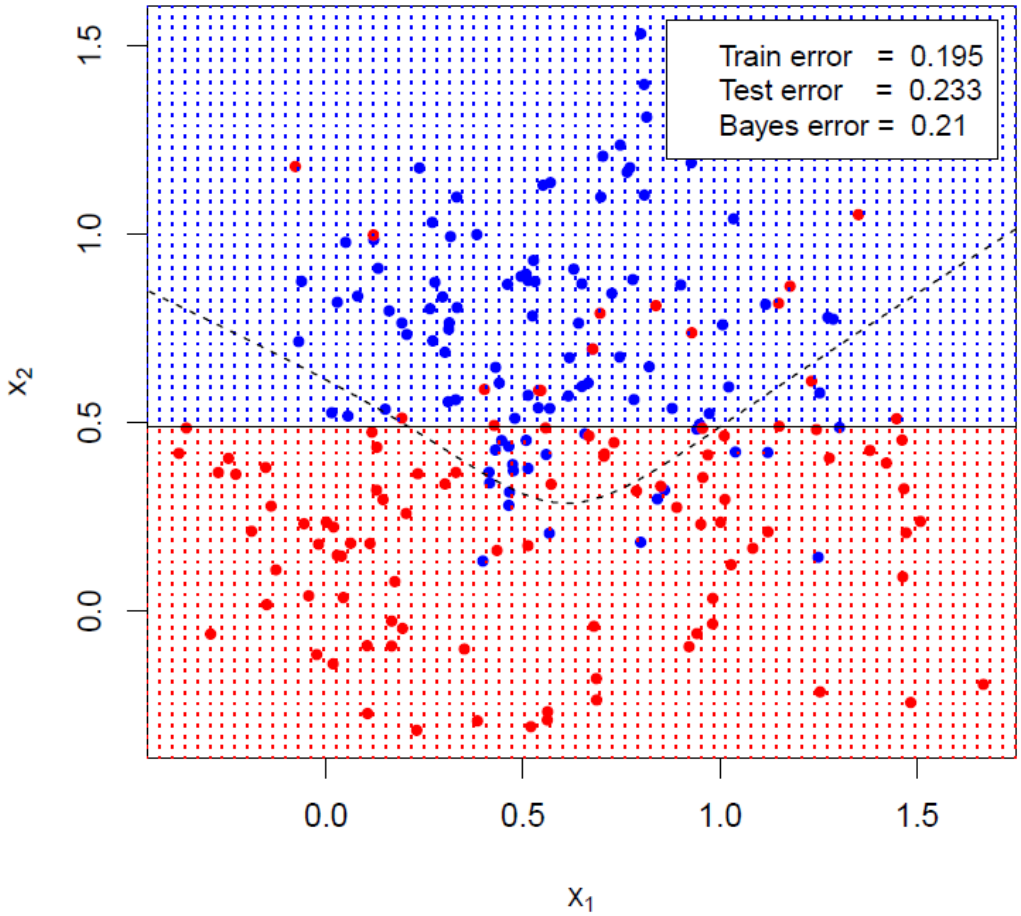
- отсев выбросов по критерию увеличения веса
- градиентный бустинг с произвольной функцией потерь

Общие замечания:

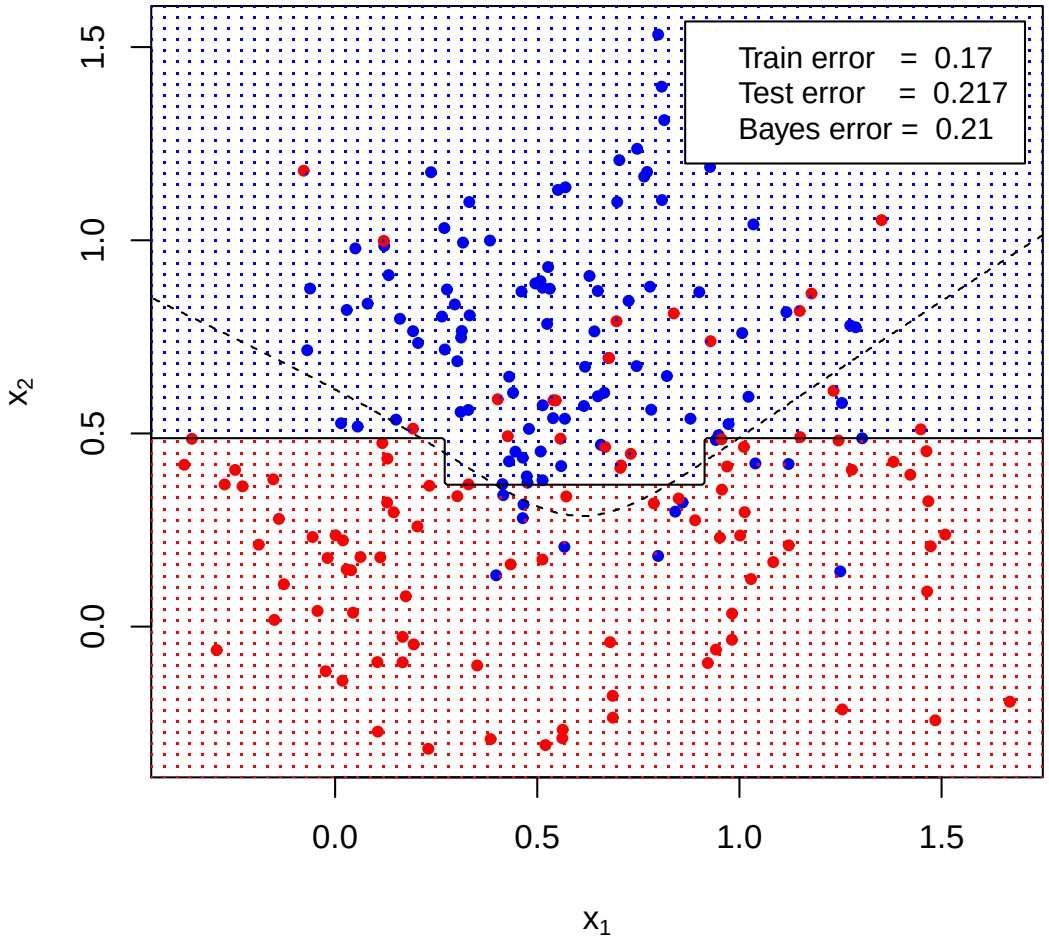
- **обобщающая способность бустинга не ухудшается с ростом сложности**
- бустинг лучше для классов с границей сложной формы, а бэггинг лучше для коротких обучающих выборок
- настройка весов не так сильно влияет на точность

Пример

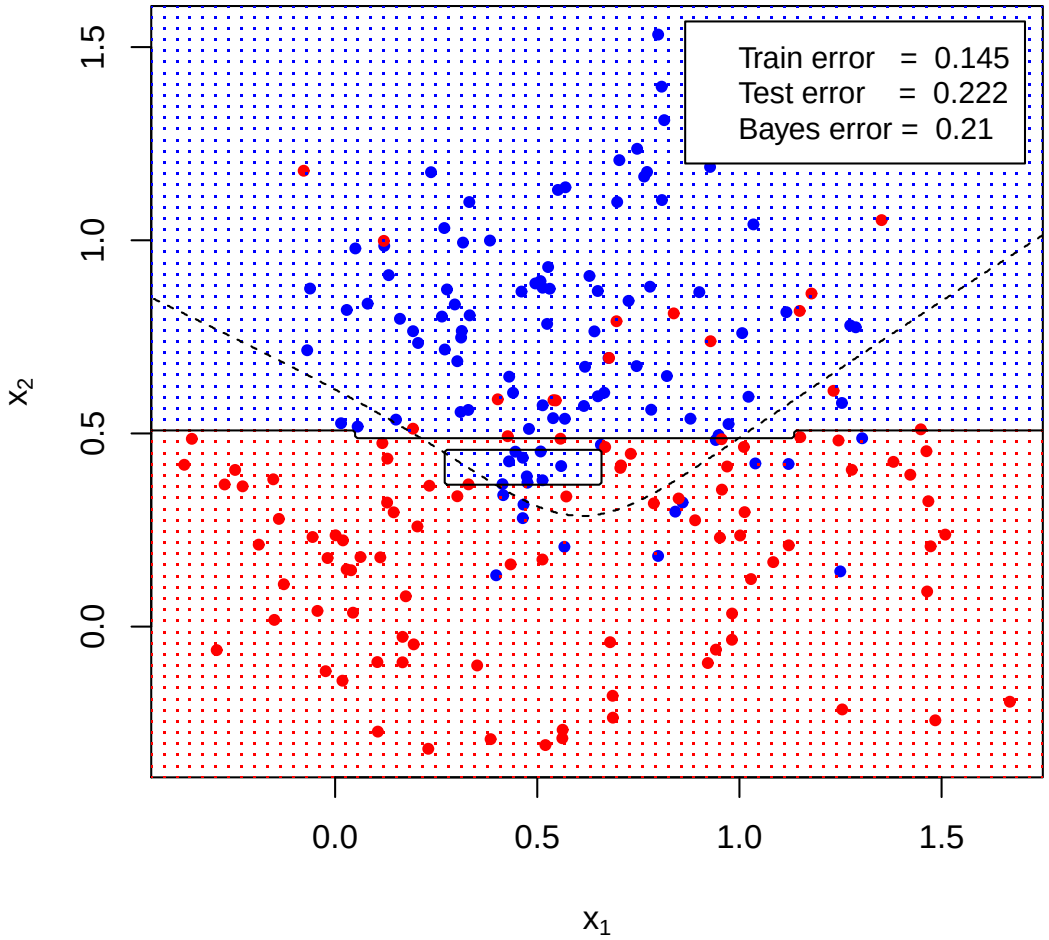
Слабые классификаторы — деревья решений высоты 1 (stumps)
$M = 1$



$M = 25$

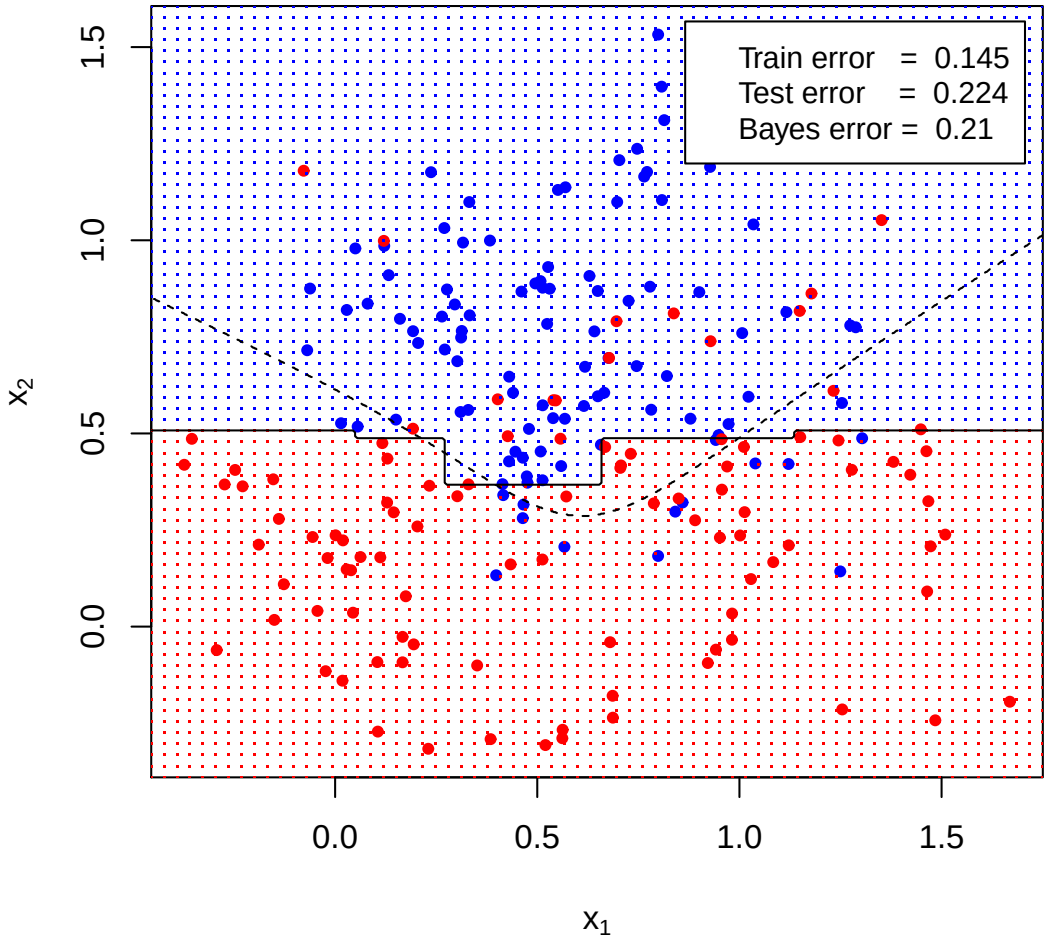

Train error = 0.17
Test error = 0.217
Bayes error = 0.21
0.0
0.5
1.0
1.5
x_1
x_2

$M = 50$

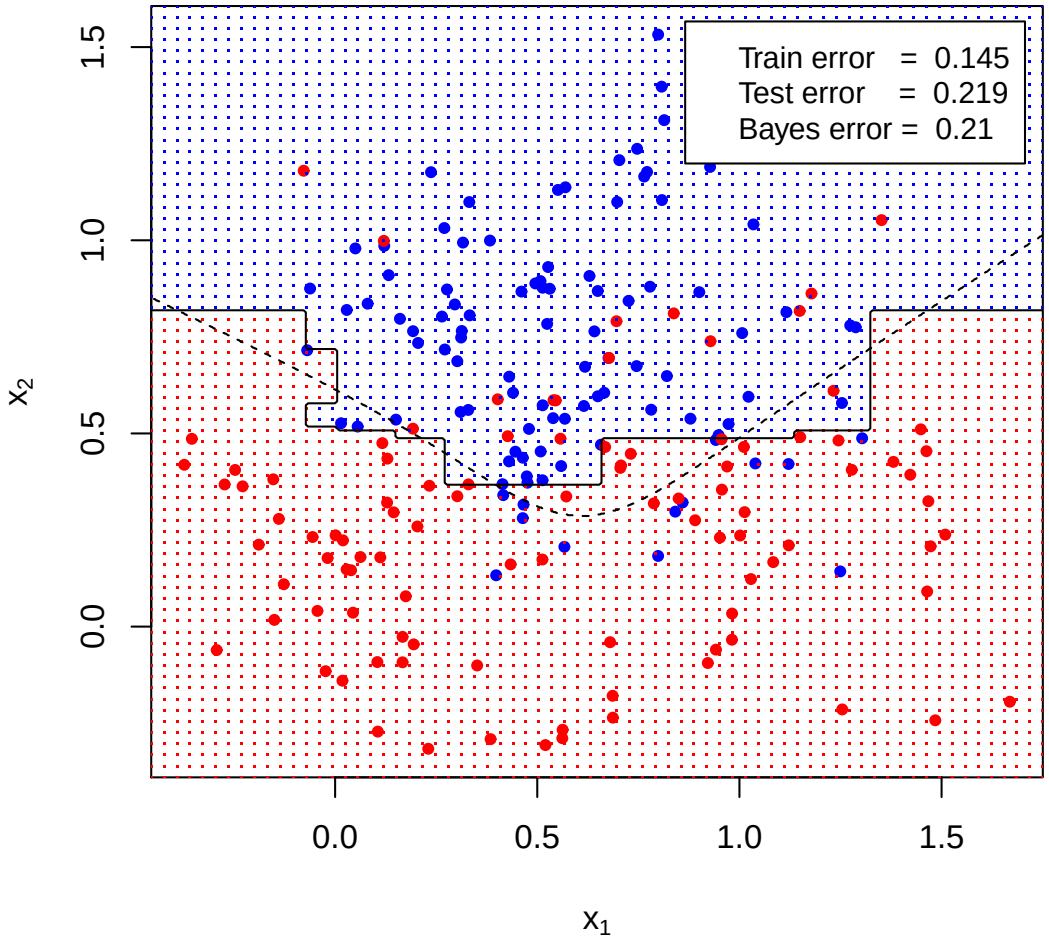

Train error = 0.145
Test error = 0.222
Bayes error = 0.21
0.0
0.5
1.0
1.5
x_1
x_2

Бустинг

$M = 75$

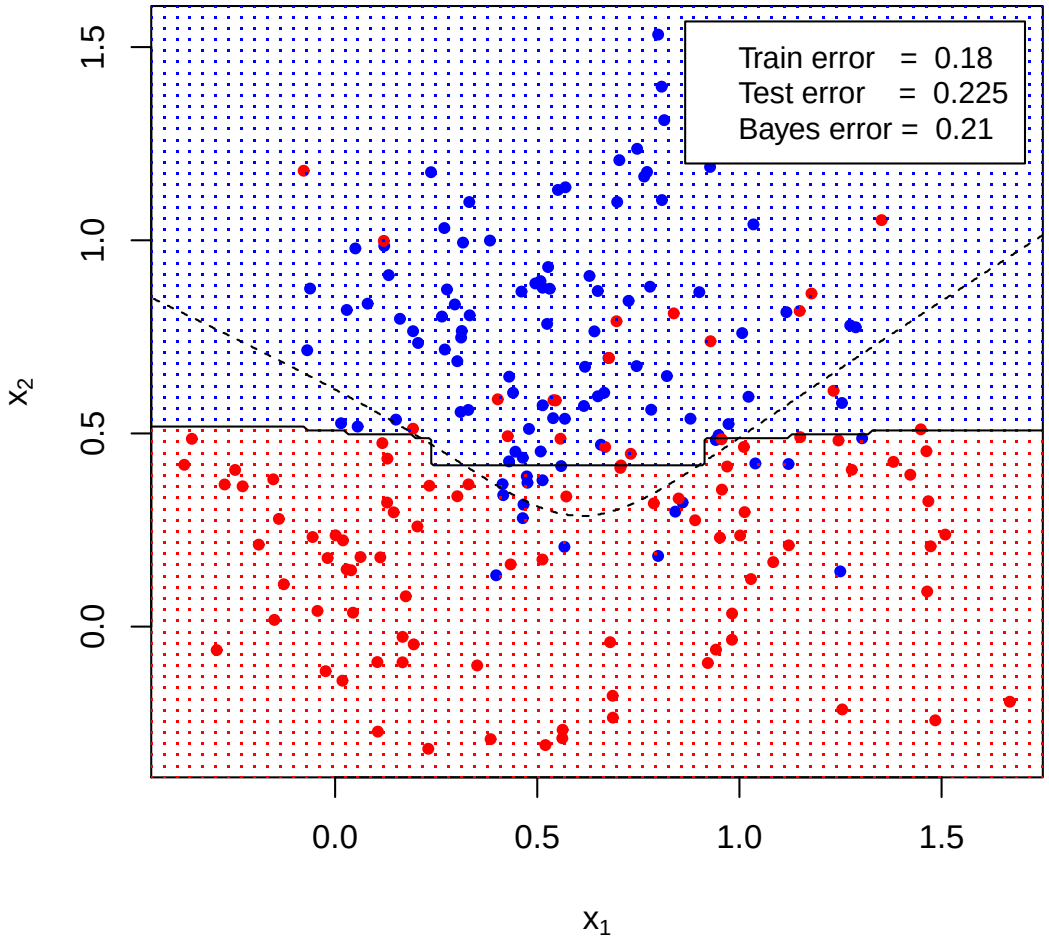

Train error = 0.145
Test error = 0.224
Bayes error = 0.21
0.0
0.5
1.0
1.5
x_1
x_2

$M = 100$


Train error = 0.145
Test error = 0.219
Bayes error = 0.21
0.0
0.5
1.0
1.5
x_1
x_2

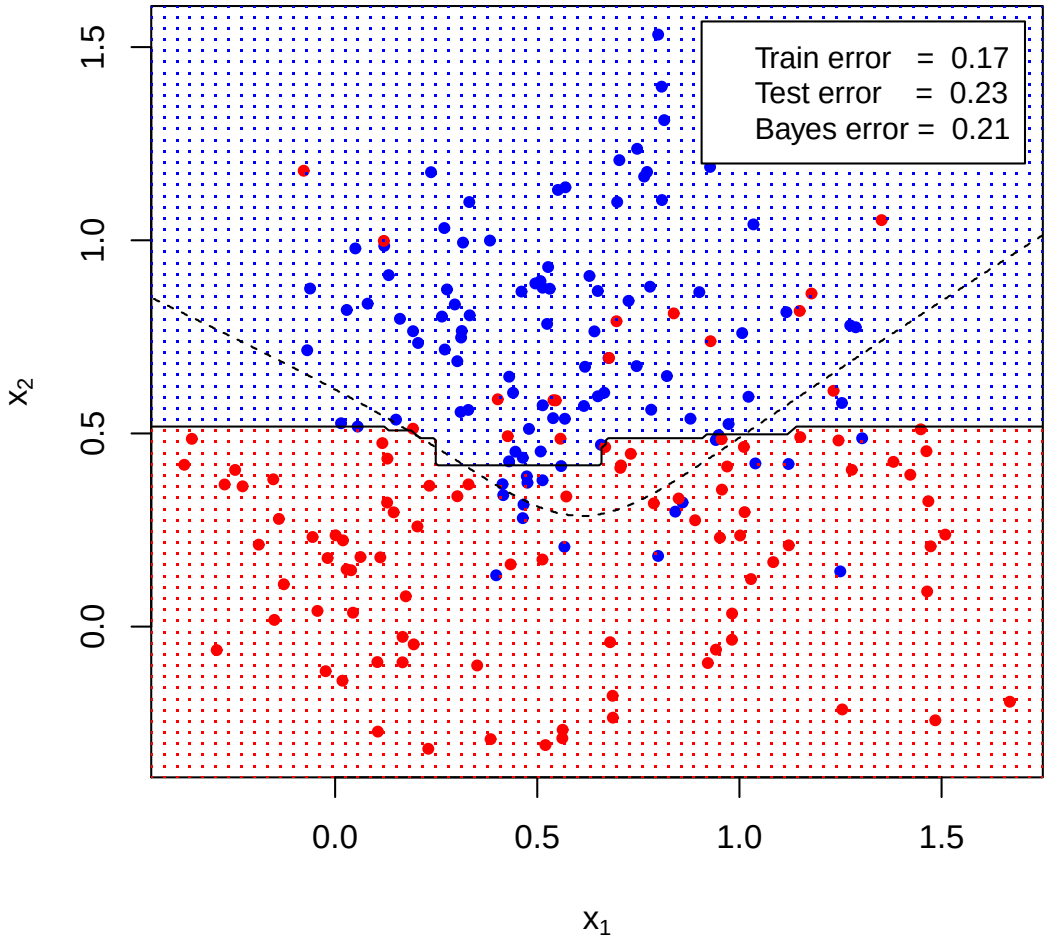
Бустинг

$M = 150$



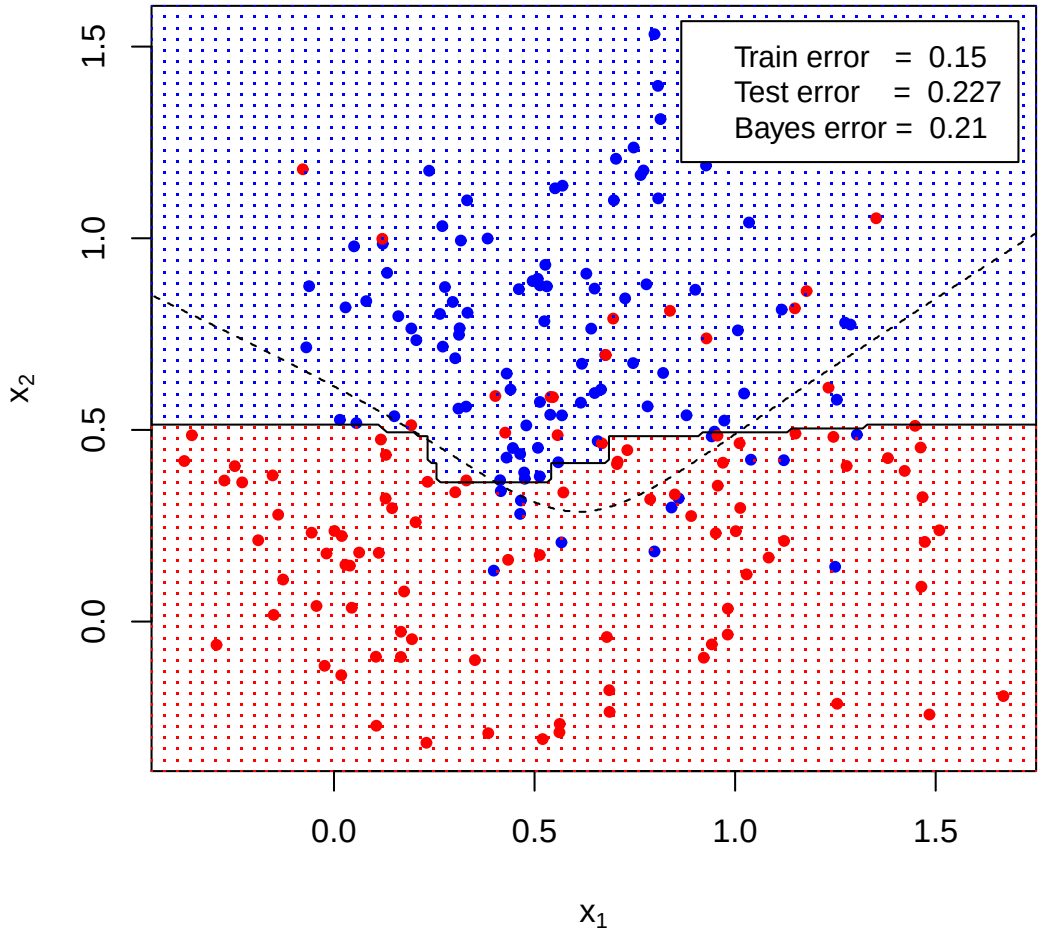
Бустинг

$M = 200$



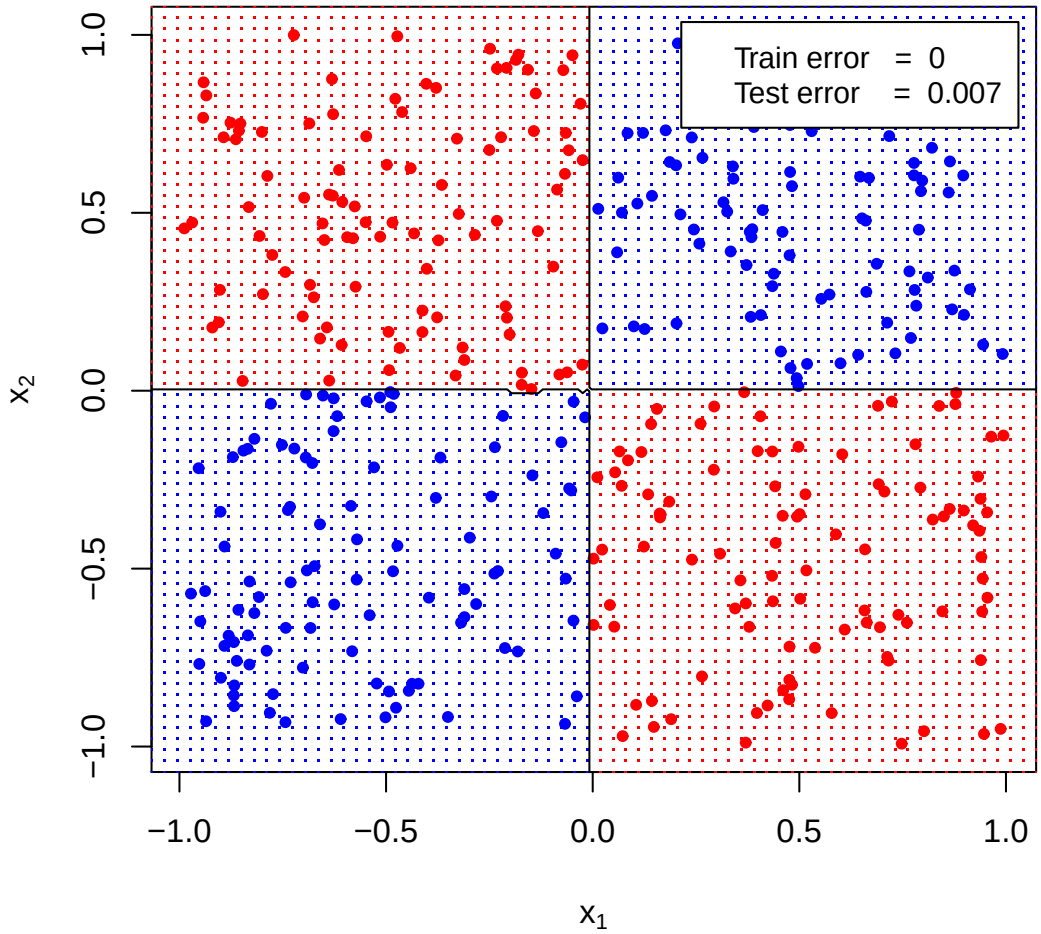
Бустинг

$M = 1000$



Бустинг

50 деревья решений высоты 2



(Если брать деревья решений высоты 1 — не хватает даже 1000)

Градиентный бустинг

Friedman, Jerome H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. // Annals of Statistics, 29(5), p. 1189–1232

Метод градиентного бустинга работает для любых дифференцируемых функций потерь и является одним из наиболее мощных и универсальных на сегодняшний день

Градиентный бустинг в задаче регрессии
Рассмотрим задачу минимизации квадратичного функционала:

$$\sum_{i=1}^{m}(a_i - y_i)^2 \to \min$$

Градиентный бустинг

Будем искать итоговый алгоритм в виде суммы базовых алгоритмов $b_s(x)$:

$$f_M(x) = \sum_{s=1}^{M} b_s(x).$$

Построим первый базовый алгоритм:

$$b_1(x) = \arg\min_b \sum_{i=1}^{m} (b(x_i) - y_i)^2$$

Теперь мы можем посчитать остатки на каждом объекте — расстояния от ответа нашего алгоритма до истинного ответа (ошибки первого алгоритма):

$$\delta_i^{(1)} = y_i - b_1(x_i),\ i = 1, \ldots, m.$$

Градиентный бустинг

Если прибавить эти остатки к ответам построенного алгоритма, то он не будет допускать ошибок на обучающей выборке. Значит, будет разумно построить второй алгоритм так, чтобы его ответы были как можно ближе к остаткам:

$$b_2(x) = \arg\min_b \sum_{i=1}^{m} \left(b(x_i) - \delta_i^{(1)} \right)^2$$

Каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

$$\delta_i^{(s-1)} = y_i - \sum_{j=1}^{s-1} b_j(x_i) = y_i - b_{s-1}(x_i), \ i = 1, \ldots, m;$$

$$b_s(x) = \arg\min_b \sum_{i=1}^{m} \left(b(x_i) - \delta_i^{(s-1)} \right)^2$$

Градиентный бустинг

Остатки могут быть найдены как антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенной композиции:

$$\delta_i^{(s)} = y_i - b_{s-1}(x_i) = -\frac{\partial}{\partial h}(h - y_i)\Big|_{h=b_{s-1}(x_i)}$$

Получается, что выбирается такой базовый алгоритм, который как можно сильнее уменьшит ошибку композиции — это свойство вытекает из его близости к антиградиенту функционала на обучающей выборке.

Градиентный бустинг

Пусть дана некоторая дифференцируемая функция потерь $L(y,h)$. Будем строить взвешенную сумму базовых алгоритмов:

$$b_M(x) = \sum_{s=1}^{M} \gamma_s b_s(x).$$

В композиции имеется начальный алгоритм b_0. Как правило коэффициент при нем берут $\gamma_0 = 1$, а сам алгоритм выбирают очень простым, например:

- нулевым $b_0(x) = 0$;
- возвращают самый популярный класс (для классификации): $b_0(x) = \arg\max_y \sum_{i=1}^{m} \mathrm{I}(y_i = y)$
- возвращают средний ответ (для регрессии): $b_0(x) = \frac{1}{m} \sum_{i=1}^{m} y_i$

Градиентный бустинг

Допустим, мы построили композицию $b_{s-1}(x)$ из $s-1$ алгоритма, и хотим выбрать следующий базовый алгоритм так, чтобы как можно сильнее уменьшить ошибку:

$$\sum_{i=1}^{m} L\left(y_i, b_{s-1}(x_i) + \gamma_s b_s(x_i)\right) \to \min_{b_s, \gamma_s}$$

Ответим в первую очередь на следующий вопрос:
если бы в качестве алгоритма b_s мы могли выбрать совершенно любую функцию, то какие значения ей следовало бы принимать на объектах обучающей выборки?
Иными словами, нам нужно понять, какие числа δ_i надо выбрать для решения следующей задачи:

$$\sum_{i=1}^{m} L\left(y_i, b_{s-1}(x_i) + \delta_i\right) \to \min_{\delta_i}$$

Градиентный бустинг

Можно требовать, чтобы $\delta_i = y_i - b_{s-1}(x_i)$, но такой подход никак не учитывает особенностей функции $L(h,y)$ и требует лишь совпадения предсказаний и истинных ответов. Более разумно потребовать, чтобы сдвиг δ_i был противоположен производной функции потерь в точке $h = b_{s-1}(x_i)$:

$$\delta_i = -\frac{\partial L}{\partial h}(h - y_i)\Big|_{h=b_{s-1}(x_i)}$$

В этом случае мы сдвинемся в сторону скорейшего убывания функции потерь. Заметим, что вектор сдвигов $\delta = (\delta_1, \ldots, \delta_m)$ совпадает с антиградиентом:

$$\left(\frac{\partial L}{\partial h}(h - y_i)\Big|_{h=b_{s-1}(x_i)}\right)_{i=1}^{m} = -\nabla_h \sum_{i=1}^{m} L(h_i,y_i)\Big|_{h_i=b_{s-1}(x_i)}$$

Градиентный бустинг

При таком выборе сдвигов δ_i мы, по сути, сделаем один шаг градиентного спуска, двигаясь в сторону наискорейшего убывания ошибки на обучающей выборке.

Отметим, что речь идет о градиентном спуске в m-мерном пространстве предсказаний алгоритма на объектах обучающей выборки. Поскольку вектор сдвига будет свой на каждой итерации, правильнее обозначать его как $\delta_i^{(s)}$.

Далее строится базовый алгоритм, приближающий градиент функции потерь на обучающей выборке:

$$b_s(x) = \arg\min_b \sum_{i=1}^{m} (b(x_i) - \delta_i)^2.$$

Градиентный бустинг

Здесь мы оптимизируем квадратичную функцию потерь независимо от функционала исходной задачи — вся информация о функции потерь $L(\cdot)$ находится в антиградиенте δ_i, а на данном шаге лишь решается задача аппроксимации функции по m точкам. После того, как новый базовый алгоритм найден, можно подобрать коэффициент при нем по аналогии с наискорейшим градиентным спуском:

$$\gamma_s = \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{m} L\left(y_i, b_{s-1}(x_i) + \gamma b_s(x_i)\right)$$

Описанный подход с аппроксимацией антиградиента базовыми алгоритмами и называется градиентным бустингом

Градиентный бустинг

Градиентный бустинг представляет собой поиск лучшей функции, восстанавливающей истинную зависимость ответов от объектов, в пространстве всех возможных функций. Ищем мы данную функцию с помощью «псевдоградиентного» спуска — каждый шаг делается вдоль направления, задаваемого некоторым базовым алгоритмом. При этом сам базовый алгоритм выбирается так, чтобы как можно лучше приближать антиградиент ошибки на обучающей выборке.

Регуляризация

На практике оказывается, что градиентный бустинг очень быстро строит композицию, ошибка которой на обучении выходит на асимптоту, после чего начинает настраиваться на шум и переобучаться. Это явление можно объяснить одной из двух причин:

- Если базовые алгоритмы очень простые (например, решающие деревья небольшой глубины), то они плохо приближают вектор антиградиента, градиентный бустинг может свестись к случайному блужданию в пространстве
- Если базовые алгоритмы сложные (например, глубокие решающие деревья), то они способны за несколько шагов бустинга идеально подогнаться под обучающую выборку — что, очевидно, будет являться переобучением

Способы регуляризации

1. **Сокращение шага** (shrinkage)

Хорошо зарекомендовавшим себя способом решения данной проблемы является сокращение шага: вместо перехода в оптимальную точку в направлении антиградиента делается укороченный шаг

$$f_s(x) = f_{s-1}(x) + \eta \cdot \gamma_s b_s(x),$$

где $\eta \in (0,1]$ — темп обучения (learning rate)

Сокращение шага, по сути, позволяет понизить доверие к направлению, восстановленному базовым алгоритмом

2. **Стохастический градиентный бустинг**

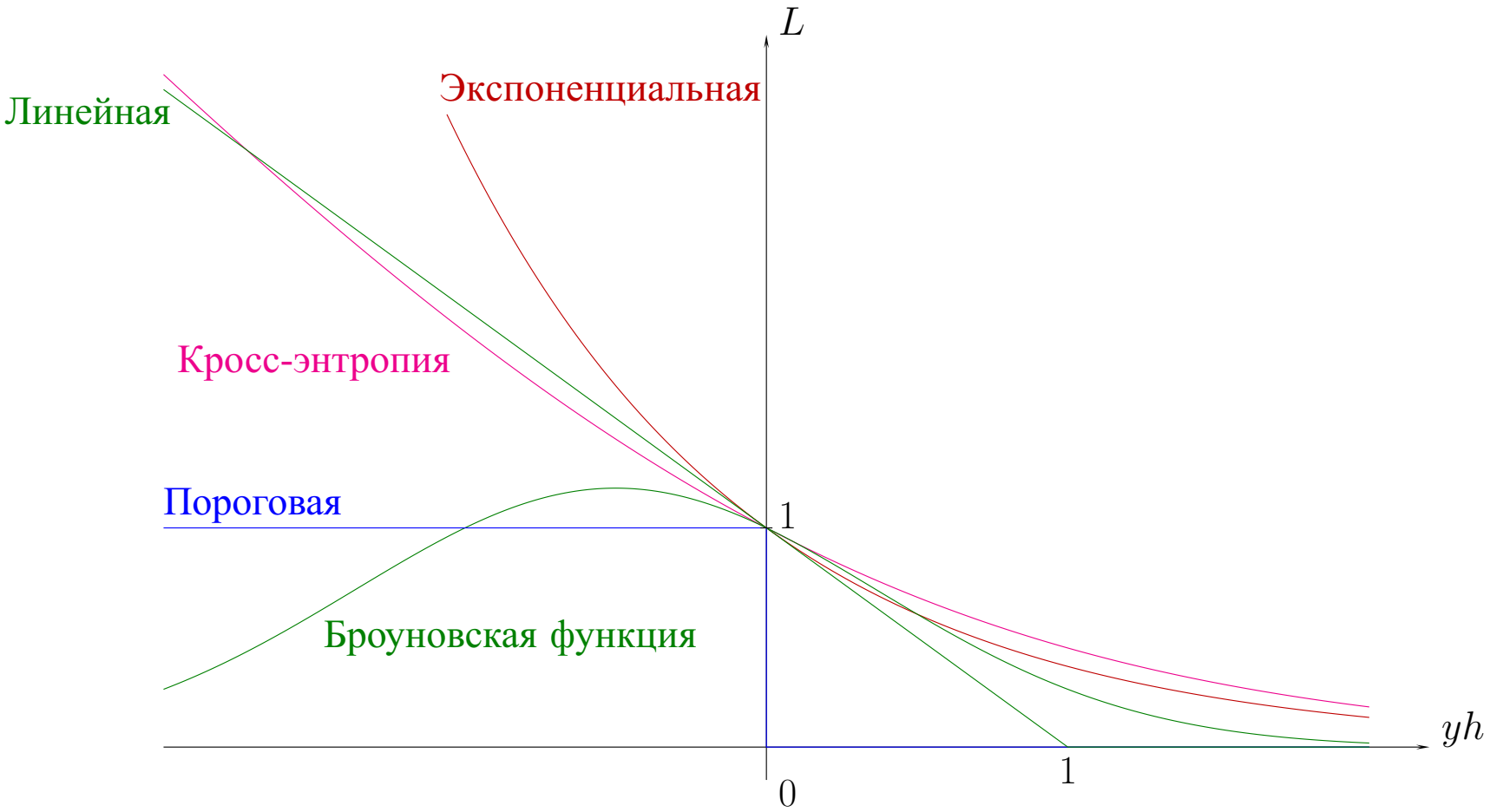
Ещё одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов

Алгоритм b_s обучается не по всей выборке $\mathcal{X}$, а лишь по ее случайному подмножеству $\mathcal{X}_s \subset \mathcal{X}$

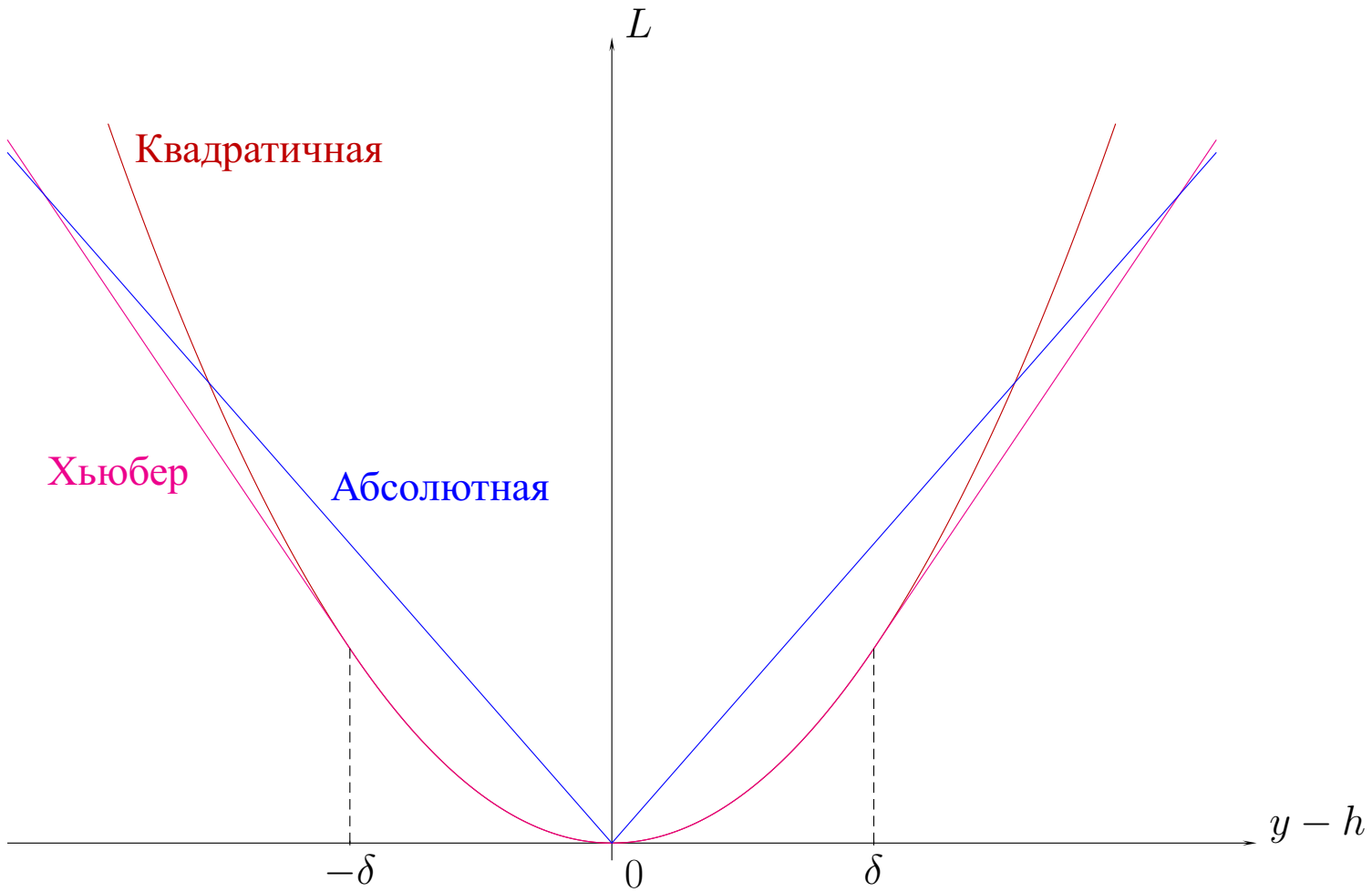
Функции потерь:

- Задача классификации:
  - Пороговая функция: $L(h,y) = \mathrm{I}(yh < 0)$
  - Линейная функция: $L(h,y) = (1 - yh)\,\mathrm{I}(yh < 1)$
  - Кросс-энтропия: $L(h,y) = \log_2(1 + e^{-yh})$
  - Экспоненциальная функция: $L(h,y) = e^{-yh}$
  - Броуновская функция: $L(h,y) = \exp\{-cyh(yh + d)\}$
- Задача регрессии:
  - Квадратичная функция: $L(h,y) = (y - h)^2$
  - Абсолютная функция: $L(h,y) = 2|y - h|$
  - Функция Хьюбера

    $$L(h,y) = (y - h)^2\,\mathrm{I}(|y - h| \le \delta) + \delta(2|y - h| - \delta)\,\mathrm{I}(|y - h| > \delta)$$


L
Экспоненциальная
Линейная
Кросс-энтропия
Пороговая
1
Броуновская функция
yh
1
0

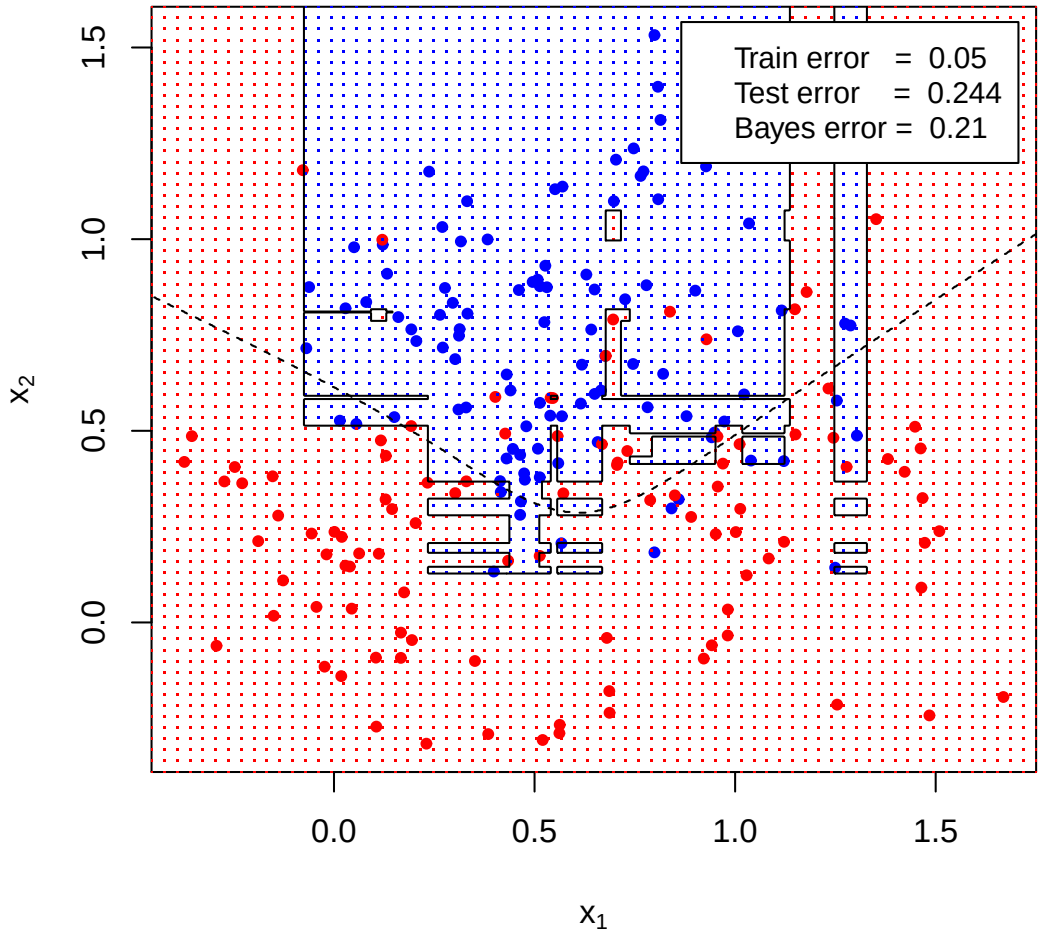
$\delta = 1.3$



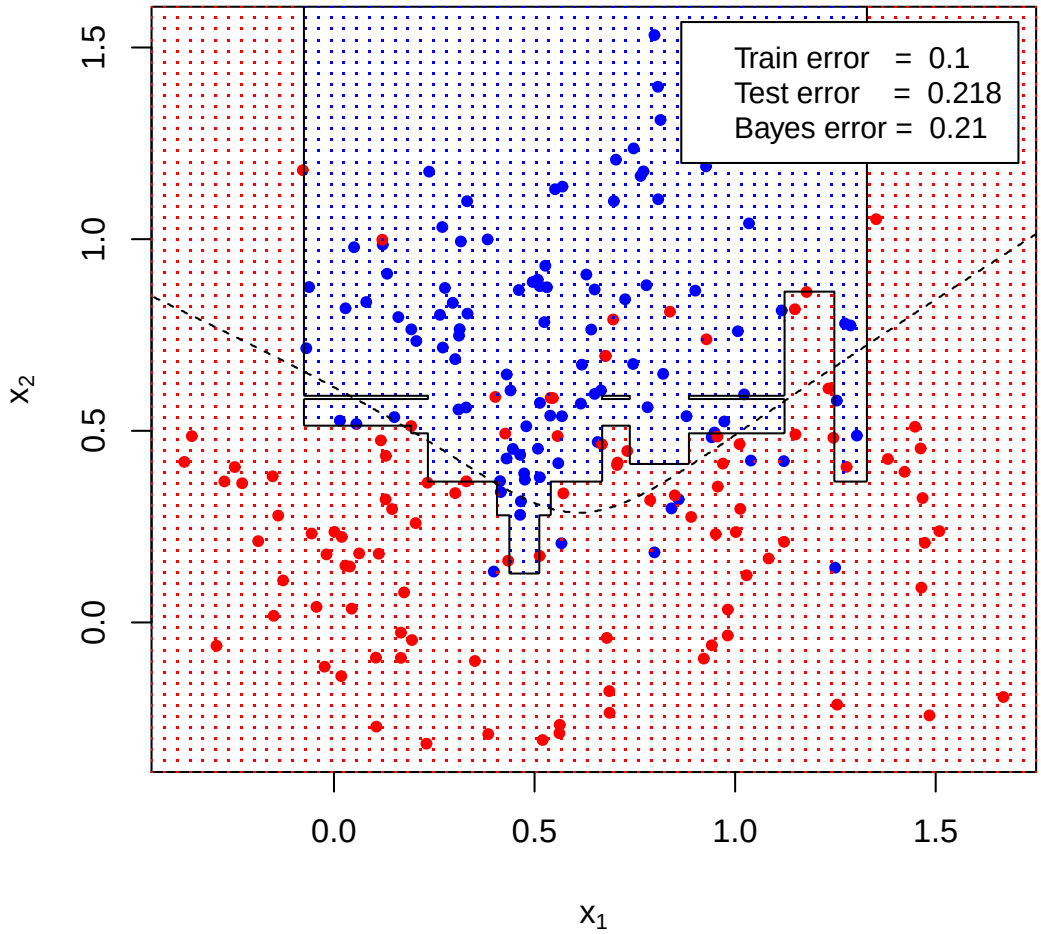
Выводы

- В случайных лесах используются глубокие деревья, поскольку от базовых алгоритмов требуется низкое смещение; разброс же устраняется за счёт усреднения ответов различных деревьев
- Бустинг работает несколько иначе — каждый следующий алгоритм целенаправленно понижает ошибку композиции, и даже при использовании простейших базовых моделей композиция может оказаться достаточно сложной
- Как правило, в бустинге используются неглубокие решающие деревья (3-6 уровней)

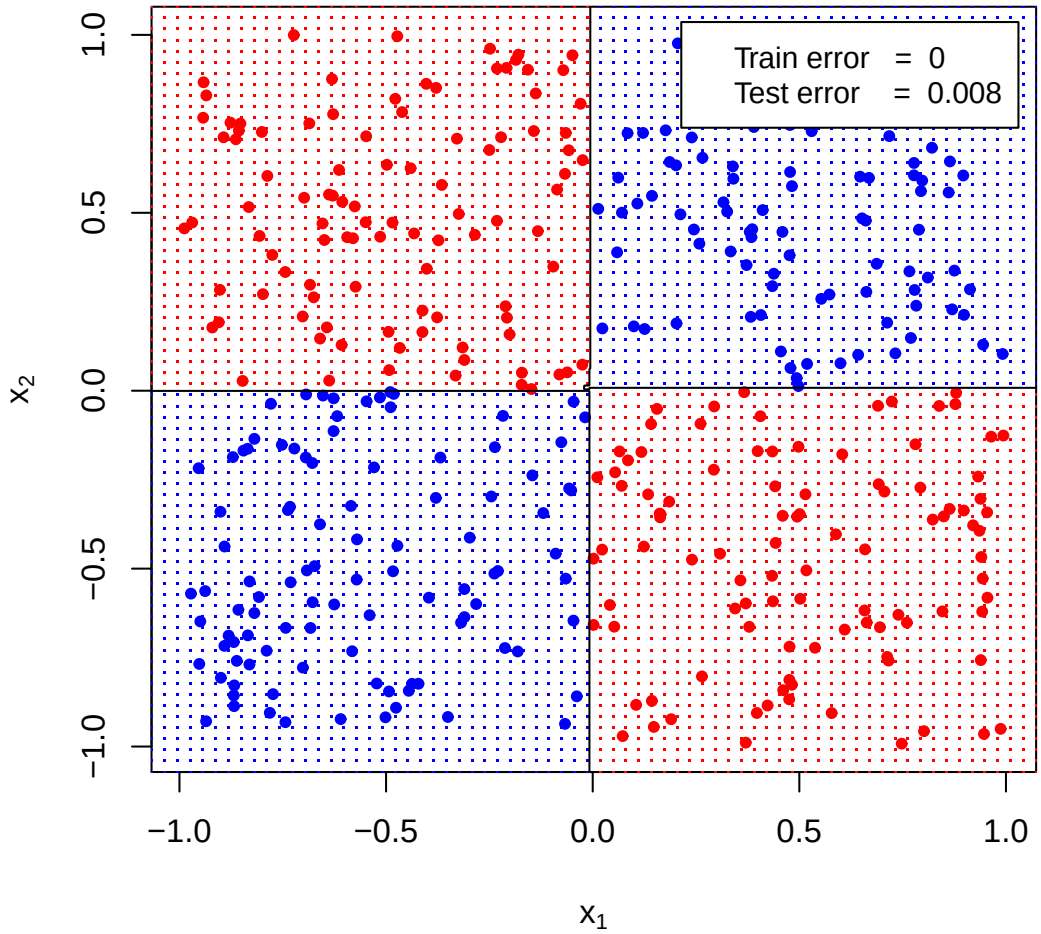
$M = 100$, штрафная функция — кросс-энтропия, stumps



$M = 100$, shrinkage $= 0.5$



50 деревья решений высоты 2, no shrinkage



Методы второго порядка

- Базовый алгоритм приближает направление, посчитанное с учётом вторых производных функции потерь
- Функционал регуляризуется — добавляются штрафы за количество листьев и за норму коэффициентов
- Очень быстрая реализация за счёт аналитических формул

Основные алгоритмы бустинга

- XGBoost — экстремальный градиентный бустинг
  https://xgboost.ai/
- LightGBM — для обучения на больших данных
  https://lightgbm.readthedocs.io/en/latest/
- CatBoost — для большого количества категориальных признаков
  https://catboost.ai/
- Сравнение бустингов
  https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Стекинг и блендинг

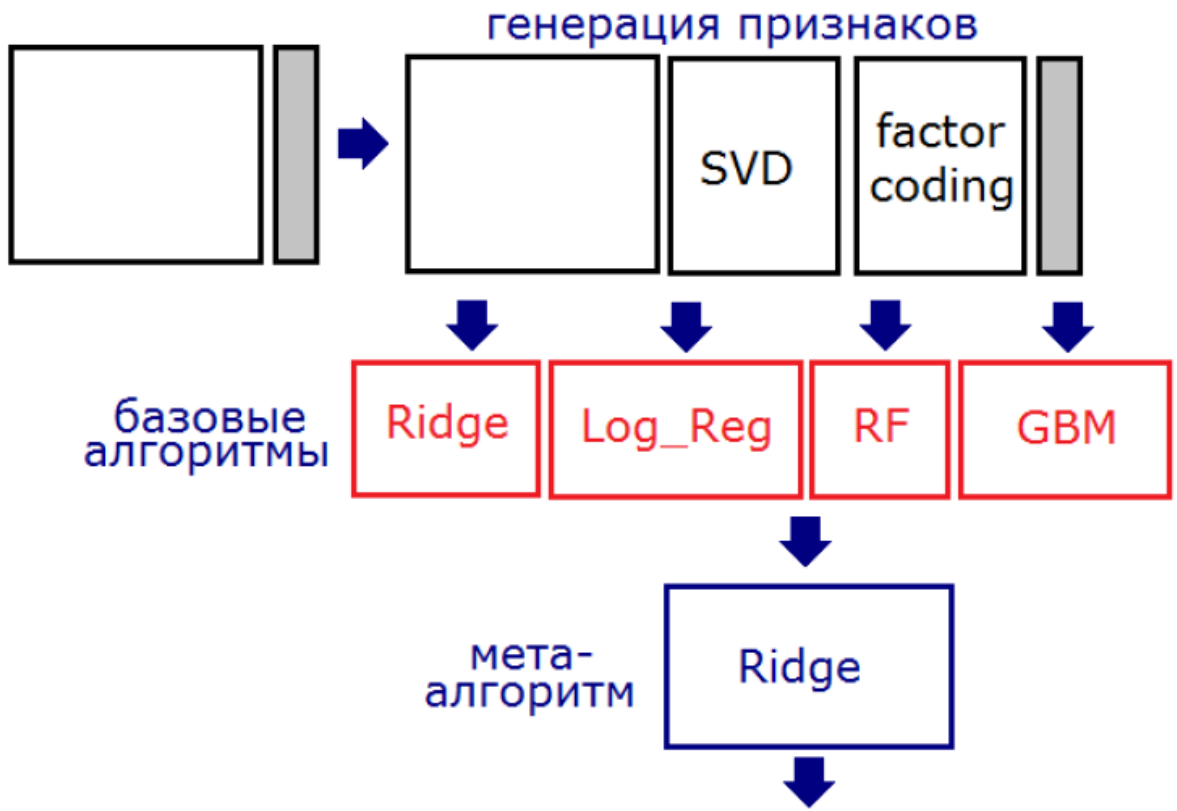
Хорошо усреднять алгоритмы, но почему именно усреднять?

Существуют способы построения композиций помимо бустинга и бэггинга. Большую популярность имеет стекинг, в котором прогнозы алгоритмов объявляются новыми признаками, и поверх них обучается ещё один алгоритм (который иногда называют мета-алгоритмом). Стекинг очень популярен в соревнованиях по анализу данных, поскольку позволяет агрегировать разные модели (различные композиции, линейные модели, нейросети и т.д.; иногда в качестве базовых алгоритмов могут выступать результаты градиентного бустинга с разными значениями гиперпараметров).

$$f(x) = a(b_1(x), \ldots, b_M(x))$$

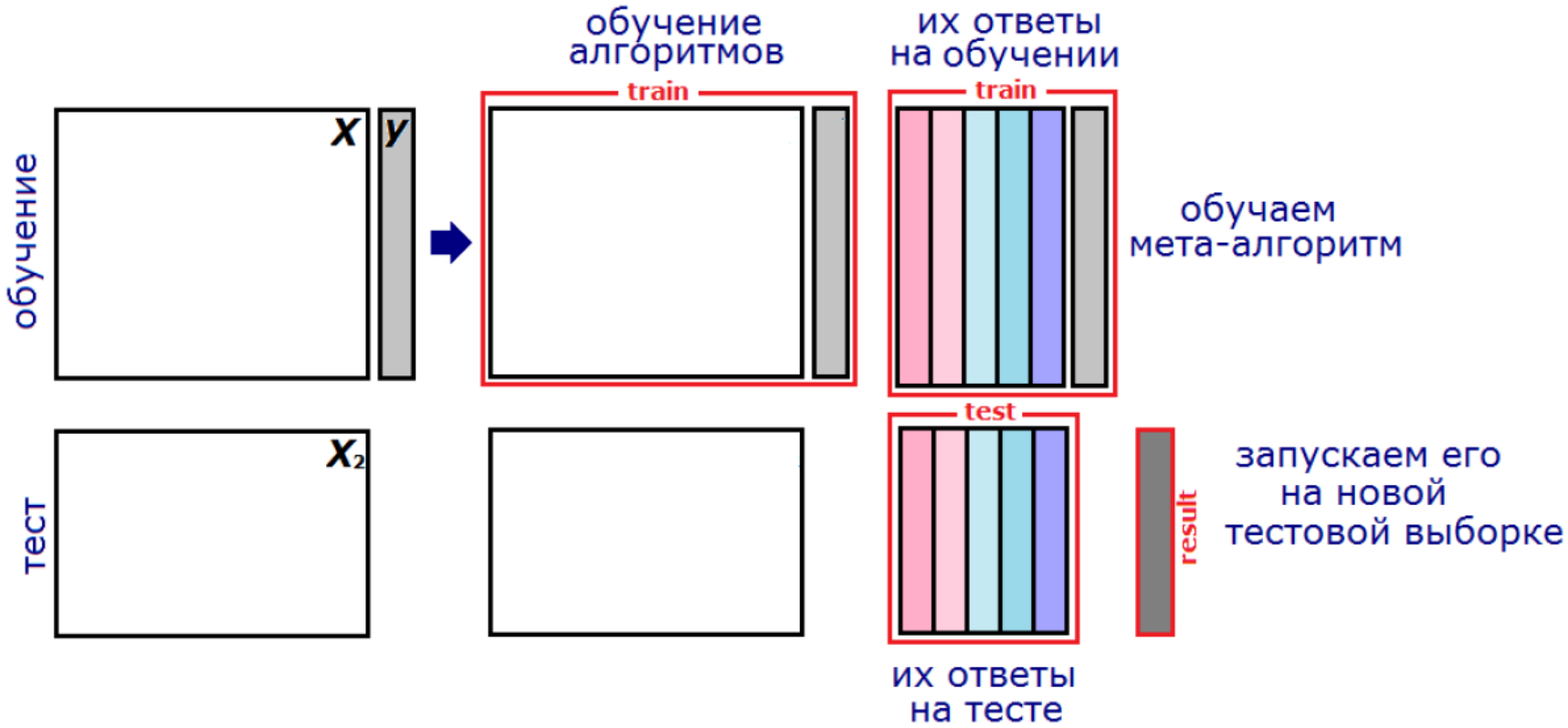
$a(\cdot)$ — мета-алгоритм, который нужно отдельно настроить!

Стекинг



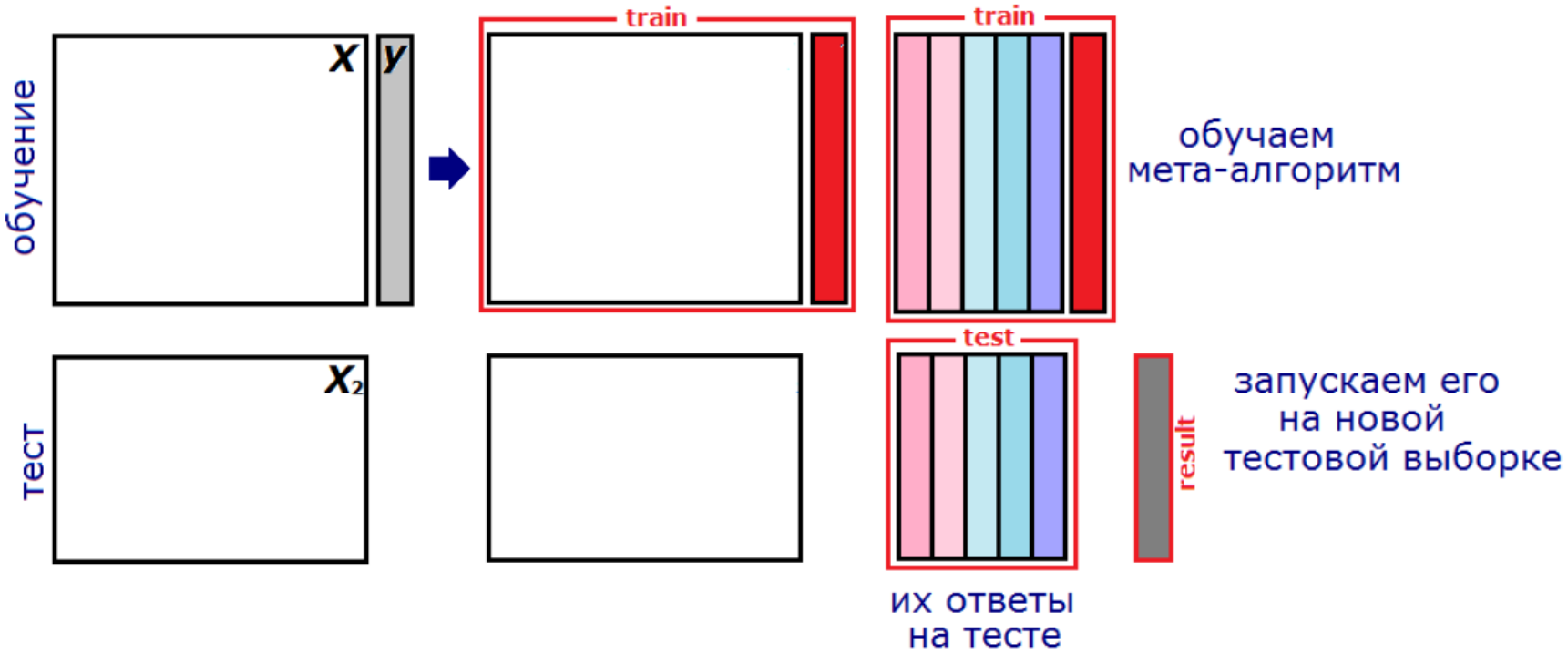
Используем ответы алгоритмов как признаки для нового мета-алгоритма машинного обучения

Наивный стекинг



что здесь неправильно?

Наивный стекинг

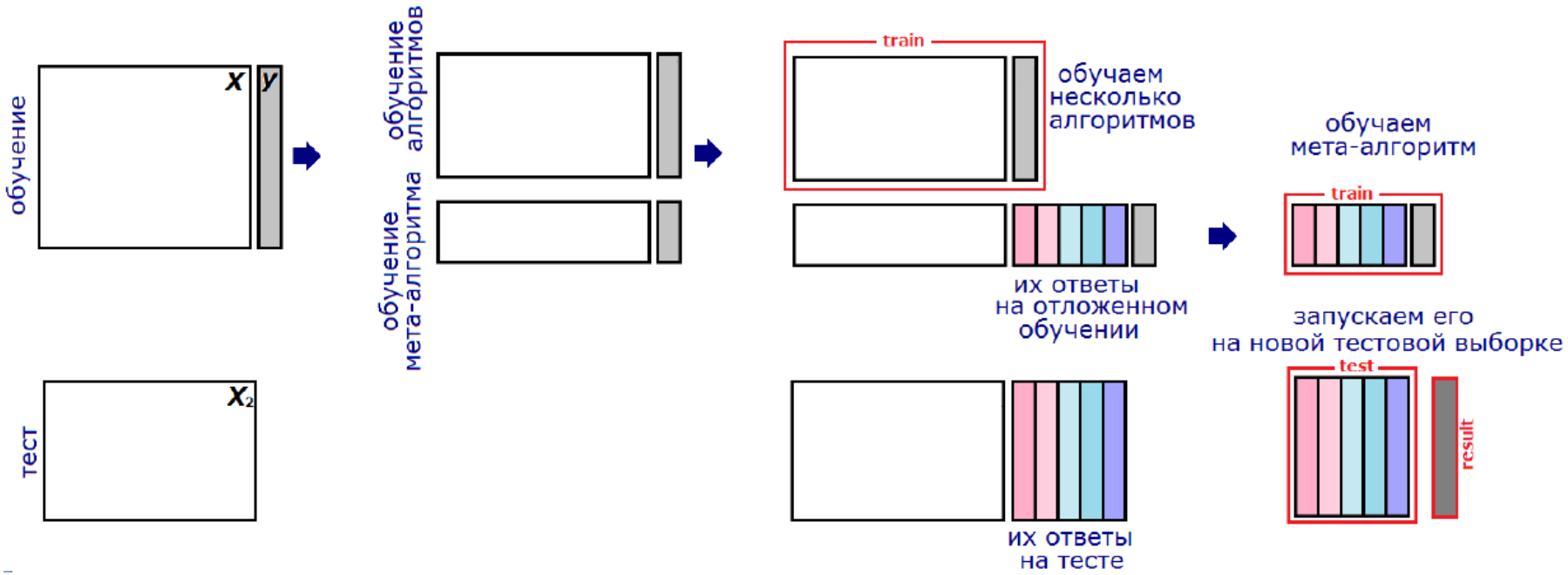


Метки целевой переменной два раза участвуют в обучении

Блендинг (простейшая форма стекинга)

Идея: набор базовых алгоритмов подаём на вход любому алгоритму машинного обучения

Проблема: такой мета-алгоритм нельзя обучать на тех же данных, что и базовые алгоритмы



Блендинг

Термин введен победителями конкурса Netflix
https://www.netflixprize.comhttps://www.netflixprize.com/

https://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf

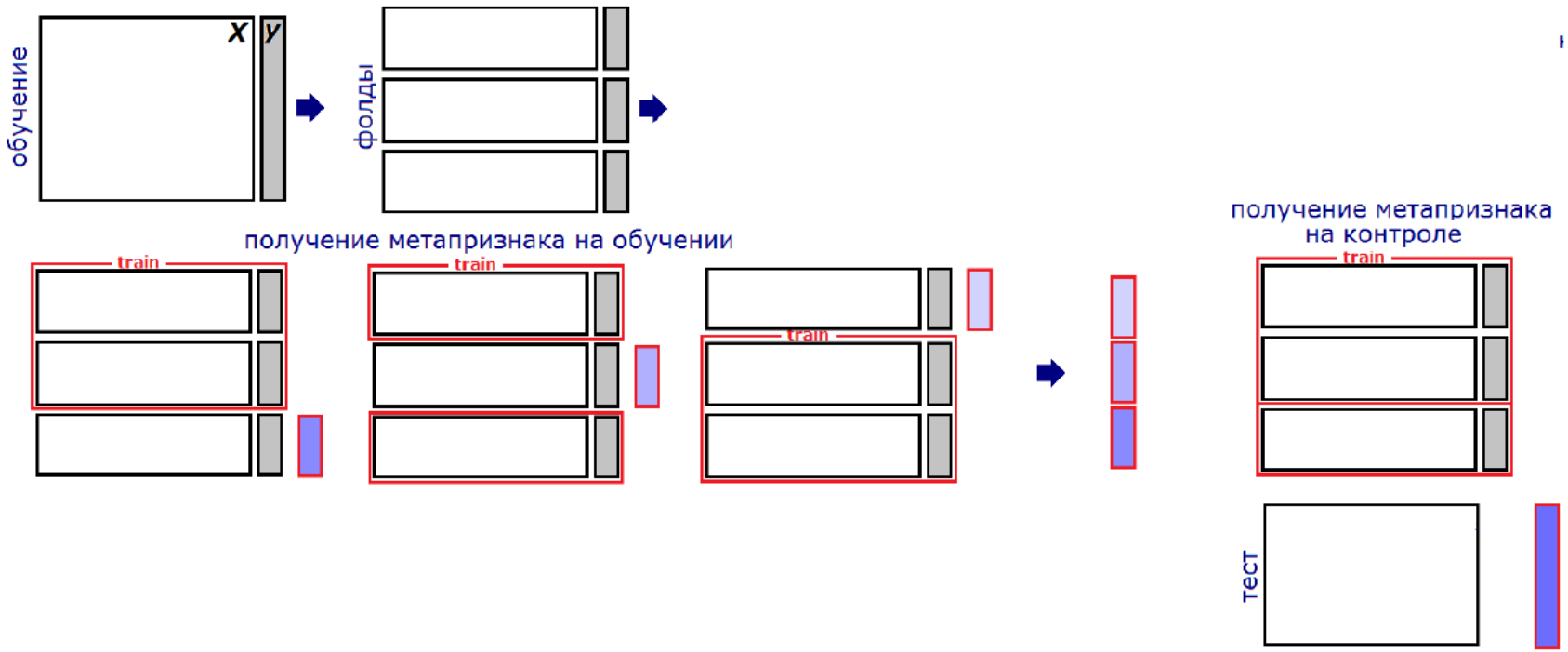
Сейчас блендингом называют в основном простейшие формы стекинга

Сама по себе идея не новая и придумана давным-давно

Недостаток: используется не вся обучающая выборка
можно «состыковать» несколько блендингов или перейти к стекингу

Стекинг

Новая проблема: для обучения используется не вся выборка
Решение: Выборка разбивается на k-блоков и получается классический стекинг



Нелинейные методы ансамблирования

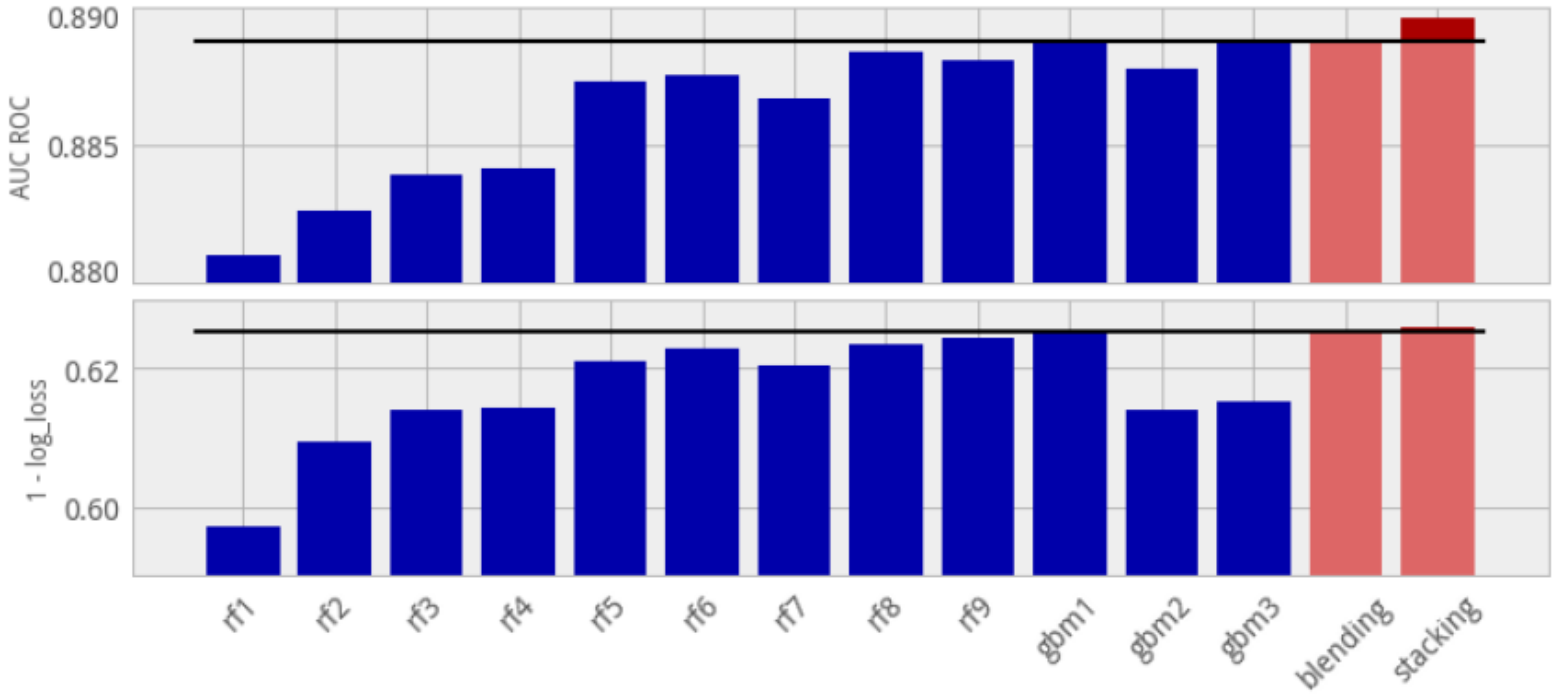
Новые проблемы:

- вместо одного мета-признака у нас получилось k похожих варианты решения: усреднение мета-признаков
- метапризнаки на обучении и тесте разные
  варианты решения: регуляризация или добавить шум к мета-признакам

Также можно брать разные разбиения и усреднять

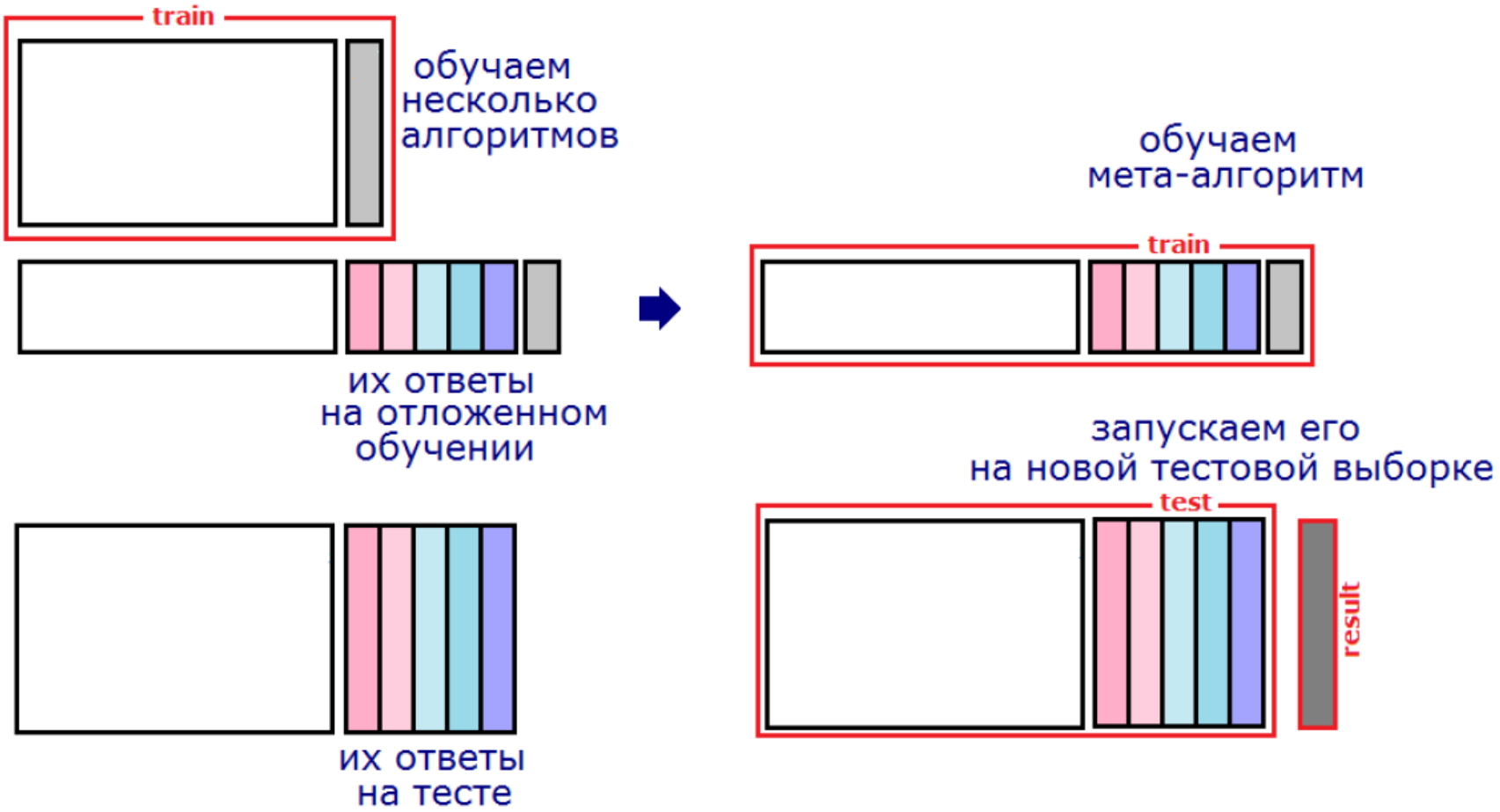
Вместе с мета-признаками можно использовать исходные признаки

Стекинг



На данных реальной задачи ML Boot Camp

Использование признаков с мета-признаками



Стекинг

- нужны достаточно большие выборки
- можно работать с алгоритмами разной природы
- хорошо на практике (бизнес-задачи)
  Пример: регрессоры + RF = устойчивость к аномальным значениям признаков
- метаалгоритм должен минимизировать целевую функцию задачи
- многоуровневый стекинг (как правило не реализуем в бизнес-приложениях)
- пространство мета-признаков удобнее исходного, но они сильно коррелированы
- стекинг не столько повышает точность, сколько придаст устойчивость ансамблю

Нелинейные методы ансамблирования

Нет хорошей теории на тему пространства метапризнаков

Практика стекинга

- используют, как правило, регрессоры, базовые алгоритмы не сильно оптимизируют
- настраиваются не на целевой признак (на его квадрат, на разницу между каким-то признаком и целевым)
- используют модели ориентированные на разные функционалы качества
- пополняют множество базовых алгоритмов алгоритмами, которые решают другую задачу (например, кластеризаторами)
- появляются дополнительные параметры: количество фолдов, уровень шума

Итог

Градиентный бустинг

https://alexanderdyakonov.files.wordpress.com/2017/06/book_boosting_pdf.pdf

Визуализация GBoost

http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html

Некоторые библиотеки

ML-Ensemble http://ml-ensemble.com/

mlxtend http://rasbt.github.io/mlxtend/

H2O Stacked Ensembles

Стекинг

https://dyakonov.org/2017/03/10/

https://arxiv.org/pdf/0911.0460.pdf

Github, поиск: stacking