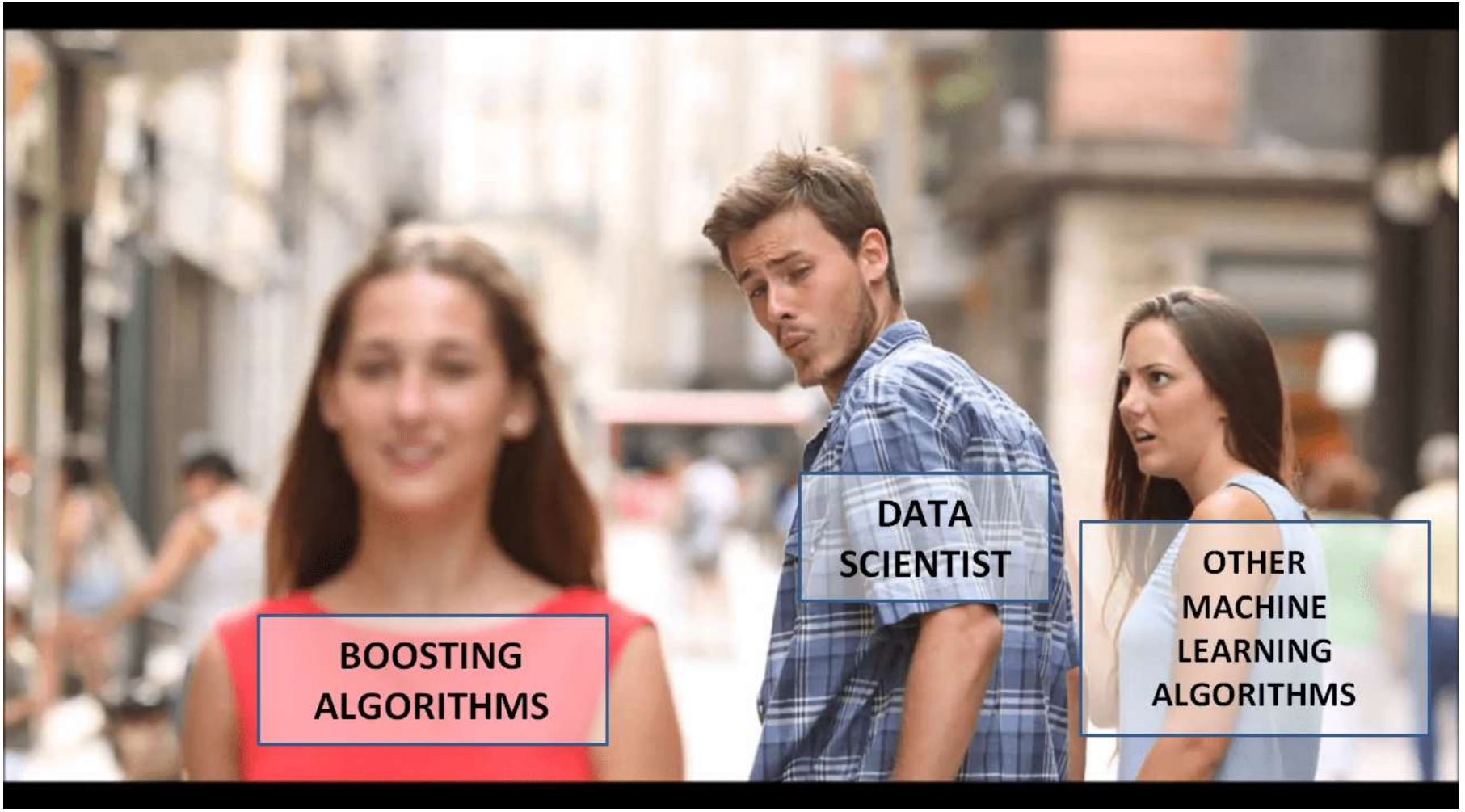
Тверской государственный технический университет

Интеллектуальные информационные системы

Развитие ансамблевых методов. Градиентный бустинг. Стекинг.

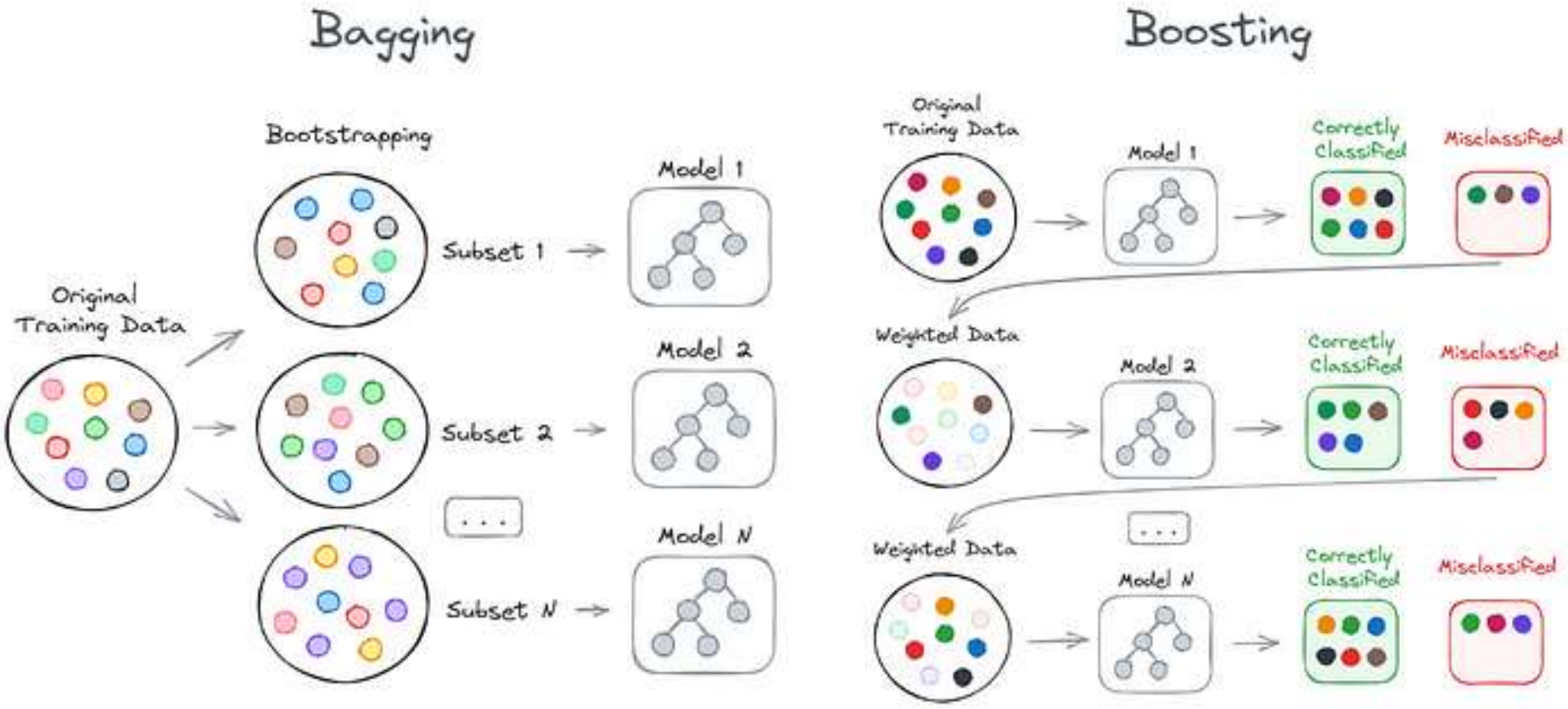
2025 г.


BOOSTING ALGORITHMS
DATA SCIENTIST
OTHER MACHINE LEARNING ALGORITHMS

Boosting

- Идея бустинга – базовые алгоритмы строятся *последовательно*, чтобы каждый следующий базовый алгоритм *уменьшал общую ошибку всех предыдущих* и повышал качество ансамбля. Итоговый ансамбль будет иметь меньшее смещение, чем отдельный базовый алгоритм, уменьшение разброса также может происходить.
- Основная цель бустинга – уменьшение смещения, поэтому в качестве базовых алгоритмов выбираем алгоритмы с *высоким смещением* и *низким разбросом*

Boosting vs Bagging



Бустинг в задачах регрессии

Задача минимизации квадратичного функционала:

$$\frac{1}{2}\sum_{i=1}^{l}(a(x_i)-y_i)^2 \to \min_a$$

Итоговый алгоритм представим как сумму базовых моделей $b_n(x)$:

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

Строим первый базовый алгоритм:

$$b_1(x) = \underset{b\in A}{argmin}\frac{1}{2}\sum_{i=1}^{l}(b(x_i)-y_i)^2$$

Оцениваем остатки на каждом объекте (расстояние от ответа нашего алгоритма до истинного ответа):

$$s_i^{(1)} = y_i - b_1(x_i)$$

Второй базовый алгоритм строим так, чтобы его ответы были как можно ближе к остаткам:

$$b_2(x) = \underset{b \in A}{argmin} \frac{1}{2} \sum_{i=1}^{l} \left(b(x_i) - s_i^{(1)} \right)^2$$

Повторяем для каждого последующего алгоритма:

$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i), \qquad i = 1, \dots, l$$

$$b_N(x) = \underset{b \in A}{argmin} \frac{1}{2} \sum_{i=1}^{l} \left(b(x_i) - s_i^{(N)} \right)^2$$

Остатки $s_i^{(N)}$ можно оценить как антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенного ансамбля:

$$s_i^{(N)} = y_i - a_{N-1}(x_i) = -\frac{\partial}{\partial z} \frac{1}{2} (z - y_i)^2, \quad z = a_{N-1}(x_i)$$

Выбирается базовый алгоритм, который как можно сильнее уменьшит ошибку ансамбля

Некоторые алгоритмы реализации бустинга

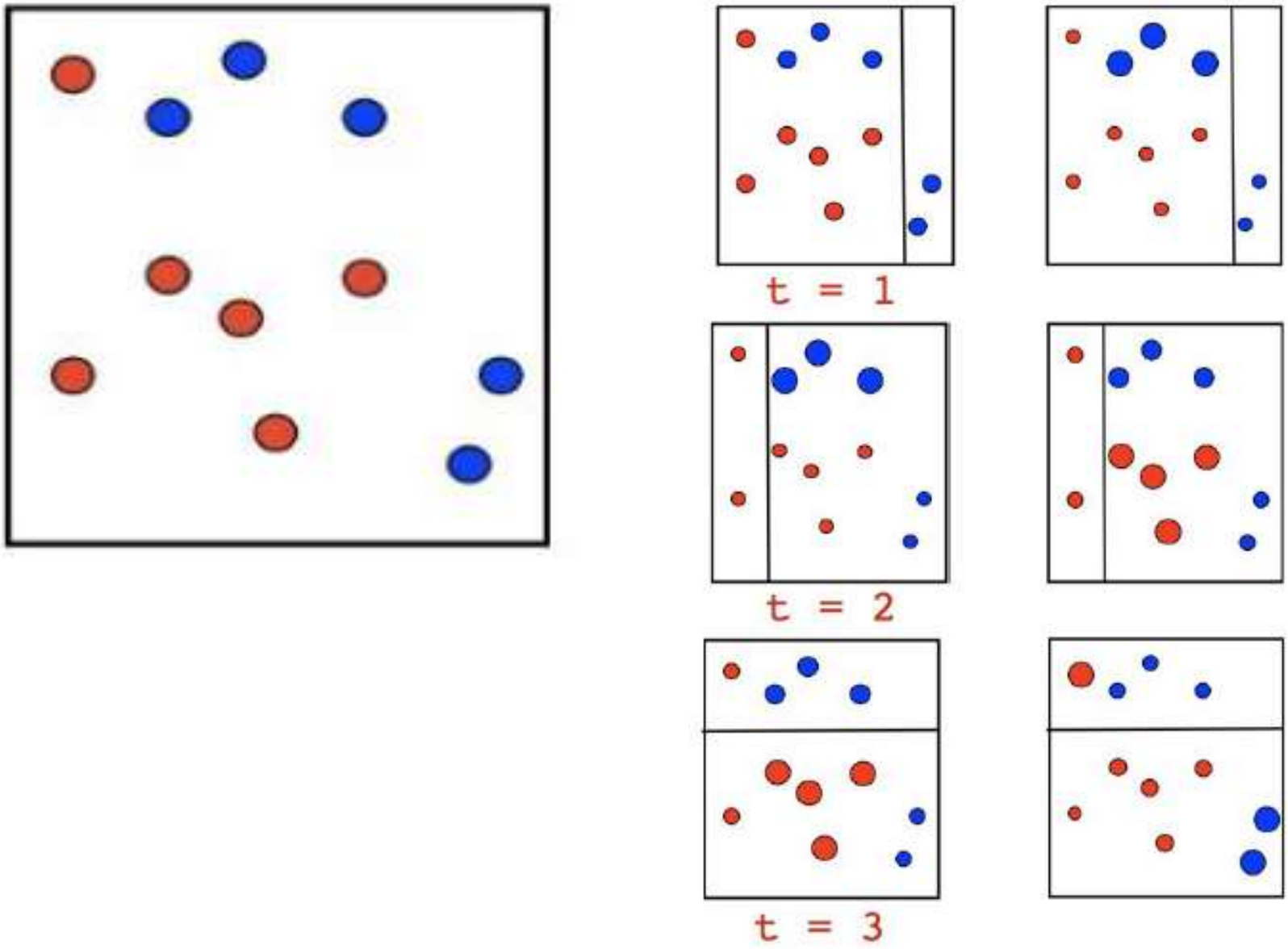
- AdaBoost (adaptive boosting)
- BrownBoost
- LogitBoost
- …

1999 г. – Yoav Freund, Robert Schapire – A short introduction to Boosting – обосновали алгоритм бустинга AdaBoost

AdaBoost – жадное построение линейной комбинации базовых алгоритмов путем перевзвешивания входных данных. Каждый последующий базовый алгоритм строится таким образом, чтобы придавать больший вес объектам, на которых была ошибка

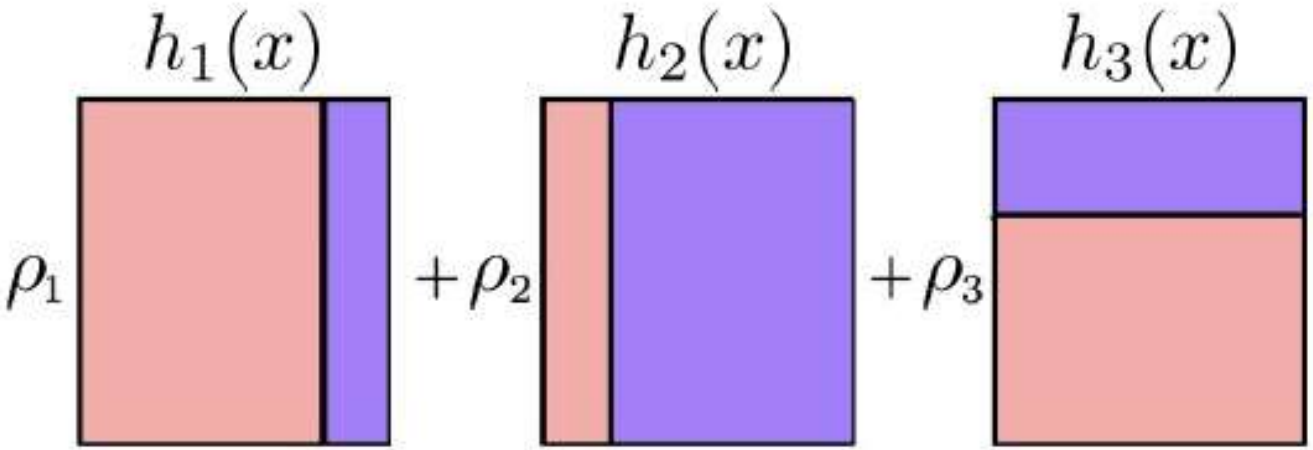
Источник: https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

Визуализация на примере бинарной классификации с использованием решающих деревьев глубиной равной 1

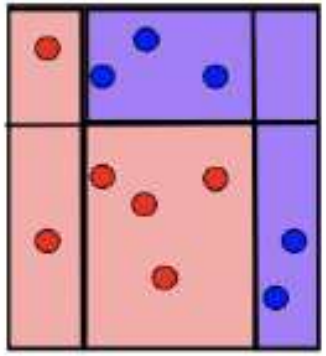


Источник: https://habr.com/ru/companies/ods/articles/327250/

Взвешенное голосование построенных базовых алгоритмов



$$\hat{f}_T(x) = \sum_{t=1}^{T} \rho_t h_t(x) \quad =$$

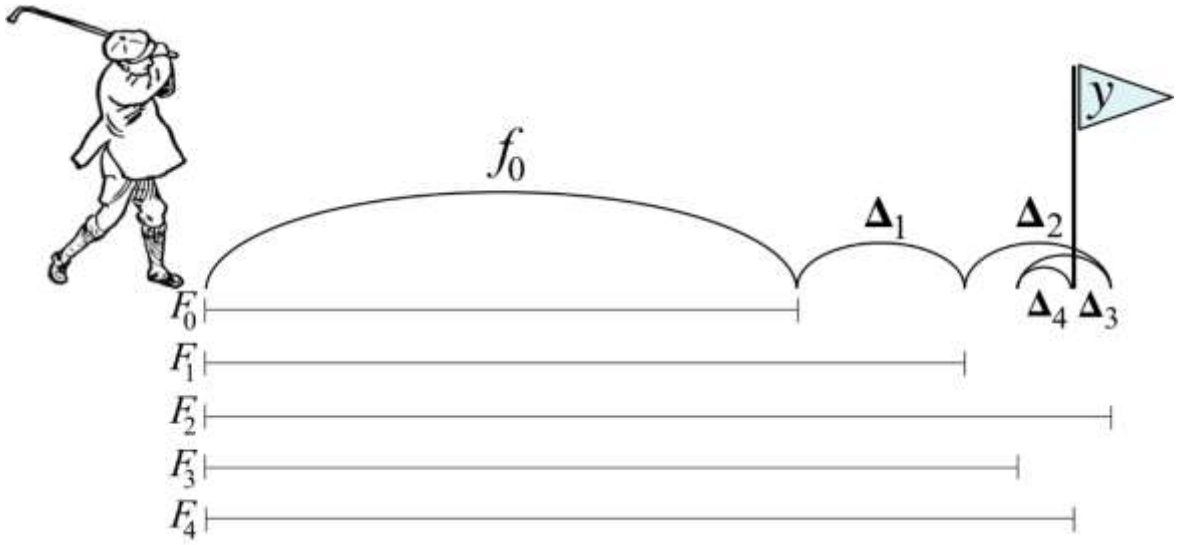


Источник: https://www.youtube.com/watch?v=k4G2VCuOMMg

1999 REITZ LECTURE

GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE[1]

BY JEROME H. FRIEDMAN



$$\begin{aligned}\hat{y} &= f_0(\mathbf{x}) + \Delta_1(\mathbf{x}) + \Delta_2(\mathbf{x}) + \ldots + \Delta_M(\mathbf{x}) \\ &= f_0(\mathbf{x}) + \sum_{m=1}^{M} \Delta_m(\mathbf{x}) \\ &= F_M(\mathbf{x})\end{aligned}$$

Stage m	Boosted Model	Model Output $\hat{y}$	Train Δ_m on $y - F_{m-1}$	Noisy Prediction Δ_m
0	F_0	70		
1	$F_1 = F_0 + \Delta_1$	70+15=85	100-70=30	$\Delta_1 = 15$
2	$F_2 = F_1 + \Delta_2$	85+20=105	100-85=15	$\Delta_2 = 20$
3	$F_3 = F_2 + \Delta_3$	105-10=95	100-105=**-5**	$\Delta_3 =$ **-10**
4	$F_4 = F_3 + \Delta_4$	95+5=100	100-95=5	$\Delta_4 = 5$

Источник: https://explained.ai/gradient-boosting/L2-loss.html

Градиентный бустинг

Дана некоторая дифференцируемая функция потерь L(y,z), строим взвешенную сумму базовых алгоритмов:

$$\boldsymbol{a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)}$$

Инициализация нулевого (начального) базового алгоритма $b_0(x)$:

- $\gamma_0 = 1$
- Для классификации – самый популярный класс

$$b_0(x) = argmax \sum_{i=1}^{l} [y_i = y]$$

- Для регрессии – выборочное среднее

$$b_0(x) = \frac{1}{l} \sum_{i=1}^{l} y_i$$

Следующий базовый алгоритм строим так, чтобы как можно *сильнее уменьшать ошибку:*

$$\sum_{i=1}^{l} L\big(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)\big) \to \min_{\gamma_N, b_N}$$

$$\searrow$$

$$= s_i$$

Как выбрать значения $s_1, \dots, s_l$?

Если выбираем $s_i = y_i - a_{N-1}(x_i)$ то никак не учитываем особенности функции потерь, требуем точного совпадения предсказаний и истинных ответов.

Лучше требовать, чтобы сдвиг s_i был противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$:

$$s_i = -\frac{\partial L}{\partial z}$$

Вектор сдвигов $s = (s_1, \dots, s_l)$ совпадает с антиградиентом, т.е. сдвигаемся в сторону наискорейшего убывания функции потерь

По сути, выполняем один шаг градиентного спуска в

ℓ-мерном пространстве предсказаний алгоритма на объектах обучающей выборки

Приближаем градиент функции потерь на обучающей выборке с помощью среднеквадратичной ошибки:

$$b_N(x) = argmin \sum_{i=1}^{l} (b(x_i) - s_i)^2$$

Оптимизируем квадратичную функцию потерь независимо от функционала исходной задачи, вся информация о функции потерь L находится в антиградиенте s_i, на данном шаге лишь решается задача аппроксимации функции по ℓ точкам. Допускается использовать и другие функционалы

После получения базового алгоритма b_N, вес для него подбирается по аналогии с наискорейшим градиентным спуском:

$$\gamma_n = argmin \sum_{i=1}^{l} L(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

Аппроксимация антиградиента базовыми алгоритмами – поэтому бустинг градиентный. Данный метод представляет собой поиск лучшей функции, восстанавливающей истинную зависимость ответов от объектов, в пространстве всех возможных функций. Ищем функцию с помощью «псевдоградиентного» спуска – каждый шаг делается вдоль направления, задаваемого некоторым базовым алгоритмом. Сам базовый алгоритм выбирается так, чтобы как можно лучше приближать антиградиент ошибки на обучающей выборке

Регуляризация в общем случае градиентного бустинга

Необходимо найти компромисс между слишком простым и слишком сложным базовым алгоритмом

Слишком простой – плохо приближает вектор антиградиента

Слишком сложный – идеально подгоняется под обучающую выборку

- Регулируем темп обучения (learning rate, eta, shrinkage):

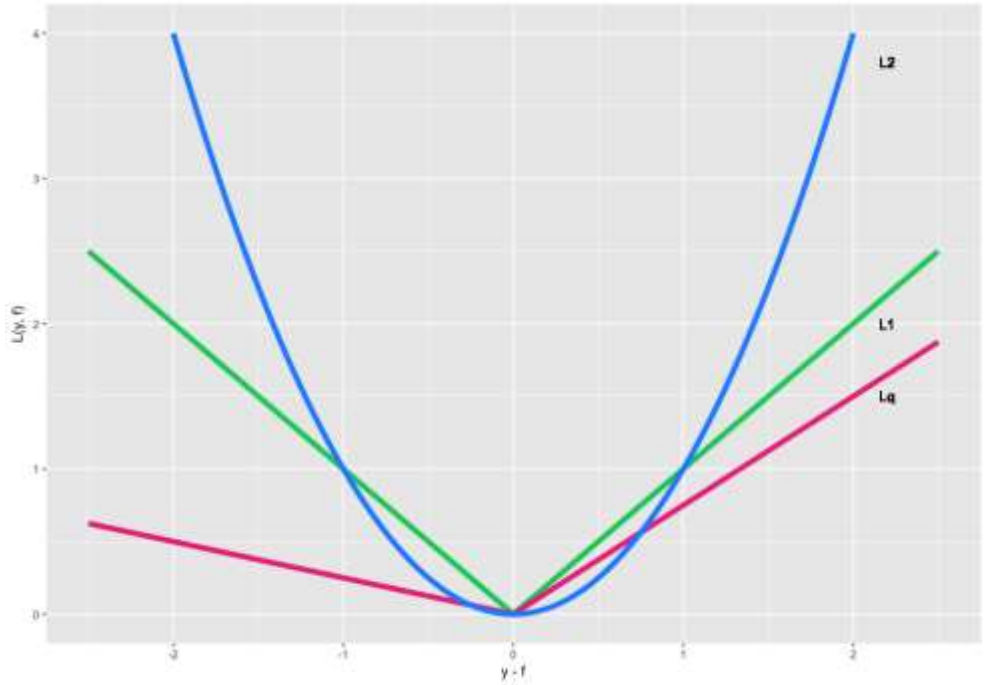
$$a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x)$$

- Ограничиваем число итераций градиентного бустинга, оцениваем качество на тесте

Функции потерь

Регрессия:

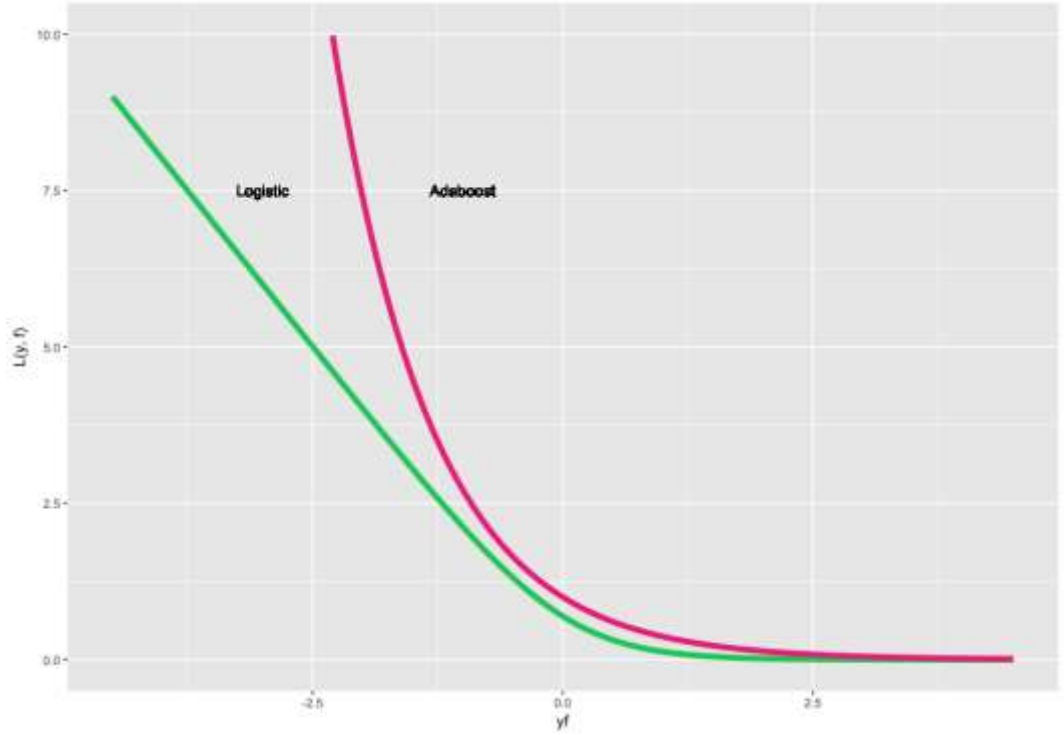
1. Квадратичная $L_2(y,z) = (y-z)^2$ - *Gaussian loss*
2. Модуль отклонения $L_1(y,z) = |y-z|$, антиградиент вычисляется как $s_i^{(N)} = -sign(a_{N-1}(x_i) - y_i)$ – *Laplacian loss*
3. *Quantile loss* $L_Q(y,z) = \begin{cases} (1-a)|y-z|, & \text{при } y-z \le 0 \\ a|y-z|, & \text{при } y-z > 0 \end{cases}$



Функции потерь

Классификация:

1. Логистическая функция потерь $L(y, z) = \log(1 + \exp(-yz)) -$ Logistic loss
2. Экспоненциальная функция потерь $L(y, z) = -ye^{-yz}$ - Adaboost loss

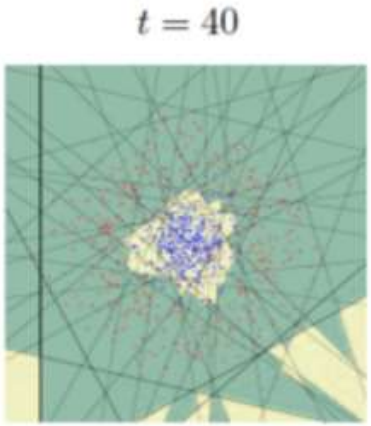


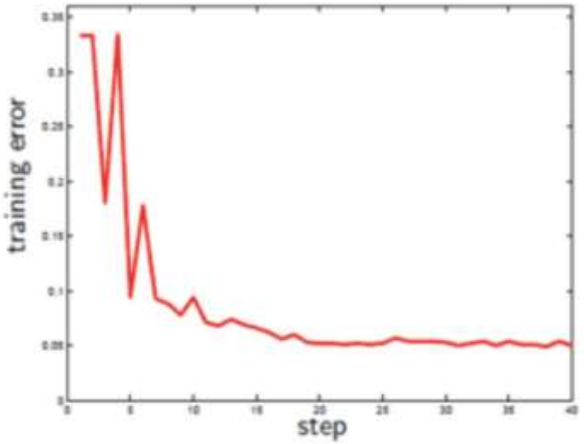
Можно обобщить подход построения бустинга на произвольную дифференцируемую функцию потерь

Какие алгоритмы можно использовать в качестве базовых в градиентном бустинге?

Вообще говоря, любые, но нужно учесть особенности:

1. Подходят под определение «слабого (weak)» алгоритма – низкий разброс
2. Сложность модели, особенности данных, время обучения
3. Можно использовать даже «самостоятельные» модели – knn, random forest и другие в качестве базовых
4. Линейные модели – ансамбль линейных моделей будет так же линейной комбинацией линейных моделей – линейной моделью. Но если использовать линейную модель с индикаторной функцией (например, логистическую регрессию), то ансамбль таких базовых алгоритмов уже не будет линейным

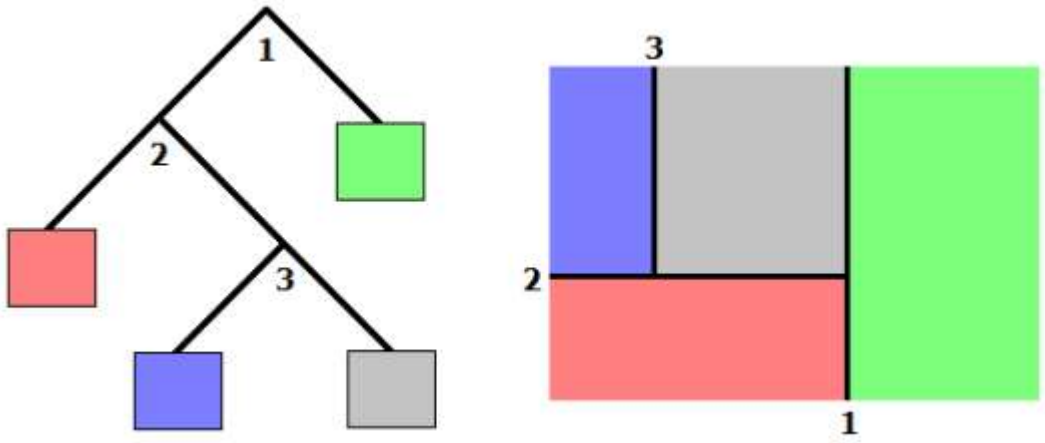




Градиентный бустинг над решающими деревьями

GBDT, gradient boosting on decision trees - один из самых универсальных и сильных методов машинного обучения

Число листьев в дереве определяет, на сколько областей дерево делит пространство объектов. Для каждой области мы подбираем свое значение алгоритма в ней, что соответствует сложности модели (level of interaction between variables). Построение дерева определяет выбор областей пространства



Источник: https://alexanderdyakonov.wordpress.com/wp-content/uploads/2017/06/book_boosting_pdf.pdf

Стохастический градиентный бустинг

Перенесение идеи бэггинга на бустинг – каждый новый алгоритм настраивать на подвыборке размером (0,1]

- Может улучшить качество ансамбля
- Уменьшает время построения базовых алгоритмов
- Позволяет оценивать out-of-bag ошибки

Размер подвыборки подбирается в процессе валидации, начать можно со значения 0.5

Основные реализации градиентного бустинга

LightGBM

- Строит деревья по принципу «на каждом шаге делим вершину с наилучшим скором», критерий остановки – максимально допустимое количество вершин в дереве. Деревья получаются несимметричными, то есть поддеревья могут иметь разную глубину. Быстро, но более подвержены переобучению
- Быстрый и нетребовательный к ресурсам (реализованы методы Gradient-based One-side Sampling – используем меньше наблюдений, Exclusive Feature Bunding – используем меньше признаков)

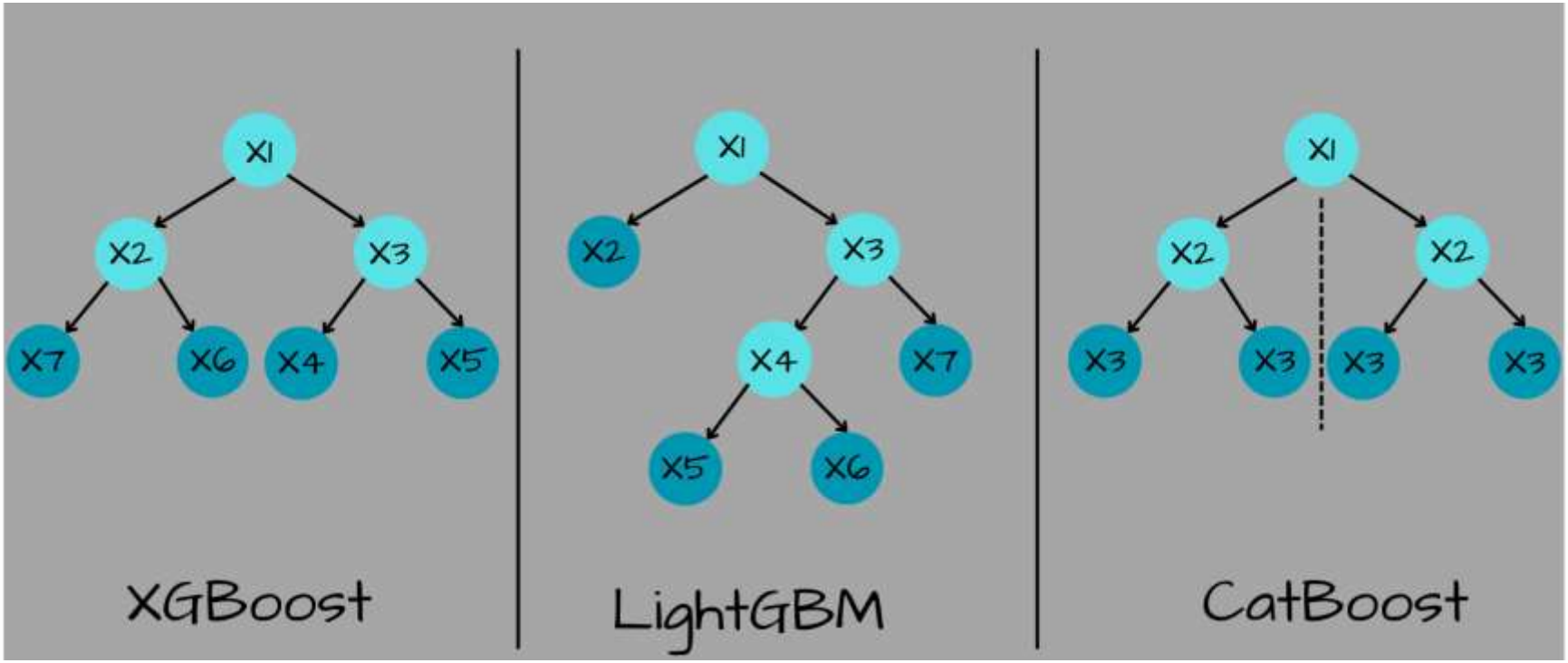
XGBoost

- Строит деревья по принципу «строим дерево последовательно по уровням до достижения максимальной глубины». Более устойчивы к переобучению
- Гибкость настройки, выбор базовых алгоритмов

Источник: https://education.yandex.ru/handbook/ml/article/gradientnyj-busting

CatBoost

- Строит деревья по принципу «все вершины имеют одинаковый предикат». За счет этого быстрый инференс. Устойчивы к переобучению
- Обработка категориальных признаков



(Гипер)Параметры градиентного бустинга

- Выбор функции потерь и метрику (*loss, eval_metric*)
- Выбор нулевого (*init*) и базового алгоритмов (*booster*)
- Выбор способа построения деревьев: порядок построения дерева (*grow policy*), критерий выбора расщепления (*criterion*)
- Темп обучения (*eta / learning rate*)
- Число итераций бустинга (*n_estimators / num_iterations*)
- Опция ранней остановки (*early_stopping_round*)

Параметры, ограничивающие сложность дерева:

1. Максимальная глубина (*max_depth*)
2. Максимальное число вершин в дереве (*max_leaves, num_leaves, max_leaf_nodes*)
3. Порог на уменьшение функции ошибки при расщеплении в дереве (gamma / min_gain_to_split)
4. Минимальное число объектов в листе (min_samples_leaf)
5. Минимальное число объектов, при котором делается расщепление (min_samples_split)
6. Порог прироста информативности для расщепления (min_impurity_split)

Параметры формирования подвыборок:

1. Доля объектов для обучения одного дерева (subsample)
2. Доля признаков для построения одного дерева (colsample_bytree)
3. Доля признаков для построения расщепления в дереве (colsample_bylevel)

Параметры регуляризации:

1. Alpha (L1)
2. Lambda (L2)

Как настраивать параметры?

1. Фиксируем число деревьев – подбираем темп обучения и другие параметры. Затем увеличиваем число деревьев и подбираем только темп обучения, оставляя остальные параметры неизменными
2. Фиксируем темп обучения, подбираем параметры – последовательно добавляем деревья в ансамбль

Дополнительные материалы:

1. Gradient Boosting Interactive Playground
2. Gradient Boosting Explained Demo

Stacking

Стекинг – алгоритм ансамблирования:

1. Использует алгоритмы разного типа, а не только из фиксированного семейства
2. Результаты базовых алгоритмов объединяются в один с помощью обучаемой мета-модели, а не с помощью какого-либо обычного способа агрегации

Обучение стекинга:

1. Train-test split
2. Train делится на k фолдов, (k-1) для обучения, 1 для получения предсказаний базовых алгоритмов (мета-признаков)

train_data

	x_0	x_1	x_2	x_3	y
fold_1	0.94	0.27	0.80	0.34	1
	0.02	0.22	0.17	0.84	0
fold_2	0.83	0.11	0.23	0.42	1
	0.74	0.26	0.03	0.41	0
fold_3	0.08	0.29	0.76	0.37	0
	0.71	0.76	0.43	0.95	1
fold_4	0.08	0.71	0.97	0.04	0
	0.84	0.97	0.89	0.05	1

Training
m1, m2, m3

train_data_new_features

pred_m1	pred_m2	pred_m3
0.96	0.86	0.66
0.03	0.13	0.30
0.90	0.90	0.85
0.12	0.10	0.11
0.03	0.09	0.08
0.77	0.50	0.67
0.19	0.65	0.20
0.91	0.89	0.90

3. На полученных мета-признаках обучается мета-модель. Кроме мета-признаков, она может принимать на вход и признаки из исходного датасета

New train_data for meta-model

pred_m1	pred_m2	pred_m3	y
0.96	0.86	0.66	1
0.03	0.13	0.30	0
0.90	0.90	0.85	1
0.12	0.10	0.11	0
0.03	0.09	0.08	0
0.77	0.50	0.67	1
0.19	0.65	0.20	0
0.91	0.89	0.90	1

Train
meta-model

Meta-model

Если данных достаточно, то можно просто разделить обучающие данные на две непересекающиеся части:

1. Для обучения базовых алгоритмов
2. На которой они делают свои предсказания и обучается мета-модель

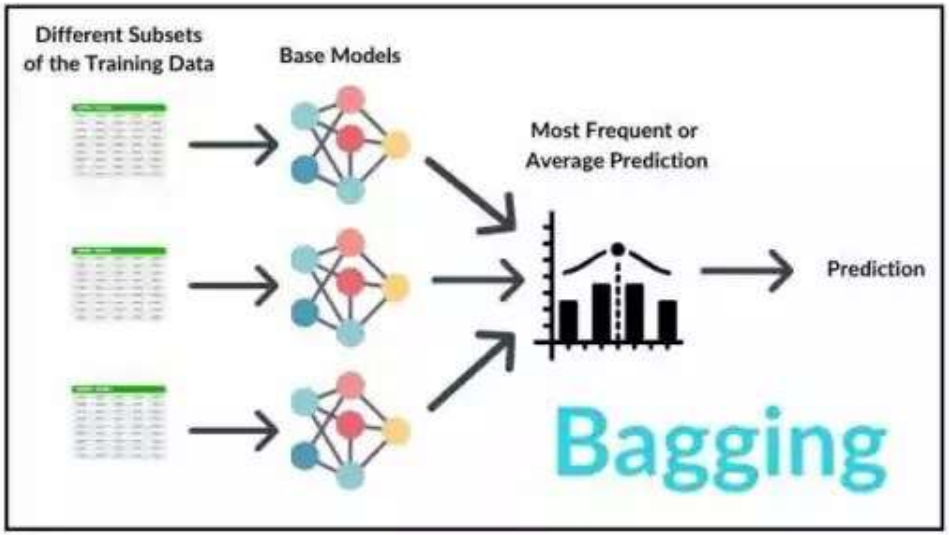
Использование такого разбиения называется *блендинг (blending)*

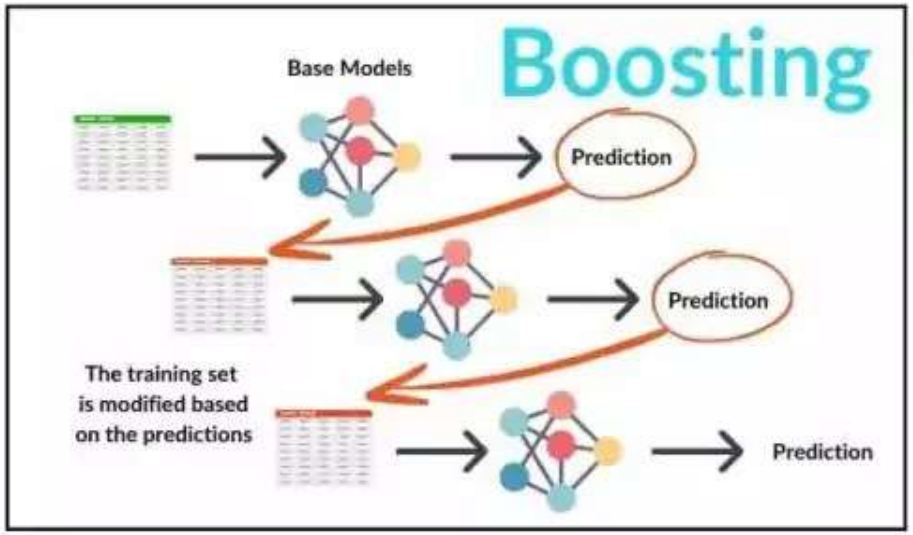
С точки зрения смещения и разброса стекинг не имеет прямого обоснования, так как не минимизирует ни одну из этих компонент. Более того, стекинг не всегда существенно повышает качество лучшего из базовых алгоритмов.

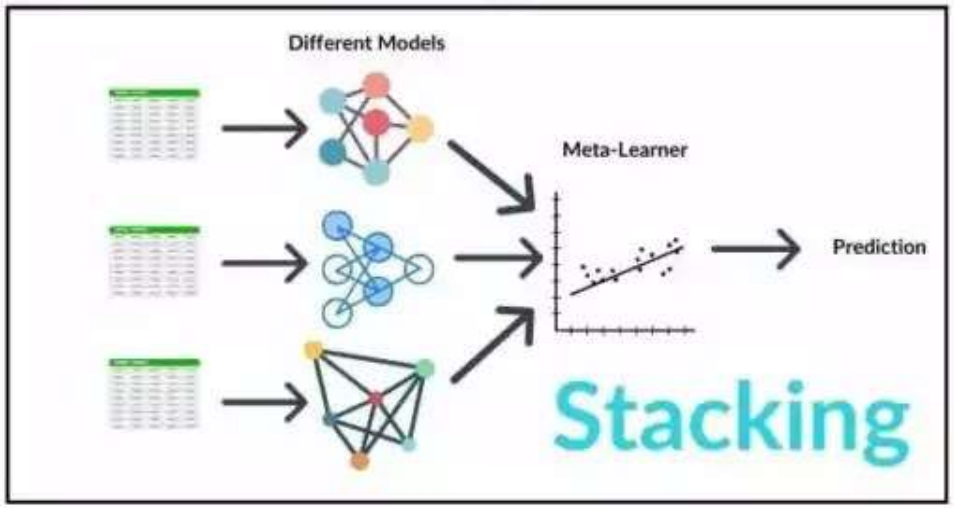
You're a ~~wizard~~ Kaggle Grandmaster, Harry

Источник:
https://alexanderdyakonov.wordpress.com/2017/03/10/c%d1%82%d0%b5%d0%ba%d0%b8%d0%bd%d0%b3-stacking-%d0%b8-%d0%b1%d0%bb%d0%b5%d0%bd%d0%b4%d0%b8%d0%bd%d0%b3-blending/

Boosting vs Bagging vs Stacking







Boosting vs Bagging vs Stacking

	Bagging	Boosting	Stacking
Purpose	Reduce Variance	Reduce Bias	Improve Accuracy
Base Learner Types	Homogeneous	Homogeneous	Heterogeneous
Base Learner Training	Parallel	Sequential	Meta Model
Aggregation	Max Voting, Averaging	Weighted Averaging	Weighted Averaging