In [1]:
```
import IPython.display
IPython.display.display_latex(IPython.display.Latex(filename="../macros.tex"))
```

Ансамбли

Ансамбли моделей - алгоритмы машинного обучения которые тренируют набор моделей и ответ дают как комбинацию предсказаний этих моделей.

Bagging (Bootstrap aggregation)

Идея натренировать несколько независимых класификаторов, и ответ давать как их *голосование*.

Теорема Кондорсе «о жюри присяжных» (1784) Если каждый член жюри присяжных имеет независимое мнение, и если вероятность правильного решения члена жюри больше 0.5, то тогда вероятность правильного решения присяжных в целом возрастает с увеличением количества членов жюри, и стремиться к единице. Если же вероятность быть правым у каждого из членов жюри меньше 0.5, то вероятность принятия правильного решения присяжными в целом монотонно уменьшается и стремится к нулю с увеличением количества присяжных

- N - число членов жюри
- p - вероятность правильного решения одного члена жюри
- μ - вероятность правильного решения жюри
- $$\mu = \sum_{i=m}^{N} C_N^i * p^i * (1-p)^{N-i}$$

- Если $p > 0.5$, то $\mu > p$.
- Если $N \to \infty$, то $\mu \to 1$.

Давайте рассмотрим ещё один пример ансамблей — "Мудрость толпы". Фрэнсис Гальтон в 1906 году посетил рынок, где проводилась некая лотерея для крестьян.

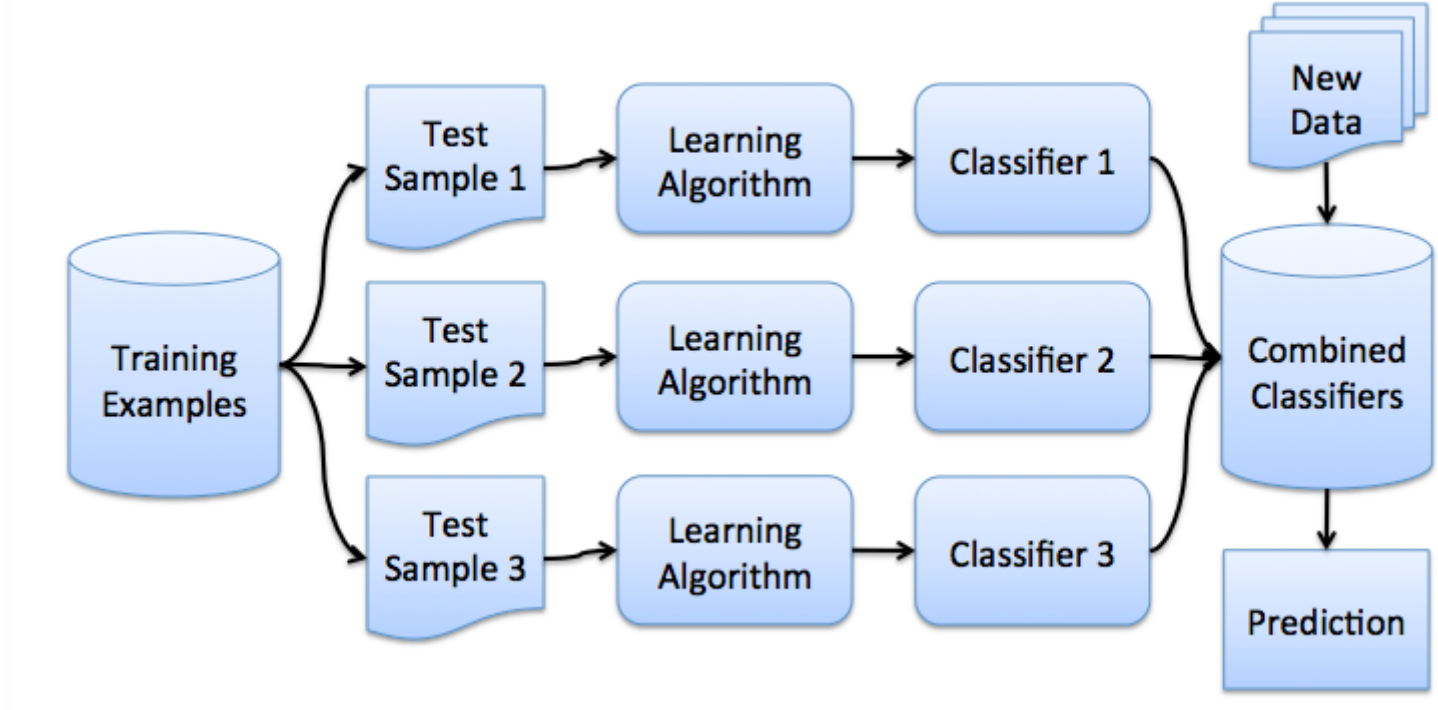
Их собралось около 800 человек, и они пытались угадать вес быка, который стоял перед ними. Бык весил 1198 фунтов. Ни один крестьянин н не дал более-менее точное число, но если посчитать среднее от их предсказаний, то получим 1197 фунтов.

Дано: выборка X.

Генерируем m подвыборок $X_1, X_2, \ldots, X_m$. На каждой подвыборке учим модель $\alpha_i(x)$.

Baging model:

$$\alpha(x) = \frac{1}{m} \sum_{i=1}^{m} \alpha_i(x)$$



Если ошибки базисных алгоритмов не смещены и нескоррелированны, то усреднение ответов позволяет уменьшить средний квадрат ошибки в m раз.

Бэггинг позволяет снизить дисперсию (variance) обучаемого классификатора.

- ошибки взаимно компенсируются
- объекты выбросы не попадают в некоторые обучающие подвыборки

Бэггинг на подпространствах

Идея

Делим пространство фич на подпространства. На каждом подпространстве обучаем классификатор.

Bagging

- уменьшает ошибку когда дисперсия ошибки базовых алгоритмов большая. Дает хороший результат при "слабых" базовых классификаторах.
- параллельное обучение алгоритмов
- не очень сложен в разработке
- сложно интерпретируем

decision trees критерий качества:

$$D = \frac{1}{l}\sum_{i=1}^{l}\left(y_i - \frac{1}{l}\sum_{i=1}^{l} y_i\right)^2$$

- l - число объектов в листе Минимизируем дисперсию вокруг среднего, ищем признаки, разбивающие выборку таким образом, что значения целевого признака в каждом листе примерно равны.

Random forest (**Случайный Лес**)

Алгоритм на m деревьев:

дано: выборка X размера N

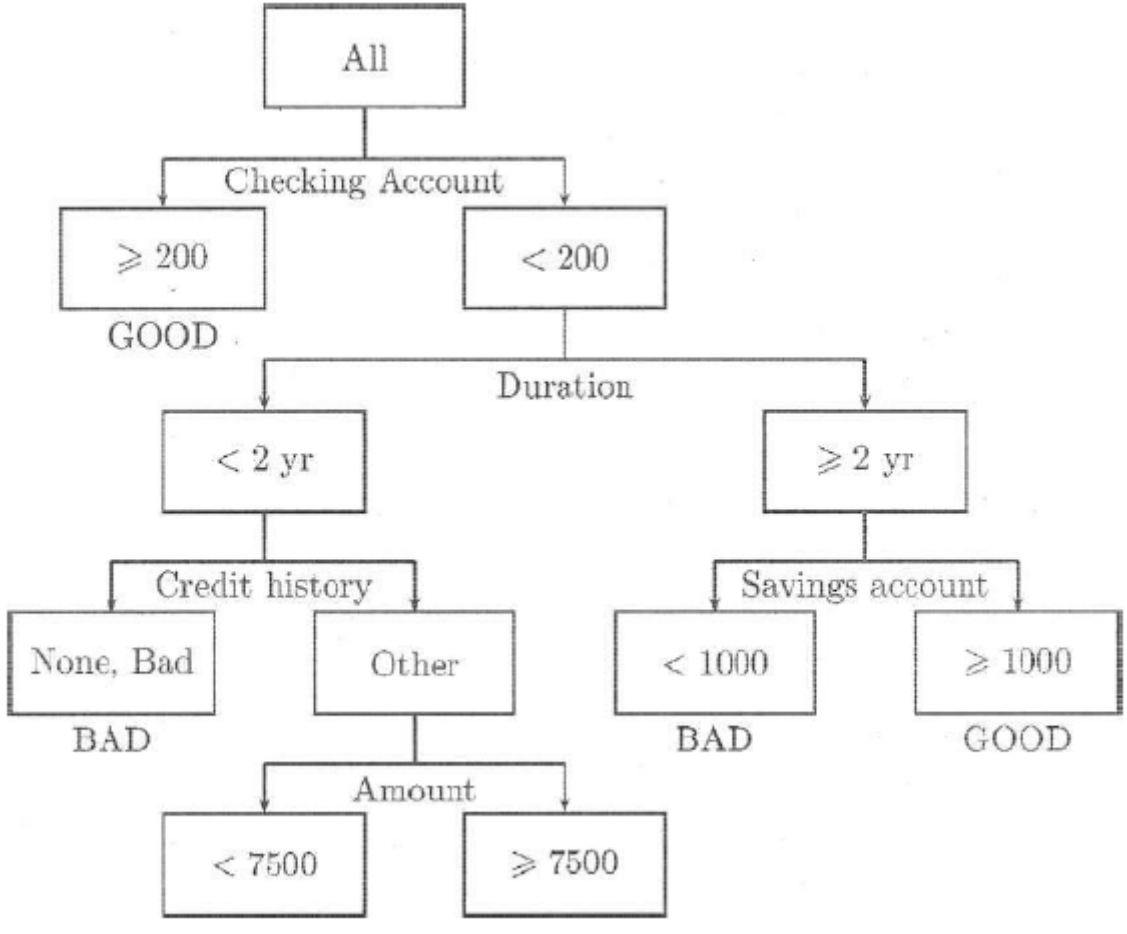
- повторяем m раз:
  - генерируем подвыборку
  - тренируем дерево на подвыборке, где $max features = n$
- result of prediction is average of predictions of all trees

Random forest - bagging (улучшенный) на деревьях.

Recommendations:

- for classification $n = \sqrt{M}$, M - number of features
- for regression $n = \frac{M}{3}$
- min_samples_leaf is 1 for classification 5 for regression

важность признаков



Чем больше уменьшается точность предсказаний из-за исключения (или перестановки) одной переменной

Advantages RF

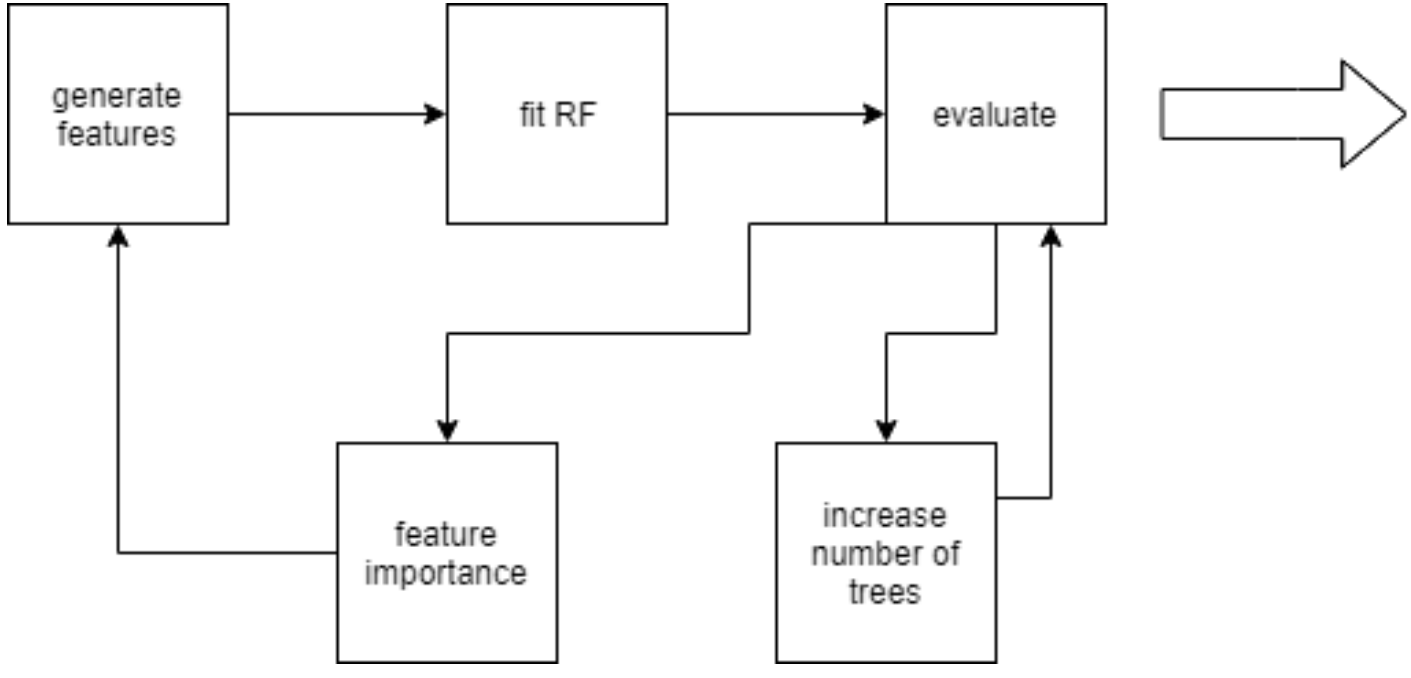
- хорошее качество предсказаний
- дает важность фич
- выбросы не проблема, так как выбираются случайные подвыборки
- нет необходимости нормализовать данные
- не так много не обучаемых параметров
- хорошо работает с большим числом фич и объектов
- хорошо работает с разными типами признаков(непрерывные дискретные)
- редко переобучение, на практике добавление деревьев только улучшает результат
- может работать с пропусками в данных
- можно балансировать обучающую выборку
- параллельно обучать алгоритмы

disadvantages RF

- не интерпретируем
- не очень хорошо работает с разряженными данными (линейные классификаторы)
- память для хранения модели

One of the best algorithm to build *baseline*.

- no difficult params for fixing
- suitable for a lot of tasks (classification, regression)
- it gives feature importance
- it gives a good result
- it gives an understanding of the level of solution (error)
- increase in the number of trees improves the result


generate features
fit RF
evaluate
feature importance
increase number of trees

boosting

Итерационный алгоритм, реализующий “сильный” классификатор, который позволяет добиться произвольно малой ошибки обучения (на обучающей выборке) на основе композиции “слабых” классификаторов, каждый из которых лучше, чем просто угадывание, т.е. вероятность правильной классификации больше 0.5.

Бустинг - обучающая выборка на каждой итерации определяется, исходя из ошибок классификации на предыдущих итерациях.

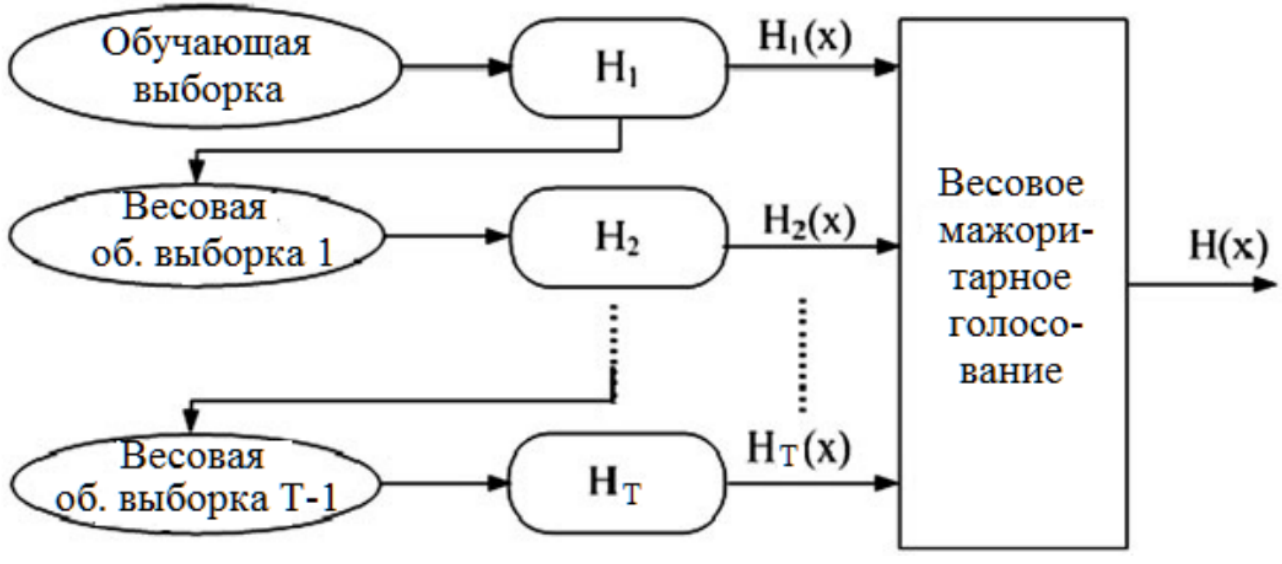
Особенности бустинга

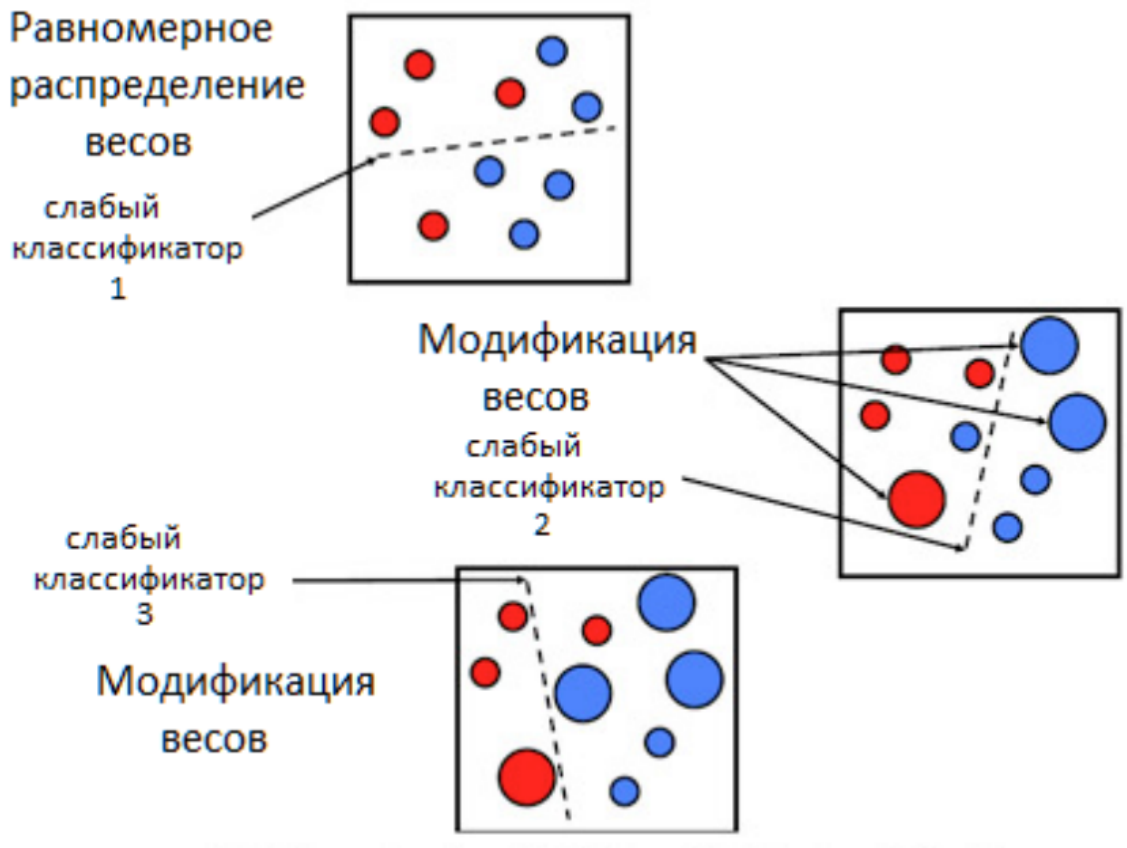
- **Ключевая идея**: использование весовой версии одних и тех же обучающих данных вместо случайного выбора их подмножества.
- Слабые классификаторы образуются последовательно, различаясь только весами обучающих данных, которые зависят от точности предыдущих классификаторов.

- Большие веса назначаются “плохим” примерам, что позволяет на каждой итерации сосредоточиться на примерах, неправильно классифицированных.

- Базовые классификаторы должны быть слабыми, из сильных хорошую композицию не построить

- Причины этого:
  - сильный классификатор, давая нулевую ошибку на обучающих данных, не адаптируется и композиция будет состоять из одного классификатора
  - один, даже сильный, классификатор может дать “плохое” предсказание на данных тестирования, давая “хорошие” результаты на на обучающих данных.


Обучающая выборка
H_1
$H_1(x)$
Весовая об. выборка 1
H_2
$H_2(x)$
Весовая об. выборка T-1
H_T
$H_T(x)$
Весовое мажори-тарное голосо-вание
H(x)



$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$

Как классифицировать с весами?

$$Q(\alpha, W) = \sum_{i=1}^{l} w_i * L(\alpha(x_i), y_i) = \sum_{i=1}^{l} w_i * [\alpha(x_i) = -y_i]$$

Общий вид бустинга

$Y = \{-1, 1\}$

$$\alpha(x) = sign(F(\alpha_1(x), \alpha_2(x), \ldots, \alpha_T(x))) = sign\Big(\sum_{t=1}^{T} \beta_t * \alpha_t(x)\Big)$$

$$Q_T = \sum_{i=1}^{l} \Big[y_i * \sum_{t=1}^{T} \beta_t * \alpha_t(x) < 0\Big]$$

Пороговая функция потерь в функционале Q_T аппроксимируется (заменяется) непрерывно дифференцируемой оценкой сверху

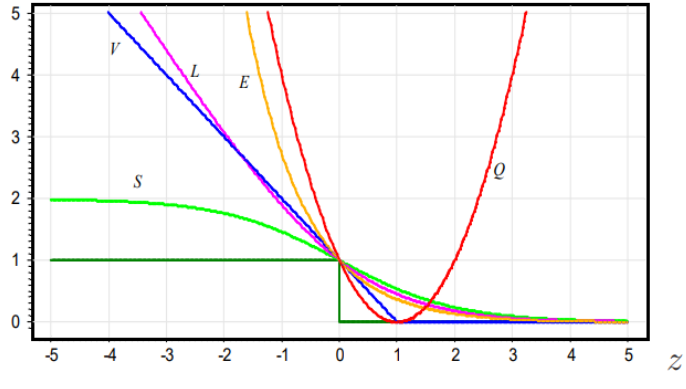


Рис. 1. Гладкие верхние аппроксимации пороговой функции потерь $[z < 0]$:
$S(z) = 2(1 + e^z)^{-1}$ — сигмоидная;
$L(z) = \log_2(1+e^{-z})$ — логарифмическая;
$V(z) = (1 - z)_+$ — кусочно-линейная;
$E(z) = e^{-z}$ — экспоненциальная;
$Q(z) = (1 - z)^2$ — квадратичная.

AdaBoost

we have:binary classification, sample X size N

initialize weights $W = \left\{\frac{1}{N}\right\}^N$ Repeat:

$$\alpha_t = argmin_\alpha Q(\alpha, W)$$

$$\beta_t = \frac{1}{2} ln \frac{1 - Q(\alpha, W)}{Q(\alpha, W)}$$

$$w_i := w_i exp\big(-\beta_t y_i \alpha_t(x_i)\big) for\ all\ i = 1...N$$

$$normalize\ weights$$

$\alpha_{adaboost}(x) = sign(\sum_i \alpha_i(x)\beta_i)$

После нескольких шагов можно проанализировать веса объектов. Объекты с очень большими весами будут выбросами, и можно их удалить. Вообще, бустинг можно использовать как универсальный метод фильтрации выбросов перед применением любого другого метода классификации

Регрессия

AdaBoost.RT(Идея)

вводим порог τ для относительной ошибки:

$$e_i = |f(x_i) - y_i|/y_i$$

если ошибка больше порога, то вес увеличиваем.

Решающее правило это сумма.

$W = \{\frac{1}{N}\}^N$ Repeat:

- строим регрессию
$$\alpha_t$$
используя веса W
- $$e_i(t) = |f_t(x_i) - y_i|/y_i$$
относительная ошибка для каждого примера
- считаем общую ошибку с учетом порога ϵ_t
- $$\beta_t = ln\frac{1-\epsilon_t}{\epsilon_t}$$
- $w_i := w_i * exp\big(-\beta_t(1-e_i(t))\big)\ where\ e_i(t) \leq \tau$
else
$$w_i := w_i$$

$$normalize\ weights$$

$\alpha_{adaboost}(x) = \sum_i \alpha_i(x)\beta_i$

AdaBoost

- Хорошая обобщающая способность. В реальных задачах (не всегда, но часто) удаётся строить композиции, превосходящие по качеству базовые алгоритмы. Обобщающая способность может улучшаться (в некоторых задачах) по мере увеличения числа базовых алгоритмов.
- Простота реализации.
- Накладные расходы бустинга невелики. Время построения композиции практически полностью определяется временем обучения базовых алгоритмов.
- Возможность идентифицировать выбросы. Это наиболее «трудные» объекты xi, для которых в процессе наращивания композиции веса wi принимают наибольшиезначения.

- AdaBoost склонен к переобучению при наличии значительного уровня шума в данных. Экспоненциальная функция потерь слишком сильно увеличивает веса «наиболее трудных» объектов, на которых ошибаются многие базовые алгоритмы. Однако именно эти объекты чаще всего оказываются шумовыми выбросами. Проблема решается путём удаления выбросов или применения менее «агрессивных» функций потерь. В частности, алгоритм LogitBoost использует логистическую функцию.
- AdaBoost требует достаточно длинных обучающих выборок. Например, бэггинг, зачастую способен строить алгоритмы сопоставимого качества по меньшим выборкам данных.
- Бустинг может приводить к построению громоздких композиций, состоящих из сотен алгоритмов.
    - не интерпретируемо,
    - требуют больших объёмов памяти для хранения базовых алгоритмов и существенных затрат времени на вычисление классификаций.

Gradient boosting

Идея каждый следующий алгоритм обучать на расхождении с реальными ответами, двигаться в сторону уменьшения эмперического риска.

let it be $h(x, \theta)$ - based algorithms.

$$\alpha_R(x) = \sum_{i=1}^{R} \beta_i h(x, \theta)$$

$$Q = \sum_{i=1}^{N} L(\alpha, y_i)$$

boosting step for $r = 1..R$

$$\nabla Q = \Big[\frac{\partial L(\alpha_{r-1}, y_i)}{\partial \alpha_{r-1}} (x_i) \Big]_{i=1}^{N}$$

$$\theta_r = learn(X, \nabla Q)$$

$$\beta_r = argmin_\beta \sum_{i=1}^{N} \big(L(\alpha_{r-1}(x_i) + \beta * h(x_i, \theta_r)), y_i \big)$$

$$\alpha_r(x) = \alpha_{r-1}(x) + \beta_r * h(x, \theta_r)$$

SGB(Стохастический градиентный бустинг) - На каждом шаге алгоритма новое слагаемое считается опираясь не на всю обучающую выборку, а лишь на случайную подвыборку фиксированного размера.

Все дело в том какую лосс функцию мы возьмем.

TreeBoost

базовые алгоритмы - деревья. $H = \{h_1, \dots\}$

$h(x, \{\theta_j, R_j\}_{j=1}^J) = \sum_j \theta_j [x \in R_j]$ - дерево с J вершинами

$$F_m(x) = F_{m-1}(x) + \beta_m * \sum_j \theta_j [x \in R_j] =$$

$$F_{m-1}(x) + \sum_j c_{m,j} [x \in R_j] \text{ где } c_{m,j} = \beta_m * \theta_j$$

То есть просто добавив дерево с другими метками без коэффициента.

$$c_{m,j} = argmin_c \sum_{i=1}^N L(y_i, F_{m-1}(x) + \sum_j c_{m,j} [x \in R_j])$$

Но R_j попарно не пересекаются, и значит

$$cc_{m,j} = argmin_c \sum_{x_i \in R_{j,m}} L(y_i, F_{m-1}(x_i) + c)$$

Stacking

Идея использовать предсказания алгорима как фичи для другого алгоритма.

Blending

sample X divided into 2 part:

$$X_1 \cap X_2 = \emptyset, X_1 \cup X_2 = X$$

$\alpha_1, \alpha_2, \ldots \alpha_n$ fit on (X_1, Y_1)

$$\tilde{X_2} = \left[\alpha_1(X_2)|\alpha_2(X_2)|\ldots|\alpha_n(X_2)\right]$$

meta algorithm $\tilde{\alpha}$ fit on $(\tilde{X_2}, Y_2)$

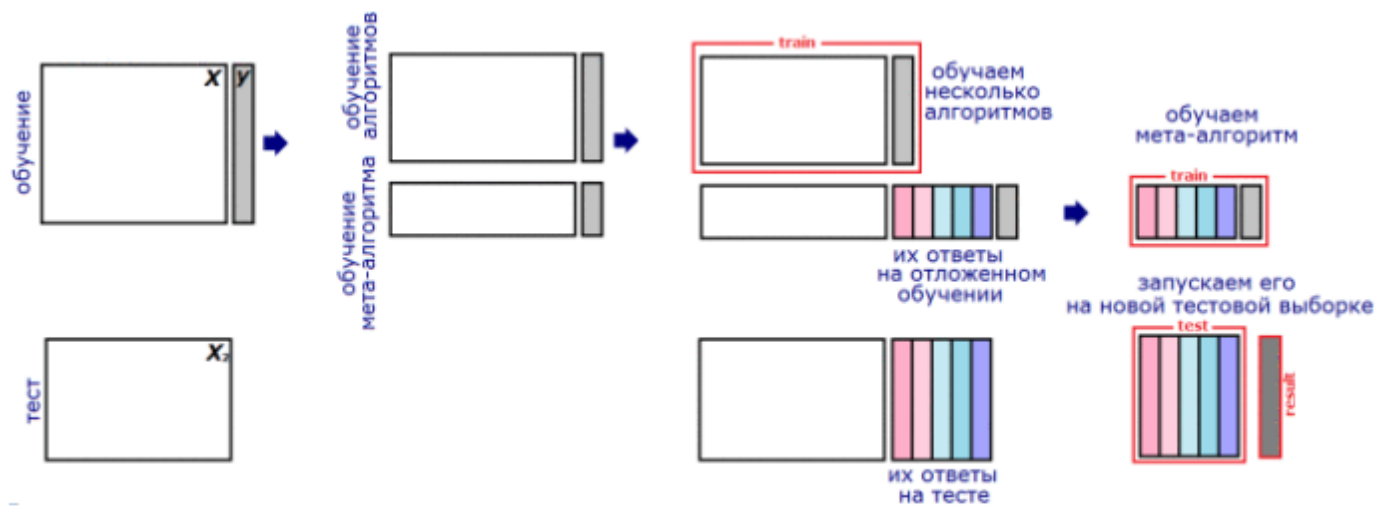


Рис.1. Схема классического блендинга.

проблема Деление обучающей выборки, не используем всего объема обучения

for $i = 1, \dots N$:

- случайно разбиваем выборку на две $X_1 \cap X_2 = \emptyset, X_1 \cup X_2 = X$
- $\alpha_i = blending(X_1, X_2)$

$\alpha = \frac{1}{N} \sum \alpha_i$

Или, например, вместо усреднения брать конкатенацию таблиц:

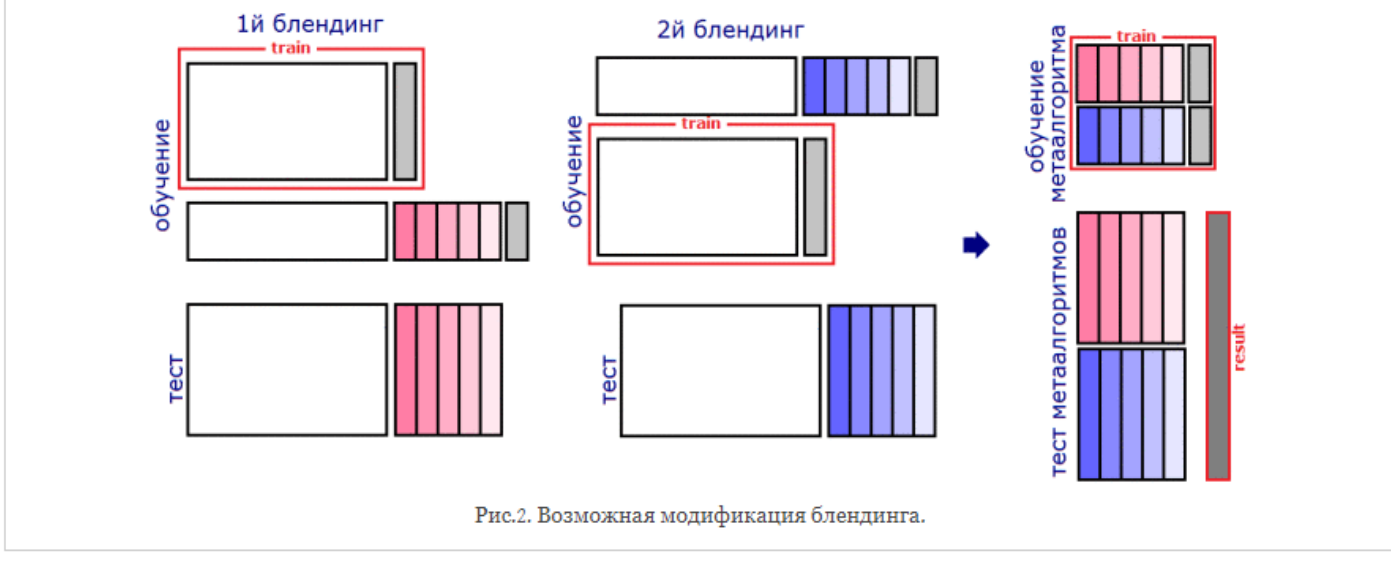


Рис.2. Возможная модификация блендинга.

Stacking

sample X divided into k fold:

$$X_1 \cap X_2 \cap \ldots \cap X_k = \emptyset, X_1 \cup X_2 \cup \ldots \cup X_k = X$$

α_1 for 1 fold: take $k-1$ other folds fit on it and predict on 1 fold.
Than for each fold do the same.$\rightarrow$ 1 meta feature

Than for each α do the same.

Meta algorithm fit as in blending.

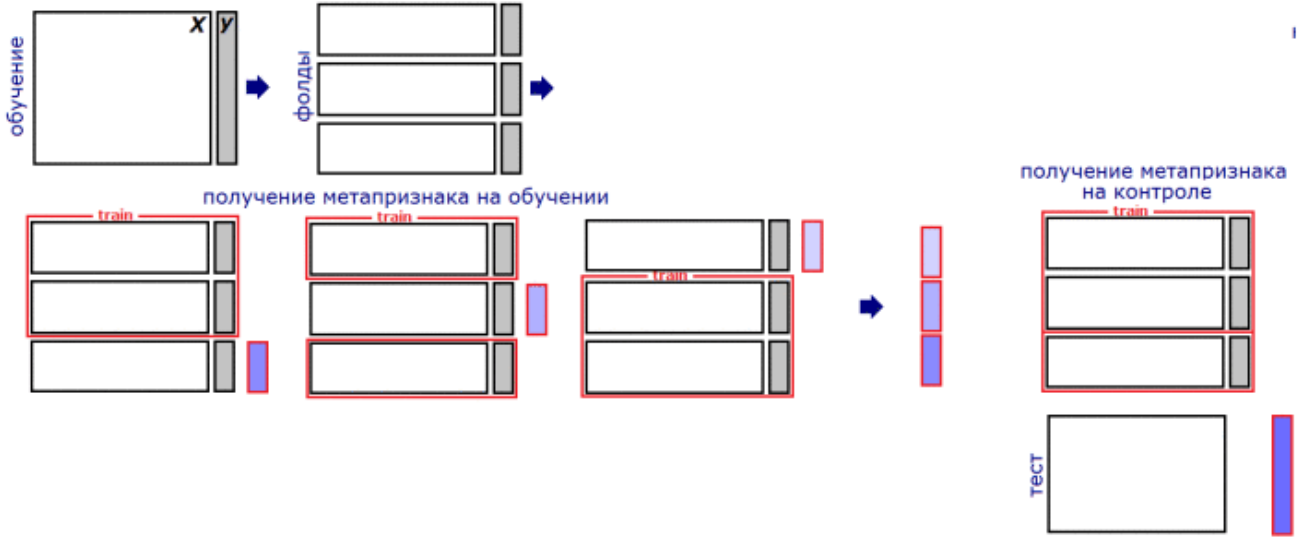


Рис.3. Получение метапризнака в классическом стекинге.

недостаток метапризнаки на обучении и на тесте разные.

Если проблема, то можно просто регуляризировать:

- соответственный параметр
- к метапризнакам добавить шум из нормального распределения

базовые алгоритмы

- хорошо использовать алгоритмы разной природы (линейные + RF)
- сложные и простые алгоритмы
- использовать разные признаковые пространства (в деревьях категориальные признаки как есть а в линейных выполнить one-hot кодировку)

метаалгоритмы

- Линейные алгоритмы (logistic regression) базовый/традиционный путь
- Не линейные алгоритмы. Нелинейные алгоритмы могут находить полезные взаимодействия между оригинальными функциями и функциями метамодели.
  - градиентный бустинг
  - KNN
  - RF
  - NN
  - ...

Идея Иногда помимо ответов базовых алгоритмов, в признаки для метаалгоритма могут добавить и исходные признаки.

- все
- некоторые
- новые (не используемые) - киллер-фичи

Деформация признаков Преобразование (деформация) метапризнакового пространства. Вместо стандартных метапризнаков можно использовать одночлены над ними (например, все попарные произведения).

метамета алгоритмы 9 категорий 93 признака. Level1: 33 модели + 8 созданных признаков — в основном полученные кластеризацией исходных данных (и сам признак — метки полученных кластеров) либо учитывающие расстояния до ближайших представителей каждого класса.